The Trichotomy of Regular Property Testing

Gabriel Bathie ✉ ⌂
LaBRI, Université de Bordeaux
DIENS, Paris, France

Nathanaël Fijalkow ✉ ⌂
LaBRI, CNRS, Université de Bordeaux, France

Corto Mascle ✉ ⌂
LaBRI, Université de Bordeaux, France
MPI-SWS, Kaiserslautern, Germany

—— **Abstract** ——

Property testing is concerned with the design of algorithms making a sublinear number of queries to distinguish whether the input satisfies a given property or is far from having this property. A seminal paper of Alon, Krivelevich, Newman, and Szegedy in 2001 introduced property testing of formal languages: the goal is to determine whether an input word belongs to a given language, or is far from any word in that language. They constructed the first property testing algorithm for the class of all regular languages. This opened a line of work with improved complexity results and applications to streaming algorithms. In this work, we show a trichotomy result: the class of regular languages can be divided into three classes, each associated with an optimal query complexity. Our analysis yields effective characterizations for all three classes using so-called minimal blocking sequences, reasoning directly and combinatorially on automata.

1 Introduction

Property testing was introduced by Goldreich, Goldwasser and Ron [19] in 1998: it is the study of randomized approximate decision procedures that must distinguishing objects that have a given property from those that are *far* from having it. Because of this relaxation on the specification, the field focuses on very efficient decision procedures, typically with sublinear (or even constant) running time – in particular, the algorithm does not even have the time to read the whole input.

In a seminal paper, Alon et al. [5] introduced property testing of formal languages: given a language L of finite words, the goal is to determine whether an input word u belongs to the language or is ε-far[1] from it, where ε is the precision parameter. The model assumes random access to the input word: a *query* specifies a position in the word and asks for the letter at that position, and the *query complexity* of the algorithm is the worst-case number of queries it makes to the input. Alon et al. [5] showed a surprising result: under the Hamming distance, all regular languages are testable with $\mathcal{O}(\log^3(\varepsilon^{-1})/\varepsilon)$ queries, where the $\mathcal{O}(\cdot)$ notation hides constants that depend on the language, but, crucially, not on the length of the input word. In that paper, they also identified the class of *trivial* regular languages, those for which the answer is always *yes* or always *no* for sufficiently large n, e.g.

---

[1] Informally, u is ε-far from L means that even by changing an ε-fraction of the letters of u, we cannot obtain a word in L. See Section 2 for a formal definition.

finite languages or the set of words starting with an a, and showed that testing membership in a *non-trivial* regular language requires $\Omega(1/\varepsilon)$ queries.

The results of Alon et al. [5] leave a multiplicative gap of $\mathcal{O}(\log^3(1/\varepsilon))$ between the best upper and lower bounds. We set out to improve our understanding of property testing of regular languages by closing this gap. Bathie and Starikovskaya obtained in 2021 [9] the first improvement over the result of Alon et al. [5] in more than 20 years:

▶ **Fact 1.1** (From [9, Theorem 5]). *Under the edit distance, every regular language can be tested with* $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ *queries.*

Testers under the edit distance are weaker than testers under the Hamming distance, hence this result does not exactly improve the result of Alon et al. [5]. We overcome this shortcoming later in this article: Theorem 4.16 extends the above result to the case of the Hamming distance.

Bathie and Starikovskaya [9] also showed that this upper bound is tight, in the sense that there is a regular language L_0 for which this complexity cannot be further improved, thereby closing the query complexity gap.

▶ **Fact 1.2** (From [9, Theorem 15]). *There is a regular language* L_0 *with query complexity* $\Omega(\log(\varepsilon^{-1})/\varepsilon)$ *under the edit distance[2], for all small enough* $\varepsilon > 0$.

Furthermore, it is easy to find specific non-trivial regular languages for which there is an algorithm using only $\mathcal{O}(1/\varepsilon)$ queries, e.g. $L = a^*$ over the alphabet $\{a, b\}$, $L = (ab)^*$ or $L = (aa + bb)^*$.

Hence, these results combined with those of Alon et al. [5] show that there exist trivial languages (that require 0 queries for large enough n), *easy* languages (with query complexity $\Theta(1/\varepsilon)$) and *hard* languages (with query complexity $\Theta(\log(\varepsilon^{-1})/\varepsilon)$). This raises the question of whether there exist languages with a different query complexity (e.g. $\Theta(\log\log(\varepsilon^{-1})/\varepsilon)$), or if every regular is either trivial, easy or hard. This further asks the question of giving a characterization of the languages that belong to each class, inspired by the recent success of exact characterizations of the complexity of sliding window [16] recognition and dynamic membership [7] of regular languages.

In this article, we answer both questions: we show a trichotomy of the complexity of testing regular languages under the Hamming distance[3], showing that there are only the three aforementioned complexity classes (trivial, easy and hard), we give a characterization of all three classes using a combinatorial object called *blocking sequences*, and show that this characterization can be decided in polynomial space (and that it is complete for PSPACE). This trichotomy theorem closes a line of work on improving query complexity for property testers and identifying easier subclasses of regular languages.

1.1   Related work

A very active branch of property testing focuses on graph properties, for instance one can test whether a given graph appears as a subgraph [3] or as an induced subgraph [4], and more generally every monotone graph property can be tested with one-sided error [6]. Other families of objects heavily studied under this algorithmic paradigm include probabilistic

---

[2] Note that, as opposed to testers, lower bounds for the edit distance are stronger than lower bounds of the Hamming distance.

[3] We consider one-sided error testers, also called testing with perfect completeness, see definitions below.

distributions [25, 11] combined with privacy constraints [2], numerical functions [10, 28], and programs [13, 12]. We refer to the book of Goldreich [18] for an overview of the field of property testing.

Testing formal languages. Building upon the seminal work of Alon et al. [5], Magniez et al. [23] gave a tester using $\mathcal{O}(\log^2(\varepsilon^{-1})/\varepsilon)$ queries for regular languages under the edit distance with moves, and François et al. [15] gave a tester using $\mathcal{O}(1/\varepsilon^2)$ queries for the case of the weighted edit distance. Alon et al. [5] also show that context-free languages cannot be tested with a constant number of queries, and subsequent work has considered testing specific context-free languages such as the DYCK languages [26, 14] or regular tree languages [23]. Property testing of formal languages has been investigated in other settings: Ganardi et al. [17] studied the question of testing regular languages in the so-called "sliding window model", while others considered property testing for subclasses of context-free languages in the streaming model: Visibly Pushdown languages [15], DYCK languages [21, 22, 24] or DLIN and LL(k) [8]. A recent application of property testing of regular languages was to detect race conditions in execution traces of distributed systems [30].

1.2 Main result and overview of the paper

We start with a high-level presentation of the approach, main result, and key ideas. In this section we assume familiarity with standard notions such as finite automata; we will detail notations in Section 2.

Let us start with the notion of a property tester for a language L: the goal is to determine whether an input word u belongs to the language L, or whether it is ε-far from it. We say that u of length n is ε-*far from L* with respect to a metric d over words if all words $v \in L$ satisfy $d(u, v) \geq \varepsilon n$, written $d(u, L) \geq \varepsilon n$. Throughout this work and unless explicitly stated otherwise, we will consider the case where d is the Hamming distance, defined for two words u and v as the number of positions at which they differ if they have the same length, and as $+\infty$ otherwise. In that case, $d(u, L) \geq \varepsilon n$ means that one cannot change a proportion ε of the letters in u to obtain a word in L. We assume random access to the input word: a query specifies a position in the word and asks for the letter in this position. A ε-*property tester* (or for short, simply a *tester*) T for a language L is a randomized algorithm that, given an input word u of length n, always answers "yes" if $u \in L$ and answers "no" with probability bounded away from 0 when u is ε-far from L. As in previous works on this topic, we measure the complexity of a tester by its *query complexity*. It is the maximum number of queries that T makes on an input of length n, as a function of n and ε, in the worst case over all words of length n and all possible random choices.

We can now formally define the classes of *trivial, easy* and *hard* regular languages.

▶ **Definition 1.3** (Hard, easy and trivial languages). *Let L be a regular language. We say that:*
- *L is* hard *if the optimal query complexity for a property tester for L is $\Theta(\log(\varepsilon^{-1})/\varepsilon)$.*
- *L is* easy *if the optimal query complexity for a property tester for L is $\Theta(1/\varepsilon)$.*
- *L is* trivial *if there exists $\varepsilon_0 > 0$ such that for all positive $\varepsilon < \varepsilon_0$, there is a property tester and some $n \in \mathbb{N}$ such that the tester makes 0 queries on words of length $\geq n$.*

▶ **Remark 1.4.** If L is finite, then it is trivial: since there is a bound B on the lengths of its words, a tester can reject words of length at least $n_0 = B + 1$ without querying them. For that reason, we only consider *infinite* languages in the rest of the article.

Our characterization of those three classes uses the notion of *blocking sequence* of a language L. Intuitively, they are sequences of words such that finding those words as factors of a word w proves that w is not in L. We also define a partial order on them, which gives us a notion of *minimal* blocking sequence.

▶ **Theorem 1.5.** *Let L be an infinite regular language recognized by an NFA $\mathcal{A}$ and let $\mathsf{MBS}(\mathcal{A})$ denote the set of minimal blocking sequences of $\mathcal{A}$. The complexity of testing L is characterized by $\mathsf{MBS}(\mathcal{A})$ as follows:*

1. *L is trivial if and only if $\mathsf{MBS}(\mathcal{A})$ is empty;*
2. *L is easy if and only if $\mathsf{MBS}(\mathcal{A})$ is finite and nonempty;*
3. *L is hard if and only if $\mathsf{MBS}(\mathcal{A})$ is infinite.*

In the case where L is recognised by a strongly connected automaton, blocking sequences can be replaced by *blocking factors*. A blocking factor is a single word that is not a factor of any word in L.

Section 2 defines the necessary terms and notations. The rest of the paper is structured as follows. In Sections 3 and 4, we delimit the set of hard languages, that is, the ones that require $\Theta(\log(\varepsilon^{-1})/\varepsilon)$ queries. More precisely, Section 3 focuses on the subcase of languages defined by *strongly connected automata*. First, we combine the ideas of Alon et al. [5] with those presented in [9] to obtain a property tester that uses $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries for any language with a strongly connected automaton, under the Hamming distance. Second, we show that if the language of a strongly connected automaton has infinitely many blocking factors then it requires $\Omega(\log(\varepsilon^{-1})/\varepsilon)$ queries. This result generalizes the result of Bathie and Starikovskaya [9], which was for a single language, to all regular languages with infinitely many minimal blocking factors. We use Yao's minimax principle [31]: this involves constructing a hard distribution over inputs, and showing that any deterministic property testing algorithms cannot distinguish between positive and negative instances against this distribution.

In Section 4, we extends those results to all automata. The interplay with the previous section is different for the upper and the lower bound.For the upper bound of $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries, we use a natural but technical extension of the proof in the strongly connected case. Note that this result is an improvement over the result of Bathie and Starikovskaya [9], which works under the edit distance, and testers for the Hamming distance are also testers for the edit distance. For the lower bound of $\Omega(\log(\varepsilon^{-1})/\varepsilon)$ queries for languages with infinitely many minimal blocking sequences, we reduce to the strongly connected case. The main difficulty is that it is not enough to consider strongly connected components in isolation: there exists finite automata that contain a strongly connected component that induces a hard language, yet the language of the whole automaton is easy. We solve this difficulty by carefully defining the notion of minimality for a blocking sequence.

Section 5 completes the trichotomy, by characterising the easy and trivial languages. We show that languages of automata with finitely many blocking sequences can be tested with $\mathcal{O}(1/\varepsilon)$ queries. We also prove that if an automaton has at least one blocking sequence, then it requires $\Omega(1/\varepsilon)$ queries to be tested, by showing that the languages that our notion of trivial language coincides with the one given by Alon et al. [5]. By contrast, we show that automata without blocking sequences recognize trivial languages.

Once we have the trichotomy, it is natural to ask whether it is effective: given an automaton $\mathcal{A}$, can we determine if its language is trivial, easy or hard? The answer is yes, and we show in Section 6 that all three decision problems are PSPACE-complete, even for strongly connected automata.

2 Preliminaries

Words and automata. We write Σ^* (resp. Σ^+) for the set of finite words (resp. non-empty words) over the alphabet Σ. The length of a word u is denoted $|u|$, and its ith letter is denoted $u[i]$. The empty word is denoted γ. Given $u \in \Sigma^*$ and $0 \le i, j \le |u| - 1$, define $u[i..j]$ as the word $u[i]u[i+1]\ldots u[j]$ if $i \le j$ and γ otherwise. Further, $u[i..j)$ denotes the word $u[i..j-1]$. A word w is a *factor* (resp. *prefix*, *suffix*) of u is there exist indices i, j such that $w = u[i..j]$ (resp. with $i = 0, j = |u| - 1$). We use $w \preccurlyeq u$ to denote "w is a factor of u". Furthermore, if w is a factor of u and $w \ne u$, we say that w is a *proper factor* of u.

A *nondeterministic finite automaton* (NFA) $\mathcal{A}$ is a transition system defined by a tuple $(Q, \Sigma, \delta, q_0, F)$, with Q a finite set of states, Σ a finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ the transition function, $q_0 \in Q$ the initial state and $F \subseteq Q$ the set of final states. The semantics is as usual [27]. When there is a path from a state p to a state q in $\mathcal{A}$, we say that q is reachable from p and that p is co-reachable from q. In this work, we assume w.l.o.g. that all NFA $\mathcal{A}$ are *trim*, i.e., every state is reachable from the initial state and co-reachable from some final state.

Property testing.

▶ **Definition 2.1.** *Let L be a language, let u be a word of length n, let $\varepsilon > 0$ be a precision parameter and let $d : \Sigma^* \times \Sigma^* \to \mathbb{N} \cup \{+\infty\}$ be a metric. We say that the word u is ε-far from L w.r.t. d if $d(u, L) \ge \varepsilon n$, where*

$$d(u, L) := \inf_{v \in L} d(u, v).$$

We assume random access to the input word: a query specifies a position in the word and asks for the letter in this position.

Throughout this work and unless explicitly stated otherwise, we will consider the case where d is the Hamming distance, defined for two words u and v as the number of positions at which they differ if they have the same length, and as $+\infty$ otherwise. In that case, $d(u, L) \ge \varepsilon n$ means that one cannot change an ε-fraction of the letters in u to obtain a word in L.

A ε-*property tester* (or simply a *tester*) T for a language L is a randomized algorithm that, given an input word u, always answers "yes" if $u \in L$ and answers "no" with probability bounded away from 0 when u is ε-far from L.

▶ **Definition 2.2.** *A property tester for the language L with precision $\varepsilon > 0$ is a randomized algorithm T that, for any input u of length n, given random access to u, satisfies the following properties:*

$$\text{if } u \in L, \text{ then } T(u) = 1,$$
$$\text{if } u \text{ is } \varepsilon\text{-far from } L, \text{ then } \mathbb{P}\left(T(u) = 0\right) \ge 2/3.$$

The query complexity of T is a function of n and ε that counts the maximum number of queries that T makes over all inputs of length n and over all possible random choices.

We measure the complexity of a tester by its *query complexity*. Let us emphasize that throughout this article we focus on so-called "testers with perfect completeness", or "one-sided error": if a word is in the language, the tester answers positively (with probability 1). In particular our characterization applies to this class. Because they are based on

the notion of blocking factors that we will discuss below, all known testers for regular languages [5, 23, 15, 9] have perfect completeness.

In this paper, we assume that the automaton $\mathcal{A}$ that describes the tested language L is *fixed*, and not part of the input. Therefore, we consider its number of states m as a constant.

Graphs and periodicity.   We now recall tools introduced by Alon et al. [5] to deal with periodicity in finite automata.

Let $G = (V, E)$ with $E \subseteq V^2$ be a directed graph. A *strongly connected component* (or SCC) of G is a maximal set of vertices that are all reachable from each other. It is *trivial* if it contains a single state with no self-loop on it. We say that G is *strongly connected* if its entire set of vertices is an SCC.

The period $\lambda = \lambda(G)$ of a non-trivial strongly connected graph G is the greatest common divisor of the length of the cycles in G. Following the work of Alon et al. [5], we will use the following property of directed graphs.

▶ **Fact 2.3** (From [5, Lemma 2.3]). *Let $G = (V, E)$ be a non-empty, non-trivial, strongly connected graph with finite period $\lambda = \lambda(G)$. Then there exists a partition $V = Q_0 \sqcup \ldots \sqcup Q_{\lambda-1}$ and a reachability constant $\rho = \rho(G)$ that does not exceed $3|V|^2$ such that:*

1. *For every $0 \leq i, j \leq \lambda - 1$ and for every $s \in Q_i, t \in Q_j$, the length of any directed path from s to t in G is equal to $(j - i) \mod \lambda$.*
2. *For every $0 \leq i, j \leq \lambda - 1$, for every $s \in Q_i, t \in Q_j$ and for every integer $r \geq \rho$, if $r = (j - i) \pmod{\lambda}$, then there exists a directed path from u to v in G of length r.*

The sets $(Q_i : i = 0, \ldots, \lambda - 1)$ are the *periodicity classes* of G. In what follows, we will slightly abuse notation and use Q_i even when $i \geq \lambda$ to mean $Q_{i \pmod \lambda}$.

An automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ defines an underlying graph $G = (Q, E)$ where $E = \{(p, q) \in Q^2 \mid \exists a \in \Sigma : q \in \delta(p, a)\}$. In what follows, we naturally extend the notions defined above to finite automata through this graph G. Note that the numbering of the periodicity classes is defined up to a shift mod λ: we can thus always assume that Q_0 is the class that contains the initial state q_0. The period of $\mathcal{A}$ is written $\lambda(\mathcal{A})$.

Positional words and positional languages.   Consider the language $L_3 = (ab)^*$. The word $v = ab$ can appear as a factor of a word $u \in L_3$ if v occurs at an even position (e.g. position 0, 2, etc.) in u. However, if v occurs at an *odd* position in u, then u cannot be in L_3. Therefore, v can be used to witness that u is not in L_3, but only if we find it at an odd position. This example leads us to introducing *p-positional words*, which additionally encode information about the index of each letter modulo an integer p.

More generally, we will associate to each regular language a period λ, and working with λ-positional words will allow us to define blocking factors in a position-dependent way without explicitly considering the index at which the factor occurs.

