Regular Languages in the Sliding Window Model

Received Feb 26, 2024
Revised Nov 11, 2024
Accepted Jan 19, 2025
Published Mar 6, 2025

Key words and phrases
Regular Languages, Streaming Algorithms, Sliding Window Model

Moses Ganardi[a]
Danny Hucke[b]
Markus Lohrey[b]
Konstantinos Mamouras[c]
Tatiana Starikovskaya[d]

a Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

b Universität Siegen, Germany

c Rice University, Houston, USA

d DI/ENS, PSL Research University, Paris, France

ABSTRACT. We study the space complexity of the following problem: For a fixed regular language L, we receive a stream of symbols and want to test membership of a sliding window of size n in L. For deterministic streaming algorithms we prove a trichotomy theorem, namely that the (optimal) space complexity is either constant, logarithmic or linear, measured in the window size n. Additionally, we provide natural language-theoretic characterizations of the space classes. We then extend the results to randomized streaming algorithms and we show that in this setting, the space complexity of any regular language is either constant, doubly logarithmic, logarithmic or linear. Finally, we introduce sliding window testers, which can distinguish whether a sliding window of size n belongs to the language L or has Hamming distance $> \epsilon n$ to L. We prove that every regular language has a deterministic (resp., randomized) sliding window tester that requires only logarithmic (resp., constant) space.

1. Introduction

1.1 The sliding window model

Streaming algorithms process a data stream $a_1a_2a_3\cdots$ of elements a_i from left to right and have at time t only direct access to the current element a_t. In many streaming applications, elements are outdated after a certain time, i.e., they are no longer relevant. The sliding window model is a simple way to model this. A sliding window algorithm computes for each time instant t a value that only depends on the relevant past (the so-called active window) of $a_1a_2\cdots a_t$. There are several formalizations of the relevant past. One way to do this is to fix a window size n.

https://theoretics.episciences.org
DOI 10.46298/theoretics.25.8

Then the active window consists at each time instant t of the last n elements $a_{t-n+1}a_{t-n+2}\cdots a_t$ (here we assume that a_i is a fixed padding symbol if $i \leqslant 0$). In the literature this is also called the fixed-size model. Another sliding window model that can be found in the literature is the variable-size model; see e.g. [7]. In this model, the arrival of new elements and the expiration of old elements can happen independently, which means that the window size can vary.[1] This allows to model for instance time-based windows, where data items arrive at irregular time instants and the active window contains all data items that arrive in the last n seconds for a fixed n. The special case of the variable-size model, where old symbols do not expire, is the classical streaming model.

A general goal in the area of sliding window algorithms is to avoid the explicit storage of the active window, which would require $\Omega(n)$ space for a window size n, and, instead, to work in considerably smaller space, e.g. polylogarithmic space with respect to the window size n. A detailed introduction into the sliding window model can be found in [1, Chapter 8].

The (fixed-size) sliding window model was introduced in the seminal paper of Datar et al. [27] where the authors considered the basic counting problem: Given a window size n and a stream of bits, maintain a count of the number of 1's in the window. One can easily observe that an exact solution would require $\Theta(n)$ bits. Intuitively, the reason is that the algorithm cannot see the bit which is about to expire (the n-th most recent bit) without storing it explicitly; this is in fact the main difficulty in most sliding window algorithms. However, Datar et al. show that with $O(\frac{1}{\epsilon} \cdot \log^2 n)$ bits one can maintain an approximate count up to a multiplicative factor of $1 \pm \epsilon$. In Section 1.3.6 below we briefly discuss further work on sliding window algorithms.

A foundational problem that has been surprisingly neglected so far is the language recognition problem over sliding windows: Given a language $L \subseteq \Sigma^*$ and a stream of symbols over a finite alphabet Σ, maintain a data structure which allows to query membership of the active window in L. In other words, we want to devise a streaming algorithm which, after every input symbol, either accepts if the current active window belongs to L, or rejects otherwise. This problem finds applications in complex event processing, where the goal is to detect patterns in data streams. These patterns are usually described in some language based on regular expressions; see e.g. [26, 89] for more details. For the standard streaming model, where input symbols do not expire, some work on language recognition problems has been done; see Section 1.3.7 below.

EXAMPLE 1.1. Consider the analysis of the price of a stock in order to identify short-term upward momentum. The original stream is a time series of stock prices, and it is pre-processed in the following way: over a sliding window of 5 seconds compute the linear regression of the prices and discretize the slope into the following values: P_2 (high positive), P_1 (low positive), Z (zero), N_1 (low negative), N_2 (high negative). This gives rise to a derived stream of symbols in

1 The reader can also think of a queue data structure, where letters can be added to the right and removed at the left, the latter without retrieving the identity of the letter.

the alphabet $\Sigma = \{\mathsf{P}_2, \mathsf{P}_1, \mathsf{Z}, \mathsf{N}_1, \mathsf{N}_2\}$, over which we describe the *upward trend* pattern: (i) no occurrence of N_2, (ii) at most two occurrences of N_1, (iii) and any two occurrences of Z or N_1 are separated by at least three positive symbols. The upward trend pattern can be described as the intersection of the language defined by the following regular expression e_{ii} and the complement of the language defined by e_i:

$$e_i = \Sigma^* \cdot \mathsf{N}_1 \cdot \Sigma^* \cdot \mathsf{N}_1 \cdot \Sigma^* \cdot \mathsf{N}_1 \cdot \Sigma^*$$
$$e_{ii} = (\mathsf{P}_2 + \mathsf{P}_1)^* \cdot \big((\mathsf{Z} + \mathsf{N}_1) \cdot (\mathsf{P}_2 + \mathsf{P}_1)^{[3,\infty)}\big)^* \cdot (\varepsilon + \mathsf{Z} + \mathsf{N}_1) \cdot (\mathsf{P}_2 + \mathsf{P}_1)^*$$

We want to monitor continuously whether the window of the last hour matches the upward trend pattern, as this is an indicator to buy the stock and ride the upward momentum. Our results will show that there is a space efficient streaming algorithm for this problem, which can be synthesized from the regular expressions e_i and e_{ii}. ◆

In this paper we focus on querying regular languages over sliding windows. Unfortunately, there are simple regular languages which require $\Omega(n)$ space in the sliding window model, i.e., one cannot avoid maintaining the entire window explicitly. However, for certain regular languages, such as for the upward trend pattern from above, we present sublinear space streaming algorithms. Before we explain our results in more detail, let us give examples of sliding window algorithms for simple regular languages.

