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 sub-linear 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.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases property testing, regular languages

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. 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.

[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.

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 4.12 extends the above result to the case of the Hamming distance.

Bathie and Starikovskaya 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, is is easy to find specific non-trivial regular languages for which there is an algorithm using only $\mathcal{O}(1/\varepsilon)$ queries. Some examples of languages that can require $\mathcal{O}(1/\varepsilon)$ queries are a^* over the alphabet $\{a, b\}$, $(ab)^*$ or $(aa + bb)^*$.

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 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.

[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.

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 [29].

1.2 Main result and overview of the paper

▶ **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 $\geq n$.*

Our characterisation 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 an order on them, which gives us a notion of *minimal* blocking sequence.

▶ **Theorem 1.4.** *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 show that the language of every strongly connected automaton is testable with $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries.
- Then, 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.

Then, Section 4 extends those results to all automata. The interplay with the previous section is different for the upper and the lower bound

- We show the upper bound of $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries to all automata by a natural extension of the proof in the strongly connected case.

- We obtain a lower bound of $\Omega(\log(\varepsilon^{-1})/\varepsilon)$ queries for languages with infinitely many minimal blocking sequences by a reduction to the strongly connected case. We show that we can use a tester for the general automaton to test one of its hard strongly connected components.

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.
- Finally, we prove that if an automaton has at least one blocking sequence, then it requires $\Omega(1/\varepsilon)$ queries to be tested. By contrast, automata with no blocking sequence recognise 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 that all three decision problems are PSPACE-complete in Section 6.

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 $u[i]u[i+1]\dots u[j]$ 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 $w \neq u$, 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 article, 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. 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 a distance d.

▶ **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. Here, $d(u, L) \ge \varepsilon n$ means that one cannot change an ε-fraction of the letters in u to obtain a word in L.

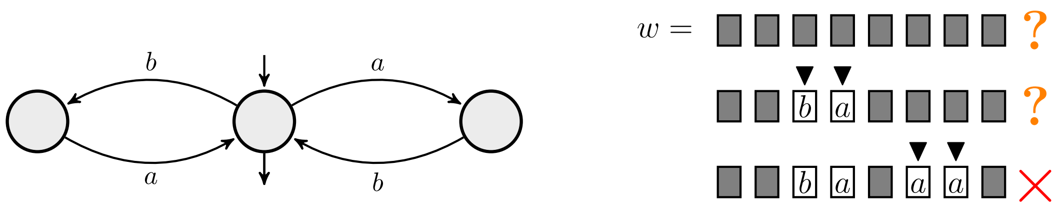

Figure 1 An automaton $\mathcal{A}$ recognizing the language $L = (ab)^* + (ba)^*$. A witness that a word is not in this language is a factor aa or bb. We query factors of size 2, and reject if we find one of those factors. We can show that a word ε-far from L must have many such factors, and infer that we can get a constant probability of rejecting it by sampling only $\mathcal{O}(1/\varepsilon)$ factors.

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:*

$$\text{if } u \in L, \text{ then } T(u) = 1, \tag{1}$$

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

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.

▶ Remark 2.3. If L is finite, then it is trivial: since there is a bound B on the lengths of its words, it suffices to read entirely words of length $\leq B$ and answer No for words of greater length. For that reason, in the rest of the paper we only consider infinite languages.

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 2.4** (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. Take a look at the language $L = (ab)^*$. The word $v = ab$ can appear as a factor of a word $u \in L$ is v occurs at an even position in u. However, if v occurs at an *odd* position in u, then u cannot be in L. 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 motivates the introduction of *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 p, and working with p-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 2.5** (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* $(n \pmod{p}, a_0)((n+1) \pmod{p}, a_1) \cdots ((n+\ell) \pmod{p}, a_\ell)$*. If* $u = a_0 \cdots a_\ell$*, we call this word* $(n : u)$ *(when* p *is clear from context).*

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, recognising $\{(0 : u) \mid u \in \mathcal{L}(\mathcal{A})\}$. 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 2.6.** *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. Note that a 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. We define 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 $\mathcal{TL}(\mathcal{A})$. We show that all strongly connected NFA have a query complexity in $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$, and that the ones which have infinitely many minimal blocking factors have a query complexity of $\Theta(\log(\varepsilon^{-1})/\varepsilon)$.

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, $(i : u)$ is blocking for $\mathcal{A}$ if u does not label any path in $\mathcal{A}$ from a state of Q_i. (This property is formally established later in Lemma A.1.) 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 3.1 and the lower bound in Section 3.2.

3.1 An efficient property tester for strongly connected NFAs.

In this section, we show the following result.

▶ **Theorem 3.3.** *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.*

As mentioned in the overview, this result supersedes the one that was obtained in [9]: while both testers use the same number of queries, the tester in [9] works under the edit distance, while the one of Theorem 3.3 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 our result supersedes and generalizes theirs. Our proof is similar to that of [9], with one notable improvement: we use a new method for sampling factors in u, which greatly simplifies the correctness analysis.

The algorithm for Theorem 3.3 is Algorithm 1. The procedure is fairly simple: the algorithm samples at random factors of various lengths in u, and rejects if and only if at least one of these factors is blocking. On the other hand, the correctness of the tester is far from trivial. The lengths and the number of factors of each lengths are chosen so that the number of queries is $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ and the probability of finding a blocking factor is maximized, regardless of their repartition in u.

We now turn to formally establishing these properties.

▷ Claim 3.4. The tester given in Algorithm 1 makes $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries to u.

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 $\mathcal{O}(1/\varepsilon)$ (see Appendix A.2 for a proof).

▶ **Lemma 3.5.** *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 $12m^2/\varepsilon$.*

For the correctness analysis, we assume that u is ε-far from $\mathcal{L}(\mathcal{A})$, and show that Algorithm 1 finds at least one of the blocking factors given by Lemma 3.5 with probability at least 2/3.

▶ **Lemma 3.6.** *In the last `Else` block, if u is ε-far from $\mathcal{L}(\mathcal{A})$, then Algorithm 1 rejects with probability at least 2/3.*

Algorithm 1 Generic ε-property tester using $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries

function SAMPLE(u, ℓ)
  $i \leftarrow$ UNIFORM$(0, n-1)$
  $l \leftarrow \max(i-\ell, 0), r \leftarrow \min(i+\ell, n-1)$
  return $(l : u[l..r])$
function TESTER(u, ε)
  $\beta \leftarrow 12m^2/\varepsilon$
  if $\mathcal{L}(\mathcal{A}) \cap \Sigma^n = \emptyset$ **then**
    Reject
  else if $n < \beta$ **then**
    Query all of u and run $\mathcal{A}$ on it
    Accept if and only if $\mathcal{A}$ accepts
  else
    $T \leftarrow \lceil \log(\beta) \rceil$
    for $t = 0$ to T **do**
      $\ell_t \leftarrow 2^t, r_t \leftarrow \lceil 2\ln(3)\beta/\ell_t \rceil$
      Call SAMPLE(u, ℓ_t) r_t times
    Reject if and only if any call to SAMPLE returned a blocking factor for $\mathcal{A}$.

Proof sketch. Assume that u is ε-far from $\mathcal{L}(\mathcal{A})$, that $\mathcal{L}(\mathcal{A}) \cap \Sigma^n$ is not empty (i.e. $\mathcal{L}(\mathcal{A})$ contains a word of length n) and that $n \geq \beta$. For $t = 0, \ldots, T$, let B_t denote the subset of $\mathcal{B}$ of blocking factors of length at most $\ell_t = 2^t$.

For each t, if in a call to SAMPLE(u, ℓ_t) the random position i is inside a blocking factor of B_t, then the returned factor is blocking. Therefore, as the factors given by Lemma 3.5 are disjoint, each one of the $r_t = 2\ln(3)\beta/\ell_t$ calls to SAMPLE(u, ℓ_t) has a probability $p_t \geq \frac{1}{n}\sum_{\tau \in B_t} |\tau|$ to return a blocking factor.

Careful calculations let us show that the probability that no blocking factor is found is at most $1/3$. They are detailed in Appendix A.2. ◀

3.2 Lower bound when there are infinitely many minimal blocking words

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

We start from 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 a 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 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, which are hard to identify for property testers. In [9], the authors proved a lower bound of $\Omega(\log(\varepsilon^{-1})/\varepsilon)$ on the query complexity of any property tester for L, matching the upper bound in the same paper. The upper bound of [9] is for the edit distance, we show that the result can be extended to the Hamming distance.

The minimal blocking factors of L include all words for the form cvd where $v \notin P$: there are infinitely many such words. This is no coincidence: we show in this article than the lower bound from [9] can be lifted to any languages with infinitely many minimal blocking sequences (or minimal blocking factors in the case of strongly connected automata).

▶ **Theorem 3.7.** *Let $\mathcal{A}$ be a strongly connected NFA. If MBF($\mathcal{A}$) is infinite, then there exists $\varepsilon_0 > 0$ such that for all $\varepsilon < \varepsilon_0$, every ε-property tester for $L = \mathcal{L}(\mathcal{A})$ uses $\Omega(\log(\varepsilon^{-1})/\varepsilon)$ queries.*

The proof of this result is a full generalization of the lower bound against the "repeated parity" language that was published in [9, Theorem 15].

Our proof is based on (a consequence of) Yao's Minmax Principle [30]: 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* with q queries errs with probability at least p on some input u.

To prove Theorem 3.7, we first exhibit such a distribution $\mathcal{D}$ for $q = \Theta(\log(\varepsilon^{-1})/\varepsilon)$ (see Appendix A.3.2 for its construction). 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 3.8),
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. we combine the above two results to prove Theorem 3.7 via Yao's Minmax principle.

Constructing a Hard Distribution $\mathcal{D}$

To sample an input w.r.t. $\mathcal{D}$, we first sample a uniformly random bit π: if $\pi = 1$, we construct a word u that belongs to L, and if $\pi = 0$, u will be far from L with high probability. The input word is then conceptually divided into εn disjoint intervals. For the j-th interval, we sample a random variable $\kappa_j \in \{0, 1, \ldots, \log(1/\varepsilon)\}$: this random variable describes the content of the interval. If $\pi = 1$, we fill the interval with *non-blocking* factors of length $\mathcal{O}(2^{\kappa_j})$, and if $\pi = 0$, we instead fill the interval with *blocking* factors of the same length, chosen to be very similar to the non-blocking factors, making it hard to distinguish between the two cases with few queries. By carefully choosing the distribution for κ_j, we can ensure that when $\pi = 0$, the resulting instance u is ε-far from L with high probability. (Proofs for this section can be found in Appendix A.3.)