▶ **Definition 2.4** (Positional words). *Let p be a positive integer. A p-positional word is a word over the alphabet $\mathbb{Z}/p\mathbb{Z} \times \Sigma$ of the form $(n \pmod p, a_0)((n+1) \pmod p, a_1) \cdots ((n+\ell) \pmod p, a_\ell)$ for some non-negative integer n. If $u = a_0 \cdots a_\ell$, we write $(n : u)$ to denote this word.*

With this definition, if $u = abcd$ and we consider the 2-positional word $\tau = (0 : u)$, the factor bc appears at position 1 in u and is mapped to the factor $\mu = (1, a)(0, b)$. In this case, even when taking factors of μ, we still retain the (congruence classes of the) indices in the original word τ.

Any strongly connected finite automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ can naturally be extended into an automaton $\widehat{\mathcal{A}}$ over $\lambda(\mathcal{A})$-positional words with $\lambda(\mathcal{A})|Q|$ states. It suffices to keep track in the states of the current state of $\mathcal{A}$ and the number of letters read modulo $\lambda(\mathcal{A})$.

We call the language recognized by $\widehat{\mathcal{A}}$ the *positional language of* $\mathcal{A}$, and denote it $\mathcal{TL}(\mathcal{A})$. This definition is motivated by the following property:

▶ **Property 2.5.** *For any word $u \in \Sigma^*$, we have $u \in \mathcal{L}(\mathcal{A})$ if and only if $(0 : u) \in \mathcal{TL}(\mathcal{A})$.*

Positional words make it easier to manipulate factors with positional information, hence we phrase our property testing results in terms of positional languages. Notice that a property tester for $\mathcal{TL}(\mathcal{A})$ immediately gives a property tester for $\mathcal{L}(\mathcal{A})$, as one can simulate queries to $(0 : u)$ with queries to u by simply pairing the index of the query modulo $\lambda(\mathcal{A})$ with its result.

3   Hard Languages for Strongly Connected NFAs

Before considering the case of arbitrary NFAs, we first study the case of strongly connected NFAs, which are NFAs such that for any pair of states $p, q \in Q$, there exists a word w such that $p \xrightarrow{w} q$. We will later generalize the results of this section to all NFAs.

We show that the query complexity of the language of such an NFA $\mathcal{A}$ can be characterized by the cardinality of the set of *minimal blocking factors* of $\mathcal{A}$, which are factor-minimal $\lambda(\mathcal{A})$-positional words that witness the fact that a word does not belong to $\mathcal{TL}(\mathcal{A})$. In this section, we consider a fixed NFA $\mathcal{A}$ and simply use "positional words" to refer to λ-positional words, where $\lambda = \lambda(\mathcal{A})$ is the period of $\mathcal{A}$.

▶ **Definition 3.1** (Blocking factors). *Let $\mathcal{A}$ be a strongly connected NFA. A positional word τ is a* blocking factor *of $\mathcal{A}$ if for any other positional word μ we have $\tau \preccurlyeq \mu \Rightarrow \mu \notin \mathcal{TL}(\mathcal{A})$.*

Further, we say that τ is a minimal *blocking factor of $\mathcal{A}$ if no proper factor of τ is a blocking factor of $\mathcal{A}$. We use $\mathsf{MBF}(\mathcal{A})$ to denote the set of all minimal blocking factors of $\mathcal{A}$.*

Intuitively and in terms of automata, the positional word $(i : u)$ is blocking for $\mathcal{A}$ if it does not label any transition in $\mathcal{A}$ labeled by u starting from a state of Q_i. (This property is formally established later in Lemma 3.5.) The main result of this section is the following:

▶ **Theorem 3.2.** *Let L be an infinite language recognised by a strongly connected NFA $\mathcal{A}$. If $\mathsf{MBF}(\mathcal{A})$ is infinite, then L is hard, i.e., it has query complexity $\Theta(\log(\varepsilon^{-1})/\varepsilon)$*

This result gives both an upper bound of $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ and a lower bound of $\Omega(\log(\varepsilon^{-1})/\varepsilon)$ on the query complexity of a tester for L: we prove the upper bound in Section 3.2 and the lower bound in Section 3.3.

3.1   Positional words, blocking factors and strongly connected NFAs

We first establish some properties of positional words that will help us ensure that we are creating well-formed positional words, that is, positional words where the index i of a letter $(i : a)$ is equal to $j + 1 \pmod{\lambda}$, where j is the index of the previous letter. In Section 3.2, we highlight the connection between property testing and blocking factors in strongly connected NFAs.

We start with the following properties, which are consequences of Fact 2.3.

▶ **Corollary 3.3.** *Let n be a nonnegative integer, let w be a word of length n. If for some states $p \in Q_i, q \in Q_j$ of $\mathcal{A}$ we have $p \xrightarrow{w} q$, then the indices i, j satisfy the equation*

$$j - i = |w| \pmod{\lambda}$$

▶ **Corollary 3.4.** *Let $\tau = (i : u)$ and $\mu = (j : v)$ be positional words. If $\tau \preccurlyeq \mu$, then there exists positional words η, η' with $|\eta| = i - j \pmod{\lambda}$ such that $\mu = \eta \tau \eta'$. In particular, this implies that there exists words w, w' with $|w| = i - j \pmod{\lambda}$ such that $v = wuw'$.*

These properties allows us to formalize the intuition we gave earlier about blocking factors.

▶ **Lemma 3.5.** *A positional word $\tau = (i : u)$ is a blocking factor for $\mathcal{A}$ iff for every states $p \in Q_i, q \in Q$, we have $p \xrightarrow{u} \!\!\!\!\!/ \ \ q$.*

Proof. We first show that if there exists states $p \in Q_i, q \in Q$ such that $p \xrightarrow{u} q$, then τ is not blocking, i.e. there exists $\mu \in \mathcal{TL}(\mathcal{A})$ such that $\tau \preccurlyeq \mu$. As $\mathcal{A}$ is strongly connected, there exist positional words η, η' such that $q_0 \xrightarrow{\eta} p$ and $q \xrightarrow{\eta'} q_f$ for some $q_f \in F$. By Fact 2.3, the positional word $\mu = \eta \tau \eta'$ is well formed. Furthermore, it labels a transition from q_0 to q_f, hence it is in $\mathcal{TL}(\mathcal{A})$, and τ is not blocking.

For the converse, assume that τ is non-blocking: we show that there exists two states $p \in Q_i, q \in Q$ such that $p \xrightarrow{u} q$. As τ is non-blocking, there exists a positional word $\mu = (0 : w)$ such that $\tau \preccurlyeq \mu$ and there exists a final state r such that $q_0 \xrightarrow{\mu} r$, and equivalently, $q_0 \xrightarrow{w} r$. By Corollary 3.4, since $\tau \preccurlyeq \mu$, there exists words v, v' such that $w = vuv'$ and the length of v is equal to i modulo λ. In particular, the path $q_0 \xrightarrow{w} r$ can be decomposed into $q_0 \xrightarrow{v} p \xrightarrow{u} q \xrightarrow{w} r$, and we have $p \xrightarrow{u} q$. It only remains to show that p is in Q_i: this follows by Corollary 3.3 since $|v| = i \pmod{\lambda}$.   ◀

Next, we show that the Hamming distance between u and $\mathcal{L}(\mathcal{A})$ is the same as the (Hamming) distance between $(0 : u)$ and $\mathcal{TL}(\mathcal{A})$.

▷ Claim 3.6.   For any word $u \in \Sigma^*$, we have $d(u, \mathcal{L}(\mathcal{A})) = d((0 : u), \mathcal{TL}(\mathcal{A}))$.

Proof. The $\leq$ part is straightforward. For the reverse inequality, if suffices to see that in any minimal substitution sequence from $(0 : u)$ to a positional word in $\mathcal{TL}(\mathcal{A})$, no operation changes only the index of an (index, letter) pair.   ◁

The above claim allows us to interchangeably use the statements "u is ε-far from $\mathcal{L}(\mathcal{A})$" and "$(0 : u)$ is ε-far from $\mathcal{TL}(\mathcal{A})$".

3.2   An efficient property tester for strongly connected NFAs.

In this section, we show that for any strongly connected NFA $\mathcal{A}$, there exists an ε-property tester for $\mathcal{L}(\mathcal{A})$ that uses $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries.

▶ **Theorem 3.7.** *Let $\mathcal{A}$ be a strongly connected NFA. For any $\varepsilon > 0$, there exists an ε-property tester for $\mathcal{L}(\mathcal{A})$ that uses $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries.*

Our proof is similar to the one given in [9], with one notable technical improvement: we use a new method for sampling factors in u, which greatly simplifies the correctness analysis.

3.2.1   An efficient sampling algorithm

We first introduce a sampling algorithm (Algorithm 1) that uses few queries and has a large probability of finding at least one factor from a large set $\mathcal{S}$ of disjoint "special" factors. Using this algorithm on a large set of disjoint blocking factors gives us an efficient property tester for strongly connected NFAs. We will re-use this sampling procedure later in the case of general NFAs (Theorem 4.16).

The procedure is fairly simple: the algorithm samples factors of various lengths in u at random. On the other hand, the correctness of the tester is far from trivial. The lengths and the number of factors of each length are chosen so that the number of queries is minimized and the probability of finding a "special" factor is maximized, regardless of their repartition in u. (In what follows, the "special" factors are blocking factors.)

■ **Algorithm 1** Efficient generic sampling algorithm

1: **function** ONESAMPLE(u, ℓ)
2:      $i \leftarrow$ UNIFORM$(0, n - 1)$
3:      $l \leftarrow \max(i - \ell, 0), r \leftarrow \min(i + \ell, n - 1)$
4:      **return** $u[l . . r]$
5: **function** SAMPLER(u, N, L)
6:      $n \leftarrow |u|$
7:      $\beta \leftarrow n/N$
8:      $T \leftarrow \lceil \log(L) \rceil$
9:      $F \leftarrow \emptyset$
10:      **for** $t = 0$ to T **do**
11:          $\ell_t \leftarrow 2^t, r_t \leftarrow \lceil 2 \ln(3) \beta / \ell_t \rceil$
12:          **for** $i = 0$ to r_t **do**
13:              $\mathcal{F} \leftarrow \mathcal{F} \cup \{$ONESAMPLE$(u, \ell_t)\}$
14:      **return** $\mathcal{F}$

▷ **Claim 3.8.** A call to SAMPLER(u, N, L) (Algorithm 1) makes $\mathcal{O}(n \log(L)/N)$ queries to u.

Proof. A call to ONESAMPLE(u, ℓ_t) makes at most $2\ell_t$ queries to u. Furthermore, the function SAMPLER(u, N, L) makes $r_t = 2 \ln(3) \cdot \beta / \ell_t = 2 \ln(3) \cdot n/(N\ell_t)$ calls to ONESAMPLE(u, ℓ_t) for each $t = 0, \ldots, T$, where $T = \lceil \log(L) \rceil$. This adds up to

$$\sum_{t=0}^{T} r_t \cdot \ell_t = \lceil \log(L) \rceil \cdot 2 \ln(3) \cdot n/N = \mathcal{O}(n \log(L)/N)$$

queries to u. ◁

▶ **Lemma 3.9.** *Let u be a word of length n, and consider a set $\mathcal{S}$ containing at least N disjoint factors of u, each of length at most L. A call to the function SAMPLER(u, N, L) (Algorithm 1) returns a set $\mathcal{F}$ of factors of u such that there exists a word of $\mathcal{S}$ that is a factor of some word of $\mathcal{F}$, with probability at least $2/3$.*

Proof. We conceptually divide the blocking factors in $\mathcal{S}$ into different categories depending on their length: let $T = \lceil \log(L) \rceil$, and for $t = 0, \ldots, T$, let S_t denote the subset of $\mathcal{S}$ which contains factors of length at most $\ell_t = 2^t$. We then carefully analyze the probability that randomly sampled factors of length $2\ell_t$ contains a factor from S_t, and show that over all t, at least one sampled factor contains a factor of $\mathcal{S}$, with probability at least $2/3$.

▷ **Claim 3.10.** If in a call to ONESAMPLE, the value i is such that there exists indices l and r such that $l \leq i \leq r$ and $u[l, r]$ contains a factor in $\mathcal{S}$, then the set $\mathcal{F}$ returned by the algorithm has the desired property.

As the factors given in $\mathcal{S}$ are disjoint, the probability p_t that the factor returned by ONE-SAMPLE contains a factor from $\mathcal{S}$ is lower bounded by $p_t \geq \frac{1}{n} \sum_{v \in S_t} |v|$. The ONESAMPLE

function is called $r_t = 2\ln(3)\beta/\ell_t$ times independently for each t, hence the probability p that the algorithm samples a factor containing a factor from $\mathcal{S}$ satisfies the following:

$$(1-p) = \prod_{t=0}^{T}(1-p_t)^{r_t} \le \exp\left(-\sum_{t=0}^{T} p_t r_t\right)$$

$$\le \exp\left(-\frac{2\ln(3)\beta}{n}\sum_{t=0}^{T}\frac{1}{\ell_t}\sum_{v\in S_t}|v|\right)$$

$$= \exp\left(-\frac{2\ln(3)\beta}{n}\sum_{v\in\mathcal{S}}|v|\sum_{t=\lceil\log|v|\rceil}^{T}2^{-t}\right).$$

Now, inverting the order of summation, and lower bounding the sum of powers of 2 by its first term, we obtain:

$$(1-p) \le \exp\left(-\frac{2\ln(3)\beta}{n}\sum_{v\in\mathcal{S}}|v|\cdot 2^{-\lceil\log|v|\rceil}\right)$$

$$\le \exp\left(-\frac{2\ln(3)\beta}{n}\sum_{v\in\mathcal{S}}|v|\frac{1}{2|v|}\right)$$

$$= \exp\left(-\frac{2\ln(3)\beta}{n}\cdot\frac{|\mathcal{S}|}{2}\right) \le \exp\left(-\frac{\ln(3)\beta N}{n}\right)$$

$$= \exp\left(-\ln(3)\right) = 1/3$$

It follows that $p \ge 2/3$, which concludes the proof. ◀

3.2.2 The tester

The algorithm for Theorem 3.7 is given in Algorithm 2.

◼ **Algorithm 2** Generic ε-property tester that uses $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries

---

1: **function** TESTER(u, ε)
2:     $n \leftarrow |u|, m \leftarrow |Q|$
3:     $L \leftarrow 12m^2/\varepsilon$
4:     **if** $\mathcal{L}(\mathcal{A}) \cap \Sigma^n = \emptyset$ **then**
5:         Reject
6:     **else if** $n < L$ **then**
7:         Query all of u and run $\mathcal{A}$ on it
8:         Accept if and only if $\mathcal{A}$ accepts
9:     **else**
10:        $\mathcal{F} \leftarrow$ SAMPLER$((0 : u), n/L, L)$
11:        Reject if and only if $\mathcal{F}$ contains a blocking factor for $\mathcal{A}$.

We now show that Algorithm 2 is a property tester for $\mathcal{L}(\mathcal{A})$ that uses $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries. In what follows, we use n to denote the length of the input word u and m to denote the number of states of $\mathcal{A}$.

▷ Claim 3.11.   The tester given in Algorithm 2 makes $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries to u.
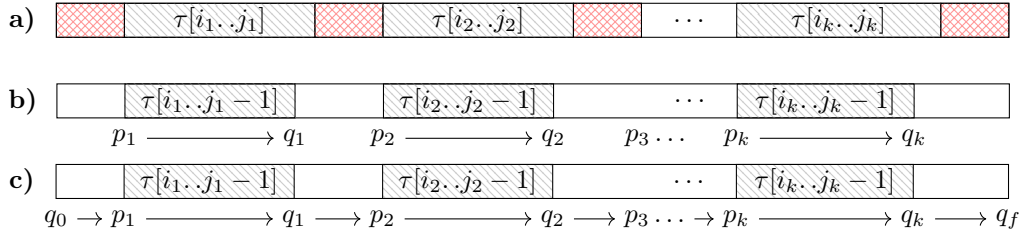
Proof. If $n \le L$, then the tester makes $n \le L = \mathcal{O}(1/\varepsilon)$ queries, and the claim holds. Otherwise, the number of queries is given by the call to SAMPLER$(u, n/L, L)$: by Claim 3.8, this uses $\mathcal{O}(\frac{n\log L}{n/L}) = \mathcal{O}(L\log L) = \mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries. ◀

Alon et al. [5, Lemma 2.6] first noticed that if a word u is ε-far from $\mathcal{L}(\mathcal{A})$, then it contains $\Omega(\varepsilon n)$ short factors that witness the fact that u is not in $\mathcal{L}(\mathcal{A})$. We start by translating the lemma of Alon et al. on "short witnesses" to the framework of blocking factors. More precisely, we show that if u is ε-far from $\mathcal{L}(\mathcal{A})$, then $(0 : u)$ contains many disjoint (i.e. non-overlapping) blocking factors.

▶ **Lemma 3.12.** *Let $\varepsilon > 0$, let u be a word of length $n \geq 6m^2/\varepsilon$ and assume that $\mathcal{L}(\mathcal{A})$ contains at least one word of length n. If $\tau = (0 : u)$ is ε-far from $\mathcal{TL}(\mathcal{A})$, then τ contains at least $\varepsilon n/(6m^2)$ disjoint blocking factors.*

Proof. We build a set $\mathcal{P}$ of disjoint blocking factors of τ as follows: we process u from left to right, starting at index $i_1 = \rho$, where ρ is the reachability constant of $\mathcal{A}$ (see Fact 2.3). Next, at iteration t, set j_t to be the smallest integer greater than or equal to i_t and smaller than $n - \rho$ such that $\tau[i_t..j_t]$ is a blocking factor. If there is no such integer, we stop the process. Otherwise, we add $\tau[i_t..j_t + \rho - 1]$ to the set $\mathcal{P}$, and iterate starting from the index $i_{t+1} = j_t + \rho$.

Let k denote the size of $\mathcal{P}$. We will show that we can substitute at most $3(k+1)m^2$ positions in τ to obtain a word in $\mathcal{TL}(\mathcal{A})$. (See Figure 1 for an illustration of this construction.) Using the assumption that τ is ε-far from $\mathcal{TL}(\mathcal{A})$ (which follows from Claim 3.6) will give us the desired bound on k.

Figure 1 a) The decomposition process returns k factors $\tau[i_1, j_t], \ldots, \tau[i_k, j_k]$ (represented as diagonally hatched in gray regions), separated together and with the start of the text by padding regions of $\rho - 1$ letters (red crosshatched regions). **b)** If we exclude the last letter of each blocking factor, we obtain factors that label transitions between some pair of states p_t, q_t for each $t = 1, \ldots, k$. **c)** We use the padding regions to bridge between consecutive factors as well as the start and end of the word.