EXAMPLE 1.2. Let $\Sigma = \{a, b\}$ be the alphabet. In the following examples we refer to the fixed-size sliding window model.

(i) Let $L = \Sigma^* a$ be the set of all words ending with a. A streaming algorithm can maintain the most recent symbol of the stream in a single bit, which is also the most recent symbol of the active window. Hence, the space complexity of L is $O(1)$ in the sliding window model.

(ii) Let $L = \Sigma^* a \Sigma^*$ be the set of all words containing a. If $n \in \mathbb{N}$ is the window size, then a streaming algorithm can maintain the position $1 \leqslant i \leqslant n$ (from right to left) of the most recent a-symbol in the active window or set $i = \infty$ if the active window contains no a-symbols. Let us assume that the initial window is b^n and initialize $i := \infty$. On input a we set $i := 1$ and on input b we increment i and then set $i := \infty$ if $i > n$. The algorithm accepts if and only if $i \leqslant n$. Since the position i can be stored using $O(\log n)$ bits, we have shown that L has space complexity $O(\log n)$ in the sliding window model.

(iii) Let $L = a\Sigma^*$ be the set of all words starting with a. We claim that any (deterministic) sliding window algorithm for L and window size $n \in \mathbb{N}$ (let us call it $\mathcal{P}_n$) must store at least n bits, which matches the complexity of the trivial solution where the window is stored explicitly. More precisely, the claim is that $\mathcal{P}_n$ reaches two distinct memory states on any two distinct words $x = a_1 \cdots a_n \in \Sigma^n$ and $y = b_1 \cdots b_n \in \Sigma^n$. Suppose that $a_i \neq b_i$. Then we simulate $\mathcal{P}_n$ on the streams xb^{i-1} and yb^{i-1}, respectively. The suffixes (or windows) of length n for the two streams are $a_i \cdots a_n b^{i-1}$ and $b_i \cdots b_n b^{i-1}$. Since exactly one of the two windows belongs to L, the algorithm $\mathcal{P}_n$ must accept exactly one of the streams xb^{i-1} and yb^{i-1}. In particular, $\mathcal{P}_n$ must reach two distinct memory states on the words xb^{i-1} and yb^{i-1}, and therefore $\mathcal{P}_n$ must have also reached two distinct memory states on the prefixes x and y, as claimed above. Therefore, $\mathcal{P}_n$ must have at least $|\Sigma^n| = 2^n$ memory states, which require n bits of memory. ◆

1.2 Results

Let us now present the main results of this paper. The precise definitions of all used notions can be found in the main part of the paper. We denote by $\mathsf{F}_L(n)$ (resp., $\mathsf{V}_L(n)$) the space complexity (measured in bits) of an optimal sliding window algorithm for the language L in the fixed-size (resp., variable-size) sliding window model. Here, n denotes the fixed window size for the fixed-size model, whereas for the variable-size model n denotes that maximal window size among all time instants when reading an input stream.

Our first result is a trichotomy theorem for the sliding window model, stating that the deterministic space complexity is always either constant, logarithmic, or linear. This holds for both the fixed- and the variable-size model. Furthermore, we provide natural characterizations for the three space classes. For this, we need the following language classes:

- **Reg** is the class of all regular languages.
- **Len** is the class of all regular length languages, i.e., regular languages $L \subseteq \Sigma^*$ such that for every $n \geqslant 0$, either $\Sigma^n \subseteq L$ or $\Sigma^n \cap L = \emptyset$.
- **ST** is the class of all suffix testable languages [79, Section 5.3], i.e., finite Boolean combinations of languages of the form $\Sigma^* w$ where $w \in \Sigma^*$ (note that these languages are regular).[2]
- **LI** is the class of all regular left ideals, i.e., languages of the form $\Sigma^* L$ for $L \subseteq \Sigma^*$ regular.

We emphasize that the three defined language properties only make sense with respect to an underlying alphabet. If $\mathbf{L}_1, \ldots, \mathbf{L}_n$ are classes of languages over some alphabet Σ, then $\langle \mathbf{L}_1, \ldots, \mathbf{L}_n \rangle$ denotes the *Boolean closure* of the classes $\mathbf{L}_1, \ldots, \mathbf{L}_n$, which is the class of all finite Boolean combinations of languages $L \in \bigcup_{i=1}^n \mathbf{L}_i$. We also use the following asymptotic notation in our results: For functions $f, g \colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$, $f(n) = \Omega^\infty(g(n))$ holds if $f(n) \geqslant c \cdot g(n)$ for some $c > 0$ and infinitely many $n \in \mathbb{N}$. Furthermore, $f(n) = \Theta^\infty(g(n))$ holds if $f(n) = O(g(n))$ and $f(n) = \Omega^\infty(g(n))$. Now we can state our first main result.

2 For the results presented in this section, one could equivalently define **ST** as the class of all languages $\Sigma^* w$ without taking the Boolean closure.