▶ **Lemma 3.8.** *Conditioned on $\pi = 0$, the probability of the event $\mathcal{F} = \{u$ is ε-far from $\mathcal{TL}(\mathcal{A})\}$ goes to 1 as n goes to infinity.*

▶ **Corollary 3.9.** *For large enough n, we have $\mathbb{P}(\mathcal{F}) \geq 5/12$.*

Intuitively, our distribution is hard to test because positive and negative instance 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 3.10.** *Let T be a deterministic tester with perfect completeness (i.e. one sided error, 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 $\mathcal{M} = \{\forall j$ s.t. $\kappa_j > 0, q_j < 2^{\kappa_j}\}$, the probability that T accepts u is 1.*

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

▶ **Lemma 3.11.** *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 $\frac{1}{72} \cdot \log(\varepsilon^{-1})/\varepsilon$ queries, i.e. $\sum_j q_j \leq \frac{1}{72} \cdot \log(\varepsilon^{-1})/\varepsilon$. The probability of the event $\mathcal{M} = \{\forall j$ s.t. $\kappa_j > 0, q_j < 2^{\kappa_j}\}$ is at least $11/12$.*

We are now ready to prove Theorem 3.7.

Proof of Theorem 3.7. We want to show that any tester with perfect completeness for $\mathcal{L}(\mathcal{A})$ requires at least $\frac{1}{72} \cdot \log(\varepsilon^{-1})/\varepsilon$ 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 $\frac{1}{72} \cdot \log(\varepsilon^{-1})/\varepsilon$ queries errs on u when $u \sim \mathcal{D}$ with probability at least 1/3, and conclude using Yao's Minmax 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 3.9, we have $\mathbb{P}(\mathcal{F}) \geq 5/12$, and by Lemma 3.11, $\mathbb{P}(\mathcal{M})$ 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 3.7, 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 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

We start with an example that highlights why we need to use blocking sequences instead of blocking factors.

▶ **Example 4.1.** Let us study the automaton $\mathcal{A}$ depicted in Figure 2. The set of minimal blocking factors of $\mathcal{L}(\mathcal{A})$ is infinite: it is the language ba^*c. Yet, L 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, then it can be decomposed $u = u_1 u_2$ where u_1 contains $\Omega(\varepsilon n)$ letters b and u_2 contains $\Omega(\varepsilon n)$ letters c.*

The pair of factors (c, a) 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 must contains many disjoint blocking sequences – this property (Lemma 4.11) 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, in the same order as in σ, then u is not in L[4]. While L has infinitely many minimal blocking factors, it has a single minimal blocking sequence $\sigma = (b, c)$.

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

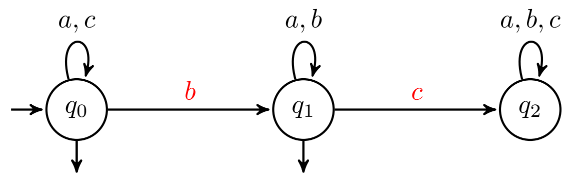

Figure 2 An automaton $\mathcal{A}$ recognizing the language $L = (a+c)^*(a+b)^*$.

4.1.1 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 traverses a strongly connected component, and *SCC-paths*, that describe a succession of portals. Appendix B presents several examples motivating and illustrating the definitions below.

We fix an NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$. Let $\mathscr{S}$ be the set of SCCs of $\mathcal{A}$. 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. From now on, we use "positional words" for p-positional words.

▶ **Definition 4.3** (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

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

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

▶ **Definition 4.4.** *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.*

Portals describe the behavior of a run in a single strongly connected component.

▶ **Definition 4.5** (SCC-path). *An* SCC-path *π of $\mathcal{A}$ is a sequence of portals linked by transitions $\pi = s_0, x_0 \rightsquigarrow t_0, y_0 \xrightarrow{a_1} s_1, x_1 \rightsquigarrow t_1, y_1 \cdots \xrightarrow{a_k} s_k, x_k \rightsquigarrow t_k, y_k$, such that for all $i \in \{1, \ldots, k\}$, $x_i = y_{i-1} + 1 \pmod{p}$ and $s_i \in \delta(t_{i-1}, a_i)$.*

Intuitively, an SCC-path is a description of the states and positions at which a path through the automaton enters and leaves each SCC. See Appendix B for some examples.

▶ **Definition 4.6.** *The language $\mathcal{L}(\pi)$ of an SCC-path $\pi = s_0, x_0 \rightsquigarrow t_0, y_0 \xrightarrow{a_1} \cdots s_k, x_k \rightsquigarrow t_k, y_k$ is the set $\mathcal{L}(\pi) = \mathcal{L}(s_0, x_0 \rightsquigarrow t_0, y_0) a_1 \mathcal{L}(s_1, x_1 \rightsquigarrow t_1, y_1) a_2 \cdots \mathcal{L}(s_k, x_k \rightsquigarrow t_k, y_k)$*

We say that π is accepting *if $x_0 = 0$, $s_0 = q_0$, $t_k \in F$ and $\mathcal{L}(\pi)$ is non-empty.*

▶ Remark 4.7. We have $\mathcal{L}(\mathcal{A}) = \bigcup_{\pi \text{ accepting}} \mathcal{L}(\pi)$.

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 sequences for automata.

▶ **Definition 4.8.** *We say that a sequence $(\mu_1, \ldots, \mu_\ell)$ 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 $i_0 \leq i_1 \leq \cdots \leq i_k$ such that for every j, μ_{i_j} is blocking for P_j.*

Crucially, in this definition, 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 B.1 in the appendix, where the language is easy because consecutive SCCs share blocking factors.

▶ **Definition 4.9** (Blocking sequence for $\mathcal{A}$). *A sequence of positional words $\sigma = (\mu_1, \ldots, \mu_\ell)$ is blocking for $\mathcal{A}$ if it is blocking for all SCC-paths of $\mathcal{A}$.*

See Example B.2) for an example illustrating these definitions.

The goal of the next two lemmas is to show that we can reduce property testing of $\mathcal{L}(\mathcal{A})$ to a search for blocking sequences in the word. They are proven in Appendix C.

- If we find a few blocking sequences in a word then it is not in the language and we can answer no (Lemma 4.10).
- A word that is far from the language contains many blocking sequences (Lemma 4.11). Hence if we do not find blocking sequences in the word, it is unlikely to be far from the language (Lemma 4.11). In fact, the lemma is more precise: we can find blocking sequences organised in a particular way.

▶ **Lemma 4.10.** *If μ contains $|\mathcal{A}|$ disjoint blocking sequences for $\mathcal{A}$ then $\mu \notin \mathcal{L}(\mathcal{A})$.*

▶ **Lemma 4.11.** *Let μ be a word of length n. There exist constants E, K such that if $+\infty > d(\mu, \mathcal{L}(\mathcal{A})) \geq \varepsilon n$ and $n \geq E/\varepsilon$, 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 $\frac{\varepsilon n}{E}$ disjoint occurrences of words $\nu_{i,1}, \nu_{i,2} \ldots$, each of length $\mathcal{O}(1/\varepsilon)$, such that for any choice of $j_0, j_1, \ldots, j_K$, the sequence $(\nu_{0,j_0}, \nu_{1,j_1}, \ldots, \nu_{K,j_K})$ is a blocking sequence for $\mathcal{A}$.*

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 4.12.** *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.*

The property tester behind this theorem uses the property tester for strongly connected NFAs as a subroutine, and its correctness is based on Lemma 4.11, an extension of Lemma 3.5 to blocking sequences.

Recall that by Lemma 4.10, if we can find $N = |\mathcal{A}|$ disjoint blocking sequences in μ, then we know $\mu \notin L$, and the tester can reject. By Lemma 4.11, if we find N disjoint $\nu_{i,j}$'s in each of the μ_i, then we have found N disjoint blocking sequences. Since all of the $\nu_{i,j}$ have length at most $\mathcal{O}(1/\varepsilon)$ and each μ_i contains $\Omega(\varepsilon n)$ of them, we can adapt the algorithm of Theorem 3.3 by tweaking the constants so that it has a constant probability of finding at least one $\nu_{i,j}$ in a fixed μ_i, using $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries. By repeating $\mathcal{O}(N)$ times for each of the $K + 1$ possible values of i, the algorithm finds the N $\nu_{i,j}$ for each i with probability at least $2/3$, and uses a total of $\mathcal{O}(\log(\varepsilon^{-1})/\varepsilon)$ queries, as N and K are constants.

4.3 Lower bound

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

▶ **Definition 4.13** (Minimal blocking sequence). *Let $\sigma = (\mu_1, \ldots, \mu_k)$ and $\sigma' = (\mu'_1, \ldots, \mu'_t)$ be blocking sequences of $\mathcal{A}$. 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 $\mathcal{A}$ is minimal *if it is a minimal element of $\trianglelefteq$ among blocking sequences of $\mathcal{A}$. The set of minimal blocking sequences of $\mathcal{A}$ is written $\mathsf{MBS}(\mathcal{A})$.*

▶ Remark 4.14. If $\sigma \trianglelefteq \sigma'$ and σ is a blocking sequence for an SCC-path π then σ' is also a blocking sequence for π.

To prove a lower bound on the number of queries necessary to test a language when $\mathsf{MBS}(\mathcal{A})$ 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 timed 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. Intuitively, the left effect of a sequence σ on an SCC-path π is 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 4.15. 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 $(\sigma_l \gg \pi) > (\pi \ll \sigma)$.

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 $\pi = P_0 \xrightarrow{a_1} \cdots P_k$ and two sequences of positional words σ_l, σ_r, we say that the portal P_i *survives* (σ_l, σ_r) if $(\sigma_l \gg \pi) < i < (\pi \ll \sigma_r)$.

▶ **Definition 4.16.** *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 all accepting SCC-path π in $\mathcal{A}$, every portal in π which survives (σ_l, σ_r) is $\simeq$-equivalent to P.*

We use the existence of P, σ_l and σ_r with those properties as an intermediate step to show that if $\mathsf{MBS}(\mathcal{A})$ is infinite then $\mathcal{L}(\mathcal{A})$ is hard.

▶ **Lemma 4.17.** *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.*

▶ **Lemma 4.18.** *If there exist $P = s, x \rightsquigarrow t, y$ and σ_l, σ_r satisfying properties P1, P2 and P3 then $\mathcal{L}(\mathcal{A})$ is hard.*

▶ **Proposition 4.19.** *If $\mathcal{A}$ has infinitely many minimal blocking sequences, then $\mathcal{L}(\mathcal{A})$ is hard.*

Proof. We combine Lemmas 4.17 and 4.18. ◀

5 Trivial and Easy languages

5.1 Upper bound for easy languages

We establish that if $\mathsf{MBS}(\mathcal{A})$ is finite then $\mathcal{A}$ is testable with $\mathcal{O}(1/\varepsilon)$ queries.

▶ **Lemma 5.1.** *If $\mathcal{A}$ has finitely many minimal blocking sequences, then there is a tester for $\mathcal{L}(\mathcal{A})$ using $\mathcal{O}(1/\varepsilon)$ queries.*

Proof sketch. The full proof is in Appendix C. As the number of minimal blocking sequences of $\mathcal{A}$ is bounded, so is the maximum length of a minimal blocking sequence. Let K be the upper bound on the length and P the bound on the number of minimal blocking sequences.