For every t, we chose j_t to be minimal so that $\tau[i_t..j_t]$ is blocking, hence $\tau[i_t..j_t - 1]$ is not blocking, and therefore $\tau[i_t..j_t - 1]$ labels a run from some state $p_t \in Q_{i_t}$ to some state $q_t \in Q_{j_t}$. Therefore, using the strong connectivity of $\mathcal{A}$ and Fact 2.3, we can substitute the letters in $\tau[j_t..j_t + \rho - 1]$ to obtain a factor that labels a transition from q_t to p_{t+1}. After this transformation, the word $\tau[i_t..j_t + \rho - 1]$ labels a transition from p_t to p_{t+1}. Using the ρ letters at the start and the end of the word, we add transitions from an initial state to p_1 and from q_k to a final state: the assumption that $\mathcal{L}(\mathcal{A})$ contains a word of length n ensures that Q_n contains a final state, hence this is always possible. The resulting word is in $\mathcal{TL}(\mathcal{A})$ and was obtained from τ using $(k+1)\rho \leq 3(k+1)m^2$ substitutions. As τ is ε-far from $\mathcal{TL}(\mathcal{A})$, we obtain the following bound on k:

$$3(k+1)m^2 \geq \varepsilon n \implies k \geq \frac{\varepsilon n}{3m^2} - 1$$
$$\implies k \geq \frac{\varepsilon n}{6m^2}$$

The last implication uses the assumption that $n \geq 6m^2/\varepsilon$.   ◀

Next, we show that if u is ε-far from $\mathcal{L}(\mathcal{A})$, then $(0:u)$ contains $\Omega(\varepsilon n)$ blocking factors, each of length $\mathcal{O}(1/\varepsilon)$.

▶ **Lemma 3.13.** *Let $\varepsilon > 0$, let u be a word of length $n \geq 6m^2/\varepsilon$ and assume that $\mathcal{L}(\mathcal{A})$ contains at least one word of length n. If u is ε-far from $\mathcal{L}(\mathcal{A})$, then the positional word $(0:u)$ contains at least $\varepsilon n/(12m^2)$ disjoint blocking factors of length at most $12m^2/\varepsilon$.*

Proof. Let $u, \mathcal{A}$ be a word and an automaton satisfying the above hypotheses. By Lemma 3.12, $(0:u)$ contains at least $\varepsilon n/(6m^2)$ *disjoint* blocking factors. As these factors are disjoint, at most half of them (that is, $\varepsilon n/(12m^2)$ of them) can have length greater than $12m^2/\varepsilon$, as the sum of their lengths cannot exceed n.   ◀

Proof of Theorem 3.7. First, note that if $u \in \mathcal{L}(\mathcal{A})$, $(0:u)$ cannot contain a blocking factor for $\mathcal{A}$, hence Algorithm 2 always accepts u. Next, if $\mathcal{L}(\mathcal{A}) \cap \Sigma^n$ is empty or if $|u| \leq L = 12m^2/\varepsilon$, the tester has the same output as $\mathcal{A}$, hence it is correct.

In the remaining case, u is long enough and ε-far from $\mathcal{L}(\mathcal{A})$, hence Lemma 3.13 gives us a large set of short blocking factors in $(0:u)$: this is exactly what the SAMPLER function needs to find at least one factor containing a blocking factor with probability at least $2/3$. More precisely, by Lemma 3.13, $(0:u)$ contains at least $\varepsilon n/(12m^2) = n/L$ blocking factors of length at most $L = 12m^2/\varepsilon$, hence the conditions of Lemma 3.9 are satisfied.

As a factor containing a blocking factor is also a blocking factor, the set $\mathcal{F}$ computed on line 10 of Algorithm 2 contains at least one blocking factor with probability at least $2/3$, and Algorithm 2 satisfies Definition 2.2.   ◀

3.3   Lower bound from infinitely many minimal blocking factors

We now show that languages with infinitely many *minimal* blocking factors are hard, i.e. any tester for such a language requires $\Omega(\log(\varepsilon^{-1})/\varepsilon)$ queries.

Let us first give an example that will motivate our construction. Consider the parity language P consisting of words that contain an even number of b's, over the alphabet $\{a, b\}$. Distinguishing $u \in P$ from $u \notin P$ requires $\Omega(n)$ queries, as changing the letter at single position can change membership in P. However, P is trivial to test, as any word is at distance at most 1 from P, for the same reason. Now, consider language L_2 consisting of words over $\{a, b, c, d\}$ such that between a c and the next d, there is a word in P. Intuitively, this language encodes multiple instances of P, hence we can construct words ε-far from L_2, and each instance is hard to recognize for property testers, hence the whole language is. In [9, Theorem 15], Bathie and Starikovskaya proved a lower bound of $\Omega(\log(\varepsilon^{-1})/\varepsilon)$ on the query complexity of any property tester for L_2, matching the upper bound in the same paper.

The minimal blocking factors of L_2 include all words for the form cvd where $v \notin P$: there are infinitely many such words. This is no coincidence: we show that this lower bound can be lifted to any language with infinitely many minimal blocking factors, under the Hamming distance.

▶ **Theorem 3.14.** *Let $\mathcal{A}$ be a strongly connected NFA. If $MBF(\mathcal{A})$ is infinite, then there exists a constant ε_0 such that for any $\varepsilon < \varepsilon_0$, any ε-property tester for $L = \mathcal{L}(\mathcal{A})$ uses $\Omega(\log(\varepsilon^{-1})/\varepsilon)$ queries.*

The proof of this result is full generalization of the lower bound against the "repeated Parity" example given above.

Our proof is based on (a consequence of) Yao's Minimax Principle [31]: if there is a distribution $\mathcal{D}$ over inputs such that any *deterministic* algorithm that makes at most q queries errs on $u \sim \mathcal{D}$ with probability at least p, then any *randomized* algorithm with q queries errs with probability at least p on some input u.

To prove Theorem 3.14, we first exhibit such a distribution $\mathcal{D}$ for $q = \Theta(\log(\varepsilon^{-1})/\varepsilon)$. We take the following steps:
1. we show that with high probability, an input u sampled w.r.t. $\mathcal{D}$ is either in or ε-far from L (Lemma 3.21),
2. we show that with high probability, any deterministic tester that makes fewer than $c \cdot \log(\varepsilon^{-1})/\varepsilon$ queries (for a suitable constant c) cannot distinguish whether the instance u is positive or ε-far, hence it errs with large probability.
3. combine the above two results to prove Theorem 3.14 via Yao's Minimax principle.

3.3.1 The structure of $\mathsf{MBF}(\mathcal{A})$

Before diving into the proof of Theorem 3.14, we show that if $\mathsf{MBF}(\mathcal{A})$ is infinite, then we can find minimal blocking factors with a "regular" structure, a crucial ingredient for our proof. First, we prove that the set of minimal blocking factors of an automaton is a regular language, recognized by an automaton that is possibly exponentially larger than $\mathcal{A}$. We first prove the result for blocking factors of the form $(i : u)$ for a fixed $i \in \mathbb{Z}/\lambda\mathbb{Z}$.

▶ **Lemma 3.15.** *Let* $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ *be a strongly connected NFA with* m *states and let* $\lambda = \lambda(\mathcal{A})$. *For every* $i \in \mathbb{Z}/\lambda\mathbb{Z}$, *the set of minimal blocking factors of* $\mathcal{A}$ *of the form* $(i : u)$ *is a regular language recognized by a NFA of size* $2^{\mathcal{O}(m)}$.

Proof. We call blocking factors of $\mathcal{A}$ of the form $(i : u)$ its *i-blocking factors*.

We first show that the set of i-blocking factors of $\mathcal{A}$, but not necessarily minimal ones, is a regular language recognized by an NFA $\mathcal{A}_i$ with $m + 1$ states. The result follows by using standard constructions for complement and intersection of automata [27, Chapter 1, Section 3]: these constructions give an automaton of size $2^{\mathcal{O}(m)}$ that recognizes words in L that have no proper factor in L.

Consider the NFA $\mathcal{A}_i$ obtained by adding a new sink state $\perp$ to $\mathcal{A}$, making it the only accepting state, with set of initial states Q_i. Formally, $\mathcal{A}_i$ is defined as $\mathcal{A}_i = (Q \cup \{\perp\}, \Sigma, \delta', Q_i, \{\perp\})$, where δ' is defined as follows:

$$\forall p \in Q, \forall a \in \Sigma : \delta'(p, a) = \begin{cases} \{\perp\} & \text{if } \delta(p, a) = \emptyset, \\ \delta(p, a) & \text{otherwise.} \end{cases}$$

This automaton[4] recognizes the set of i-blocking factors of $\mathcal{A}$ and has size $\mathcal{O}(m)$. Applying the aforementioned construction to $L = \mathcal{L}(\mathcal{A}_i)$ yields the desired automaton, of size $2^{\mathcal{O}(m)}$.
◀

It follows that the set of minimal blocking factors of $\mathcal{A}$ is also a regular language.

▶ **Corollary 3.16.** *Let* $\mathcal{A}$ *be an NFA with* m *states. The set of minimal blocking factors of* $\mathcal{A}$ *is a regular language recognized by an NFA of size* $2^{\mathcal{O}(m)}$.

---

[4] Our definition of NFAs does not allow for multiple initial states. As there is no constraint of strong connectivity for $\mathcal{A}_i$, this can be solved using a simple construction that adds a new initial state.

Therefore, if $\mathsf{MBF}(\mathcal{A})$ is infinite, we can use the Pumping Lemma [27, Chapter 1, Proposition 2.2] to find an infinite family of minimal blocking factors with a shared structure $\{\phi\nu^r\chi, r \in \mathbb{N}\}$, for some non-empty positional words ϕ, ν and χ. We will use this property later, when proving a lower bound against the language of automata with infinitely many blocking factors.

▶ **Lemma 3.17.** *If $\mathsf{MBF}(\mathcal{A})$ is infinite, then there exist positional words ϕ, ν_+, ν_-, χ such that:*

1. *the words ν_+ and ν_- have the same length,*
2. *there exists a constant $S = 2^{\mathrm{poly}(m)}$ such that $|\phi|, |\nu_+|, |\nu_-|, |\chi| \leq S$,*
3. *there exists an index $i_* \in \mathbb{Z}/\lambda\mathbb{Z}$ and a state $q_* \in Q_{i_*}$ such that for every integer $r \geq 1$, the positional word $\tau_{-,r} = \phi(\nu_-)^r\chi$ is blocking for $\mathcal{A}$, and for every $s < r$, we have*

$$q_* \xrightarrow{\tau_{+,r,s}} q_* \text{ where } \tau_{+,r,s} = \phi(\nu_-)^j\nu_+(\nu_-)^{r-1-s}\chi.$$

In particular, $\tau_{+,r,s}$ is not blocking for $\mathcal{A}$.

Note that here, the state q_* is the same for *every* integers r, s.

Proof. As $\mathsf{MBF}(\mathcal{A})$ is infinite, there must exist an integer i_* such that $\mathcal{A}$ has infinitely many minimal i_*-blocking factors; we fix such i_* in what follows.

Let us recall the Pumping Lemma [27, Chapter 1, Proposition 2.2], with a formulation adapted to our purpose.

▶ **Fact 3.18** (Pumping Lemma). *Let L be a regular language recognized by an automaton of size T. There exists an integer $S = \mathcal{O}(T)$ such that any word u of L of length at least S can be factorized as $u = xyz$ such that $|xy| \leq S, |y| \geq 1$ and, for all $k \geq 0, xy^kz \in L$.*

As the set of minimal i_*-blocking factors is an infinite regular language recognized by an NFA of size $T = 2^{\mathcal{O}(m)}$. Let $S = 2^{\mathcal{O}(m)}$ be the constant given by the Pumping Lemma: since the language is infinite, it contains at least one positional word of length greater than S. Hence, there exist positional words τ, μ and η, with $|\mu| \geq 1$, such that for any non-negative integer $k, \tau\mu^k\eta$ is a minimal i_*-blocking factor. By removing factors that label loops in the automaton, we can assume that each of them has length at most S. Furthermore, we can assume w.l.o.g. that neither τ nor η is empty, otherwise we set their value to μ: after this modification, $\tau\mu^k\eta$ is still a minimal i_*-blocking factor for every $k \geq 0$.

Notice that the word $\tau\mu^m$ is not a blocking factor, as a proper factor of the minimal blocking factor $\tau\mu^m\eta$. Therefore, by the pigeonhole principle, there exist integers $k_0, k_1 \geq 1$ with $k_0 + k_1 = m$ and states p, p_1 such that we have

$$p \xrightarrow{\tau\mu^{k_0}} p_1 \xrightarrow{\mu^{k_1}} p_1.$$

Note that, by Fact 2.3, $p_1 \xrightarrow{\mu^{k_1}} p_1$ implies that $k_1 \cdot |\mu| = 0 \pmod{\lambda}$.

Similarly, the word $\mu^m\eta$ is not a blocking factor, since it is a proper factor of the minimal i_*-blocking factor $\tau\mu^m\eta$. Again, there exist integers $k_2 \geq 1, k_3$ summing to m and states p_2 and q such that

$$p_2 \xrightarrow{\mu^{k_2}} p_2 \xrightarrow{\mu^{k_3}\eta} q.$$

Now, define $\phi = \tau\mu^{k_0}, \chi = \mu^{k_3}\eta$ and $\nu_- = \mu^K$, where $K = \rho \cdot k_1 \cdot k_2$. As there are transitions starting from p_1 and p_2 labeled by μ, p_1 and p_2 belong to the same periodicity class. Therefore, by Fact 2.3, as $K \geq \rho$ and $K \cdot |\mu| = 0 \pmod{\lambda}$, there exists a word ν_+ of length $K \cdot |\mu|$ such that $p_1 \xrightarrow{\nu_+} p_2$. This choice of ϕ, ν_+, ν_- and χ satisfies all the conditions of the lemma. ◄

3.3.2 Constructing a Hard Distribution $\mathcal{D}$

Let $\varepsilon > 0$ be sufficiently small and let n be a large enough integer. In what follows, m denotes the number of states of $\mathcal{A}$. To construct the hard distribution $\mathcal{D}$, we will use an infinite family of blocking factors that share a common structure, given by Lemma 3.17.

The crucial property here is that $\tau_{-,r}$ and $\tau_{+,r,s}$ are very similar: they have the same length, differ in at most S letters, yet one of them is blocking and the other is not.

We now use the words $\tau_{-,r}$ and $\tau_{+,r,s}$ and the constant S to describe how to sample an input $\mu = (0 : u)$ of length n w.r.t. $\mathcal{D}$.

Let π be a uniformly random bit. If $\pi = 1$, we will construct a positive instance $\mu \in \mathcal{TL}(\mathcal{A})$, and otherwise the instance will be ε-far from $\mathcal{TL}(\mathcal{A})$ with high probability. We divide the interval $[0\mathbin{.\,.}n-1]$ into $k = \varepsilon n$ intervals of length $\ell = 1/\varepsilon$, plus small initial and final segments μ_i and μ_f of length $\mathcal{O}(\rho)$ to be specified later. For the sake of simplicity, we assume that k and ℓ are integers and that λ divides ℓ. For $j = 1, \ldots, k$, let a_j, b_j denote the endpoints of the j-th interval. For each interval, we sample independently at random a variable κ_j with the following distribution:

$$\kappa_j = \begin{cases} t, & \text{with prob. } p_t = 3 \cdot 2^t S\varepsilon / \log((S\varepsilon)^{-1}) \text{ for } t = 1, 2, \ldots, \log((S\varepsilon)^{-1}), \\ 0, & \text{with prob. } p_0 = 1 - \sum_{t=1}^{\log((S\varepsilon)^{-1})} p_t. \end{cases} \tag{1}$$

The event $\kappa_j > 0$ means that the j-th interval is filled with $N \approx 2^{-\kappa_j}/\varepsilon$ "special" factors. When $\pi = 0$, these "special" factors will be minimal blocking factors $\tau_{-,r}$ for $r = 2^{\kappa_j}$, whereas when $\pi = 1$, they will instead be similar non-blocking factors $\tau_{+,r,s}$ for a uniformly random s: they will be hard to distinguish with few queries. On the other hand, the event $\kappa_j = 0$ means that the j-th interval contains no specific information. More precisely, we choose a positional word η_* of length ℓ such that $q_* \xrightarrow{\eta_*} q_*$: by Fact 2.3, this is possible as $\ell = 0 \pmod{\lambda}$. Then, if $\kappa_j = 0$, we set $\mu[a_j \mathbin{.\,.} b_j] = \eta_*$, regardless of the value of π.

Formally, if $\kappa_j > 0$, let $r = 2^{\kappa_j}$, $N = 2^{-\kappa_j}/(S\varepsilon)$ and let η be a word of length $\ell - N \cdot |\tau_{-,r}|$ such that $q_* \xrightarrow{\eta} q_*$: such a word exists as λ divides ℓ and $|\tau_{-,r}|$. We construct the j-th interval as follows:

- if $\pi = 0$, we set $\mu[a_j \mathbin{.\,.} b_j] = (\tau_{-,r})^N \eta$,
- if $\pi = 1$, we select $s \in [0 \mathbin{.\,.} r-1]$ uniformly at random, and set $\mu[a_j \mathbin{.\,.} b_j] = (\tau_{+,r,s})^N \eta$.

Finally, the initial and final fragments μ_i and μ_f of μ are chosen to be the shortest words that label a transition from q_0 to q_* and q_* to a final state, respectively.

3.3.3 Properties of the distribution $\mathcal{D}$

Next, we establish that the distribution $\mathcal{D}$ has the desired properties.

▶ **Observation 3.19.** *If ε is small enough, $\mathcal{D}$ is well-defined, i.e. for every t between 0 and $\log((S\varepsilon)^{-1})$, we have $0 \le p_t \le 1$.*

▶ **Observation 3.20.** *If $\pi = 1$, then $\mu \in \mathcal{TL}(\mathcal{A})$.*

▶ **Lemma 3.21.** *Conditioned on $\pi = 0$, the probability of the event $\mathcal{F} = \{\mu$ is ε-far from $\mathcal{TL}(\mathcal{A})\}$ goes to 1 as n goes to infinity.*

Proof. When $\pi = 0$, the procedure for sampling μ puts blocking factors of the form $(i_* : x)$ at positions equal to $i_* \mod \lambda$. Any word containing such a factor at such a position is not in $\mathcal{TL}(\mathcal{A})$, therefore any sequence of substitutions that transforms μ into a word of $\mathcal{TL}(\mathcal{A})$ must make at least one substitution in every such factor. Consequently, the distance

between μ and $\mathcal{TL}(\mathcal{A})$ is at least the number of blocking factors in μ. To prove the lemma, we show that this number is at least εn with high probability, by showing that it is larger than εn by a constant factor in expectation and using a concentration argument.