THEOREM 1.3. *Let $L \subseteq \Sigma^*$ be regular. The space complexity $\mathsf{F}_L(n)$ is either $\Theta(1)$, $\Theta^\infty(\log n)$, or $\Theta^\infty(n)$. Moreover, we have:*

$$\begin{aligned}
\mathsf{F}_L(n) &= \Theta(1) &&\iff L \in \langle \mathbf{ST}, \mathbf{Len}\rangle\\
\mathsf{F}_L(n) &= \Theta^\infty(\log n) &&\iff L \in \langle \mathbf{LI}, \mathbf{Len}\rangle \setminus \langle \mathbf{ST}, \mathbf{Len}\rangle\\
\mathsf{F}_L(n) &= \Theta^\infty(n) &&\iff L \in \mathbf{Reg} \setminus \langle \mathbf{LI}, \mathbf{Len}\rangle
\end{aligned}$$

The space complexity $\mathsf{V}_L(n)$ is either $\Theta(1)$, $\Theta(\log n)$, or $\Theta(n)$. Moreover, we have:

$$\begin{aligned}
\mathsf{V}_L(n) &= \Theta(1) &&\iff L \in \{\emptyset, \Sigma^*\}\\
\mathsf{V}_L(n) &= \Theta(\log n) &&\iff L \in \langle \mathbf{LI}, \mathbf{Len}\rangle \setminus \{\emptyset, \Sigma^*\}\\
\mathsf{V}_L(n) &= \Theta(n) &&\iff L \in \mathbf{Reg} \setminus \langle \mathbf{LI}, \mathbf{Len}\rangle
\end{aligned}$$

Theorem 1.3 describes which regular patterns can be queried over sliding windows in sublinear space: Regular left ideals over a sliding window express statements of the form "recently in the stream some regular event happened". Dually, complements of left ideals over a sliding window express statements of the form "at all recent times in the stream some regular event happened".

Most papers on streaming algorithms make use of randomness. For many problems, randomized streaming algorithms are more space efficient than deterministic streaming algorithms; see e.g. [3] and the remarks at the beginning of Section 4. So, it is natural to consider randomized sliding window algorithms for regular languages. Our randomized sliding window algorithms have a two-sided or one-sided error of $1/3$ (any constant error probability below $1/2$ would yield the same results). For a one-sided error we obtain exactly the same space trichotomy for regular languages as for deterministic algorithms (Theorem 4.18). This changes if we allow a two-sided error. With $\mathsf{F}^{\mathsf{r}}_L(n)$ we denote the optimal space complexity of a randomized sliding window algorithm for L in the fixed size model and with two-sided error. Our second main result says that the functions $\mathsf{F}^{\mathsf{r}}_L(n)$ for L regular fall into four randomized space complexity classes: constant, doubly logarithmic, logarithmic, and linear space. A language L is *suffix-free* if $xy \in L$ and $x \neq \varepsilon$ implies $y \notin L$. We denote by **SF** the class of all regular suffix-free languages.

THEOREM 1.4. *Let $L \subseteq \Sigma^*$ be regular. The randomized space complexity $\mathsf{F}^{\mathsf{r}}_L(n)$ of L in the fixed-size sliding window model is either $\Theta(1)$, $\Theta^\infty(\log\log n)$, $\Theta^\infty(\log n)$, or $\Theta^\infty(n)$. Furthermore:*

$$\begin{aligned}
\mathsf{F}^{\mathsf{r}}_L(n) &= \Theta(1) &&\iff L \in \langle \mathbf{ST}, \mathbf{Len}\rangle\\
\mathsf{F}^{\mathsf{r}}_L(n) &= \Theta^\infty(\log\log n) &&\iff L \in \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len}\rangle \setminus \langle \mathbf{ST}, \mathbf{Len}\rangle\\
\mathsf{F}^{\mathsf{r}}_L(n) &= \Theta^\infty(\log n) &&\iff L \in \langle \mathbf{LI}, \mathbf{Len}\rangle \setminus \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len}\rangle\\
\mathsf{F}^{\mathsf{r}}_L(n) &= \Theta^\infty(n) &&\iff L \in \mathbf{Reg} \setminus \langle \mathbf{LI}, \mathbf{Len}\rangle
\end{aligned}$$

Figure 1 compares the deterministic and the randomized space complexity in the fixed-size model (we only show the upper bounds in order to not overload the figure). We also consider randomized algorithms in the variable-size model. In this setting we obtain again the same space trichotomy for regular languages as for the deterministic case; see Lemma 4.21.

By Theorems 1.3 and 1.4, some (simple) regular languages (e.g. $a\{a,b\}^*$) do not admit sublinear space (randomized) algorithms. This gives the motivation to seek for alternative approaches in order to achieve efficient algorithms for all regular languages. We take our inspiration from the property testing model introduced by Goldreich et al. [61]. In this model, the task is to decide (with high probability) whether the input has a particular property P, or is "far" from any input satisfying P, while querying as few symbols of the input as possible. Alon et al. prove that every regular language has a property tester making only $O(1)$ many queries [2]. The idea of property testing was also combined with the streaming model, yielding streaming property testers, where the objective is not to minimize the number of queries but the required memory [34, 38]. We define sliding window testers, which, using as little space as possible, must accept if the window (of size n) belongs to the language L and must reject if the window has Hamming distance at least $\gamma(n)$ from every word in L. Here $\gamma(n) \leqslant n$ is a function that is called the Hamming gap of the sliding window testers. We focus on the fixed-size model.

Two of our main results concerning sliding window testers that we show in Section 5 are the following:

THEOREM 1.5. *Let $L \subseteq \Sigma^*$ be regular.*

(i) There exists a deterministic sliding window tester for L with constant Hamming gap that uses space $O(\log n)$.

(ii) For every $\epsilon > 0$ there exists a randomized sliding window tester for L with two-sided error and Hamming gap ϵn that uses space $O(1/\epsilon)$.

Section 5 contains additional results that give a rather precise tradeoff between space complexity and the Hamming gap function $\gamma(n)$. In addition we also study sliding window testers with a one-sided error and prove optimality for most of our results by providing matching lower bounds. See Section 5.2 for a complete discussion of our results for sliding window testers.

1.3 Related work

This paper builds on four conference papers [43, 45, 46, 47]. To keep this paper coherent, we decided to omit some of the results from [43, 45, 46, 47]. In this section, we briefly discuss these results as well as other related work.