We infer from the fact that there are finitely many blocking sequences that if a word μ is ε-far from the language of $\mathcal{A}$ then it must contain $O(\varepsilon|\mu|)$ disjoint occurrence of the same minimal sequence σ.

Since each positional word in this sequence has length at most K, by sampling $O(\frac{1}{\varepsilon})$ factors of length K uniformly at random, we can show a positive constant lower bound on the probability to find σ. We can repeat this step to obtain a probability $> 1/2$ of finding $|\mathcal{A}|$ occurrences the sequence σ. This witnesses that $\mu \notin \mathcal{L}(\mathcal{A})$, by Lemma 4.10. ◀

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

At this point we can revisit the class of *trivial* regular languages identified by Alon et al. [5].

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.

We give a combinatorial characterization of trivial languages (previously identified by Alon et al. [5]) based on their set of blocking factors. Our notion of *trivial* languages coincides with the notion of trivial languages of Alon et al. [5], recalled below.

▶ **Definition 5.2** ([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 $d(w, L) \geq \varepsilon_0 n$.*

We show that triviality is equivalent to having no blocking sequence. Recall that we focus on infinite languages, since we know that all finite ones are trivial (Remark 2.3).

▶ **Lemma 5.3.** *$\mathsf{MBS}(\mathcal{A})$ is empty if and only if $L = \mathcal{L}(\mathcal{A})$ is trivial.*

Proof sketch. The full proof is in Appendix C. Suppose $\mathsf{MBS}(\mathcal{A})$ is empty. By Lemma 4.11, for all $\varepsilon > 0$ long enough words cannot be ε-far from L, hence L is trivial.

Suppose $\mathsf{MBS}(\mathcal{A})$ contains a blocking sequence $\sigma = (\mu_1, \dots, \mu_k)$. For all $N \in \mathbb{N}$ we construct a word w_N containing $k+1+N$ factors μ_1, ... , then $k+1+N$ factors μ_k, plus a few letters for padding. This word has length $\mathcal{O}(N)$. Furthermore, by changing N letters we always obtain a word with $k+1$ disjoint occurrences of σ, and thus not in L by Lemma C.4. As a result, $d(w_N, L) > N$ and thus w_N is ε-far from L for some ε independent of 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 regular languages could be classified in three classes, which characterise their query complexity. 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?

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 all three problems are PSPACE-complete.

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

The proof is presented in Appendix F. The upper bound requires us to give another characterisation of hard languages, as it is not clear that the one we provided before can be turned into a PSPACE algorithm. The hardness is proven by a reduction from NFA universality. We transform a given automaton so that it remains universal if it was at the beginning, but any word out of the language yields an infinite family of minimal blocking factors in the new automaton.

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: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: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 sampled 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 languages. *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 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`.

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

A The Case of Strongly Connected NFAs

A.1 Positional words, blocking factors and strongly connected NFAs

We establish some properties of positional words that will help us ensure that we are creating well-formed positional words. We fix a strongly connected automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and sets of states $Q_0, \ldots, Q_{\lambda(\mathcal{A})-1}$ as in Fact 2.4, with the initial state q_0 in Q_0.

▶ **Lemma A.1.** *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 $p \stackrel{u}{\not\to} q$.*

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 words w, w' such that $q_0 \xrightarrow{w} p$ and $q \xrightarrow{w'} q_f$ for some $q_f \in F$. By Fact 2.4, the positional word $(0 : wuw')$ contains τ as a factor. Since wuw' labels a path from q_0 to q_f, we have $(0 : wuw') \in \mathcal{TL}(\mathcal{A})$, and τ is not blocking.

For the converse, assume that τ is not 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, $q_0 \xrightarrow{w} r$. Since $\tau \preccurlyeq (0 : w)$, 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 $q_0 \xrightarrow{v} p \xrightarrow{u} q \xrightarrow{w} r$: in particular, we have $p \xrightarrow{u} q$. It only remains to show that p is in Q_i: this follows by Fact 2.4 since $|v| = i \pmod{\lambda}$ and $q_0 \in Q_0$. ◀

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 A.2. For any word $u \in \Sigma^*$, we have $d(u, \mathcal{L}(\mathcal{A})) = d((0 : u), \mathcal{TL}(\mathcal{A}))$.

Proof. It suffices to observe that for all $v \in \mathcal{L}(\mathcal{A})$, the ith letter differs in u and v if and only if it differs in $(0 : u)$ and $(0 : v)$. Hence $d(u, v) = d((0 : u), (0 : v))$ for all $v \in \mathcal{L}(\mathcal{A})$, and therefore $d(u, \mathcal{L}(\mathcal{A})) = d((0 : u), \mathcal{TL}(\mathcal{A}))$. ◁

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 $\mathcal{TL}(\mathcal{A})$".

A.2 Upper Bound for Strongly Connected NFAs

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 blocking factors.

▶ **Lemma A.3.** *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 $i_1 = \rho$. 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 3 for an illustration of this construction.) Using the assumption that τ is ε-far from $\mathcal{TL}(\mathcal{A})$ (which follows from Claim A.2) will give us the desired bound on k.

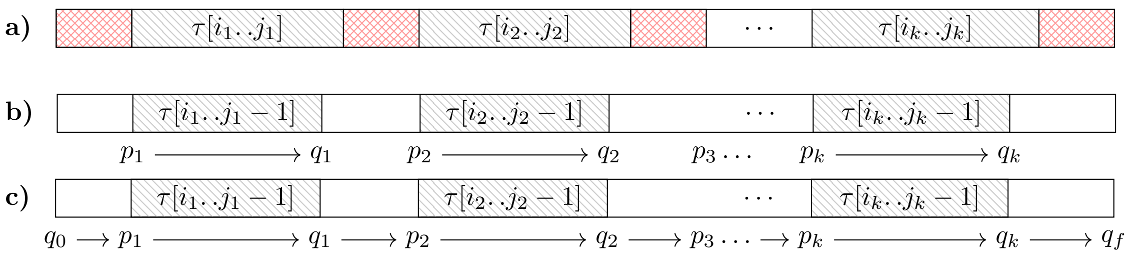

Figure 3 **a)** The decomposition process returns k factors $\tau[i_1, j_t], \ldots, \tau[i_k, j_k]$ (represented as diagonally hatched in gray regions), separated together and with the start of the word by padding regions of $\rho - 1$ letters (red cross-hatched regions). **b)** After removing the last letter, each previously blocking factor now labels a path between some pair of states p_t, q_t. **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 $p_t \in Q_k$ to some state $q_t \in Q_{k+j_t-i_t}$. Therefore, using the strong connectivity of $\mathcal{A}$ and Fact 2.4, we can substitute the letters in $\tau[j_t..j_t + \rho - 1]$ to obtain a factor that labels a path from q_t to p_{t+1}. After this transformation, the word $\tau[i_t..j_t + \rho - 1]$ labels a path 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:

$$
\begin{aligned}
3(k+1)m^2 \geq \varepsilon n &\Rightarrow k + 1 \geq \frac{\varepsilon n}{3m^2} \\
&\Rightarrow k \geq \frac{\varepsilon n}{3m^2} - 1 \\
&\Rightarrow k \geq \frac{\varepsilon n}{6m^2}
\end{aligned}
$$

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

▶ **Lemma 3.5.** *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 A.3, $(0 : u)$ 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. ◀

▶ **Lemma 3.6.** *In the last* `Else` *block, if u is ε-far from $\mathcal{L}(\mathcal{A})$, then Algorithm 1 rejects with probability at least $2/3$.*

Proof. Assume that u is ε-far from $\mathcal{L}(\mathcal{A})$. As we are in the last `Else` block of Algorithm 1, $\mathcal{L}(\mathcal{A}) \cap \Sigma^n$ is not empty (i.e., $\mathcal{L}(\mathcal{A})$ contains a word of length n) and $n \geq \beta$, therefore the conditions of Lemma 3.5 are satisfied. Let $\mathcal{B}$ denote the set of minimal blocking factors in $(0 : u)$ given by Lemma 3.5: we have $|\mathcal{B}| \geq n/\beta$. We conceptually divide the blocking factors in $\mathcal{B}$ into different categories depending on their length: for $t = 0, \ldots, T$, let B_t denote the subset of $\mathcal{B}$ of blocking factors of length at most $\ell_t = 2^t$. We then carefully analyse the probability that randomly sampled factors of length $2\ell_t$ contains a blocking factor from B_t, and show that over all t, at least one blocking factor is found with probability at least $2/3$.

▷ Claim A.4. If in a call to Sample, the value i is such that there exists indices $l, r, l \leq i \leq r$, such that $(0 : u)[l, r]$ is a blocking factor of $\mathcal{A}$ of length at most ℓ, then the factor η returned by the function is blocking for $\mathcal{A}$.

As the factors given by Lemma 3.5 are disjoint, the probability p_t that the factor returned by Sample is blocking is lower bounded by

$$p_t \geq \frac{1}{n} \sum_{\tau \in B_t} |\tau|$$

The Sample 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 blocking factor satisfies the following:

$$\begin{aligned}
(1-p) &= \prod_{t=0}^{T} (1-p_t)^{r_t} \leq \exp\left(-\sum_{t=0}^{T} p_t r_t\right) \\
&\leq \exp\left(-\frac{2\ln(3)\beta}{n} \sum_{t=0}^{T} \frac{1}{\ell_t} \sum_{\tau \in B_t} |\tau|\right) \\
&= \exp\left(-\frac{2\ln(3)\beta}{n} \sum_{\tau \in \mathcal{B}} |\tau| \sum_{t=\lceil \log|\tau| \rceil}^{T} 2^{-t}\right) \\
&\leq \exp\left(-\frac{2\ln(3)\beta}{n} \sum_{\tau \in \mathcal{B}} |\tau| \cdot 2^{-\lceil \log|\tau| \rceil}\right) \\
&\leq \exp\left(-\frac{2\ln(3)\beta}{n} \sum_{\tau \in \mathcal{B}} |\tau| \frac{1}{2|\tau|}\right) \\
&= \exp\left(-\frac{2\ln(3)\beta}{n} \cdot \frac{|\mathcal{B}|}{2}\right) \\
&\leq \exp\left(-\frac{2\ln(3)\beta}{n} \cdot \frac{n}{2\beta}\right) \\
&\leq \exp(-\ln(3)) = 1/3
\end{aligned}$$

It follows that $p \geq 2/3$, and Algorithm 1 satisfies Definition 2.2. ◀

A.3 Lower Bound for strongly connected automata with infinitely many blocking factors

Let $\mathcal{A}$ be a strongly connected NFA. Let $\widehat{\mathcal{A}}$ be an automaton recognising $\mathcal{TL}(\mathcal{A})$, as defined just after Definition 2.5. As $\mathcal{A}$ is strongly connected, so is $\widehat{\mathcal{A}}$: from every reachable state we can go back to the initial state by following a path to the initial state of $\mathcal{A}$ and adding cycles to adjust the modulo. In this whole section we will thus assume that $\mathcal{A}$ recognises $\mathcal{TL}(\mathcal{A})$ instead of just $\mathcal{L}(\mathcal{A})$.