Let B_j denote the number of blocking factors in the j-th interval: it is equal to $2^{-\kappa_j}/(S\varepsilon)$ when $\kappa_j > 0$ and to 0 otherwise.

▷ **Claim 3.22.** Let $B = \sum_{j=1}^k B_j$, and let $E = \mathbb{E}[B]$. We have $E \geq 2\varepsilon n$.

Claim proof. By direct calculation:

$$
\begin{aligned}
E &= \sum_{j=1}^{k} \mathbb{E}[B_j] = \sum_{j=1}^{k} \sum_{t=1}^{\log(S/\varepsilon)} 2^{-t}/(S\varepsilon) \cdot p_t \\
&= \sum_{j=1}^{k} \sum_{t=1}^{\log(S/\varepsilon)} 2^{-t}/(S\varepsilon) \cdot 3 \cdot 2^t \varepsilon S/\log(S/\varepsilon) = \sum_{j=1}^{k} \sum_{t=1}^{\log(S/\varepsilon)} 3/\log(S/\varepsilon) \\
&= 3k \geq 2\varepsilon n
\end{aligned}
$$

◄

We will now show that $\mathbb{P}(B < \varepsilon n)$ goes to 0 as n goes to infinity. We use Hoeffding's inequality, which we recall here:

▶ **Fact 3.23** ([20, Theorem 2]). *Let $X_1, \ldots, X_k$ be independent random variables such that for every $i = 1, \ldots, k$, we have $a_i \leq X_i \leq b_i$, and let $S = \sum_{i=1}^k X_i$. Then, for any $t > 0$, we have*

$$
\mathbb{P}(\mathbb{E}[S] - S \geq t) \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^k (b_i - a_i)^2}\right).
$$

By Claim 3.22, we have $B < \varepsilon n \Rightarrow E - B \geq \varepsilon n$, and therefore $\mathbb{P}(B < \varepsilon n) \leq \mathbb{P}(E - B \geq \varepsilon n)$. The random variable B is the sum of k independent random variables, each taking values between 0 and $1/(S\varepsilon)$. Therefore, by Hoeffding's inequality (Fact 3.23), we have

$$
\begin{aligned}
\mathbb{P}(E - B < \varepsilon n) &\leq \exp\left(-\frac{2\varepsilon^2 n^2}{k/(S\varepsilon)^2}\right) \\
&\leq \exp\left(-\frac{2S^2 \varepsilon^4 n^2}{\varepsilon n}\right) \text{ as } k \leq \varepsilon n \\
&\leq \exp\left(-2S^2 \varepsilon^3 n\right)
\end{aligned}
$$

This probability goes to 0 as n goes to infinity, which concludes the proof.   ◄

▶ **Corollary 3.24.** *For large enough n, we have $\mathbb{P}(\mathcal{F}) \geq 5/12$.*

Intuitively, our distribution is hard to test because positive and negative instances are very similar. Therefore, a tester with few queries will likely not be able to tell them apart: the perfect completeness constraint forces the tester to accept in that case. Below, we establish this result formally.

▶ **Lemma 3.25.** *Let T be a deterministic tester with perfect completeness (i.e. it always accepts $\tau \in \mathcal{TL}(\mathcal{A})$) and let q_j denote the number of queries that it makes in the j-th interval. Conditioned on the event $\mathcal{M} = \{\forall j \text{ s.t. } \kappa_j > 0, q_j < 2^{\kappa_j}\}$, the probability that T accepts $\mu \sim \mathcal{D}$ is 1.*

Proof. We proceed by contradiction, and show that if there exists a word τ with non-zero probability w.r.t. $\mathcal{D}$ under $\mathcal{M}$ that T rejects, then there exists a word $\tau' \in \mathcal{TL}(\mathcal{A})$ that T rejects that also has non-zero probability, contradicting the fact that T has perfect completeness.

Let τ be the word rejected by T: as T has perfect completeness, τ is not in $\mathcal{TL}(\mathcal{A})$, and there must be at least one interval with $\kappa_j > 0$. Consider every interval j such that $\kappa_j > 0$: it is of the form $(\tau_{-,r})^N \eta$ where $r = 2^{\kappa_j}$ and $\tau_{-,r} = \phi(\nu_-)^r \chi$. Therefore, if $q_j < 2^{\kappa_j}$, then there is a copy of ν_- that has not been queried by T across all copies of $\tau_{-,r}$. Consider the word τ' obtained by replacing this copy of ν_- with ν_+ in all N copies of $\tau_{-,r}$ in the block. The result block is of the form $(\tau_{+,r,s})^N \eta$ for some $s < r$, and by construction it is not blocking. Applying this operation to all blocks results in a word τ' that is in $\mathcal{TL}(\mathcal{A})$. Furthermore, τ' has non-zero probability under $\mathcal{D}$ conditioned on $\mathcal{M}$: it can be obtained by flipping the random bit π and choosing the right index s in every block. ◄

Next, we show that if a tester makes few queries, then the event $\mathcal{M}$ has large probability.

▶ **Lemma 3.26.** *Let T be a deterministic tester, let q_j denote the number of queries that it makes in the j-th interval, and assume that T makes at most $\frac{1}{72} \cdot \log(S/\varepsilon)/\varepsilon$ queries, i.e. $\sum_j q_j \leq \frac{1}{72} \cdot \log(S/\varepsilon)/\varepsilon$. The probability of the event $\mathcal{M} = \{\forall j \ s.t. \ \kappa_j > 0, q_j < 2^{\kappa_j}\}$ is at least $11/12$.*

Proof. We show that the probability of $\overline{\mathcal{M}}$, the complement of $\mathcal{M}$, is at most $1/12$. We have:

$$
\begin{aligned}
\mathbb{P}\left(\overline{\mathcal{M}}\right) &= \mathbb{P}\left(\exists j : \kappa_j > 0 \wedge q_j \geq 2^{\kappa_j}\right) \\
&\leq \sum_j \mathbb{P}\left(\kappa_j > 0 \wedge q_j \geq 2^{\kappa_j}\right) && \text{by union bound} \\
&\leq \sum_j \sum_{t=1}^{\lfloor \log q_j \rfloor} p_t = \sum_j \sum_{t=1}^{\lfloor \log q_j \rfloor} \frac{3 \cdot 2^t \varepsilon}{\log(S/\varepsilon)} && \text{by def. of } p_t \\
&\leq \frac{3\varepsilon}{\log(S/\varepsilon)} \sum_j \sum_{t=1}^{\lfloor \log q_j \rfloor} 2^t
\end{aligned}
$$

By upper bounding the sum of power of 2 up to $k = \lfloor \log q_j \rfloor$ by 2^{k+1}, we obtain:

$$
\begin{aligned}
\mathbb{P}\left(\overline{\mathcal{M}}\right) &\leq \frac{3\varepsilon}{\log(S/\varepsilon)} \sum_j 2q_j \\
&= \frac{3\varepsilon}{\log(S/\varepsilon)} \cdot \frac{2}{72} \cdot \frac{\log(S/\varepsilon)}{\varepsilon} \\
&\leq 1/12 && \blacktriangleleft
\end{aligned}
$$

We are now ready to prove Theorem 3.14.

Proof of Theorem 3.14. We want to show that any tester with perfect completeness for $\mathcal{L}(\mathcal{A})$ requires at least $\frac{1}{72} \cdot \log(S/\varepsilon)/\varepsilon$ queries, by showing that any tester with fewer queries errs with probability at least $1/3$. We show that any **deterministic** algorithm T with perfect completeness that makes less than $\frac{1}{72} \cdot \log(S/\varepsilon)/\varepsilon$ queries errs on u when $(0 : u) \sim \mathcal{D}$ with probability at least $1/3$, and conclude using Yao's Minimax principle.

Consider such an algorithm T. The probability that T makes an error on u is lower-bounded by the probability that u is ε-far from $\mathcal{L}(\mathcal{A})$ and T accepts, which in turn is larger

than the probability of $\mathcal{M} \cap \mathcal{F}$. By Corollary 3.24, we have $\mathbb{P}(\mathcal{F}) \geq 5/12$, and by Lemma 3.26, $\mathbb{P}(\mathcal{M})$ is at least $11/12$. Therefore, we have

$$\mathbb{P}(T \text{ errs}) \geq \mathbb{P}(\mathcal{M} \cap \mathcal{F}) \geq 1 - 7/12 - 1/12 = 4/12 = 1/3.$$

This concludes the proof of Theorem 3.14, and consequently of Theorem 3.2. ◀

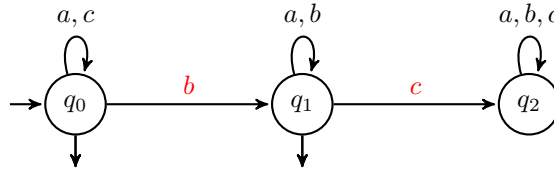## 4 Characterisation of Hard Languages for All NFAs

In this section we extend the results of the previous section to all finite automata. This extension is based on a generalization of blocking factors: we introduce *blocking sequences*, which are sequences of factors that witness the fact that we cannot take any path through the strongly connected components of the automaton. For the lower bound, we define a suitable partial order on blocking sequences, which extends the factor relation on words to those sequences, and allows us to define *minimal* blocking sequences.

4.1 Blocking sequences

4.1.1 Examples motivating blocking sequences

Before presenting the technical part of the proof, let us go through two examples, which motivate the notions that we introduce and illustrate some of the main difficulties.

▶ **Example 4.1.** Consider the automaton $\mathcal{A}_1$ depicted in Figure 2: it recognizes the language L_1 of words in which all c's appear before the first b, over the alphabet $\{a, b, c\}$.

Figure 2 An automaton $\mathcal{A}_1$ that recognizes the language $L_1 = (a + c)^*(a + b)^*$.

The set of minimal blocking factors of $\mathcal{A}_1$ is infinite: it is the language ba^*c. Yet, L_1 is easy to test: we sample $\mathcal{O}(1/\varepsilon)$ letters at random, answer "no" if the sample contains a c occurring after a b, and "yes" otherwise. To prove that this yields a property tester, we rely on the following property:

▶ **Property 4.2.** *If u is ε-far from L_1, then it can be decomposed into $u = u_1 u_2$ where u_1 contains $\Omega(\varepsilon n)$ letters b and u_2 contains $\Omega(\varepsilon n)$ letters c.*

The pair of factors (b, c) is an example of blocking sequence: a word that contains an occurrence of the first followed by an occurrence of the second cannot be in L_1. We can also show that a word ε-far from L_1 must contains many disjoint blocking sequences – this property (Lemma 4.18) underpins the algorithm for general regular languages.

What this example shows is that blocking factors are not enough: we need to consider sequences of factors, yielding the notion of *blocking sequences*. Intuitively, a blocking sequence for L is a sequence $\sigma = (v_1, \ldots, v_k)$ of (positional) words such that if each word of the sequence appears in u, with the occurrences of the v_i's ordered as in σ, then u is not in

L.[5] While L_1 has infinitely many minimal blocking factors, it has a single minimal blocking sequence $\sigma = (b, c)$.

Notice that the (unique) blocking sequence (b, c) can be visualized on Figure 2: it is composed of the red letters that label transitions between the different SCCs. This is no coincidence: in many simple cases, blocking sequences are exactly sequences that contain one blocking factors for each SCC. This fact could lead one to believe that the set of minimal blocking sequences is exactly the set of sequences of minimal blocking factors, one for each SCC. In particular, this would imply that as soon as one SCC has infinitely many minimal blocking factors, the language of the whole automaton is hard to test. We show in the next example that this is not always the case, because SCCs might share minimal blocking factors.

▶ **Example 4.3.** Consider the automaton in Figure 3: it has two SCCs and a sink state. The minimal blocking factors of the first SCC are given by $B_1 = be^*c + a$, and $B_2 = \{a\}$ for the second SCC. This automaton is easy to test: intuitively, a word that is ε-far from this language has to contain many a's, as otherwise we can make it accepted by deleting all a's, thanks to the second SCC. However, a is also a blocking factor of the first SCC, therefore, as soon as we find two a's in the word, we know that it is not in L_2.

Figure 3 An automaton $\mathcal{A}_2$ that recognizes the language $L_2 = [((c + d + e)^*b(b + e)^*d)^*a](b + c + d + e)^*$.

The crucial facts here are that the set B_2 of minimal blocking factors of the second SCC is finite and it is a subset of B_1: the infinite nature of B_1 is made irrelevant because any word far from the language contains many a's. Therefore, $\mathcal{A}_2$ has a *single* minimal blocking sequence, $\sigma = (a)$.

4.1.2 Portals and SCC-paths

Intuitively, blocking sequences are sequences of blocking factors of successive strongly connected components. To formalize this intuition, we use *portals*, which describe how a run in the automaton interacts with a strongly connected component, and *SCC-paths*, that describe a succession of portals.

---

[5] This is not quite the definition, but it conveys the right intuition.

In what follows, we fix an NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, \{q_f\})$. We assume w.l.o.g. that $\mathcal{A}$ has a unique final state q_f. Let $\mathscr{S}$ be the set of SCCs of $\mathcal{A}$. We define the partial order relation $\leq_{\mathcal{A}}$ on $\mathscr{S}$ by $S \leq_{\mathcal{A}} T$ if and only if T is reachable from S. We write $<_{\mathcal{A}}$ for its strict part $\leq_{\mathcal{A}} \setminus \geq_{\mathcal{A}}$. These relations can be naturally extended to states through their SCC: if $s \in S$ and $t \in T$, then $s \leq_{\mathcal{A}} t$ if and only if $S \leq_{\mathcal{A}} T$.

We define p as the least common multiple of the lengths of all simple cycles of $\mathcal{A}$. Given a number $k \in \mathbb{Z}/p\mathbb{Z}$, we say that a state t is k-reachable from a state s if there is a path from s to t of length k modulo p. In what follows, we use "positional words" for p-positional words with this value of p.

▶ **Remark 4.4.** In the rest of this section we will not try to optimize the constants in the formulas. They will, in fact, become quite large in some of the proofs. We make this choice to make the proofs more readable, although some of them are already technical.

For instance, the choice of p as the lcm of the lengths of simple cycles is not optimal: we could use, for instance, the lcm of the periods of the SCCs.

▶ **Definition 4.5** (Portal). *A portal is a 4-tuple $P = s, x \rightsquigarrow t, y \in (Q \times \mathbb{Z}/p\mathbb{Z})^2$, such that s and t are in the same SCC. It describes the first and last states visited by a path in an SCC, and the positions x, y (modulo p) at which it first and lasts visits that SCC.*

The positional language of a portal is the set

$$\mathcal{L}(s, x \rightsquigarrow t, y) = \{(x : w) \mid s \xrightarrow{w} t \wedge x + |w| = y \pmod{p}\}.$$

Portals were already defined by Alon et al. [5], in a slightly different way. Our definition will allow us to express blocking sequences more naturally.

▶ **Definition 4.6.** *A positional word $(n : u)$ is blocking for a portal P if it is not a factor of any word of $\mathcal{L}(P)$. In other words, there is no path that starts in s and ends in t, of length $y - x$ modulo p, which reads u after $n - x$ steps modulo p.*

The above definition matches the definition of blocking factors for strongly connected automata. This is no coincidence: we show in the next lemma that the language of a portal has a strongly connected automaton.

▶ **Lemma 4.7.** *Let $\mathcal{A}$ be an automaton and P a portal of $\mathcal{A}$. There is a strongly connected NFA with at most $p|\mathcal{A}|$ states that recognizes $L' = \mathcal{L}(P)$.*

Proof. Let S denote the SCC of s and t in $\mathcal{A}$, and let λ denote its period. By definition of p, λ divides p: let k be the integer such that $p = \lambda k$. The automaton $\mathcal{A}'$ for L' simulates the behavior of $\mathcal{A}$ restricted to S starting from the state s, while keeping track of the number of letters read modulo p, starting from x. More precisely, let $Q_0, \dots Q_{\lambda-1}$ be the partition of the states of S given by Fact 2.3. The set of states of $\mathcal{A}'$ is given by

$$Q' = \{(s', i + j\lambda) \mid s' \in Q_i \wedge i = 0 \dots, \lambda - 1 \wedge j = 0, \dots, k - 1\}.$$

It is a subset of $Q \times \mathbb{Z}/p\mathbb{Z}$, hence it has cardinality at most $p|\mathcal{A}|$. The transitions in $\mathcal{A}'$ are of the form $(s_1, i + j\lambda) \xrightarrow{(i,a)} (s_2, i + j\lambda + 1 \pmod{p})$ for any s_1, s_2 such that $s_1 \xrightarrow{a} s_2$ in $\mathcal{A}$.

Furthermore, $\mathcal{A}'$ is strongly connected. Let i_1, i_2 be indices of periodicity classes of S, and let $s_1 \in Q_{i_1}, s_2 \in Q_{i_2}$ be states of S. We show that for any $j_1, j_2 < k$, there is a path from $\sigma_1 = (s_1, i_1 + j_1\lambda)$ to $\sigma_2 = (s_2, i_2 + j_2\lambda)$ in $\mathcal{A}'$. Let ℓ be a sufficiently large integer equal to $(i_2 - i_1) + (j_2 - j_1)\lambda \pmod{p}$. As λ divides p, ℓ is equal to $(i_2 - i_1) \pmod{\lambda}$. By taking ℓ larger than the reachability constant of S, Fact 2.3 gives us that there is a path of

length ℓ from s_1 to s_2 in S, labeled by some word u. The positional word $(x : u)$ labels a transition from σ_1 to σ_2 in $\mathcal{A}'$, hence it is strongly connected.

Note that the period of $\mathcal{A}'$ is p, hence we can apply the results we obtained on strongly connected NFAs in Section 3 to portals, with $p|\mathcal{A}|$ as the number of states and p as the period. ◄

Portals describe the behavior of a run inside a single strongly connected component of the automaton. Next, we introduce SCC-paths, which describe the interaction of a run with multiple SCCs and between two successive SCCs.

▶ **Definition 4.8** (SCC-path). *An SCC-path π of $\mathcal{A}$ is a sequence of portals linked by single-letter transitions $\pi = s_0, x_0 \leadsto t_0, y_0 \xrightarrow{a_1} s_1, x_1 \leadsto t_1, y_1 \cdots \xrightarrow{a_k} s_k, x_k \leadsto t_k, y_k$, such that for all $i \in \{1, \ldots, k\}$, $x_i = y_{i-1} + 1 \pmod p$, $t_{i-1} \xrightarrow{a_i} s_i$, and $t_{i-1} <_{\mathcal{A}} s_i$.*

Intuitively, an SCC-path is a description of the states and positions at which a path through the automaton enters and leaves each SCC.