1.3.1 Uniform setting

In all our results we assume a fixed regular language L. The space complexity is only measured with respect to the window size. It is a natural question to ask how the space bounds depend on the size of a finite automaton (deterministic or nondeterministic) for L. This question is

$$\begin{tikzcd}
\text{space complexity} & \text{deterministic} \\ & \text{one-sided error} & \text{two-sided error} \\
\mathcal{O}(n) & \mathbf{Reg} \arrow[d, dash] & \mathbf{Reg} \arrow[d, dash] \\
\mathcal{O}(\log n) & \langle \mathbf{LI}, \mathbf{Len} \rangle \arrow[dd, dash] & \langle \mathbf{LI}, \mathbf{Len} \rangle \arrow[d, dash] \\
\mathcal{O}(\log\log n) & & \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle \arrow[d, dash] \\
\mathcal{O}(1) & \langle \mathbf{ST}, \mathbf{Len} \rangle & \langle \mathbf{ST}, \mathbf{Len} \rangle
\end{tikzcd}$$

Figure 1. The space complexity of regular languages in the fixed-size sliding window model. **Reg**: regular languages, **LI**: regular left ideals, **ST**: suffix testable languages, **SF**: regular suffix-free languages, **Len**: regular length languages. The angle brackets $\langle \cdot \rangle$ denote Boolean closure.

considered in [43]. It is shown that, if $\mathcal{A}$ is a DFA (resp., NFA) with m states for a language $L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$, then $\mathsf{V}_L(n) = \mathcal{O}(2^m \cdot m \cdot \log n)$ (resp., $\mathsf{V}_L(n) = \mathcal{O}(4^m \cdot \log n)$). Furthermore, for every $k \geqslant 1$ there exists a language $L_k \subseteq \{0, \ldots, k\}^*$ recognized by a deterministic automaton with $k+3$ states such that $L_k \in \langle \mathbf{LI}, \mathbf{Len} \rangle$ and $F_{L_k}(n) \geqslant (2^k - 1) \cdot (\log n - k)$. A binary encoding of the words in L_k yields a subexponential lower bound over a fixed alphabet [41, Theorem 4.45]. Further results on the uniform space complexity for languages in $\langle \mathbf{LI}, \mathbf{Len} \rangle$ as well as in $\langle \mathbf{ST}, \mathbf{Len} \rangle$ can be found in [41, Section 4.3].

1.3.2 Membership in the space classes

In view of Theorem 1.3 it is natural to ask for the complexity of checking whether a given (non)deterministic finite automaton accepts a language from $\langle \mathbf{ST}, \mathbf{Len} \rangle$ or $\langle \mathbf{LI}, \mathbf{Len} \rangle$, respectively. Both problems are shown in [43] to be NL-complete for deterministic automata and PSPACE-complete for nondeterministic automata. We remark that for the class $\langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$ from Theorem 1.4 the complexities of the corresponding membership problems are open.

1.3.3 Other models of randomness

In Section 1.2 we did not specify the underlying model of randomized sliding window algorithms (that is used in Theorem 1.4) in a precise way. Let us be a bit more specific: we require that a randomized sliding window algorithm for a language L running on an input stream s outputs at every time instant a correct answer on the question whether the current window belongs to L or not with high probability (say at least $2/3$). This is not the only model of randomized sliding window algorithms that can be found in the literature. A stronger model requires that with high probability the randomized sliding window algorithm outputs at every time instant a correct answer. So the difference is between "$\forall$ time instants: Pr[answer correct] $\geqslant 2/3$" and "Pr[$\forall$ time instants: answer correct] $\geqslant 2/3$". A randomized sliding window algorithm that fulfills the latter (stronger) correctness criterion is called strictly correct in [44]. This model is for instance implicitly used in [13, 27]. In [44] it is shown that every strictly correct randomized sliding window algorithm can be derandomized without increasing the space complexity. This result is shown in a very general context for arbitrary approximation problems. The proof in [44] needs input streams of length doubly exponential in the window size for the derandomization. In contrast, if one restricts to input streams of length polynomial in the window size then strictly correct randomized sliding window algorithms can be more space efficient than ordinary randomized sliding window algorithms (as defined in this paper) [44]. The intermediate case of exponentially long input streams is open.

Finally, we emphasize that our randomized sliding window algorithms are not necessarily *adversarially robust*, i.e., an adversary may fool the algorithm by observing the internal memory state and picking the input symbols adaptively.

1.3.4 Context-free languages

It is natural to ask to which extent our results hold for context-free languages. This question is considered in [49, 42]. Let us briefly discuss the results. In [49] it is shown that if L is a context-free language with $F_L(n) \leqslant \log n - \omega(1)$ then L must be regular and $F_L(n) = \Theta(1)$. Hence, the gap between constant space and logarithmic space for regular languages also exists for context-free languages. In contrast, the gap between logarithmic space and linear space for regular languages does not extend to all context-free languages. In [49], the authors construct examples of context-free languages L with $F_L(n) = \Theta^\infty(n^{1/c})$ and $V_L(n) = \Theta(n^{1/c})$ for every natural number $c \geqslant 2$. These languages are not deterministic context-free, but [49] also contains examples of deterministic one-turn one-counter languages L and L' with $F_L(n) = \Theta^\infty(\log^2 n)$ and $V_{L'}(n) = \Theta(\log^2 n)$. In [42], the author studies the space complexity of visibly pushdown languages (a language class strictly in-between the regular and deterministic context-free languages with good closure and decidability properties [5]). It is shown that for every visibly pushdown language the space complexity in the variable-size sliding window model is either constant, logarithmic or linear in the window size. Hence, the space trichotomy that we have seen for regular languages also holds for visibly pushdown languages in the variable-size model. Whether the visibly pushdown languages also exhibit the space trichotomy in the fixed-size model is open.

1.3.5 Update times

In this paper, we only considered the space complexity of sliding window algorithms. Another important complexity measure is the update time of a sliding window algorithm, i.e., the worst case time that is spent per incoming symbol for updating the internal data structures. In [88], it is shown that for every regular language L there exists a deterministic sliding window algorithm (for the fixed-size model) with constant update time. The underlying machine model of the sliding window algorithm is the RAM model, where basic arithmetic operations on registers of bit length $O(\log n)$ (with n the window size) need constant time. In fact, the algorithm in [88] is formulated in a more general context for any associative aggregation function. The case of a regular language L is obtained by applying the algorithm from [88] for the syntactic monoid of L. In [48] the result of [88] is extended to visibly pushdown languages.