A.3.1 The structure of MBF($\mathcal{A}$)

We prove that the set of minimal blocking factors of an automaton is a regular language, albeit recognized by an automaton that is possibly exponentially larger that $\mathcal{A}$. We first prove the result for blocking factors of the form $(i : u)$ for a fixed $i \in \mathbb{Z}/\lambda\mathbb{Z}$.

▶ **Lemma A.5.** *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.

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

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

This automaton[5] recognizes the set of i-blocking factors of $\mathcal{A}$ and has size $\mathcal{O}(m)$. We can then construct $\mathcal{B}_i$ a deterministic automaton for the same language.

The result follows by using a standard construction: build an automaton that runs $\mathcal{B}_i$ on the input word, and $\mathcal{B}_{i+1}$ on the word obtained by ignoring the first letter. It accepts if the word is accepted by $\mathcal{B}_i$, and the words obtained respectively by ignoring the first and last letters are rejected by respectively $\mathcal{B}_{i+1}$ and $\mathcal{B}_i$. A word is accepted if and only if it is a minimal blocking factor.

This 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 A.6.** *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 $2^{\mathcal{O}(m)}$.*

Therefore, if MBF($\mathcal{A}$) is infinite, we can use Kleene's lemma to find an infinite family of minimal blocking factors with a shared structure $\{\phi\nu^r\chi, r \in \mathbb{N}\}$. We will use this property later, when proving a lower bound against the language of automata with infinitely many blocking factors.

▶ **Lemma A.7.** *If MBF($\mathcal{A}$) is infinite, then there exist positional words $\phi, \nu_+, \nu_-, z, \chi$ such that:*

1. *the words ν_+ and ν_- have the same length,*
2. *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 $\tau_{-,r} = \phi(\nu_-)^r z$ is blocking for $\mathcal{A}$, and for every $s < r$, we have*

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

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

[5] 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.

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 i_* such that $\mathcal{A}$ has infinitely many minimal i_*-blocking factors; we fix such an i_* in what follows.

As the set of minimal i_*-blocking factors is an infinite regular language recognized by an NFA of size $S = 2^{\mathcal{O}(m)}$, by Kleene's Lemma, there exist positional words τ, μ, η, each of length at most S with $|\mu| \geq 1$, such that for any non-negative integer $k, \tau\mu^k\eta$ is a minimal i_*-blocking factor. 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 2.4, $p_1 \xrightarrow{\mu^{k_1}} p_1$ 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 labelled by μ, p_1 and p_2 belong to the same periodicity class. Therefore, by Fact 2.4, 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. ◀

A.3.2 Constructing a hard distribution

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 A.7.

Let S be the length of ν_- and ν_+. 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 $[0..n-1]$ 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:

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

The event $\kappa_j > 0$ means that the j-th interval is filled with 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 $q_* \xrightarrow{w_*} q_*$: by Fact 2.4, this is possible as $\ell = 0 \pmod{\lambda}$. Then, if $\kappa_j = 0$, we set $\mu[a_j..b_j] = \eta_*$, 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 $\mu[a_j..b_j] = (\tau_{-,r})^N \eta$,
- if $\pi = 1$, we select $s \in [0..r-1]$ uniformly at random, and set $\mu[a_j..b_j] = (\tau_{+,r,s})^N \eta$.

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.

A.3.3 Properties of the distribution $\mathcal{D}$

We first state the important properties of the distribution $\mathcal{D}$

▶ **Observation A.8.** *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 A.9.** *If $\pi = 1$, then $\mu \in \mathcal{TL}(\mathcal{A})$.*

We recall Hoeffding's inequality, which will be used in the next proof.

▶ **Lemma A.10** ([20, Theorem 2]). *Let $X_1, \dots, X_k$ be independent random variables such that for every $i = 1, \dots, k$, we have $a_i \le X_i \le b_i$, and let $S = \sum_{i=1}^k$. Then, for any $t > 0$, we have*

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

▶ **Lemma 3.8.** *Conditioned on $\pi = 0$, the probability of the event $\mathcal{F} = \{u$ 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 A.11. Let $B = \sum_{j=1}^k B_j$, and let $E = \mathbb{E}[B]$. We have $E \ge 2\varepsilon n$.

Claim proof. By direct calculation:

$$
\begin{aligned}
E &= \sum_{j=1}^{k} \mathbb{E}\left[B_j\right] && \text{by linearity}\\
&= \sum_{j=1}^{k} \sum_{t=1}^{\log(S/\varepsilon)} 2^{-t}/(S\varepsilon)\cdot p_t && \text{def. of expectation}\\
&= \sum_{j=1}^{k} \sum_{t=1}^{\log(S/\varepsilon)} 2^{-t}/(S\varepsilon)\cdot 3\cdot 2^{t}\varepsilon S/\log(S/\varepsilon) && \text{def. of } p_t\\
&= \sum_{j=1}^{k} \sum_{t=1}^{\log(S/\varepsilon)} 3/\log(S/\varepsilon)\\
&= 3k \geq 2\varepsilon n
\end{aligned}
$$

◀

We will now show that $\mathbb{P}(B < \varepsilon n)$ goes to 0 as n goes to infinity. By Claim A.11, 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 (Lemma A.10), 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. ◀

▶ **Lemma 3.10.** *Let T be a deterministic tester with perfect completeness (i.e. one sided error, 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 u is* 1.

Proof. We show that if there exists a 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, hence $\tau \notin \mathcal{TL}(\mathcal{A})$, 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. ◀

▶ **Lemma 3.11.** *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 $\frac{1}{72}\cdot\log(\varepsilon^{-1})/\varepsilon$ queries, i.e. $\sum_j q_j \leq \frac{1}{72}\cdot\log(\varepsilon^{-1})/\varepsilon$. The probability of the event $\mathcal{M} = \{\forall j \text{ s.t. } \kappa_j > 0, q_j < 2^{\kappa_j}\}$ 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:

$$\begin{aligned}
\mathbb{P}\left(\overline{\mathcal{M}}\right) &= \mathbb{P}\left(\exists j : \kappa_j > 0 \wedge q_j \geq 2^{\kappa_j}\right) \\
&\leq \sum_j \mathbb{P}\left(\kappa_j > 0 \wedge q_j \geq 2^{\kappa_j}\right) && \text{by union bound} \\
&\leq \sum_j \sum_{t=1}^{\lfloor \log q_j \rfloor} p_t \\
&= \sum_j \sum_{t=1}^{\lfloor \log q_j \rfloor} \frac{3 \cdot 2^t \varepsilon}{\log(S/\varepsilon)} && \text{by def. of } p_t \\
&\leq \frac{3\varepsilon}{\log(S/\varepsilon)} \sum_j \sum_{t=1}^{\lfloor \log q_j \rfloor} 2^t \\
&\leq \frac{3\varepsilon}{\log(S/\varepsilon)} \sum_j 2q_j \\
&= \frac{3\varepsilon}{\log(S/\varepsilon)} \cdot \frac{2}{72} \cdot \frac{\log(1/\varepsilon)}{\varepsilon} \\
&\leq 1/12
\end{aligned}$$

◀

B Useful examples for blocking sequences

The goal of this appendix is to provide a few examples to guide the intuition of the reader through the definitions of Section 4.1.

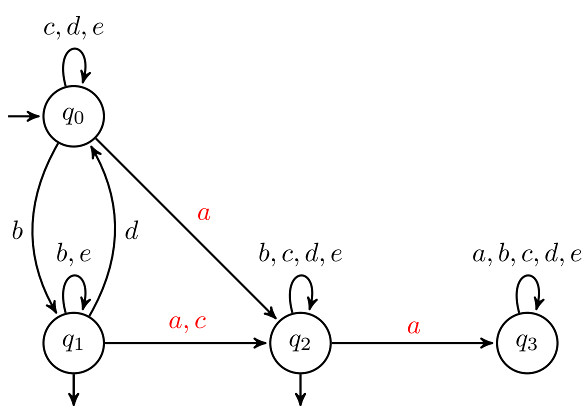

Figure 4 An automaton $\mathcal{A}_2$ recognizing the language $L_2 = [((c+d+e)^*b(b+e)^*d)^*a](b+c+d+e)^*$.

▶ **Example B.1.** Consider the automaton in Figure 4: 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, as otherwise we can make it accepted by deleting all a, thanks to the second SCC. However, a is also a blocking factor of the first SCC, therefore, as soon as we find a's in the word, we know that it is not in L_2. 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.

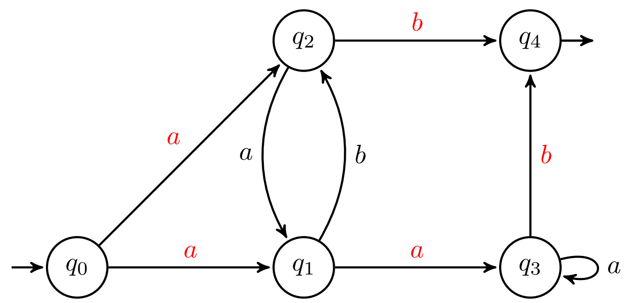

Figure 5 Automaton used for Example B.2.

▶ **Example B.2.** Consider the automaton displayed in Figure 5. 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 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_3. 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.

▶ **Example B.3.** Consider the automaton $\mathcal{A}$ displayed in Figure 6: it only has cycles of length 1, hence $p = 1$. Its states are totally ordered by $\leq_{\mathcal{A}}$.

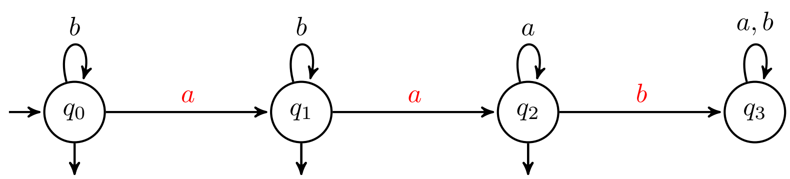

Figure 6 An automaton recognizing the language $b^* + b^*ab^*a^*$.

Observe that the sequence $\sigma = ((0 : a), (0 : b))$ is a blocking sequence for the SCC-path $\pi = q_0, 0 \rightsquigarrow q_0, 0 \xrightarrow{a} q_1, 0 \rightsquigarrow q_1, 0 \xrightarrow{a} q_2, 0 \rightsquigarrow q_2, 0$. Indeed, a is blocking for the first two portals, and b for the third.

We can verify Lemma C.4 on this example: if a word u contains $|Q| = 4$ disjoint occurrences of σ, then in particular it must contain factors a, a and b in that order, hence u

is not in $\mathcal{L}(\mathcal{A})$. Two occurrences of blocking sequences would be enough here, but note that one occurrence is not enough: the word $(0 : aba)$ contains $((0 : a), (0 : b))$, yet it is in the language of π.