▶ **Definition 4.9.** *The language $\mathcal{L}(\pi)$ of an SCC-path $\pi = P_0 \xrightarrow{a_1} P_1 \xrightarrow{a_2} \cdots P_k$ is the set*

$$\mathcal{L}(\pi) = \mathcal{L}(P_0)a_1\mathcal{L}(P_2)a_2 \cdots \mathcal{L}(P_k).$$

We say that π is *accepting* if $P_0 = s_0, x_0 \leadsto t_0, y_0$, $P_k = s_k, x_k \leadsto t_k, y_k$ with $x_0 = 0$, $s_0 = q_0$, $t_k = q_f$ and $\mathcal{L}(\pi)$ is non-empty.

▶ **Lemma 4.10.** *We have $\mathcal{TL}(\mathcal{A}) = \bigcup_{\pi \text{ accepting}} \mathcal{L}(\pi)$.*

Proof. We show that for any word μ in $\mathcal{TL}(\mathcal{A})$, there is an accepting SCC-path π whose language contains μ. Let $\mu = a_1 \cdots a_n$ be a word of length n in $\mathcal{TL}(\mathcal{A})$: there exists an accepting run $\rho = q_0 \xrightarrow{a_1} q_1 \cdots \xrightarrow{a_n} q_n = q_f$ in $\mathcal{A}$.

We define the sequence of indices $i_0 < i_1 < \ldots < i_k < i_{k+1}$ as follows:
- $i_0 = 0, i_{k+1} = n + 1$,
- for every $j = 1, \ldots, k$, i_j is the smallest index such that $q_{i_j-1} <_{\mathcal{A}} q_{i_j}$, i.e. q_{i_j-1} and q_{i_j} belong to distinct SCCs.

In other words, those are the indices at which ρ enters a new SCC. We then define the SCC-path $\pi(\rho)$ as follows:

$$\pi(\rho) = q_0, 0 \leadsto q_{i_1-1}, y_0 \xrightarrow{a_{i_1}} q_{i_1}, x_1 \leadsto q_{i_2-1}, y_1 \cdots \xrightarrow{a_{i_k}} q_{i_k}, x_k \leadsto q_n, y_k$$

where $x_j = i_j \pmod p$ and $y_j = x_{j+1} - 1 \pmod p$ for all $j = 0, \ldots, k + 1$.

By construction, $\mu \in \mathcal{L}(\pi(\rho))$ and $\pi(\rho)$ is an accepting SCC-path.

The converse inclusion follows by definition of (accepting) SCC-paths. ◄

As a consequence the distance between a word μ and the (positional) language of $\mathcal{A}$ is equal to the minimum of the distances between μ and the languages of the SCC-paths of $\mathcal{A}$.

▶ **Corollary 4.11.** *For any positional word μ, we have*

$$d(\mu, \mathcal{TL}(\mathcal{A})) = \min_{\pi \text{ accepting}} d(\mu, \mathcal{L}(\pi)).$$

Decomposing $\mathcal{A}$ as a union of SCC-paths allows us to use them as an intermediate step to define blocking sequences. We earlier defined blocking factors for portals: we now generalize this definition to blocking sequences for SCC-paths, to finally define blocking sequence of automata.

▶ **Definition 4.12** ((Strongly) Blocking Sequences for SCC-paths). *We say that a sequence* $\sigma = (\mu_1, \ldots, \mu_\ell)$ *of positional factors is blocking for an SCC-path* $\pi = P_0 \xrightarrow{a_1} \cdots P_k$ *if there is a sequence of indices* $i_0 \le i_1 \le \cdots \le i_k$ *such that for every* j, μ_{i_j} *is blocking for* P_j.

Furthermore, if there is a sequence of indices $i_0 < i_1 < \cdots < i_k$ *with the same property, then* σ *is said to be* strongly blocking *for* π.

Note that, crucially, in the definition of blocking sequences, consecutive indices i_j and i_{j+1} can be equal, i.e. a single factor of the sequence may be blocking for multiple consecutive SCCs in the SCC-path. This choice is motivated by Example 4.3, where the language is easy because consecutive SCCs share blocking factors.

We say that two occurrences of blocking sequences in a word μ are *disjoint* if the occurrences of their factors appear on disjoint sets of positions in μ.

In the strongly connected case, we had the property that if μ contains an occurrence of a factor blocking for $\mathcal{A}$, then μ is not in the language of $\mathcal{A}$. The following lemma gives an extension of this result to *strongly* blocking sequences and the language of an SCC-path.

▶ **Lemma 4.13.** *Let* π *be an SCC-path. If* μ *contains a strongly blocking sequence for* π, *then* $\mu \notin \mathcal{L}(\pi)$.

Proof. We proceed by induction on the length k of the SCC-path $\pi = P_0 \xrightarrow{a_1} P_1 \cdots \xrightarrow{a_k} P_k$. Let $\sigma = (\nu_0, \ldots, \nu_k)$ be a strongly blocking sequence for π that occurs in μ. If $k = 0$, then σ consists only of a blocking factor for P_0, hence μ is not in $\mathcal{L}(P_0)$, which is equal to $\mathcal{L}(\pi)$.

For $k > 0$, assume for the sake of contradiction that $\mu \in \mathcal{L}(\pi)$. By definition of $\mathcal{L}(\pi)$, μ can then be written as $\mu_0 a_1 \mu'$, with $\mu_0 \in \mathcal{L}(P_0)$ and $\mu' \in \mathcal{L}(\pi')$, where $\pi' = P_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} P_k$. As ν_0 is blocking for P_0, the prefix μ_0 of μ must end before the occurrence of ν_0 in μ, and the sequence $\sigma' = (\nu_1, \ldots, \nu_k)$ occurs in μ'. Furthermore, because σ is *strongly* blocking for π, σ' is strongly blocking for π'. Using the induction hypothesis on μ' and the path π' of length $k - 1$, this implies that $\mu' \notin \mathcal{L}(\pi')$, a contradiction. ◀

We can now define sequences that are blocking for an automaton: they are sequences that are blocking for *every* accepting SCC-path of the automaton.

▶ **Definition 4.14** (Blocking sequence for $\mathcal{A}$). *Let* $\sigma = (\mu_1, \ldots, \mu_\ell)$ *be a sequence of positional words. We say that* σ *is blocking for* $\mathcal{A}$ *if it is blocking for all accepting SCC-paths of* $\mathcal{A}$.

As an example, observe that the sequences $((0 : ab), (1 : ab))$ and $((0 : aa), (0 : b))$ are both blocking for the automaton displayed in Figure 4 (see Example 4.15).
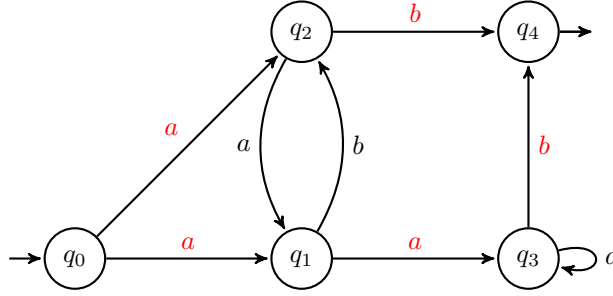
▶ **Example 4.15.** Consider the automaton displayed in Figure 4. The lcm of the lengths of its simple cycles is $p = 2$. This automaton has six accepting SCC-paths, including

$$\pi_1 = q_0, 0 \rightsquigarrow q_0, 0 \xrightarrow{a} q_1, 1 \rightsquigarrow q_1, 1 \xrightarrow{a} q_3, 0 \rightsquigarrow q_3, 0 \xrightarrow{b} q_4, 1 \rightsquigarrow q_4, 1$$

$$\pi_2 = q_0, 0 \rightsquigarrow q_0, 0 \xrightarrow{a} q_2, 1 \rightsquigarrow q_1, 0 \xrightarrow{a} q_3, 1 \rightsquigarrow q_3, 0 \xrightarrow{b} q_4, 1 \rightsquigarrow q_4, 1$$

The language of the portal π_1 is $a(ba)^*a(a^2)^*b$. A blocking sequence for this SCC-path is $((0 : aa), (0 : b))$, which is in fact blocking for all of the SCC-paths.

On the other hand, $((0 : ab))$ is not blocking for π_1, as $(0 : ab)$ is not a blocking factor for the portal $q_1, 1 \rightsquigarrow q_1, 1$. It is, however, a blocking sequence for π_2. This is because if we enter the SCC $\{q_1, q_2\}$ through q_1, a factor ab can only appear after an even number of steps, while if we enter through q_2, it can only appear after an odd number of steps.

Figure 4 Automaton used for Example 4.15.

4.2   An efficient property tester

In this section, we show that for any regular language L and any small enough $\varepsilon > 0$, there is an ε-property tester for L that uses $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries.

▶ **Theorem 4.16.** *For any NFA $\mathcal{A}$ and any small enough $\varepsilon > 0$, there exists an ε-property tester for $\mathcal{L}(\mathcal{A})$ that uses $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries.*

As mentioned in the overview, this result supersedes the one given by Bathie and Starikovskaya [9]: while both testers use the same number of queries, the tester in [9] works under the edit distance, while that of Theorem 4.16 is designed for the Hamming distance. As the edit distance never exceeds the Hamming distance, the set of words that are ε-far with respect to the former is contained in the set of words ε-far for the latter. Therefore, an ε-tester for the Hamming distance is also an ε-tester for the edit distance, and this result is stronger.

The property tester behind Theorem 4.16 uses the property tester for strongly connected NFAs as a subroutine, and its correctness is based on an extension of Lemma 3.13 to blocking sequences. We show that we can reduce property testing of $\mathcal{L}(\mathcal{A})$ to a search for blocking sequences in the word, in the following sense:

- If μ contains a strongly blocking sequence for each of the SCC-paths of $\mathcal{A}$, then it is not in the language and we can answer no (Corollary 4.17).
- If μ is ε-far from the language, then for each accepting SCC-path π of $\mathcal{A}$, μ is far from for the language of π and contains many disjoint strongly blocking sequences for π (Lemma 4.18), hence random sampling is likely to find at least one of them, and we reject μ with constant probability.

▶ **Corollary 4.17.** *If μ contains a strongly blocking sequence for each SCC-path of $\mathcal{A}$, then $\mu \notin \mathcal{TL}(\mathcal{A})$.*

Proof. This follows from Lemma 4.10.   ◀

The next lemma expresses a partial converse to Corollary 4.17 and generalizes Lemma 3.12 from the strongly connected case: if a word is far from the language, then it contains many strongly blocking sequences for any SCC-path.

▶ **Lemma 4.18.** *Let $\pi = P_0 \xrightarrow{a_1} \cdots P_k$ be an SCC-path, let $L = \mathcal{L}(\pi)$, and let μ be a positional word of length n such that $d(\mu, L)$ is finite. There is a constant C such that if $n \geq C/\varepsilon$ and μ is ε-far from L, then μ can be partitioned into $\mu = \mu_0\mu_1 \cdots \mu_k$ such that for every $i = 0, \ldots, k$, μ_i contains at least $\frac{\varepsilon n}{C}$ disjoint blocking factors for P_i, each of length at most $\mathcal{O}(1/\varepsilon)$.*

Proof. We proceed similarly to the proof of Lemma 3.12, and only sketch this proof. Starting from the left end of μ, we accumulate letters until we find a factor blocking for P_0, and iterate again starting from p positions later, where p is the lcm of the length of all cycles in $\mathcal{A}$; notably, it is a multiple of the reachability constant of a strongly connected automaton recognizing $\mathcal{L}(P_0)$. When we have found at least $K = \frac{\varepsilon n}{C}$ blocking factors (C is to be determined later) for $\mathcal{L}(P_i)$, this position marks the end of μ_i, and we iterate with the next portal in π.

Let us assume that the process ends (i.e. we reach the right end of μ) before finding enough blocking factors for all portals. We show that in this case, the distance between μ and L is at most εn. Assume that we stop before finding enough blocking factors for the i-th portal, P_i. As in the proof of Lemma 3.12, we replace the last letter of each blocking factor and use the padding between them to make the run accepted by the SCC-path: this uses at most $((i+1) \cdot (K+1) + 2)p$ substitutions. If we set $C = 4(k+3)p$, this is less than εn when $n \geq C/\varepsilon$. Therefore, if μ is ε-far from $\mathcal{L}(\pi)$, then the decomposition process finds at least K blocking factors for P_i in μ_i for each i.

Then, since all of these factors are disjoint, we can use the same technique as in Lemma 3.13 to show that at least half of these factors have length $\mathcal{O}(1/\varepsilon)$, and the result holds, up to doubling C. ◀

▶ **Corollary 4.19.** *Let $L = \mathcal{TL}(\mathcal{A})$ and let μ be a positional word of length n. If L contains a word of length n and μ is ε-far from L, then μ contains $\Omega(\varepsilon n)$ disjoint blocking sequences for $\mathcal{A}$.*

Proof. We use a proof identical to that of Lemma 4.18, except that we consider a linear ordering of all the portals of $\mathcal{A}$ given by topological ordering, instead of the linear given by an SCC-path. The graph used for the topological ordering is the graph of all portals of $\mathcal{A}$, with an edge from P to P' when P and P' appear consecutively in some SCC-path of $\mathcal{A}$. Since any two portals in an SCC-path are from different SCCs of $\mathcal{A}$, this graph is acyclic, and its vertices can be topologically ordered. ◀

We are now ready to prove Theorem 4.16.

Proof of Theorem 4.16. Our algorithm iterates over all K accepting SCC-paths $\pi = P_0 \xrightarrow{a_1} \ldots \xrightarrow{a_k} P_k$ of $\mathcal{A}$, and for each π, searches for blocking sequences for π in $\mu = (0 : u)$. If we find a strongly blocking sequence for π in μ, then by Lemma 4.13, μ is not in $\mathcal{TL}(\mathcal{A})$ and we can reject. Note that if $\mu \in \mathcal{TL}(\mathcal{A})$, then the algorithm will not reject, hence the perfect completeness property is satisfied.

Next, we show that if μ is ε-far from $\mathcal{TL}(\mathcal{A})$, then we can find a strongly blocking sequence for π with probability at least $1 - 1/(3K)$ using $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries. Our algorithm is based on the following observation:

▶ **Observation 4.20.** *Let $\pi = P_0 \xrightarrow{a_1} \cdots P_k$ be an SCC-path. Let $\nu_0, \ldots, \nu_k$ be positional words such that ν_i is blocking for P_i. Then, $\sigma = (\nu_0, \ldots, \nu_k)$ is a strongly blocking sequence of π.*

We can assume w.l.o.g. that μ has length at least $2C/\varepsilon$, where C is the constant defined in Lemma 4.18, otherwise we can read all of μ using $\mathcal{O}(1/\varepsilon)$ queries. Therefore, we can apply Lemma 4.18, and μ can be partitioned into $k+1$ words $\mu_0, \ldots, \mu_k$ such that each μ_i contains at least $\varepsilon n/C$ disjoint blocking factors for C, each of length $L = \mathcal{O}(1/\varepsilon)$.

For each i, we can use the algorithm of Lemma 3.9 to sample from μ a set $\mathcal{F}$ that contains a factor that contains a ν_i with probability at least $2/3$. By repeating the procedure

$\mathcal{O}(\ln(3K \cdot (k+1)))$ times and taking the union of all returned sets $\mathcal{F}$, we can increase this probability to $1 - \frac{1}{3K \cdot (k+1)}$. Then, by the union bound, we find a blocking factor ν_i for each P_i in the corresponding μ_i with probability at least $1 - 1/(3K)$. As observed above, the sequence $\sigma = (\nu_0, \ldots, \nu_k)$ is strongly blocking for π.

By union bound again, this algorithm finds a strongly blocking sequence for each of the K SCC-paths in $\mathcal{A}$, and therefore rejects μ, with probability at least $2/3$.

For a single μ_i of a given SCC-path, the sampling procedure uses $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries (by Claim 3.8). As the lengths and number of SCC-paths in $\mathcal{A}$ does not depend on the input length, this algorithm uses $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries in total. ◀

4.3 Lower bound

In order to characterize hard languages for all automata, we define a partial order $\trianglelefteq$ on sequences of positional factors. It is an extension of the factor partial order on blocking factors. It will let us define *minimal blocking sequences*, which we use to characterize the complexity of testing a language.

▶ **Definition 4.21** (Minimal blocking sequence)**.** *Let $\sigma = (\mu_1, \mu_2, \ldots, \mu_k)$ and $\sigma' = (\mu'_1, \ldots, \mu'_t)$ be sequences of positional words. We have $\sigma \trianglelefteq \sigma'$ if there exists a sequence of indices $i_1 \leq i_2 \leq \ldots \leq i_k$ such that μ_j is a factor of μ'_{i_j} for all $j = 1, \ldots, k$.*

A blocking sequence σ of $\mathcal{A}$ (resp. π) is minimal *if it is a minimal element of $\trianglelefteq$ among blocking sequences of $\mathcal{A}$ (resp. π). The set of minimal blocking sequences of $\mathcal{A}$ (resp. π) is written MBS($\mathcal{A}$) (resp. MBS(π)).*

▶ **Remark 4.22.** *If $\sigma \trianglelefteq \sigma'$ and σ is a blocking sequence for an SCC-path π then σ' is also a blocking sequence for π.*

We make the remark that minimal blocking sequences have a bounded number of terms. This is because if we build the sequence from left to right by adding terms one by one, the minimality implies that at each step we should block a previously unblocked portal.

▶ **Lemma 4.23.** *A minimal blocking sequence for $\mathcal{A}$ contains at most $p^2|Q|^2$ terms.*

Proof. First, remark that there at most $p^2|Q|^2$ portals in $\mathcal{A}$. Let $\sigma = (\mu_1, \ldots, \mu_\ell)$ be a minimal blocking sequence for $\mathcal{A}$. For all $i = 1, \ldots, \ell$, we define $\sigma_i = (\mu_1, \ldots, \mu_i)$, and σ_0 is the empty sequence.

Then, for each i, we consider the set $\mathcal{S}_i$ of portals P such that for all accepting SCC-path π of $\mathcal{A}$ containing P, the prefix of π ending at P is blocked by σ_i. We have $\mathcal{S}_0 = \emptyset$, and $\mathcal{S}_\ell$ is the set of all portals of $\mathcal{A}$.