1.3.6 Further work on sliding windows

We have already mentioned the seminal work of Datar et al. on the sliding window model [27], where the authors considered the problem of estimating the number of ones in the sliding window. In the same paper, Datar et al. extend their result for the basic counting problem to arbitrary functions which satisfy certain additivity properties, e.g. L_p-norms for $p \in [1, 2]$. Braverman and Ostrovsky introduced the smooth histogram framework [17], to compute so-called smooth functions over sliding windows, which include all L_p-norms and frequency moments. Further work on computing aggregates, statistics and frequent elements in the sliding window model can be found in [7, 9, 12, 13, 16, 28, 35, 55, 56]. The problem of sampling over sliding windows was first studied in [8] and later improved in [18]. As an alternative to sliding windows, Cohen and Strauss consider the problem of maintaining stream aggregates where the data items are weighted by a decay function [25].

1.3.7 Language recognition in the classical streaming model

Whereas language recognition in the sliding window model has been neglected prior to our work, there exists some work on streaming algorithms for formal languages in the standard setting, where the streaming algorithm reads an input word w and at the end has to decide whether w belongs to some language. Clearly, for regular languages, this problem can be solved in constant space. Streaming algorithms for various subclasses of context-free languages have been studied in [10, 38, 63, 69, 72]. Related to this is the work on querying XML documents in the streaming model [11, 67, 85].

1.3.8 Streaming pattern matching

Related to our work is the problem of streaming pattern matching, where the goal is to find all occurrences of a pattern (possibly with some bounded number of mismatches) in a data stream; see e.g. [66, 82, 57, 54, 19, 21, 22, 23, 24, 58, 60, 59, 80, 86] and search of repetitions in streams [33, 32, 31, 53, 52, 73, 74].

1.3.9 Dynamic membership problems for regular languages

A sliding window algorithm can be viewed as a dynamic data structure that maintains a dynamic string w (the window content) under very restricted update operations. Dynamic membership problems for more general updates that allow to change the symbol at an arbitrary position have been studied in [6, 39, 40]. As in our work, a trichotomy for the dynamic membership problem of regular languages has been obtained in [6] (but the classes appearing the trichotomy in [6] are different from the classes that appear in our work).

1.4 Outline

The outline of the paper is as follows: In Section 2 we give preliminary definitions and introduce the fixed-size sliding window model and the variable-size sliding window model. In Section 3 we study deterministic sliding window algorithms for regular languages and prove the space trichotomy and the characterizations of the space classes (Theorem 1.3). In Section 4 we turn to randomized sliding window algorithms and prove the space tetrachotomy (Theorem 1.4). Finally, in Section 5 we present deterministic and randomized sliding window property testers for regular languages (Theorem 1.5).

2. Preliminaries

2.1 Words and languages

An *alphabet* Σ is a nonempty finite set of *symbols*. A *word* over an alphabet Σ is a finite sequence $w = a_1 a_2 \cdots a_n$ of symbols $a_1, \dots, a_n \in \Sigma$. The *length* of w is the number $|w| = n$. The *empty word* is denoted by ε whereas the lunate epsilon ϵ denotes small positive numbers. The *concatenation* of two words u, v is denoted by $u \cdot v$ or uv. The set of all words over Σ is denoted by Σ^*. A subset $L \subseteq \Sigma^*$ is called a *language* over Σ.

Let $w = a_1 \cdots a_n \in \Sigma^*$ be a word. Any word of the form $a_1 \cdots a_i$ is a *prefix* of w, a word of the form $a_i \cdots a_n$ is a *suffix* of w, and a word of the form $a_i \cdots a_j$ is a *factor* of w. The concatenation of two languages K, L is $KL = \{uv \mid u \in K,\ v \in L\}$. For a language L we define L^n inductively by $L^0 = \{\varepsilon\}$ and $L^{n+1} = L^n L$ for all $n \in \mathbb{N}$. The *Kleene-star* of a language L is the language $L^* = \bigcup_{n \in \mathbb{N}} L^n$. Furthermore, we define $L^{\leqslant n} = \bigcup_{0 \leqslant k \leqslant n} L^k$ and $L^{<n} = \bigcup_{0 \leqslant k < n} L^k$.

Let $L \subseteq \Sigma^*$ be a language. We say that L *separates* two words $x, y \in \Sigma^*$ with $x \neq y$ if $|\{x, y\} \cap L| = 1$. We say that L *separates* two languages $K_1, K_2 \subseteq \Sigma^*$ if $K_1 \subseteq L$ and $K_2 \cap L = \emptyset$, or $K_2 \subseteq L$ and $K_1 \cap L = \emptyset$.

2.2 Automata and regular languages

For good introductions to the theory of formal languages and automata we refer to [15, 62, 68].

The standard description for regular languages are finite automata. Let Σ be a finite alphabet. A *nondeterministic finite automaton (NFA)* is a tuple

$$\mathcal{A} = (Q, \Sigma, I, \Delta, F),$$

where Q is the finite set of *states*, $I \subseteq Q$ is the set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is the set of *transitions*, and $F \subseteq Q$ is the set of final states. A *run* of $\mathcal{A}$ on a word $w = a_1 \cdots a_n \in \Sigma^*$ is a finite sequence $\pi = q_0 a_1 q_1 a_2 q_2 \cdots q_{n-1} a_n q_n \in Q(\Sigma Q)^*$ such that $(q_{i-1}, a_i, q_i) \in \Delta$ for all $1 \leqslant i \leqslant n$. We call π *successful* if $q_0 \in I$ and $q_n \in F$. The language *accepted* by $\mathcal{A}$ is defined as

$$\mathsf{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \text{there exists a successful run of } \mathcal{A} \text{ on } w\}.$$

A language $L \subseteq \Sigma^*$ is *regular* if it is accepted by some NFA. The *size* $|\mathcal{A}|$ is defined as the number of states.