C Properties of blocking sequences

▶ **Lemma C.1.** *Let $\mathcal{A}$ be an automaton and P a portal in $\mathcal{A}$. 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 periodicity. 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 2.4. The set of states of $\mathcal{A}'$ is given by

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

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 $s_1 \in Q_{i_1}, s_2 \in Q_{i_1}$ 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 2.4 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 periodicity 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 periodicity. ◀

In order to simplify the exposition of the proofs of later results, let us start with two technical lemmas expressing two basic properties of the Hamming distance with respect to $\mathcal{A}$. The first one extends Kleene's Lemma on 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.

▶ **Lemma C.2.** *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 4.6):

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

It follows that a word $\mu \in \mathcal{L}(\pi)$ can be written as $\mu = \mu_1 a_1 \mu_2 \dots \mu_k$ 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 2.4, 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 C.3.** *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 $\ell \pmod p$.*

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.

▶ **Lemma C.4.** *Let $\pi = P_0 \xrightarrow{a_1} P_1 \cdots P_k$ be an SCC-path. If μ contains $k+1$ disjoint occurrences of blocking sequences for π, then $\mu \notin \mathcal{L}(\pi)$.*

Proof. We prove the statement by induction on k.

For $k = 0$, the language of π is exactly the language of P_0, and blocking sequences of π contain at least a blocking factor of P_0. Therefore, a word containing a blocking sequence is not in $\mathcal{L}(P_0) = \mathcal{L}(\pi)$.

Now let $k > 0$ and suppose that the induction hypothesis holds for $k-1$. Assume for the sake of contradiction that μ is in $\mathcal{L}(\pi)$ and that it contains $k+1$ occurrences of blocking sequences. As it is in $\mathcal{L}(\pi)$, we can decompose μ into $\mu = \mu_0 a_0 \mu'$ with $\mu_0 \in \mathcal{L}(P_0)$ and $\mu' \in \mathcal{L}(\pi')$, where $\pi' = P_1 \xrightarrow{a_2} \cdots P_k$ is the tail of π. Now, we can assume w.l.o.g. that the first element ν of each of the $k+1$ blocking sequence occurring in μ is blocking for P_0, as otherwise the removal of ν still leaves a blocking sequence. It follows that each ν does not occur completely in μ_0, as μ_0 is in $\mathcal{L}(P_0)$. Furthermore, as the sequences are disjoint, at most one ν can overlap with $\mu_0 a_0$, hence at least k blocking sequences are fully contained in μ'. Our induction hypothesis implies that μ' is not in $\mathcal{L}(\pi')$, a contradiction. This concludes our induction. ◀

The next lemma expresses a kind of converse implication, which generalizes Lemma A.3 from the strongly connected case: if a word is far from the language, then it contains many blocking sequences.

▶ **Lemma C.5.** *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 $n \geq 2C/\varepsilon$ 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 $\frac{\varepsilon n}{C}$ blocking factors for P_i, for some constant C.*

Proof. We proceed similarly to the proof of Lemma A.3, 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 ρ positions later, where ρ is the reachability constant of a strongly connected automaton recognizing $\mathcal{L}(P_0)$. When we have found at least $K = \frac{\varepsilon n}{C(k+2)}$ 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 w.l.o.g. that we stop before finding enough blocking factors for P_i. As in the proof of Lemma A.3, we replace the last letter of each blocking factor and use the padding between them to make the run accepted by the automaton: this uses at most $((i+1) \cdot K + 2)\rho \leq \varepsilon n$ substitutions, if we set $C = 4(k+1)\rho$. 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. ◀

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 C.6.** *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 $\sigma = (\mu_1 \dots, \mu_\ell)$ be a minimal blocking sequence for $\mathcal{A}$. For all $i = 1, \dots, \ell$, we define $\sigma_i = (\mu_1 \dots, \mu_i)$, 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 $\ell \le p^2|Q|^2$. ◀

▶ **Lemma 4.10.** *If μ contains $|\mathcal{A}|$ disjoint blocking sequences for $\mathcal{A}$ then $\mu \notin \mathcal{L}(\mathcal{A})$.*

Proof. Let π be an accepting SCC-path through $\mathcal{A}$. Note that number ℓ of portals in π is at most $|Q|$. By definition, a blocking sequence for $\mathcal{A}$ is a blocking sequence for π. As μ contains $|Q|$ disjoint blocking sequences for $\mathcal{A}$, it contains $\ell \le |Q|$ disjoint blocking sequences for π, hence $\mu \notin \mathcal{L}(\pi)$ by Lemma C.4. As a result, μ is not in the language of any accepting SCC-path of $\mathcal{A}$, and thus not in $\mathcal{L}(\mathcal{A})$. ◀

▶ **Lemma 4.11.** *Let μ be a word of length n. There exist constants E, K such that if $+\infty > d(\mu, \mathcal{L}(\mathcal{A})) \ge \varepsilon n$ and $n \ge E/\varepsilon$, then μ can be partitioned into $\mu = \mu_0\mu_1 \cdots \mu_K$ such that for every $i = 0, \dots, K$, μ_i contains at least $\frac{\varepsilon n}{E}$ disjoint occurrences of words $\nu_{i,1}, \nu_{i,2} \dots$, each of length $\mathcal{O}(1/\varepsilon)$, such that for any choice of $j_0, j_1, \dots, j_K$, the sequence $(\nu_{0,j_0}, \nu_{1,j_1}, \dots, \nu_{K,j_K})$ is a blocking sequence for $\mathcal{A}$.*

Proof. We start by observing that an SCC-path contains at most $|\mathcal{A}|$ portals, hence there are at most $(|\mathcal{A}|^2 p^2 |\Sigma| + 1)^{|\mathcal{A}|}$ SCC-paths in $\mathcal{A}$. Let $K \le (|\mathcal{A}|^2 p^2 |\Sigma| + 1)^{|\mathcal{A}|}$ denote the number of accepting SCC-paths of $\mathcal{A}$, and let Π be the set of these SCC paths.

For each accepting SCC-path π, as $\mathcal{L}(\pi) \subseteq \mathcal{L}(\mathcal{A})$, $d(\mu, \mathcal{L}(\pi)) \ge d(\mu, \mathcal{L}(\mathcal{A}))$, hence μ contains $\frac{\varepsilon n}{C}$ disjoint blocking sequences for π, by Lemma C.5. We can apply this result as long as $D \ge 2C$: we define $D = \max(2C, K \cdot C)$. As the blocking sequences are disjoint, the sum of their lengths does not exceed n, thus up to doubling C, we have that $\frac{\varepsilon n}{C}$ of them have a total length of $\mathcal{O}(1/\varepsilon)$.

It remains to prove that $\frac{\varepsilon n}{C}$ disjoint blocking sequences for each π implies $\frac{\varepsilon n}{D}$ disjoint blocking sequences for $\mathcal{A}$. We decompose μ into $\eta_1, \dots, \eta_K$ and index the accepting SCC-paths $\Pi = \{\pi_1, \dots, \pi_K\}$ such that each η_i contains a $1/K$-fraction of the paths that are blocking for π_i, the i-th accepting SCC path of $\mathcal{A}$. We proceed as follows: for all $i < K$ we define η_i so that $\eta_1 \cdots \eta_i$ is the shortest prefix of μ such that there exist $\pi_i \in \Pi \setminus \{\pi_1, \dots, \pi_{i-1}\}$ such that there are $\ge \frac{\varepsilon n i}{KC}$ disjoint blocking sequences for π_i in $\eta_1 \cdots \eta_i$. We define η_K so that $\eta_1 \cdots \eta_K = \mu$. As all $\pi \in \Pi$ have at least $\frac{\varepsilon n}{C}$ blocking sequences in μ, this sequence is well-defined. Furthermore, the π_i are distinct by definition.

It follows that η_i contains $\frac{\varepsilon n}{KC} \ge \frac{\varepsilon n}{D}$ disjoint blocking sequences for π_i. Finally, concatenating a blocking sequence from each of the η_i yields a blocking sequence for $\mathcal{A}$.

Note that the resulting blocking sequences $\sigma = (\xi_0, \xi_1, \dots, \xi_t)$ contain at most $t \le K \cdot |\mathcal{A}|$ elements, as the blocking sequences for the π_i contain at most $|\mathcal{A}|$ elements. The word μ_0 is the prefix of μ that contains the element ξ_0 from the first $\frac{\varepsilon n}{E}$ sequences σ, where $E = D \cdot |\mathcal{A}|$; the selected ξ_0 are the $(\nu_{0,j})_j$. We then iterate on the remainder of the word, with μ_i containing the elements ξ_i from $\frac{\varepsilon n}{E}$ subsequent sequences σ. ◀

D Languages with infinitely many blocking sequences are hard

D.1 Proof of Lemma 4.17

▶ **Lemma 4.17.** *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. As $\mathcal{A}$ has infinitely many minimal blocking sequences, there exists an accepting SCC-path π with infinitely many minimal blocking sequences. By Lemma C.6, the number of terms in a minimal blocking sequence is bounded, therefore there is a portal in π with infinitely many minimal blocking factors.

If $\mathcal{A}$ has infinitely many minimal blocking sequences, let $(\sigma_j)_{j\in\mathbb{N}}$ be a family of minimal blocking sequences such that the sum of the lengths of the terms of σ_j is at least j for all j.

By Lemma C.6, a minimal blocking sequence has a bounded number of elements. We can thus extract from this sequence another one $(\sigma'_j)_{j\in\mathbb{N}}$ such that each σ'_j contains a factor of length at least j.

For each j let i_j be the index in σ'_j of a factor of length at least j, and l_j and r_j respectively the left effect of the $i_j - 1$ first factors and the right effect of the $k_j - i_j$ last ones, with k_j the length of σ'_j. As those objects are taken from bounded sets, we can obtain a third sequence $(\bar{\sigma}_j)_{j\in\mathbb{N}}$ and α and K such that the ith element of each $\bar{\sigma}_j$ has length at least j and the set of components for which it is blocking is K.

For all j let (n_j, u_j) be the ith element of $\bar{\sigma}_j$. Define $\sigma_l = (n^l_1 : u^l_1), \ldots, (n^l_k : u^l_k)$ and $\sigma_r = (n^r_1 : u^r_1), \ldots, (n^r_\ell : u^r_\ell)$ so that $\bar{\sigma}_1 = \sigma_l(n_1, u_1)\sigma_r$. For all j, $\sigma_l(n_j : u_j)\sigma_r$ is a minimal blocking sequence.

We call surviving portals the portals that survive (σ_l, σ_r) in at least one SCC-path.

▷ Claim D.1. There exists a surviving portal with infinitely many minimal blocking factors that is minimal for $\preceq$ among surviving portals.

Proof. Suppose by contradiction that all $\preceq$-minimal surviving portals have finitely many minimal blocking factors.