We claim that for every $i < \ell$, $\mathcal{S}_i$ is a proper subset of $\mathcal{S}_{i+1}$. Otherwise, if $\mathcal{S}_i = \mathcal{S}_{i+1}$, then removing μ_{i+1} from σ gives a blocking sequence σ' of $\mathcal{A}$, such that $\sigma' \trianglelefteq \sigma$, contradicting the minimality of σ. Therefore, it follows that $\ell \leq p^2|Q|^2$. ◀

4.3.1 Reducing to the strongly connected case

To prove a lower bound on the number of queries necessary to test a language when MBS($\mathcal{A}$) is infinite, we present a reduction to the strongly connected case. Under the assumption that $\mathcal{A}$ has infinitely many minimal blocking sequences, we exhibit a portal P of $\mathcal{A}$ with infinitely many minimal blocking factors and "isolate it" by constructing two sequences of positional factors σ_l and σ_r such that for all μ, $\sigma_l, (\mu), \sigma_r$ is blocking for $\mathcal{A}$ if and only if μ is a blocking factor of P. Then we reduce the problem of testing the language of this portal to the problem of testing the language of P.

To define "isolating P" formally, we define the left (and right) effect of a sequence on an SCC-path. Informally, the left effect of a sequence σ on an SCC-path π is related to the index of the first portal in π where a run can be after reading σ, because all previous portals have been blocked. The right effect represents the same in reverse, starting from the end of the run.

More formally, the *left effect* of a sequence σ on an SCC-path $\pi = P_0 \xrightarrow{a_1} \cdots P_k$ is the largest index i such that the sequence is blocking for $P_0 \xrightarrow{a_1} \cdots P_i$ (-1 if there is no such i). We denote it by $(\sigma \gg \pi)$. Similarly, the *right effect* of a sequence on π is the smallest index i such that the sequence is blocking for $P_i \xrightarrow{a_{i+1}} \cdots P_k$ ($k+1$ if there is no such i); we denote it by $(\pi \ll \sigma)$.

▶ Remark 4.24. A sequence σ is blocking for an SCC-path $\pi = P_0 \xrightarrow{a_1} \cdots P_k$ if and only if $(\sigma \gg \pi) = k$, if and only if $(\pi \ll \sigma) = 0$.

Also, given two sequences σ_l, σ_r, the sequence $\sigma_l \sigma_r$ is blocking for π if and only if $(\sigma_l \gg \pi) \geq (\pi \ll \sigma_r)$.

For the next lemma we define a partial order on portals: $P \preceq P'$ if all blocking factors of P' are also blocking factors of P. We write $\succeq$ for the reverse relation, $\simeq$ for the equivalence relation $\preceq \cap \succeq$ and $\not\simeq$ for the complement relation of $\simeq$.

Additionally, given an SCC-path $\pi = P_0 \xrightarrow{x_1} \ldots P_k$ and two sequences of positional words σ_l, σ_r, we say that the portal P_i *survives* (σ_l, σ_r) *in* π if $(\sigma_l \gg \pi) < i < (\pi \ll \sigma_r)$.

▶ **Definition 4.25.** *Let P be a portal and σ_l and σ_r sequences of positional words.*
   We define three properties that those objects may have:
P1) *$\sigma_l \sigma_r$ is not blocking for $\mathcal{A}$*
P2) *P has infinitely many minimal blocking factors*
P3) *for any accepting SCC-path π in $\mathcal{A}$, every portal in π which survives (σ_l, σ_r) is $\simeq$-equivalent to P.*

▶ **Lemma 4.26.** *If $\mathcal{A}$ has infinitely many minimal blocking sequences, then there exist a portal P and sequences σ_l and σ_r satisfying properties P1, P2 and P3.*

Proof. By Lemma 4.23, a minimal blocking sequence has a bounded number of elements. Therefore, if $\mathcal{A}$ as an infinite number of minimal blocking sequences, there exists an integer i_* and an infinite family $(\sigma_j)_{j \in \mathbb{N}}$ of minimal blocking sequences of $\mathcal{A}$ such that the length of i_*-th term of σ_j is at least j, for every j. For each j, let $\sigma_{j,l}$ denote the sequence containing the elements of σ_j, up to index $i_* - 1$, and let $\sigma_{j,r}$ denote the sequence with the elements starting from index $i_* + 1$. As there is a finite number of SCC-paths in $\mathcal{A}$, we can extract from the sequence $(\sigma_j)_j$ an infinite subsequence $(\sigma'_j)_{j \in \mathbb{N}}$ such that for all SCC-paths π of $\mathcal{A}$, all of the $\sigma_{j,l}$ have the same left effect as $\sigma_l = \sigma_{0,l}$ on π, and symmetrically for the right effect of the $(\sigma_{j,r})_j$ and $\sigma_r = \sigma_{0,r}$.

Then, we can replace $\sigma_{j,l}$ with σ_l and $\sigma_{j,r}$ with σ_r in each σ'_j, to obtain an infinite sequence of minimal blocking sequences of the form $(\sigma_l, \nu_j, \sigma_r)$, where each ν_j is a positional word of length at least j. As these blocking sequences are minimal, the pair (σ_l, σ_r) is not blocking for $\mathcal{A}$, there is an accepting SCC-path π_* and a portal P_* that survives (σ_l, σ_r) in that π_*. If there are multiple possible choices for π_* and P_*, we choose them so that P_* is $\preceq$-minimal among the possible choices. The following claim shows that we can choose such a P_* with infinitely many minimal blocking factors.

▷ Claim 4.27.   There exists such a P_* with infinitely many minimal blocking factors.

Proof. The word ν_j is blocking for all portals that survive (σ_l, σ_r), and there are arbitrarily long ν_j such that $(\sigma_l, \nu_j, \sigma_r)$ is a minimal blocking sequence. Therefore, all letters in each ν_j must belong to a minimal blocking factor of some $\preceq$-minimal portal, hence one of them has infinitely many minimal blocking factors. ◁

So far, properties P1 and P2 are satisfied. Next, we extend the sequences σ_l and σ_r until the property P3 is satisfied, while preserving properties P1 and P2.

▷ **Claim 4.28.** There exist σ_l, σ_r such that $\sigma_l \sigma_r$ is not a blocking sequence for $\mathcal{A}$, and for any accepting SCC-path π in $\mathcal{A}$, every surviving portal in π is $\simeq$-equivalent to P_*.

Proof. Note that for each $P \not\simeq P_*$, we can pick a positional word τ_P that is blocking for P but not for P_*, since P_* is $\preceq$-minimal.

We extend σ_l and σ_r as follows. While there is a surviving portal P that is not $\simeq$-equivalent to P_*:

- We pick an SCC-path $\pi = P_0 \xrightarrow{a_1} \ldots P_k$ such that P survives in π.
- Let $i_\ell = (\sigma_l \gg \pi)$ and $i_r = (\pi \ll \sigma_r)$
- If for all $i \in \{i_\ell + 1, \ldots, i_r - 1\}$, $P_i \not\simeq P_*$ then we append at the end of σ_l the sequence $\tau_{P_{i_\ell+1}}, \ldots, \tau_{P_{i_r-1}}$. The sequence $\sigma_l \sigma_r$ is now blocking for π. On the other hand, since we did not add any blocking factor for P_*, there must still be a surviving portal that is $\simeq$-equivalent to it.
- If there is an $i \in \{i_\ell + 1, \ldots, i_r - 1\}$ such that $P_i \simeq P_*$ then let c be the maximal index in $\{i_\ell + 1, \ldots, i\}$ such that P_c is not equivalent to P_* for $\simeq$, or i_ℓ if there is no such index. Symmetrically, let d the minimal index in $\{i, \ldots, i_r - 1\}$ such that $P_d \not\simeq P_*$, or i_r if there is no such index. We append at the end of σ_l the sequence $\tau_{P_{i_\ell+1}}, \ldots, \tau_{P_c}$. We append at the beginning of σ_r the sequence $\tau_{P_d}, \ldots, \tau_{P_{i_r-1}}$. Now all surviving portals in π are $\simeq$-equivalent to P_*, and P_i still survives.

We iterate this step until all surviving portals are $\simeq$-equivalent to P_*. We made sure that at least one portal was still surviving after each step, hence in the end the sequence $\sigma_l \sigma_r$ is not blocking for $\mathcal{A}$. ◁

◀

▶ **Lemma 4.29.** *Let $\pi = P_0 \xrightarrow{a_1} \cdots P_\ell$ be an accepting SCC-path, denote $P_j = s_j, x_j \leadsto t_j, y_j$ for each $j = 0, \ldots, \ell$, let $i \in \{0, \ldots, \ell\}$, and let $\sigma_l = (\nu_{1,l}, \ldots, \nu_{k,l})$ be a sequence such that $(\sigma_l \gg \pi) < i$.*

Then, for any integer $N \in \mathbb{N}$, there is a positional word w_l^* of length at most $(3|\mathcal{A}|^3 + |\mathcal{A}|)(k+1) + N(2p^2 + p)k|\mathcal{A}| + pN \sum_{t=1}^{k} |\nu_{t,l}|$ such that $|w_l^*| = x_i - x_0 \pmod{p}$, there is a run reading w_l^* from s_0 to s_i in $\mathcal{A}$, and $(x_0 : w_l^*)$ contains N occurrences of $\nu_{1,l}$, followed by N occurrences of $\nu_{2,l}$, etc. up to $\nu_{k,l}$, all disjoint.

Proof. We define w_l^* by induction on k, the length of σ_l. As π is accepting, by definition its language $\mathcal{L}(\pi)$ is nonempty, and thus for all $j \in \{0, \ldots, \ell\}$, there exists a word u_j of length $y_j - x_j \pmod{p}$ that labels a path from s_j to t_j. By Fact 2.3, there is such a word u_j of length at most $3|\mathcal{A}|^2$. As a result, for all $z \in \{0, \ldots, \ell\}$ we can form a word $w_z = u_0 a_1 u_1 \cdots a_z$, of length at most $3|\mathcal{A}|^3 + |\mathcal{A}|$, that labels a path of length $x_z - x_0 \pmod{p}$ from q_0 to s_z in $\mathcal{A}$. If $k = 0$, we can simply set $w_l^* = w_i$.

Let $k > 0$, and assume that the lemma holds for $k - 1$. Let $j = (\nu_{1,l} \gg \pi)$. As $(\nu_{1,l} \gg \pi) \le (\sigma_l \gg \pi) < i$, we have $j < i$, hence $\nu_{1,l}$ is not blocking for P_{j+1}. As a consequence, there is a word v_j that labels a path from s_j to t_j such that $\tau_j = (x_j : v_j)$ has

$\nu_{1,l}$ as a factor. We can remove cycles of length 0 (mod p) in that path, before and after reading τ_j, so we can assume that $|v_j| \leq |\nu_{1,l}| + 2p|\mathcal{A}|$. As s_j and t_j are in the same SCC, we can extend v_j into a word v'_j of length at most $|v_j| + |\mathcal{A}| \leq |\nu_{1,l}| + (2p+1)|\mathcal{A}|$ that labels a cycle from s_j to itself.

Let $\sigma' = (\nu_{2,l}, \ldots, \nu_{k,l})$ and $\pi' = P_{j+1} \xrightarrow{a_{j+2}} \cdots P_\ell$. As σ_l is the concatenation of $\nu_{1,l}$ and σ', and $j = (\nu_{1,l} \gg \pi)$, we have $(\sigma' \gg \pi') < i - j - 1$. By induction hypothesis, there is a word w' of length at most $(3|\mathcal{A}|^3 + |\mathcal{A}|)k + N(2p^2 + p)(k-1)|\mathcal{A}| + pN \sum_{t=2}^k |\nu_{t,l}|$ such that $|w'| = x_i - x_{j+1}$ (mod p), there is a run reading w' from s_{j+1} to s_i in $\mathcal{A}$, and $(x_{j+1} : w')$ contains N occurrences of $\nu_{t,l}$, all disjoint, for each $t = 2, \ldots, k$.

We set $w_l^* = w_{j+1}(v'_j)^{pN} w'$. This word has length $x_i - x_0$ (mod p), and satisfies:

$$\begin{aligned}
|w_l^*| &\leq |w_{j+1}| + pN|v'_j| + |w'| \\
&\leq 3|\mathcal{A}|^3 + |\mathcal{A}| + pN(|\nu_{1,l}| + (2p+1)|\mathcal{A}|) + |w'| \\
&\leq (3|\mathcal{A}|^3 + |\mathcal{A}|)(k+1) + N(2p^2 + p)k|\mathcal{A}| + pN \sum_{t=1}^k |\nu_{t,l}|.
\end{aligned}$$

By construction, the word $(x_0 : w_l^*)$ labels a path from s_0 to s_i, and contains N occurrences of $\nu_{1,l}$, followed by N occurrence of $\nu_{2,l}$, etc. up to $\nu_{k,l}$, all disjoint, which concludes the proof.   ◄

▶ **Lemma 4.30.** *Let $\pi = P_0 \xrightarrow{a_1} \cdots P_\ell$ be an accepting SCC-path, denote $P_j = s_j, x_j \rightsquigarrow t_j, y_j$ for each $j = 0, \ldots, \ell$, let $i \in \{0, \ldots, \ell\}$, and let $\sigma_r = (\nu_{1,r}, \ldots, \nu_{k,r})$ be a sequence such that $(\sigma_l \gg \pi) < i$.*

Then, for any integer $N \in \mathbb{N}$, there is a word w_r^* of length at most $(3|\mathcal{A}|^3 + |\mathcal{A}|)(k+1) + N(2p^2 + p)k|\mathcal{A}| + pN \sum_{i=1}^k |\nu_{i,r}|$ such that $|w_r^*| = x_i - x_0$ (mod p), there is a run reading w_r^* from s_0 to s_i in $\mathcal{A}$, and $(x_0 : w_r^*)$ contains N occurrences of $\nu_{1,r}$, followed by N occurrence of $\nu_{2,r}$, etc. up to $\nu_{k,r}$, all disjoint.

Proof. By a proof symmetric to the one of the previous lemma.   ◄

Given a sequence σ, define $||\sigma||$ as the sum of the lengths of the terms of σ.

▶ **Lemma 4.31.** *If there exist a portal P and σ_l, σ_r satisfying properties P1, P2 and P3 then $\mathcal{L}(\mathcal{A})$ is hard.*

Proof of Lemma 4.31. A direct consequence of properties P1 and P3 is that for all ν', then $\sigma_l \nu' \sigma_r$ is blocking for $\mathcal{A}$ if and only if ν' is blocking for P.

The proof goes as follows: we show that we can turn an algorithm testing $\mathcal{L}(\mathcal{A})$ with $f(\varepsilon)$ samples into an algorithm testing $\mathcal{L}(P)$ with $f(\varepsilon/X)$ samples with X a constant. We then apply Theorem 3.14 from the strongly connected case to obtain the lower bound.

Consider an algorithm testing $\mathcal{L}(\mathcal{A})$ with $f(\varepsilon)$ samples for some function f. We describe an algorithm for testing $\mathcal{L}(P)$. Say we are given a threshold ε and a word v of length n. First of all we can apply Lemmas 4.29 and 4.30 to compute two words w_l^* and w_r^* of length at most $E + \varepsilon n F$ for some constants E and F such that we can read w_l^* from q_0 to s and w_r^* from t to q_f and w_l^* contains occurrences of each element of σ_l at least εn times, all disjoint, with all occurrences of the i-th of σ_l appearing before element j for $i < j$, and similarly for w_r^* and σ_r. Let $w = w_l^* v w_r^*$, and assume that $|v| \geq \frac{6p^2|\mathcal{A}|^2}{\varepsilon}$ and that $d(v, \mathcal{L}(P)) < +\infty$.

- If $v \in \mathcal{L}(P)$ then clearly $w \in \mathcal{L}(\mathcal{A})$.

- If $d(v, \mathcal{L}(P)) \geq \varepsilon n$ then by Lemma 3.12 (in light of Lemma 4.7), $(x : v)$ contains at least $\frac{\varepsilon n}{6p^2|\mathcal{A}|^2}$ blocking factors for P. Then we have that w contains at least $\frac{\varepsilon n}{6p^2|\mathcal{A}|^2}$ disjoint blocking sequences for $\mathcal{A}$. As a result, $d(w, \mathcal{L}(\mathcal{A})) \geq \frac{\varepsilon n}{6p^2|\mathcal{A}|^2}$. We divide this by the length of w, which is at most $2E + 2F\varepsilon n + n$. We obtain that $d(w, \mathcal{L}(\mathcal{A})) \geq \frac{\varepsilon}{X}|w|$ for some constant X.

Let us now describe the algorithm for testing $\mathcal{L}(P)$.

- If $\mathcal{L}(P) \cap \Sigma^n = \emptyset$ then we reject.
- If $|v| < \frac{6p^2|\mathcal{A}|^2}{\varepsilon}$ then we read v entirely and check that it is in $\mathcal{L}(P)$.
- If $v \in \mathcal{L}(P)$ then we apply our algorithm for testing $\mathcal{L}(\mathcal{A})$ on $w = w_l^* v w_r^*$ with parameter $\varepsilon' = \frac{\varepsilon}{X}$.

The number of queries used on v is at most the number of queries needed on w, hence at most $f(\varepsilon/X)$ queries. We obtain a procedure to test $\mathcal{L}(P)$ using $f(\varepsilon/X)$ queries. By Theorem 3.14, $f(\varepsilon/X) = \Omega(\log(\varepsilon^{-1})/\varepsilon)$, hence $f(\varepsilon) = \Omega(\log(\varepsilon^{-1})/\varepsilon)$. This concludes our proof. ◀

▶ **Proposition 4.32.** *If $\mathcal{A}$ has infinitely many minimal blocking sequences, then $\mathcal{L}(\mathcal{A})$ is hard.*

Proof. We combine Lemmas 4.26 and 4.31. ◀

5   Trivial and Easy languages

5.1   Upper bound for easy languages

We first establish that an automaton with finitely many minimal blocking sequences is easy (or trivial) to test.

▶ **Lemma 5.1.** *Let $\mathcal{A}$ be an NFA with a finite number of minimal blocking sequences, let $\pi = P_0 \xrightarrow{a_1} \cdots P_k$ be an SCC-path of $\mathcal{A}$, let $L = \mathcal{L}(\pi)$, and let μ be a positional word of length n such that $d(\mu, L)$ is finite. There are constants B, D such that if $n \geq 2D/\varepsilon$ and μ is ε-far from L, then μ can be partitioned into $\mu = \tau_0 \tau_1 \cdots \tau_k$ such that for every $i = 0, \ldots, k$, τ_i contains at least $\frac{\varepsilon n}{D}$ disjoint blocking factors for P_i, each of length at most B.*

Proof. By Corollary 4.19, the positional word μ contains $N \geq \varepsilon n/C$ disjoint blocking sequences $(\sigma_j)_{j=1,\ldots,N}$ for $\mathcal{A}$, for some constant C. We can extract from each σ_j a minimal blocking sequence $\sigma_j' = (\nu_{0,j}, \ldots, \nu_{s_j,j})$. By definition of blocking sequences, σ_j' is also blocking for π.