A *(left-)deterministic finite automaton (DFA)* is an NFA $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$, where $I = \{q_0\}$ has exactly one initial state q_0, and for all $p \in Q$ and $a \in \Sigma$ there exists exactly one transition $(p, a, q) \in \Delta$. We view Δ as a *transition function* $\delta \colon Q \times \Sigma \to Q$ and write $\mathcal{A}$ in the format $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$. The transition function δ can be extended to a right action $\cdot \colon Q \times \Sigma^* \to Q$ of the free monoid Σ^* on the state set Q by setting $q \cdot \varepsilon = q$ and defining inductively $q \cdot ua = \delta(q \cdot u, a)$ for all $q \in Q$, $u \in \Sigma^*$, and $a \in \Sigma$. We write $\mathcal{A}(w)$ instead of $q_0 \cdot w$. It is known that any NFA can be turned into an equivalent DFA by the power set construction.

We also consider automata with (possibly) infinitely many states as our formal model for streaming algorithms. A *deterministic automaton* $\mathcal{A}$ has the same format $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ as a DFA but we drop the condition that Q must be finite. We use the notations from the previous paragraph for general deterministic automata as well.

It is well-known that for every regular language L there exists a minimal DFA $\mathcal{A}_L$ for L, which is unique up to isomorphism and whose states are the Myhill-Nerode classes of L. This construction can be carried out for every language L and yields a deterministic automaton $\mathcal{A}_L$ for L such that for every deterministic automaton $\mathcal{B}$ for L, we have that $\mathcal{B}(x) = \mathcal{B}(y)$ implies $\mathcal{A}_L(x) = \mathcal{A}_L(y)$ for all $x, y \in \Sigma^*$ [30, Chapter III, Theorem 5.2]. We call this automaton the *minimal deterministic automaton for L*.

2.3 Streaming algorithms

A *stream* is a finite sequence of elements $a_1 \cdots a_m$, which arrive element by element from left to right. So, it is just a finite word over some alphabet. In this paper, the elements a_i are always symbols from a finite alphabet Σ. A streaming algorithm reads the symbols of the input stream from left to right. At time instant t the algorithm only has access to the symbol a_t and the internal storage, which is encoded by a bit string. The goal of the streaming algorithm is to compute a function $\varphi \colon \Sigma^* \to Y$, where Σ is a finite alphabet and Y is a set of output values. For the remainder of this paper, we only consider the Boolean case, i.e., $Y = \{0, 1\}$; in other words, φ is the characteristic function of a language L. Furthermore, we abstract away the actual computation and only analyze the memory requirement.

Formally, a *deterministic streaming algorithm* is the same as a deterministic automaton $\mathcal{P}$ and we say that $\mathcal{P}$ is a streaming algorithm for the language $\mathsf{L}(\mathcal{P})$. The letter $\mathcal{P}$ stands for *program*. If $\mathcal{P} = (M, \Sigma, m_0, \delta, F)$ then the states from M are usually called *memory states*. We require $M \neq \emptyset$ but allow M to be infinite. The *space* of $\mathcal{P}$ (or *number of bits used by* $\mathcal{P}$) is given by $s(\mathcal{P}) = \log |M| \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$. Here and in the rest of the paper, we denote with log the logarithm with base two, i.e., we measure space in bits. If $s(\mathcal{P}) = \infty$ we will measure the space restricted to input streams where some parameter is bounded (namely the window size); see Section 2.5.

We remark that many streaming algorithms in the literature only produce a single answer after completely reading the entire stream. Also, the length of the stream is often known in advance. However, in the sliding window model we rather assume an input stream of unbounded and unknown length, and need to compute output values for *every* window, i.e., at every time instant.

In the following, we introduce the sliding window model in two different variants: the fixed-size sliding window model and the variable-size sliding window model.

2.4 Fixed-size sliding window model

We fix an arbitrary padding symbol $\square \in \Sigma$. Given a stream $x = a_1 a_2 \cdots a_m \in \Sigma^*$ and a *window size* $n \in \mathbb{N}$, we define $\mathsf{last}_n(x) \in \Sigma^n$ by

$$\mathsf{last}_n(x) = \begin{cases} a_{m-n+1} a_{m-n+2} \cdots a_m, & \text{if } n \leqslant m, \\ \square^{n-m} a_1 \cdots a_m, & \text{if } n > m, \end{cases}$$

which is called the *window of size* n, or the *active* or *current window*. In other words, $\mathsf{last}_n(x)$ is the suffix of length n, padded with $\square$-symbols on the left. We view $\square^n$ as the *initial window*; its choice is completely arbitrary.

Let $L \subseteq \Sigma^*$ be a language. The *sliding window problem* $\mathsf{SW}_n(L)$ for L and *window size* $n \in \mathbb{N}$ is the language

$$\mathsf{SW}_n(L) = \{x \in \Sigma^* \mid \mathsf{last}_n(x) \in L\}.$$

Note that for every L and every n, $\mathsf{SW}_n(L)$ is regular. A *sliding window algorithm (SW-algorithm)* for L and window size $n \in \mathbb{N}$ is a streaming algorithm for $\mathsf{SW}_n(L)$. The function $\mathsf{F}_L \colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ is defined by

$$\mathsf{F}_L(n) = \inf\{s(\mathcal{P}_n) \mid \mathcal{P}_n \text{ is an SW-algorithm for } L \text{ and window size } n\}. \tag{1}$$

It is called the *space complexity* of L *in the fixed-size sliding window model*. Note that $\mathsf{F}_L(n) < \infty$ since $\mathsf{SW}_n(L)$ is regular. A subtle point is that the space complexity $\mathsf{F}_L(n)$ of a language L in general depends on the underlying alphabet. A simple example is $L = a^*$ which has complexity

$\mathsf{F}_L(n) = O(1)$ over the singleton alphabet $\{a\}$ whereas it has complexity $\mathsf{F}_L(n) = \Theta(\log n)$ over the alphabet $\{a, b\}$ (the latter follows from our results). Here, it is also important that the padding symbol $\Box$ belongs to the alphabet Σ over which the language L is defined. An alternative definition would be to take a fresh padding symbol $\Box \notin \Sigma$ and define $\mathsf{last}_n(x)$ and $\mathsf{SW}_n(L)$ as above. For instance, for $L = a^*$ we would obtain $\mathsf{SW}_n(L) = \{a^i \mid i \geqslant n\}$, whose minimal DFA has $n + 1$ states. Thus, the space complexity would be $\Omega(\log n)$ instead of $O(1)$. Note that these differences only concern the space complexity during the first n steps (until the window is filled up). Sliding window algorithms are usually used for streams that are much longer than the window size. So it might be acceptable, if during a short initial phase the space complexity is higher than for the rest of the stream.