For all j, $(n_j : u_j)$ must be blocking for all surviving portals (otherwise $\overline{\sigma}_j$ would not be blocking for $\mathcal{A}$). Hence $(n_j : u_j)$ contains a blocking factor for each $\preceq$-minimal surviving portal. As those factors are bounded while $(n_j : u_j)$ can get arbitrarily large, there exists j such that $(n_j : u_j)$ can be split into two non-empty parts $(n_j : u_j^-)(n_j^+ : u_j^+)$ so that each $\preceq$-minimal surviving portal has a minimal blocking factor in either $(n_j : u_j^-)$ or $(n_j^+ : u_j^+)$. As a consequence, every surviving portal has a blocking factor in either $(n_j : u_j^-)$ or $(n_j^+ : u_j^+)$.

Let P be the number of portals of $\mathcal{A}$. We obtain that $\sigma_l[(n_j : u_j^-)(n_j^+ : u_j^+)]^P\sigma_r$ is a blocking sequence for $\mathcal{A}$, contradicting the minimality of $\sigma_l(n_j : u_j)\sigma_r$ for $\trianglelefteq$. In conclusion, there is a $\preceq$-minimal surviving portal with infinitely many minimal blocking factors. ◁

Let $s, x \rightsquigarrow t, y$ be a $\preceq$-minimal surviving portal with infinitely many minimal blocking factors: It satisfies P2.

The following claim shows that there is a pair of sequences (σ_l, σ_r) such that properties P1 and P3 are satisfied.

▷ Claim D.2. There exist σ_l, σ_r such that $\sigma_l\sigma_r$ is not a blocking sequence for $\mathcal{A}$, and for all accepting SCC-path π in $\mathcal{A}$, every surviving portal in π is $\simeq$-equivalent to $s, x \rightsquigarrow t, y$.

Proof. We start from the sequences σ_l, σ_r defined before and extend them so that they have the desired property.

For each $s', x' \rightsquigarrow t', y' \not\simeq s, x \rightsquigarrow t, y$, since $s, x \rightsquigarrow t, y$ is $\preceq$-minimal we can pick a positional word $(n : u)_{s',x'\rightsquigarrow t',y'}$ that is blocking for $s', x' \rightsquigarrow t', y'$ but not for $s, x \rightsquigarrow t, y$.

We extend σ_l and σ_r as follows. While there is a surviving portal $s', x' \rightsquigarrow t', y'$ that is not $\simeq$-equivalent to $s, x \rightsquigarrow t, y$:

- We pick an SCC-path π such that $s', x' \rightsquigarrow t', y'$ survives in π.
- Let $i_\ell = (\sigma_l \gg \pi)$ and $i_r = (\pi \ll \sigma_r)$
- If for all $i \in \{i_\ell + 1, \dots, i_r - 1\}$, $s_i, x_i \rightsquigarrow t_i, y_i \not\simeq s, x \rightsquigarrow t, y$ then we append at the end of σ_l the sequence $(n : u)_{s_{i_\ell+1},x_{i_\ell+1}\rightsquigarrow t_{i_\ell+1},y_{i_\ell+1}}, \dots, (n : u)_{s_{i_r-1},x_{i_r-1}\rightsquigarrow t_{i_r-1},y_{i_r-1}}$. The sequence $\sigma_l\sigma_r$ is now blocking for π. On the other hand, since we did not add any blocking factor for $s, x \rightsquigarrow t, y$, there must still be a surviving portal that is $\simeq$-equivalent to it.
- If there is an $i \in \{i_\ell + 1, \dots, i_r - 1\}$ such that $s_i, x_i \rightsquigarrow t_i, y_i \simeq s, x \rightsquigarrow t, y$ then let c be the maximal index in $\{i_\ell + 1, \dots, i\}$ such that (m_c, s_c, t_c) is not equivalent to $s, x \rightsquigarrow t, y$ for $\simeq$, or i_ℓ if there is no such index. Symmetrically, let d the minimal index in $\{i, \dots, i_r - 1\}$ such that $(m_d, s_d, t_d) \not\simeq s, x \rightsquigarrow t, y$, or i_r if there is no such index. We append at the end of σ_l the sequence $(n : u)_{s_{i_\ell+1},x_{i_\ell+1}\rightsquigarrow t_{i_\ell+1},y_{i_\ell+1}}, \dots, (n : u)_{m_c,s_c,t_c}$. We append at the beginning of σ_r the sequence $(n : u)_{s_d,x_d\rightsquigarrow t_d,y_d}, \dots, (n : u)_{s_{i_r-1},x_{i_r-1}\rightsquigarrow t_{i_r-1},y_{i_r-1}}$. Now all surviving portals in π are $\simeq$-equivalent to $s, x \rightsquigarrow t, y$, and $s_i, x_i \rightsquigarrow t_i, y_i$ still survives.

We iterate this step until all surviving portals are $\simeq$-equivalent to $s, x \rightsquigarrow t, y$. 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 $\mathcal{A}$. ◁

◀

D.2 Proof of Lemma 4.18

We show the following statement.

▶ **Lemma 4.18.** *If there exist $P = s, x \rightsquigarrow t, y$ and σ_l, σ_r satisfying properties P1, P2 and P3 then $\mathcal{L}(\mathcal{A})$ is hard.*

▶ **Lemma D.3.** *Let $\pi = s_0, x_0 \rightsquigarrow t_0, y_0 \xrightarrow{a_1} \cdots s_\ell, x_\ell \rightsquigarrow t_\ell, y_\ell$ be an accepting SCC-path, and let $i \in \{0, \dots, \ell\}$. Let $\sigma_l = (n_1^l : u_1^l), \dots, (n_k^l : u_k^l)$ a sequence such that $(\sigma_l \gg \pi) < i$ and $N \in \mathbb{N}$.*

Then there is a word w^l of length at most $(3|\mathcal{A}|^3 + |\mathcal{A}|)(k + 1) + N(2p^2 + p)k|\mathcal{A}| + pN \sum_{i=1}^{k} |u_i^l|$ 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 $(x_0 : w)$ contains N times $(n_1^l : u_1^l)$, ..., N times $(n_k^l : u_k^l)$ as disjoint factors, in that order.

Proof. We define w^l by induction on k. As π is accepting, by definition $\mathcal{L}(\pi) \neq \emptyset$, and thus for all $j \in \{0, \dots, \ell\}$ there exists a word of length $y_j - x_j \pmod p$ labelling a path from s_j to t_j. By Fact 2.4, there is such a word v_j of length at most $3|\mathcal{A}|^2$. As a result, for all $z \in \{0, \dots, \ell\}$ we can form a word $w_z = v_0 a_1 v_1 \cdots a_z$, of length at most $3|\mathcal{A}|^3 + |\mathcal{A}|$, labelling a path of length $x_z \pmod p$ from q_0 to s_z in $\mathcal{A}$. If $k = 0$, we can simply set $w^l = w_i$.

Let $k > 0$, suppose the lemma holds for $k - 1$. Let $j = ((n_1 : u_1^l) \gg \pi)$. As $((n_1 : u_1^l) \gg \pi) \leq (\sigma_l \gg \pi) < i$, we have $j < i$. By definition, $(n_1 : u_1^l)$ is not blocking for

$s_{j+1}, x_{j+1} \rightsquigarrow t_{j+1}, y_{j+1}$. As a consequence, there is a word v_j labelling a path from s_j to t_j such that $(x_j : v_j)$ has $(n_1 : u_1^l)$ as a factor. We can remove cycles of length $0 \pmod p$ in that path, before and after reading $(x_j : v_j)$, so we can assume that $|v_j| \leq |u_1^l| + 2p|\mathcal{A}|$. As s_j and t_j are in the same SCC, we can extend v_j into a word v_j' of length $\leq |v_j| + |\mathcal{A}| \leq |u_1^l| + (2p+1)|\mathcal{A}|$ that labels a cycle from s_j to itself.

Let $\sigma' = (n_2^l : u_2^l), \dots, (n_k^l : u_k^l)$ and $\pi' = s_{j+1}, x_{j+1} \rightsquigarrow t_{j+1}, y_{j+1} \xrightarrow{a_{j+2}} \cdots s_\ell, x_\ell \rightsquigarrow t_\ell, y_\ell$. By definition, we have $(\sigma' \gg \pi') = (\sigma_l \gg \pi) < i$. By induction hypothesis, there is a word w' of length $\leq (3|\mathcal{A}|^3 + |\mathcal{A}|)k + N(2p^2+p)(k-1)|\mathcal{A}| + pN\sum_{i=1}^{k-1}|u_i|$ such that $|w'| = x_i - x_j \pmod p$, there is a run reading w' from s_j to s_i in $\mathcal{A}$, and $(x_j : w)$ contains N times $(n_2^l : u_2^l)$, ..., N times $(n_k^l : u_k^l)$ as disjoint factors, in that order.

We set $w^l = w_j (v_j')^{pN} w'$. This word has length $x_i \pmod p$, and at most $|w_j| + pN|v_j'| + |w'| \leq 3|\mathcal{A}|^3 + |\mathcal{A}| + pN(|u_1^l| + (2p+1)|\mathcal{A}|) + |w'| \leq (3|\mathcal{A}|^3 + |\mathcal{A}|)(k+1) + N(2p^2+p)k|\mathcal{A}| + pN\sum_{i=1}^{k}|u_i^l|$. It labels a path from s_0 to s_i, and contains N times $(n_1^l : u_1^l)$, ..., N times $(n_k^l : u_k^l)$ as disjoint factors, in that order. ◀

▶ **Lemma D.4.** *Let $\pi = s_0, x_0 \rightsquigarrow t_0, y_0 \xrightarrow{a_1} \cdots s_\ell, x_\ell \rightsquigarrow t_\ell, y_\ell$ be an accepting SCC-path, and let $i \in \{0, \dots, \ell\}$. Let $\sigma_r = (n_1^r : u_1^r), \dots, (n_k^r : u_k^r)$ a sequence such that $(\pi \ll \sigma_r) > i$ and $N \in \mathbb{N}$.*

Then there is a word w^r of length at most $(3|\mathcal{A}|^3 + |\mathcal{A}|)(k+1) + N(2p^2+p)k|\mathcal{A}| + pN\sum_{i=1}^{k}|u_i^r|$ such that $|w^r| = y_\ell - y_i \pmod p$, there is a run reading w^r from t_i to t_ℓ in $\mathcal{A}$, and $(y_i : w^r)$ contains N times $(n_1^r : u_1^r)$, ..., N times $(n_k^r : u_k^r)$ as disjoint factors, in that order.

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.

Proof of Lemma 4.18. A direct consequence of properties P1 and P3 is that for all $(n' : u')$, $\sigma_l(n' : u')\sigma_r$ is blocking for $\mathcal{A}$ if, and only if $(n' : u')$ 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 3.7 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 D.3 and D.4 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 each element of σ_l at least εn times and w_r contains each element of σ_r at least εn times. Let $w = w^l v w^r$. Suppose $|v| \geq \frac{6p^2|\mathcal{A}|^2}{\varepsilon}$ and $d(v, \mathcal{L}(P)) < +\infty$.