As $\mathcal{A}$ has a finite number of minimal blocking sequences, hence there is a constant B such that any $\nu_{i,j}$ has length at most B.

We build the decomposition $\mu = \tau_0 \tau_1 \cdots \tau_k$ with the following iterative process. For the index $i = 0$, we set τ_0 to the shortest prefix of μ that contains the leftmost $N/(k+1)$ components of the σ_j' that are blocking for P_0. Since the $(\sigma_j)_j$ are disjoint in μ, so are the $(\nu_{i,j})_{i,j}$, and this leaves us with at least $N(1 - 1/(k+1))$ of the σ_j' that have their component blocking for P_0, and therefore also for P_1 in the part of μ outside of τ_0. We then iterate again for $i = 1, \ldots, k + 1$, with the invariant that at step i, we have $N(1 - i/(k+1))$ of the σ_j' that have their component blocking for P_i outside for $\tau_0 \ldots \tau_{i-1}$. We then take for τ_i the shortest prefix of the rest of μ that contains the leftmost $N/(k+1)$ components of these σ_j' that are blocking for P_i.

At each step, the factor τ_i contains $N/(k+1)$ blocking factors for P_i, hence the decomposition $\mu = \tau_0 \tau_1 \cdots \tau_k$ has the desired property for $D = C \cdot (k+1)$. ◀

▶ **Proposition 5.2.** *If $\mathcal{A}$ has finitely many minimal blocking sequences, then there is a tester for $\mathcal{L}(\mathcal{A})$ that uses $\mathcal{O}(1/\varepsilon)$ queries.*

Proof. We use the same algorithm that for Theorem 4.16, except that we use the factors given by Lemma 5.1, therefore, in the call to the SAMPLER function (Algorithm 1), the upper bound on the length of the factors is B instead of $\mathcal{O}(1/\varepsilon)$. In that case, the query complexity becomes $\mathcal{O}(\log(B)/\varepsilon) = \mathcal{O}(1/\varepsilon)$. ◀

This already gives us a clear dichotomy: all languages either require $\Theta(\log(\varepsilon^{-1})/\varepsilon)$ queries to be tested, or can be tested with $\mathcal{O}(1/\varepsilon)$ queries.

5.2 Separation between trivial and easy languages

It remains to show that languages that can be tested with $\mathcal{O}(1/\varepsilon)$ queries have query complexity either $\Theta(1/\varepsilon)$, or 0 for large enough n. Our proof uses the class of *trivial* regular languages identified by Alon et al. [5], which we revisit next.

An example of a trivial language is L_2 consisting of words containing at least one a over the alphabet $\{a, b\}$. For any word u, replacing any letter by a yields a word in L_2, hence $d(u, L_2) \leq 1$. Therefore, for $n > 1/\varepsilon$, no word of length n is ε-far from L_2, and the trivial property tester that answers "yes" without sampling any letter is correct.

Alon et al. [5] define non-trivial languages as follows.

▶ **Definition 5.3** ([5, Definition 3.1]). *A language L is non-trivial if there exists a constant $\varepsilon_0 > 0$, so that for infinitely many values of n the set $L \cap \Sigma^n$ is non-empty, and there exists a word $w \in \Sigma^n$ so that $d(w, L) \geq \varepsilon_0 n$.*

It is easy to see that if a language is trivial in the above sense (i.e. not non-trivial), then for large enough input length n, the answer to testing membership in L only depends n, and the algorithm does not need to query the input. Alon et al. [5, Property 2] show that if a language is non-trivial, then testing it requires $\Omega(1/\varepsilon)$ queries for small enough $\varepsilon > 0$.

To obtain our characterization of *trivial* languages, we show that $\mathsf{MBS}(\mathcal{A})$ is non-empty if and only if $\mathcal{L}(\mathcal{A})$ is non-trivial (in the above sense). It follows that if $\mathsf{MBS}(\mathcal{A})$ is empty, then testing $\mathcal{L}(\mathcal{A})$ requires 0 queries for large enough n. Furthermore, by the result of Alon et al. [5], if $\mathsf{MBS}(\mathcal{A})$ is non-empty, then testing $\mathcal{L}(\mathcal{A})$ requires $\Omega(1/\varepsilon)$ queries.

Recall that we focus on infinite languages, since we know that all finite ones are trivial (Remark 1.4).

▶ **Lemma 5.4.** *$\mathsf{MBS}(\mathcal{A})$ is empty if and only if $L = \mathcal{L}(\mathcal{A})$ is trivial.*

We prove the two directions separately.

▶ **Lemma 5.5.** *If $\mathsf{MBS}(\mathcal{A})$ is empty, then $L = \mathcal{L}(\mathcal{A})$ is trivial in the sense of Definition 5.3.*

Proof. We showed in Corollary 4.19 that if μ is long enough and ε-far from L, then μ contains $\Omega(\varepsilon n)$ disjoint blocking sequences for $\mathcal{A}$. As $\mathcal{A}$ has no minimal blocking sequences, it does not have blocking sequences either, and long enough words cannot be ε-far from L, hence it is trivial in the sense of Definition 5.3. ◀

To prove the converse property, we need the following extension of Kleene's Lemma for languages of SCC-paths: for large enough ℓ, whether $\mathcal{L}(\pi)$ contains a word of length ℓ only depends on the value of ℓ modulo p (p is the lcm of all the lengths of the simple cycles in $\mathcal{A}$).

▶ **Lemma 5.6.** *Let $\pi = P_0 \xrightarrow{a_1} \cdots P_k$ be an SCC-path. There exists a constant B such that, for all $\ell \geq B$, if there is a word μ of length ℓ in $\mathcal{L}(\pi)$, then there exists a word μ' of length $\ell - p$ and a word μ'' of length $\ell + p$ in $\mathcal{L}(\pi)$.*

Proof. Recall the definition of $\mathcal{L}(\pi)$ (Definition 4.9):

$$\mathcal{L}(\pi) = L_0 a_1 L_1 a_2 \cdots L_k, \text{ where } L_i = \mathcal{L}(P_i) \text{ for } i = 0, \ldots, k.$$

It follows that a word $\mu \in \mathcal{L}(\pi)$ can be written as $\mu = \mu_1 a_1 \mu_2 \ldots \mu_k$ with $\mu_i \in L_i$. Each L_i is recognized by a strongly connected automaton $\mathcal{A}_i$ with at most $p|\mathcal{A}|$ states. Let $B = 5(p|\mathcal{A}|)^2$. If the length ℓ of μ exceeds B, then the run of μ in each of the $\mathcal{A}_i$'s contains simple loops with sum of lengths greater than $p + 3(p|\mathcal{A}|)^2$. Let $\ell_0 + p$ denote the sum of the length of these simple cycles: by construction ℓ_0 is greater than $3(p|\mathcal{A}|)^2$. We remove these simple cycles from the run: the resulting run is still in $\mathcal{L}(\pi)$. Next, select any non-trivial SCC S_i in π and let s be a state of S_i used by the run. As $\ell_0 \geq 3(p|\mathcal{A}|)^2$, by Fact 2.3, there is a path of length ℓ_0 from s to itself in $\mathcal{A}_i$. Adding this path to the run yields an accepting run of length $\ell - (\ell_0 + p) + \ell_0 = \ell - p$: the word labeling this run is the desired word μ'.

To obtain μ'', consider any simple cycle in the run of μ in $\mathcal{A}$, and let m denote the length of this cycle. By definition of p, m divides p. Iterating this cycle p/m times yields a word μ'' of length $\ell + p$ that is in $\mathcal{L}(\pi)$. ◀

▶ **Corollary 5.7.** *Let π be an SCC path. For large enough ℓ, whether there is an word of length ℓ in $\mathcal{L}(\pi)$ only depends on the value of $\ell \pmod p$.*

To finish our characterization of trivial languages, we show that if $\mathsf{MBS}(\mathcal{A})$ is not empty, then $L = \mathcal{L}(\mathcal{A})$ is non-trivial in the sense of Alon et al. [5].

▶ **Lemma 5.8.** *Let $\mathcal{A}$ be a trim NFA such that $L = \mathcal{L}(\mathcal{A})$ is infinite. If $\mathcal{A}$ admits a blocking sequence, then there exists $\varepsilon_0 > 0$, such that for infinitely many n there exist words in $\mathcal{L}(\mathcal{A}) \cap \Sigma^n$ and there exists $w \in \Sigma^n$ such that $d(w, \mathcal{L}(\mathcal{A})) \geq \varepsilon_0 n$*

Proof. Let $\sigma = (\mu_1, \ldots, \mu_k)$ be a blocking sequence for $\mathcal{A}$. We can assume w.l.o.g. that σ is strongly blocking for every accepting π of $\mathcal{A}$, as we can make it strongly blocking by concatenating σ to itself K times, where K is the maximum length of an accepting SCC-path in $\mathcal{A}$. Let C be the maximum length of a μ_i's. As L is infinite, there exists an accepting SCC-path π in $\mathcal{A}$ and $w \in \mathcal{L}(\pi)$ with $|w| \geq t$ for arbitrary t. By Corollary 5.7, for all sufficiently large ℓ such that $\ell = |w| \pmod p$, there exists $w' \in \mathcal{L}(\pi)$ with $|w'| = \ell$.

For all $i = 1, \ldots, k$, let ν_i be a shortest word of the form $(0 : v_i)$, for some v_i, and of length ℓ_i equal to 0 modulo p, such that μ_i is a factor of ν_i. By minimality, ℓ_i is at most $C + 2p$. Then, for any integer $N \in \mathbb{N}$, let $w_N = \nu_1^N \cdots \nu_k^N (0 : a^{|w|})$, where a is an arbitrary letter.

As w_N is of length $|w| \pmod p$, there is a word of the same length in $\mathcal{L}(\mathcal{A})$, i.e. $\mathcal{L}(\mathcal{A}) \cap \Sigma^n$ is nonempty. On the other hand, it contains N disjoint occurrences of σ, which is a strongly blocking sequence for every accepting SCC-path of $\mathcal{A}$, therefore, the distance between w_N and $\mathcal{L}(\mathcal{A})$ is at least N. Furthermore, the length of w_N is less than $|w| + N(C + 2p)$. Therefore, if we let $\varepsilon_0 = \frac{1}{C + 2p + |w|}$, then we have $\varepsilon_0 |w_N| \leq N \leq d(w_N, \mathcal{L}(\mathcal{A}))$, i.e. w_N is ε-far from L for any $\varepsilon \leq \varepsilon_0$ and any N. ◀

It is easy to see that if a language is trivial in the above sense, then for large enough input length n, membership in L only depends n, and the algorithm does not need to query the input. Alon et al. [5] show that if a language is non-trivial, then testing it requires $\Omega(1/\varepsilon)$ queries for small enough $\varepsilon > 0$. As a corollary of that lower bound, we obtain that if $\mathsf{MBS}(\mathcal{A})$ is non-empty, then testing $\mathcal{L}(\mathcal{A})$ requires $\Omega(1/\varepsilon)$ queries.

6 Hardness of classifying

In the previous sections, we have shown that testing some regular languages (*easy* ones) that requires fewer queries than testing others (*hard* ones). Therefore, given the task of testing a word for membership in $\mathcal{L}(\mathcal{A})$, it is natural to first try to determine if the language of $\mathcal{A}$ is easy, and if this is the case, run the appropriate ε-tester, that uses fewer queries. In this section, we investigate the computational complexity of checking which class of the trichotomy the language of a given automaton belongs to. We formalize this question as the following decision problems:

▶ **Problem 6.1** (Triviality problem)**.** Given an finite automaton $\mathcal{A}$, is $\mathcal{L}(\mathcal{A})$ trivial?

▶ **Problem 6.2** (Easiness problem)**.** Given an finite automaton $\mathcal{A}$, is $\mathcal{L}(\mathcal{A})$ easy?

▶ **Problem 6.3** (Hardness problem)**.** Given an finite automaton $\mathcal{A}$, is $\mathcal{L}(\mathcal{A})$ hard?

In these problems, the automaton $\mathcal{A}$ is the input and is no longer fixed. We show that, our combinatorial characterization based on minimal blocking sequences is effective, in the sense that all three problems are decidable. However, it does not lead to efficient algorithms, as both problems are PSPACE-complete.

▶ **Theorem 6.4.** *The triviality and easiness problems are both PSPACE-complete, even for strongly connected NFAs.*

In Section 6.1 we show the PSPACE upper bounds on the hardness and triviality problems (Propositions 6.11 and 6.13). The upper bound on the easiness problem follows immediately, as the three properties form a trichotomy.

In Section 6.2, we show that all three problems are PSPACE-hard (Lemma 6.15 and Corollary 6.17).

6.1 A PSPACE upper-bound

6.1.1 Testing hardness

A naive algorithm to check hardness of a language $\mathcal{L}(\mathcal{A})$ would be to construct an automaton recognising blocking sequences of $\mathcal{L}(\mathcal{A})$ (exponential in $\mathcal{A}$), and use it to get an automaton recognising the minimal ones (which requires complementation and could yield another exponential blow-up). This would a priori not give a PSPACE algorithm, since we obtain a doubly-exponential state space. We solve this by providing another characterisation of automata with hard languages, resulting in a recursive PSPACE algorithm to test it.

▶ **Lemma 6.5.** *Let $\pi = P_0 \xrightarrow{a_1} \cdots P_\ell$ be an SCC-path, i an index, Π a set of SCC-paths and $(\sigma_{\pi'})_{\pi' \in \Pi}$ a family of sequences of positional words such that $(\sigma_{\pi'} \gg \pi) < i$ for all π'.*
There exists a sequence of positional words σ such that:

- $(\sigma \gg \pi) < i$
- $(\sigma_{\pi'} \gg \pi') \leq (\sigma \gg \pi')$ *for all $\pi' \in \Pi$.*

Proof. We prove this by induction on the sum of the lengths of the elements of Π. If Π is empty or contains only empty sequences, then we can set σ as the empty sequence.

If not, let π^* be such that the first term ν_1 of $\sigma_{\pi^*} = (\nu_1, \ldots, \nu_k)$ has the least left effect on π among all SCC-paths in Π; let $\pi^* = P_0' \xrightarrow{a_1'} \cdots P_{\ell'}'$. We consider the effect of ν_1 (as a single-element sequence) on π^* and π: let $j = (\nu_1 \gg \pi^*)$ and $r = (\nu_1 \gg \pi)$.

Next, we build a set Π' of SCC-paths as follows. Let $\bar{\pi}$ denote the part of π^* that survives ν_1, if any, i.e. $\bar{\pi} = P'_{j+1} \xrightarrow{a'_{j+1}} \cdots P'_{\ell'}$. We define $\Pi' = \Pi \setminus \{\pi^*\} \cup \{\bar{\pi}\}$ if $j < \ell'$ and $\Pi' = \Pi \setminus \{\pi^*\}$ otherwise. In the first case the sequence associated with $\bar{\pi}$ is $\sigma_{\bar{\pi}} = (\nu_2, \ldots, \nu_k)$.

We now wish to apply the induction hypothesis to the set Π' and the part of π that survives ν_1, i.e. on $\tilde{\pi} = P_{r+1} \xrightarrow{a_{r+1}} \ldots \to P_\ell$, with a target left effect of $i - r - 1$. By construction, the sum of the lengths of the elements in Π' is smaller than that of Π. The following claim shows that, for any π' in Π', the left effect of $\sigma_{\pi'}$ on $\tilde{\pi}$ is at most $i - r - 1$.

▷ **Claim 6.6.** For all $\pi' \in \Pi'$, we have $(\sigma_{\pi'} \gg \tilde{\pi}) < i - r - 1$.

Proof. Let $\pi' \in \Pi \setminus \{\pi^*\}$, and let $\sigma_{\pi'} = (\nu'_1, \ldots, \nu'_m)$. Since the first term of σ_{π^*} was the one with the least left effect on π, the first term of every other sequence has a left effect at least r on it. Formally, let $z = (\nu'_1 \gg \pi)$: we have $z \geq r$.

In other words, ν'_1 is blocking for all portals in $\tilde{\pi}$ up to P_z. Therefore, the sequence $(\nu'_2, \ldots, \nu'_m)$ will be applied to the same portals in π and in $\tilde{\pi}$. Since portal P_i survives in π, it must also survive in $\tilde{\pi}$, and we have $(\sigma_{\pi'} \gg \tilde{\pi}) < i - r - 1$. ◁

By induction hypothesis, we obtain a sequence $\tilde{\sigma}$ such that
- $(\tilde{\sigma} \gg \tilde{\pi}) < i - r - 1$
- $(\sigma_{\pi'} \gg \pi') \leq (\tilde{\sigma} \gg \pi')$ for all $\pi' \in \Pi'$.

Then, the sequence obtained by prepending ν_1 to $\tilde{\sigma}$ satisfies both conditions of the lemma, as $\tilde{\pi}$ is the part of π that survives ν_1, and prepending ν_1 cannot decrease the left effect of a sequence. ◀

▶ **Lemma 6.7.** *An automaton $\mathcal{A}$ is hard if and only if there exists an accepting SCC-path π containing a portal P such that:*
- *P has infinitely many minimal blocking factors.*
- *For any accepting SCC-path π' there exist sequences $\sigma_{l,\pi'}, \sigma_{r,\pi'}$ such that:*
  - *P survives $(\sigma_{l,\pi'}, \sigma_{r,\pi'})$ in π*
  - *All portals surviving $(\sigma_{l,\pi'}, \sigma_{r,\pi'})$ in π' are $\simeq$-equivalent to P*

Proof. The left-to-right direction follows from Proposition 5.2, by taking $\sigma_{l,\pi'} = \sigma_l$ and $\sigma_{r,\pi'} = \sigma_r$ for every π'.

Let us now prove the other direction. Suppose we have π and P satisfying the conditions of the lemma. We only need to construct two sequences σ_l, σ_r such that properties P1 and P3 are satisfied. The result follows by Lemma 4.31.

Let Π be the set of accepting SCC-paths in $\mathcal{A}$. Consider families of sequences $(\sigma_{l,\pi'})_{\pi' \in \Pi}$ and $(\sigma_{r,\pi'})_{\pi' \in \Pi}$ such that for all $\pi' \in \Pi$:
- P survives $(\sigma_{l,\pi'}, \sigma_{r,\pi'})$ in π
- All portals surviving $(\sigma_{l,\pi'}, \sigma_{r,\pi'})$ in π' are $\simeq$-equivalent to P

Let i be the index of P in π. By Lemma 6.5 we can build a sequence σ_l such that
- $(\sigma_l \gg \pi) < i$, and
- $(\sigma_{l,\pi'} \gg \pi') \leq (\sigma_l \gg \pi')$ for all $\pi' \in \Pi$.