We draw similarities to circuit complexity, where a language $L \subseteq \{0, 1\}^*$ is recognized by a family of circuits $(C_n)_{n\in\mathbb{N}}$ in the sense that C_n recognizes the slice $L \cap \{0, 1\}^n$. Similarly, the sliding window problem $\mathsf{SW}_n(L)$ is solely defined by the slice $L \cap \Sigma^n$. If we speak of an SW-algorithm for L and omit the window size n, then this parameter is implicitly universally quantified, meaning that there exists a family of streaming algorithms $(\mathcal{P}_n)_{n\in\mathbb{N}}$ such that every $\mathcal{P}_n$ is an SW-algorithm for L and window size n.

LEMMA 2.1. *For any language L we have $\mathsf{F}_L(n) = O(n)$.*

PROOF. A trivial SW-algorithm $\mathcal{P}_n$ for L explicitly stores the active window of size n in a queue so that the algorithm can always test whether the window belongs to L. Formally, the state set of $\mathcal{P}_n$ is Σ^n and it has transitions of the form (bu, a, ua) for $a, b \in \Sigma$, $u \in \Sigma^{n-1}$. Viewed as an edge-labeled graph this automaton is also known under the name *de Bruijn graph* [20]. Since every word $w \in \Sigma^n$ can be encoded with $O(\log |\Sigma| \cdot n)$ bits and $|\Sigma|$ is a constant, the algorithm uses $O(n)$ bits. ■

Depending on the language L there are more space efficient solutions. Usually, sliding window algorithms are devised in the following way:

— Specify some information or property $I(w)$ of the active window w and show that it can be *maintained* by a streaming algorithm. This means that given $I(bu)$ and $a \in \Sigma$ one can compute $I(ua)$.
— Show that one can decide $w \in L$ from the information $I(w)$.

Notice that the complexity function $\mathsf{F}_L(n)$ is not necessarily monotonic. For instance, let L be the intersection of $a\Sigma^*$ and the set of words with even length. By Example 1.2(iii), we have $\mathsf{F}_L(2n) = \Theta(n)$ but clearly we have $\mathsf{F}_L(2n + 1) = O(1)$ since for odd window sizes the algorithm can always reject. Therefore, we can only show $\mathsf{F}_L(n) = \Theta^\infty(n)$ (instead of $\mathsf{F}_L(n) = \Theta(n)$ which is false here), where $\Theta^\infty(g(n))$ was defined in the introduction.

Note that the fixed-size sliding window model is a *nonuniform* model: for every window size we have a separate streaming algorithm and these algorithms do not have to follow a common pattern. Working with a nonuniform model makes lower bounds stronger. In contrast, the variable-size sliding window model that we discuss next is a uniform model in the sense that there is a single streaming algorithm that works for every window size. Let us remark that all presented upper bounds for the fixed-size model will be realized by uniform families of algorithms.

2.5 Variable-size sliding window model

For an alphabet Σ we define the extended alphabet $\Sigma_\downarrow = \Sigma \cup \{\downarrow\}$. In the variable-size model the *active window* $\mathsf{wnd}(u) \in \Sigma^*$ for a stream $u \in \Sigma_\downarrow^*$ is defined as follows, where $a \in \Sigma$:

$$\begin{aligned} \mathsf{wnd}(\varepsilon) &= \varepsilon & \qquad \mathsf{wnd}(u{\downarrow}) &= \varepsilon, \text{ if } \mathsf{wnd}(u) = \varepsilon \\ \mathsf{wnd}(ua) &= \mathsf{wnd}(u)a & \qquad \mathsf{wnd}(u{\downarrow}) &= v, \text{ if } \mathsf{wnd}(u) = av \end{aligned}$$

The symbol $\downarrow$ represents the *pop operation*. We emphasize that a pop operation on an empty window leaves the window empty. The *variable-size sliding window problem* $\mathsf{SW}(L)$ of a language $L \subseteq \Sigma^*$ is the language

$$\mathsf{SW}(L) = \{u \in \Sigma_\downarrow^* \mid \mathsf{wnd}(u) \in L\}. \tag{2}$$

Note that in general, $\mathsf{SW}(L)$ is not a regular language (even if L is regular). A *variable-size sliding window algorithm (variable-size SW-algorithm)* $\mathcal{P}$ for L is a streaming algorithm for $\mathsf{SW}(L)$.

There are various possible definitions for the space complexity of a variable-size SW-algorithm. Here, we measure the space complexity as a function in the *maximum* window size over all read prefixes. This definition enjoys the property that every language L has a variable-size SW-algorithm with *smallest* complexity among all variable-size SW-algorithms for L. If one would measure the space complexity in the *current* window size instead, this does not hold anymore, since the memory state encodings of any SW-algorithm can be permuted to yield an algorithm whose complexity is incomparable to the original one.

To be more formal, for a stream $u = a_1 \cdots a_m \in \Sigma_\downarrow^*$ let

$$\mathsf{mwl}(u) = \max\{|\mathsf{wnd}(a_1 \cdots a_i)| \mid 0 \leqslant i \leqslant m\}$$

be the *maximum window size* of all prefixes of u. If $\mathcal{P} = (M, \Sigma, m_0, \delta, F)$ is a streaming algorithm over $\Sigma_\downarrow$ we define

$$M_n = \{\mathcal{P}(w) \mid w \in \Sigma_\downarrow^*, \mathsf{mwl}(w) = n\}. \tag{3}$$

and $M_{\leqslant n} = \bigcup_{0 \leqslant k \leqslant n} M_k$. The *space complexity* of $\mathcal{P}$ in the variable-size sliding window model is

$$v(\mathcal{P}, n) = \log |M_{\leqslant n}| \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}.$$

In other words: when we say that the space complexity of a variable-size SW-algorithm is bounded by $f(n)$, we mean that the algorithm never has to store more than $f(n)$ bits when it processes a stream $u \in \Sigma_{\downarrow}^*$ such that for every prefix of u the size of the active window never exceeds n.