- If $v \in \mathcal{L}(\mathcal{A})$ then clearly $w \in \mathcal{L}(\mathcal{A})$.
- If $d(v, \mathcal{L}(P)) \geq \varepsilon n$ then by Lemma A.3 (in light of Lemma C.1), $(x : v)$ 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 samples used on v is at most the number of samples needed on w, hence $f(\varepsilon/X)$. We obtain a procedure to test $\mathcal{L}(P)$ using $f(\varepsilon/X)$ samples. By Lemma C.1, $\mathcal{L}(P)$ is recognised by a strongly connected automaton. Furthermore by P2, P has infinitely many blocking factors. By Theorem 3.7, $f(\varepsilon/X) = \Omega(log(\varepsilon^{-1})/\varepsilon)$, hence $f(\varepsilon) = \Omega(log(\varepsilon^{-1})/\varepsilon)$. This concludes our proof. ◀

E Trivial and easy languages

E.1 Proof of Lemma 5.1

▶ **Lemma 5.1.** *If $\mathcal{A}$ has finitely many minimal blocking sequences, then there is a tester for $\mathcal{L}(\mathcal{A})$ using $\mathcal{O}(1/\varepsilon)$ queries.*

Proof. Let us start by proving that a word that is ε-far from $\mathcal{L}(\mathcal{A})$ must contain many times some minimal blocking sequence σ. We first check whether $\mathcal{A}$ contains a word of length n. If not, we reject, and otherwise, the distance between μ and $\mathcal{L}(\mathcal{A})$ is finite. Therefore, by Lemma 4.11, if μ is ε-far from $\mathcal{L}(\mathcal{A})$, then μ contains $N = \Omega(\varepsilon n)$ disjoint blocking sequences, organised as stated in the lemma. We can extract from each group of $|\mathcal{A}|$ such blocking sequence a minimal blocking sequence, hence μ contains at least N disjoint occurrences of minimal blocking sequences. By pigeonhole principle, there exists a minimal blocking sequence σ such that μ contains $N/P = \Omega(\varepsilon n)$ disjoint occurrences of σ.

If $|\mu| \leq \frac{D}{\varepsilon}$ then we read μ entirely, and accept iff it is in $\mathcal{L}(\mathcal{A})$: this uses $\mathcal{O}(1/\varepsilon)$ queries.

Otherwise, for each minimal blocking sequence σ, we sample uniformly at random $\mathcal{O}(1/\varepsilon)$ intervals of length K in μ. We reject if we find $|\mathcal{A}|$ disjoint occurrences of σ. If we have gone through every minimal blocking sequence without rejecting, we accept.

If $\mu \in \mathcal{L}(\mathcal{A})$, then by Lemma C.4 it cannot contain $|\mathcal{A}|$ disjoint blocking sequences, hence the algorithm will accept.

If the word is ε-far from $\mathcal{L}(\mathcal{A})$ (but within a finite distance), then there exists a minimal blocking sequence σ such that μ contains $\Omega(\varepsilon n)$ disjoint occurrences of σ. By sampling $\mathcal{O}(1/\varepsilon)$ factors of length K at random, we have a probability bounded away from zero of finding $|Q|$ of those occurrences, and subsequently rejecting μ.

We can iterate this procedure a constant number of times to obtain a procedure using $\mathcal{O}(\frac{1}{\varepsilon})$ samples that accepts every word in the language and rejects with probability at least 1/2 words that are ε-far from the language. ◀

E.2 Proof of Lemma 5.3

▶ **Lemma 5.3.** *$\mathsf{MBS}(\mathcal{A})$ is empty if and only if $L = \mathcal{L}(\mathcal{A})$ is trivial.*

We prove the two directions separately.

▶ **Lemma E.1.** *If $\mathsf{MBS}(\mathcal{A})$ is empty, then $L = \mathcal{L}(\mathcal{A})$ is trivial in the sense of Definition 5.2.*

Proof. We showed in Lemma 4.11 that if μ is long enough and ε-far from L, then μ contains $\Omega(\varepsilon n)$ disjoint blocking sequences for $\mathcal{A}$. As there are no blocking sequences for $\mathcal{A}$, long enough words cannot be ε-far from L, hence it is trivial in the sense of Definition 5.2. ◀

Alon et al. [5] show that if a language is non-trivial (in their sense), then testing it requires $\Omega(1/\varepsilon)$ queries for small enough $\varepsilon > 0$. 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.

▶ **Lemma E.2.** *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 $\mathcal{A}$, and let C be the maximum length of a μ_i's. As L is infinite, there exists an SCC-path π in $\mathcal{A}$ and $w \in \mathcal{L}(\pi)$ with $|w| \geq |\mathcal{A}|$. By Corollary C.3, 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 $(0 : v_i)$ 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 blocking sequence for $\mathcal{A}$. Therefore, the distance between w_N and $\mathcal{A}$ is at least $N - |\mathcal{A}|$. 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 for $N \geq 2|\mathcal{A}|$, we have $\varepsilon_0 |w_N| \leq N - |\mathcal{A}| \leq d(w_N, \mathcal{L}(\mathcal{A}))$, i.e. w_N is ε-far from L. ◀

F Complexity proofs

In this section we prove Theorem 6.4:

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

In Appendix F.1 we show the PSPACE upper bounds on the hardness and triviality problems (Propositions F.7 and F.9). The upper bound on the easiness problem follows immediately, as the three properties form a trichotomy.

In Appendix F.2, we show that all three problems are PSPACE-hard (Lemma F.11 and Corollary F.13).

F.1 A PSPACE upper-bound

F.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 $\mathcal{L}(\mathcal{A})$. However, there are subtleties in the construction due to the fact that the number of SCC-paths can be exponential in the $|\mathcal{A}|$. For instance, an automaton computing the set of left effects of a given sequence on SCC-paths of $\mathcal{A}$ would be doubly exponential. This would a priori not give a PSPACE algorithm, since we obtain a doubly-exponential state space.

We choose to prove another characterisation of automata with hard languages, which straightforwardly gives a recursive PSPACE algorithm to test it.

▶ **Lemma F.1.** *Let $\pi = s_0, x_0 \rightsquigarrow t_0, y_0 \xrightarrow{a_1} \cdots s_\ell, x_\ell \rightsquigarrow t_\ell, y_\ell$ 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 then we can set σ as the empty sequence.

If not, let π_{min} be such that the first term of $\sigma_{\pi_{min}}$ has the least left effect on π. Let $\sigma_{\pi_{min}} = (n_1 : u_1), \dots, (n_k : u_k)$ and $\pi_{min} = s'_0, x'_0 \rightsquigarrow t'_0, y'_0 \xrightarrow{a_1} \cdots s'_\ell, x'_\ell \rightsquigarrow t'_\ell, y'_\ell$. Let $j = ((n_1 : u_1) \gg \pi_{min})$ and $r = ((n_1 : u_1) \gg \pi)$.

Let $\pi' = s'_{j+1}, x'_{j+1} \rightsquigarrow t'_{j+1}, y'_{j+1} \xrightarrow{a_1} \cdots s'_\ell, x'_\ell \rightsquigarrow t'_\ell, y'_\ell$. Define $\Pi' = \Pi \setminus \{\pi_{min}\} \cup \{\pi'\}$ if $j < \ell$ and $\Pi' = \Pi \setminus \{\pi_{min}\}$ otherwise. In the first case the sequence associated with π' is $\sigma_{\pi'} = (n_2 : u_2), \dots, (n_k : u_k)$.

▷ Claim F.2. For all $\overline{\pi} \in \Pi \setminus \{\pi_{min}\}$, we have $(\sigma_{\overline{\pi}} \gg \pi) = r + (\sigma_{\overline{\pi}} \gg s_{r+1}, x_{r+1} \rightsquigarrow t_{r+1}, y_{r+1} \xrightarrow{a_{r+2}} \cdots s_k, x_k \rightsquigarrow t_k, y_k)$

Proof. Since the first term of $\sigma_{\pi'}$ 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.

Let $\overline{\pi} \in \Pi \setminus \{\pi_{min}\}$, let $\sigma_{\overline{\pi}} = (\overline{n}_1 : \overline{u}_1), \dots, (\overline{n}_m : \overline{u}_m)$. Let $z = ((\overline{n}_1 : \overline{u}_1) \gg \pi)$. This means $(\overline{n}_1 : \overline{u}_1)$ is not a blocking factor for $s_{z+1}, x_{z+1} \rightsquigarrow t_{z+1}, y_{z+1}$.

We have $(\sigma_{\overline{\pi}} \gg \pi) = z + ((\overline{n}_2 : \overline{u}_2), \dots, (\overline{n}_m : \overline{u}_m) \gg s_{z+1}, x_{z+1} \rightsquigarrow t_{z+1}, y_{z+1})$ and $(\sigma_{\overline{\pi}} \gg s_{r+1}, x_{r+1} \rightsquigarrow t_{r+1}, y_{r+1} \xrightarrow{a_{r+2} \cdots}) = z - r + ((\overline{n}_2 : \overline{u}_2), \dots, (\overline{n}_m : \overline{u}_m) \gg s_{z+1}, x_{z+1} \rightsquigarrow t_{z+1}, y_{z+1}) = (\sigma_{\overline{\pi}} \gg \pi) - r$.

◁

As a consequence of this claim, we have that $(\sigma_{\overline{\pi}} \gg s_{r+1}, x_{r+1} \rightsquigarrow t_{r+1}, y_{r+1} \xrightarrow{a_{r+2}} \cdots s_k, x_k \rightsquigarrow t_k, y_k) < i - r$ for all $\overline{\pi} \in \Pi \setminus \{\pi'\}$.

By induction hypothesis, we obtain a sequence σ' such that

- $(\overline{\sigma} \gg s_{r+1}, x_{r+1} \rightsquigarrow t_{r+1}, y_{r+1} \xrightarrow{a_1} \cdots s_\ell, x_\ell \rightsquigarrow t_\ell, y_\ell) < i - r$
- $(\sigma_{\pi'} \gg \pi') \leq (\sigma' \gg \pi')$ for all $\pi' \in \Pi'$.

The sequence $(n_1 : u_1), \sigma'$ satisfies both conditions of the lemma. ◀

▶ **Lemma F.3.** *An automaton $\mathcal{A}$ is hard if and only if there exists an accepting SCC-path π containing a portal $s, x \rightsquigarrow t, y$ such that:*

- *$s, x \rightsquigarrow t, y$ has infinitely many minimal blocking factors.*
- *For all accepting SCC-path π' there exist sequences σ^l, σ^r such that:*
  - *$s, x \rightsquigarrow t, y$ survives (σ^l, σ^r) in π*
  - *All portals surviving (σ^l, σ^r) in π' are $\simeq$-equivalent to $s, x \rightsquigarrow t, y$*

Proof. Let us start with the left-to-right direction. If $\mathcal{A}$ is hard then by Lemma 5.1 it has infinitely many minimal blocking sequences. Then by Lemma 4.17 we have a portal $s, x \rightsquigarrow t, y$ and sequences σ^l, σ^r satisfying properties P1, P2 and P3.