Using a symmetric argument, we build a sequence σ_r such that
- $i < (\pi \ll \sigma_r)$, and
- $(\pi' \ll \sigma_{r,\pi'}) \geq (\pi' \ll \sigma_r)$ for all $\pi' \in \Pi$.

As a consequence, for all accepting SCC-path $\pi' \in \Pi$, all portals surviving (σ_l, σ_r) in π' are $\simeq$-equivalent to P. Furthermore, P survives (σ_l, σ_r) in π.

We have shown that P and (σ_l, σ_r) satisfy properties P1 and P3. P2 is immediate by assumption. We simply apply Lemma 4.31 to obtain the result. ◀

Next, we establish that the items listed in the previous lemma can all be checked in polynomial space in $|\mathcal{A}|$.

▶ **Lemma 6.8.** *Given a portal P, we can check whether it has infinitely many minimal blocking factors in space polynomial in $|\mathcal{A}|$.*

Proof. Recall that, by Lemma 4.7, $L = \mathcal{L}(P)$ is recognized by a strongly connected automaton $\mathcal{A}'$ with at most $p|\mathcal{A}|$ states. While this number may be exponential in $|\mathcal{A}|$, the transition function of $\mathcal{A}'$ can be computed in polynomial space from the polynomial-sized representation of a state. Furthermore, in this case, we can show that the same property holds for the construction used in Lemma 3.15, as in the determinization step, all states share the index modulo p.

We then simply need to check if the resulting automaton has an infinite language, which is the case if and only if it has a cycle reachable from the initial state and from which a final state is reachable. This can be checked by exploring the state space of the automaton, in non-deterministic polynomial space (in $|\mathcal{A}|$), and applying Savitch's theorem [29, Theorem 1]. ◀

▶ **Lemma 6.9.** *Given two SCC-paths π and π', one can check in PSPACE whether there is a sequence σ that is blocking for π and not π'.*

Proof. The algorithm relies on the following property.

▷ **Claim 6.10.** There is a sequence σ that is blocking for $\pi = P_0 \xrightarrow{a_1} \cdots P_k$ and not $\pi' = P'_0 \xrightarrow{a'_1} \cdots P'_\ell$ if and only if either:

- there is a positional word μ that is a blocking factor for P_0 and not P'_0 and there is a sequence σ' that is blocking for $P_1 \xrightarrow{a_2} \cdots P_k$ and not π',
- or there is a positional word μ that is a blocking factor for P_0 and P'_0 and there is a sequence σ' that is blocking for $P_1 \xrightarrow{a_2} \cdots P_k$ and not $P'_1 \xrightarrow{a'_2} \cdots P'_\ell$.

Proof. The right-to-left direction is clear (just take $\sigma = \mu\sigma'$ in both cases).

For the left-to-right direction, consider a sequence σ that is blocking for π and not π', of minimal length. Let σ_+ and μ be such that $\sigma = \mu\sigma_+$.

- If μ is not blocking for P_0 then σ_+ is blocking for π and not π', contradicting the minimality of σ.
- If μ is blocking for P_0 and not P'_0 then we set $\sigma' = \sigma$. We know that σ is not blocking for π'. On the other hand, as σ is blocking for π, it is also blocking for $P_1 \xrightarrow{a_2} \cdots P_k$.
- If μ is blocking for both P_0 and P'_0 then we set $\sigma' = \sigma$. As σ is blocking for π, it is also blocking for $P_1 \xrightarrow{a_2} \cdots P_k$. On the other hand, if σ was blocking for $P'_1 \xrightarrow{a'_2} \cdots P'_\ell$, then it would also be blocking for π', a contradiction. Hence σ is not blocking for $P'_1 \xrightarrow{a'_2} \cdots P'_\ell$ ◁

The claim above lets us define a recursive algorithm.

- First check if there is a positional word μ that is blocking for P_0 and not P'_0. If it is the case, make a recursive call to check if there is a sequence σ' that is blocking for $P_1 \xrightarrow{a_2} \cdots P_k$ and not π'. If it is the case, answer yes.

- Then check if there is a positional word μ that is a blocking factor for P_0 and P'_0. If so, make a recursive call to check if there is a sequence σ' that is blocking for $P_1 \xrightarrow{a_2} \cdots P_k$ and not $P'_1 \xrightarrow{a'_2} \cdots P'_\ell$. If it is the case, answer yes.

  If both items fail, answer no.

The existence of those positional words can be checked in polynomial space using the automaton $\mathcal{B}$ constructed in the proof of Lemma 6.8. The depth of the recursive calls is at most the sum of the lengths of π and π', which is bounded by $2|\mathcal{A}|$. In consequence, this algorithm runs in polynomial space.

◀

▶ **Proposition 6.11.** *The hardness problem is in* PSPACE.

Proof. Our algorithm is based on Lemma 6.7. We use the following algorithm to check whether the characterization holds.
1. First, we nondeterministically guess an SCC-path $\pi = P_0 \xrightarrow{a_1} \cdots P_k$ and an index i.
2. Using Lemma 6.8, we check that P_i has infinitely many minimal blocking factors.
3. For each accepting SCC-path $\pi' = P'_0 \xrightarrow{a'_1} \cdots P'_\ell$ of $\mathcal{A}$, we guess indices j_l and j_r, and check that every portal P'_j with $j_l < j < j_r$ is $\simeq$-equivalent to P_i.
4. Then, we use Lemma 6.9 to check that there is a sequence σ_l that is blocking for $P'_0 \xrightarrow{a'_1} \cdots P'_{j_l}$ and not $P_0 \xrightarrow{a_1} \cdots P_i$. Symmetrically, we check that there is a sequence σ_r that is blocking for $P'_{j_r} \xrightarrow{a'_1} \cdots P'_\ell$ and not $P_i \xrightarrow{a_{i+1}} \cdots P_k$.

If all those tests succeed, we answer "yes", otherwise we answer "no". This algorithm is correct and complete by Lemma 6.7. ◀

6.1.2 Testing triviality

We show the PSPACE upper bound on the complexity of checking if a language is trivial. It is based on the characterisation of trivial languages given by Lemma 5.8, and uses the following result.

▶ **Lemma 6.12.** *Given a portal P, we can check whether it has a blocking factor in space polynomial in $|\mathcal{A}|$.*

Proof. We proceed as in the proof of Lemma 6.8, except that we only need to check whether some final state is reachable from the final state. ◀

▶ **Proposition 6.13.** *The triviality problem is in* PSPACE.

Proof. Recall that $\mathcal{L}(\mathcal{A})$ is trivial if and only if $\mathcal{A}$ has no blocking sequences.

▷ **Claim 6.14.** There is an accepting SCC-path π of $\mathcal{A}$ that contains a portal P with no blocking factors if and only if $\mathcal{A}$ has no blocking sequence.

Proof. Any blocking sequence of $\mathcal{A}$ is blocking for π, therefore it contains a blocking factor for P. ◁

Therefore, it suffices to enumerate all accepting SCC-paths π in the automaton, and then check that all portals in π have at least one blocking factor, using Lemma 6.12. ◀

6.2   Hardness of classifying automata

We prove hardness of the triviality problem and easiness problems, concluding on their PSPACE-completeness. We reduce from the universality problem for NFAs, which is well-known to be PSPACE-complete (see e.g. [1, Theorem 10.14]).

▶ **Lemma 6.15.** *The triviality and hardness problems are PSPACE-hard.*

Proof. Consider an NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ on an alphabet Σ. Without loss of generality, we assume that $\mathcal{A}$ is trim (up to removing unreachable or non-co-reachable states) and that it accepts all words of length less than 2: this can be checked in polynomial time and does not affect the PSPACE-hardness of universality. Let $\#$ and $!$ be two letters that are not in Σ. We apply the following transformations to $\mathcal{A}$:

- add a transition labeled by $!$ from every final state to the initial state q_0
- add a self-loop labeled by $\#$ to each state.

We call the resulting automaton $\mathcal{B} = (Q, \Sigma \cup \{!, \#\}, \delta', q_0, F)$. Note that $\mathcal{B}$ is strongly connected: consider any two states $q, q' \in Q$, we show that q' is reachable from q. As $\mathcal{A}$ is trim, there exists $q_f \in F$ that is reachable from q, and q' is reachable from the initial state q_0. Furthermore, we have put a $!$ transition from q_f to q_0, hence q' is reachable from q.

Recall that the language of a strongly connected automaton is trivial if and only if it has no minimal blocking factor, and hard if and only if it has infinitely many minimal blocking factors.

Hence, to complete the proof, we show that $\mathsf{MBF}(\mathcal{B})$ is empty when $\mathcal{A}$ is universal and infinite otherwise.

First, let us describe the language recognized by $\mathcal{B}$. It is given by

$$\mathcal{L}(\mathcal{B}) = \{u_1!u_2!\cdots!u_n \mid \forall i, u_i \in (\Sigma \cup \{\#\})^* \wedge \pi_\Sigma(u_i) \in \mathcal{L}(\mathcal{A})\},$$

where $\pi_\Sigma(u)$ is the word in Σ^* obtained by removing all letters not in Σ from u.

▷ Claim 6.16.   If $\mathcal{A}$ is universal, then $\mathcal{B}$ is also universal.

Proof. Indeed, any word in u in can be uniquely decomposed into $u = u_1!u_2!\cdots!u_n$ where each u_i does not contain the letter "$!$". As $\#$ is idempotent on $\mathcal{B}$, $\delta'(q_0, u_i)$ is equal to $\delta(q_0, \pi_\Sigma(u_i))$ for every i. Since $\mathcal{A}$ is universal, each of the $\delta'(q_0, u_i)$ contains a final state, hence $\delta'(q_0, u_i!) = \{q_0\}$. Therefore, the set $\delta'(q_0, u)$ is equal to $\delta'(q_0, u_n)$, which contains a final state, and u is in $\mathcal{L}(\mathcal{B})$, which shows that $\mathcal{B}$ is universal.   ◁

This shows that if $\mathcal{A}$ is universal, then $\mathsf{MBF}(\mathcal{B})$ is empty.

Now we show that a word $w \in \Sigma^*$ not in $\mathcal{L}(\mathcal{A})$ induces infinitely many minimal blocking factors for $\mathcal{B}$. Consider such a w of minimal size. As we assumed that $\mathcal{A}$ accepts all words of size less than 2, $|w| \geq 2$. Let u, v be words of length at least 1 such that $w = uv$. For all $n \in \mathbb{N}$, at least one of $u\#^n v, !u\#^n v, u\#^n v!, !u\#^n v!$ is a minimal blocking factor (depending respectively on whether w is not a factor of any word of $\mathcal{L}(\mathcal{A})$ or is a prefix/suffix of a word of $\mathcal{L}(\mathcal{A})$ or not). As a consequence, $\mathcal{B}$ has infinitely many blocking factors, and is thus hard to test by Theorem 3.2.

In summary, $\mathcal{A}$ is universal if and only if $\mathcal{B}$ is trivial to test, and $\mathcal{A}$ is *not* universal if and only if $\mathcal{B}$ is hard to test. This shows the PSPACE-hardness of the triviality problem.   ◀

The above proof can be extended to show the PSPACE-hardness of the easiness problem.

▶ **Corollary 6.17.** *The easiness problem is PSPACE-hard.*

Proof. We proceed as in the proof of Lemma 6.15: given an automaton $\mathcal{A}$ over an alphabet Σ, we build an automaton $\mathcal{B}$ over the alphabet $\Sigma \cup \{!, \#\}$ such that if $\mathcal{A}$ is universal, $\mathsf{MBF}(\mathcal{B})$ is empty, and if $\mathcal{A}$ is not universal, then $\mathsf{MBF}(\mathcal{B})$ is infinite.

To show the hardness of the easiness problem, let $\flat$ denote a new letter not in $\Sigma \cup \{\#, !\}$ and consider the automaton $\mathcal{B}'$ equal to $\mathcal{B}$ but taken over the alphabet $\Sigma \cup \{\#, !, \flat\}$. As there are no transitions labeled by $\flat$ in $\mathcal{B}'$, the word $\flat$ is always a minimum blocking factor of $\mathcal{B}'$. As a result, we have $\mathsf{MBF}(\mathcal{B}') = \mathsf{MBF}(\mathcal{B}) \cup \{\flat\}$, hence $\mathcal{A}$ is universal if and only if $\mathsf{MBF}(\mathcal{B}')$ is finite but non-empty: by Theorem 3.2, this is equivalent to $\mathcal{L}(\mathcal{B}')$ is easy to test. Therefore, the easiness problem is also PSPACE-hard. ◀

This concludes the proof of Theorem 6.4

7 Conclusion

We presented an effective classification of regular languages in three classes, each associated with an optimal query complexity for property testing. We thus close a line of research aiming to determine the optimal complexity of regular languages. All our results are with respect to the Hamming distance. We conjecture that they can be adapted to the edit distance. We use non-deterministic automata to represent regular languages. A natural open question is the complexity of classifying languages represented by *deterministic* automata.

References

1   Alfred V. Aho and John E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., 1974.

2   Maryam Aliakbarpour, Ilias Diakonikolas, and Ronitt Rubinfeld. Differentially private identity and equivalence testing of discrete distributions. In *International Conference on Machine Learning, ICML*, 2018. URL: https://proceedings.mlr.press/v80/aliakbarpour18a.html.

3   Noga Alon, Richard A. Duke, Hanno Lefmann, Vojtech Rödl, and Raphael Yuster. The algorithmic aspects of the regularity lemma. *Journal of Algorithms*, 16(1):80–109, 1994. doi:10.1006/JAGM.1994.1005.

4   Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000. doi:10.1007/s004930070001.

5   Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6):1842–1862, 2001. doi:10.1109/SFFCS.1999.814641.

6   Noga Alon and Asaf Shapira. Every monotone graph property is testable. *SIAM Journal of Computing*, 38(2):505–522, 2008. doi:10.1137/050633445.

7   Antoine Amarilli, Louis Jachiet, and Charles Paperman. Dynamic membership for regular languages. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 116:1–116:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.ICALP.2021.116, doi:10.4230/LIPICS.ICALP.2021.116.

8   Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013.

9   Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *International Colloquium on Automata, Languages, and Programming, ICALP*, 2021. `doi:10.4230/LIPIcs.ICALP.2021.119`.

10   Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993. `doi:https://doi.org/10.1016/0022-0000(93)90044-W`.

11   Ilias Diakonikolas and Daniel M Kane. A new approach for testing properties of discrete distributions. In *IEEE Symposium on Foundations of Computer Science, FOCS*, pages 685–694. IEEE, 2016. `doi:10.1109/FOCS.2016.78`.

12   Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. In *International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM*, pages 97–108. Springer, 1999. `doi:10.1007/978-3-540-48413-4_10`.

13   Funda Ergün, Sampath Kannan, S.Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000. `doi:https://doi.org/10.1006/jcss.1999.1692`.

14   Eldar Fischer, Frédéric Magniez, and Tatiana Starikovskaya. Improved bounds for testing Dyck languages. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1529–1544. SIAM, 2018.

15   Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proceedings of the 24th Annual European Symposium on Algorithms*, volume 57 of *LIPIcs*, pages 43:1–43:17, 2016. `doi:10.4230/LIPIcs.ESA.2016.43`.

16   Moses Ganardi, Danny Hucke, and Markus Lohrey. Randomized sliding window algorithms for regular languages. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs*, pages 127:1–127:13, 2018. `doi:10.4230/LIPIcs.ICALP.2018.127`.

17   Moses Ganardi, Danny Hucke, Markus Lohrey, and Tatiana Starikovskaya. Sliding Window Property Testing for Regular Languages. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation (ISAAC 2019)*, volume 149 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ISAAC.2019.6`, `doi:10.4230/LIPIcs.ISAAC.2019.6`.

18   Oded Goldreich. *Introduction to property testing*. Cambridge University Press, 2017. `doi:10.1017/9781108135252`.

19   Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, jul 1998. `doi:10.1145/285055.285060`.

20   Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *The collected works of Wassily Hoeffding*, pages 409–426, 1994.

21   Rahul Jain and Ashwin Nayak. The space complexity of recognizing well-parenthesized expressions in the streaming model: The index function revisited. *IEEE Transactions on Information Theory*, 60(10):6646–6668, Oct 2014. `doi:10.1109/TIT.2014.2339859`.

22   Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming algorithms for recognizing nearly well-parenthesized expressions. In *Proc. of MFCS 2011*, volume 6907 of *LNCS*, pages 412–423. Springer, 2011. `doi:10.1007/978-3-642-22993-0\_38`.

23   Frédéric Magniez and Michel de Rougemont. Property testing of regular tree languages. *Algorithmica*, 49(2):127–146, 2007. `doi:10.1007/s00453-007-9028-3`.

24   Frédéric Magniez, Claire Mathieu, and Ashwin Nayak.   Recognizing well-parenthesized
           expressions in the streaming model.   *SIAM J. Comput.*, 43(6):1880–1905, 2014.
           `doi:10.1137/130926122`.

25   Liam Paninski.   A coincidence-based test for uniformity given very sparsely sam-
           pled discrete data.   *IEEE Transactions on Information Theory*, 54(10):4750–4755, 2008.
           `doi:10.1109/TIT.2008.928987`.

26   Michal Parnas, Dana Ron, and Ronitt Rubinfeld.   Testing membership in parenthesis lan-
           guages. *Random Struct. Algorithms*, 22(1):98–138, 2003. `doi:10.1002/rsa.10067`.

27   Jean-Éric Pin, editor.   *Handbook of Automata Theory*.   European Mathematical Society
           Publishing House, Zürich, Switzerland, 2021.   URL: `https://doi.org/10.4171/Automata`,
           `doi:10.4171/AUTOMATA`.

28   Ronitt Rubinfeld and Madhu Sudan.   Robust characterizations of polynomials with
           applications to program testing.   *SIAM Journal on Computing*, 25(2):252–271, 1996.
           `doi:10.1137/S0097539793255151`.

29   Walter J. Savitch.   Relationships between nondeterministic and deterministic tape
           complexities.   *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
           `doi:https://doi.org/10.1016/S0022-0000(70)80006-X`.

30   Mosaad Al Thokair, Minjian Zhang, Umang Mathur, and Mahesh Viswanathan. Dynamic
           race detection with O(1) samples. *Proc. ACM Program. Lang.*, 7(POPL):1308–1337, 2023.
           `doi:10.1145/3571238`.

31   Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complex-
           ity. In *Symposium on Foundations of Computer Science, SFCS*, pages 222–227. IEEE, 1977.
           `doi:10.1109/SFCS.1977.24`.