By P1, $\sigma^l \sigma^r$ is not blocking for $\mathcal{A}$, thus there exists an SCC-path $\pi = s_0, x_0 \rightsquigarrow t_0, y_0 \xrightarrow{a_1} \cdots s_k, x_k \rightsquigarrow t_k, y_k$ and an index i such that $(\sigma^l \gg \pi) < i < (\pi \ll \sigma^r)$.

As a consequence, we have $s_i, x_i \rightsquigarrow t_i, y_i \simeq s, x \rightsquigarrow t, y$, by P3. We can assume without loss of generality that $s_i, x_i \rightsquigarrow t_i, y_i = s, x \rightsquigarrow t, y$. As a result, for all accepting SCC-path π' we have that $s, x \rightsquigarrow t, y$ survives (σ^l, σ^r) in π and all portals surviving (σ^l, σ^r) in π' are $\simeq$-equivalent to $s, x \rightsquigarrow t, y$ (we use the same pair (σ^l, σ^r) for all π').

Let us now prove the other direction. Suppose we have π and $s, x \rightsquigarrow t, y$ 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 4.18.

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 $\pi' \in \Pi$:

- $s, x \rightsquigarrow t, y$ survives $(\sigma^l_{\pi'}, \sigma^r_{\pi'})$ in π
- All portals surviving $(\sigma^l_{\pi'}, \sigma^r_{\pi'})$ in π' are $\simeq$-equivalent to $s, x \rightsquigarrow t, y$

Let i be the index of $s, x \rightsquigarrow t, y$ in π. By Lemma F.1 we can build a sequence σ^l such that

- $(\sigma^l \gg \pi) < i$
- $(\sigma^l_{\pi'} \gg \pi') \leq (\sigma^l \gg \pi')$ for all $\pi' \in \Pi$.

Using a symmetric argument, we build a sequence σ^r such that

- $i < (\pi \ll \sigma^r)$
- $(\pi' \ll \sigma^r_{\pi'}) \geq (\pi' \ll \sigma^r)$ 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 $s, x \rightsquigarrow t, y$. Furthermore, $s, x \rightsquigarrow t, y$ survives (σ^l, σ^r) in π.

We have shown that $s, x \rightsquigarrow t, y$ and (σ^l, σ^r) satisfy properties P1 and P3. P2 is immediate by assumption. We simply apply Lemma 4.18 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 F.4.** *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 C.1, $L = \mathcal{L}(P)$ 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, the same property holds for the construction used in Lemma A.5, 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. ◀

▶ **Lemma F.5.** *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. We prove an intermediate result that yields a recursive PSPACE algorithm.

▷ Claim F.6. There is a sequence σ that is blocking for $\pi = s_0, x_0 \rightsquigarrow t_0, y_0 \xrightarrow{a_1} \cdots s_k, x_k \rightsquigarrow t_k, y_k$ and not $\pi' = s'_0, x'_0 \rightsquigarrow t'_0, y'_0 \xrightarrow{a'_1} \cdots s'_\ell, x'_\ell \rightsquigarrow t'_\ell, y'_\ell$ if and only if either:

- there is a positional word $(n : w)$ that is a blocking factor for $s_0, x_0 \rightsquigarrow t_0, y_0$ and not $s'_0, x'_0 \rightsquigarrow t'_0, y'_0$ and there is a sequence σ' that is blocking for $s_1, x_1 \rightsquigarrow t_1, y_1 \xrightarrow{a_2} \cdots s_k, x_k \rightsquigarrow t_k, y_k$ and not π',
- or there is a positional word $(n : w)$ that is a blocking factor for $s_0, x_0 \rightsquigarrow t_0, y_0$ and $s'_0, x'_0 \rightsquigarrow t'_0, y'_0$ and there is a sequence σ' that is blocking for $s_1, x_1 \rightsquigarrow t_1, y_1 \xrightarrow{a_2} \cdots s_k, x_k \rightsquigarrow t_k, y_k$ and not $s'_1, x'_1 \rightsquigarrow t'_1, y'_1 \xrightarrow{a'_2} \cdots s'_\ell, x'_\ell \rightsquigarrow t'_\ell, y'_\ell$.

Proof. The right-to-left direction is clear (just take $\sigma = (n : w), \sigma'$ 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 $(n : w)$ be such that $\sigma = (n : w)\sigma_+$.

- If $(n : w)$ is not blocking for $s_0, x_0 \rightsquigarrow t_0, y_0$ then σ_+ is blocking for π and not π', contradicting the minimality of σ.
- If $(n : w)$ is blocking for $s_0, x_0 \rightsquigarrow t_0, y_0$ and not $s'_0, x'_0 \rightsquigarrow t'_0, y'_0$ 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 $s_1, x_1 \rightsquigarrow t_1, y_1 \xrightarrow{a_2} \cdots s_k, x_k \rightsquigarrow t_k, y_k$.
- If $(n : w)$ is blocking for both $s_0, x_0 \rightsquigarrow t_0, y_0$ and $s'_0, x'_0 \rightsquigarrow t'_0, y'_0$ then we set $\sigma' = \sigma$. As σ is blocking for π, it is also blocking for $s_1, x_1 \rightsquigarrow t_1, y_1 \xrightarrow{a_2} \cdots s_k, x_k \rightsquigarrow t_k, y_k$. On the other hand, if σ was blocking for $s'_1, x'_1 \rightsquigarrow t'_1, y'_1 \xrightarrow{a'_2} \cdots s'_\ell, x'_\ell \rightsquigarrow t'_\ell, y'_\ell$, then it would also be blocking for π', a contradiction. Hence σ is not blocking for $s'_1, x'_1 \rightsquigarrow t'_1, y'_1 \xrightarrow{a'_2} \cdots s'_\ell, x'_\ell \rightsquigarrow t'_\ell, y'_\ell$

◁

The claim above lets us define a recursive algorithm.

- First check if there is a positional word $(n : w)$ that is blocking for $s_0, x_0 \rightsquigarrow t_0, y_0$ and not $s'_0, x'_0 \rightsquigarrow t'_0, y'_0$. If it is the case, make a recursive call to check if there is a sequence σ' that is blocking for $s_1, x_1 \rightsquigarrow t_1, y_1 \xrightarrow{a_2} \cdots s_k, x_k \rightsquigarrow t_k, y_k$ and not π'. If it is the case, answer yes.
- Then check if there is a positional word $(n : w)$ that is a blocking factor for $s_0, x_0 \rightsquigarrow t_0, y_0$ and $s'_0, x'_0 \rightsquigarrow t'_0, y'_0$. If so, make a recursive call to check if there is a sequence σ' that is blocking for $s_1, x_1 \rightsquigarrow t_1, y_1 \xrightarrow{a_2} \cdots s_k, x_k \rightsquigarrow t_k, y_k$ and not $s'_1, x'_1 \rightsquigarrow t'_1, y'_1 \xrightarrow{a'_2} \cdots s'_\ell, x'_\ell \rightsquigarrow t'_\ell, y'_\ell$. 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 F.4. 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 F.7.** *The hardness problem is in PSPACE.*

Proof. We use Lemma F.3. We guess an SCC-path $\pi = s_0, x_0 \rightsquigarrow t_0, y_0 \xrightarrow{a_1} \cdots s_k, x_k \rightsquigarrow t_k, y_k$ and an index i.

We check that $s_i, x_i \rightsquigarrow t_i, y_i$ has infinitely many minimal blocking factors, using Lemma F.4.

We then enumerate all SCC-paths in $\mathcal{A}$. For each one $\pi' = s'_0, x'_0 \rightsquigarrow t'_0, y'_0 \xrightarrow{a'_1} \cdots s'_\ell, x'_\ell \rightsquigarrow t'_\ell, y'_\ell$ we guess indices j^l and j^r. We check that every portal $s'_j, x'_j \rightsquigarrow t'_j, y'_j$ with $j^l < j < j^r$ is $\simeq$-equivalent to $s, x \rightsquigarrow t, y$.

Then, we use Lemma F.5 to check that there is a sequence σ^l that is blocking for $s'_0, x'_0 \rightsquigarrow t'_0, y'_0 \xrightarrow{a'_1} \cdots s'_{j^l}, x'_{j^l} \rightsquigarrow t'_{j^l}, y'_{j^l}$ and not $s_0, x_0 \rightsquigarrow t_0, y_0 \xrightarrow{a_1} \cdots s_i, x_i \rightsquigarrow t_i, y_i$.

Symmetrically, we check that there is a sequence σ^r that is blocking for $s'_{j^r}, x'_{j^r} \rightsquigarrow t'_{j^r}, y'_{j^r} \xrightarrow{a'_1} \cdots s'_\ell, x'_\ell \rightsquigarrow t'_\ell, y'_\ell$ and not $s_i, x_i \rightsquigarrow t_i, y_i \xrightarrow{a_{i+1}} \cdots s_k, x_k \rightsquigarrow t_k, y_k$.

If all those tests succeed, we answer yes, otherwise we answer no. This algorithm is correct and complete by Lemma F.3. ◀

F.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 5.3, and uses the following result.

▶ **Lemma F.8.** *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 F.4, except that we only need to check whether some final state is reachable from the initial state. ◀

▶ **Proposition F.9.** *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 F.10. $\mathcal{A}$ has a blocking sequence if and only if for all accepting SCC-path π of $\mathcal{A}$, every portal in π has a blocking factor.

Proof. If $\mathcal{A}$ has a blocking sequence then by definition it is blocking for all accepting SCC-paths, and thus contains blocking factors for all portals along those SCC-paths.

Suppose that every portal π of every accepting SCC-path has a blocking factor. Then by taking the sequence of those blocking factors we obtain a blocking sequence for π. Then, since there are finitely many accepting SCC-paths, we can concatenate all those sequences to obtain a sequence that is blocking for all accepting SCC-paths. By definition, that sequence is blocking for $\mathcal{A}$. ◁

Therefore, it suffices to enumerate all accepting SCC-paths π in the automaton, and then check whether all portals in π have at least one blocking factor, using Lemma F.8. ◀

F.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 F.11.** *The triviality and hardness problems are PSPACE-hard.*

Proof. Consider an NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$. 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.

As $\mathcal{A}$ is strongly connected, it is easy to check that its set of minimal blocking sequences is exactly its set of minimal blocking factors.

Therefore 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. 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 $\#$ from u.

▷ Claim F.12. If $\mathcal{A}$ is universal, then $\mathcal{B}$ is also universal.

Proof. Indeed, any word in u in $\Sigma \cup \{\#, !\}$ can be 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, and thus $u_i \in \mathcal{L}(\mathcal{A})$.

By the description of $\mathcal{L}(\mathcal{B})$ given above, we have $u \in \mathcal{B}$. ◁

This shows that if $\mathcal{A}$ is universal, then so is $\mathcal{B}$ and thus $\mathsf{MBF}(\mathcal{B})$ 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 both the triviality problem and the hardness problem. ◀

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

▶ **Corollary F.13.** *The easiness problem is PSPACE-hard.*

Proof. We proceed as in the proof of Lemma F.11: 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