Notice that $v(\mathcal{P}, n)$ is a monotonic function. To prove upper bounds above $\log n$ for the space complexity of $\mathcal{P}$ it suffices to bound $\log |M_n|$ as shown in the following.

LEMMA 2.2. *If $s(n) \geqslant \log n$ is a monotonic function and $\log |M_n| = \mathcal{O}(s(n))$ then $v(\mathcal{P}, n) = \mathcal{O}(s(n))$.*

PROOF. Since $M_{\leqslant n} = M_0 \cup M_1 \cup \cdots \cup M_n$, we have

$$\begin{aligned}
\log |M_{\leqslant n}| &= \log \sum_{i=0}^{n} |M_i| \\
&\leqslant \log \left((n+1) \cdot \max_{0 \leqslant i \leqslant n} |M_i| \right) \\
&= \log(n+1) + \max_{0 \leqslant i \leqslant n} \log |M_i| \\
&\leqslant \log(n+1) + \max_{0 \leqslant i \leqslant n} \mathcal{O}(s(i)) \\
&\leqslant \log(n+1) + \mathcal{O}(s(n)) = \mathcal{O}(s(n)),
\end{aligned}$$

which proves the statement. ■

LEMMA 2.3. *For every language $L \subseteq \Sigma^*$ there exists a space-optimal variable-size SW-algorithm $\mathcal{P}$, i.e., $v(\mathcal{P}, n) \leqslant v(\mathcal{Q}, n)$ for every variable-size SW-algorithm $\mathcal{Q}$ for L and every $n \in \mathbb{N}$.*

PROOF. Let $\mathcal{P}$ be the minimal deterministic automaton $\mathcal{A}_{\mathsf{SW}(L)}$ for $\mathsf{SW}(L)$. If $\mathcal{Q}$ is any deterministic automaton for $\mathsf{SW}(L)$ then $\mathcal{Q}(x) = \mathcal{Q}(y)$ implies $\mathcal{P}(x) = \mathcal{P}(y)$. Then, we obtain

$$\begin{aligned}
v(\mathcal{P}, n) &= \log |\{\mathcal{P}(w) \mid w \in \Sigma_{\downarrow}^*, \mathsf{mwl}(w) \leqslant n\}| \\
&\leqslant \log |\{\mathcal{Q}(w) \mid w \in \Sigma_{\downarrow}^*, \mathsf{mwl}(w) \leqslant n\}| = v(\mathcal{Q}, n),
\end{aligned}$$

which proves the statement. ■

One could also define the space complexity $v(\mathcal{P}, n)$ as the number of bits required to encode a state of $\mathcal{P}$ where the *current* window size is n. It is not difficult to see that Lemma 2.3 fails for this definition.

We define the space complexity of L in the variable-size sliding window model by $\mathsf{V}_L(n) = v(\mathcal{P}, n)$ where $\mathcal{P}$ is a space-optimal variable-size SW-algorithm for $\mathsf{SW}(L)$ from Lemma 2.3. It is a monotonic function.

LEMMA 2.4. *For any language $L \subseteq \Sigma^*$ and $n \in \mathbb{N}$ we have $\mathsf{F}_L(n) \leqslant \mathsf{V}_L(n)$.*

PROOF. If $\mathcal{P}$ is a space-optimal variable-size SW-algorithm for L then one obtains an SW-algorithm $\mathcal{P}_n$ for window size $n \in \mathbb{N}$ as follows. Let us assume $n \geqslant 1$ (for $n = 0$ we use the trivial

SW-algorithm). First one simulates $\mathcal{P}$ on the initial window $\Box^n$. For every incoming symbol $a \in \Sigma$ we perform a pop operation $\downarrow$ in $\mathcal{P}$, followed by inserting a. Since the maximum window size is bounded by n on any stream, the space complexity is bounded by $v(\mathcal{P}, n) = \mathsf{V}_L(n)$. ■

The following lemma states that in the variable-size model one must at least maintain the current window size if the language is neither empty nor universal. The issue at hand is performing a pop operation on an empty window.

LEMMA 2.5. *Let $\mathcal{P}$ be a variable-size SW-algorithm for a language $\emptyset \subsetneq L \subsetneq \Sigma^*$. Then, $\mathcal{P}(x)$ determines*[3] $|\mathsf{wnd}(x)|$ *for all $x \in \Sigma_\downarrow^*$ and therefore $\mathsf{V}_L(n) \geqslant \log(n+1)$.*

PROOF. Let $y \in \Sigma^+$ be a length-minimal nonempty word such that $|\{\varepsilon, y\} \cap L| = 1$. Consider streams $x_1, x_2 \in \Sigma_\downarrow^*$ with $|\mathsf{wnd}(x_1)| < |\mathsf{wnd}(x_2)| = m$ and assume $\mathcal{P}(x_1) = \mathcal{P}(x_2)$. Then, we also have $\mathcal{P}(x_1 y \downarrow^m) = \mathcal{P}(x_2 y \downarrow^m)$. But $\mathsf{wnd}(x_2 y \downarrow^m) = y$ whereas $\mathsf{wnd}(x_1 y \downarrow^m)$ is a proper suffix of y. Now, by the choice of y one these two words belongs to L whereas the other does not, which contradicts $\mathcal{P}(x_1 y \downarrow^m) = \mathcal{P}(x_2 y \downarrow^m)$.

For the second statement: if the algorithm reads any stream $a_1 \cdots a_n \in \Sigma^n$ it must visit $n+1$ pairwise distinct memory states and hence $v(\mathcal{P}, n) \geqslant \log(n+1)$. ■

Alternative definitions of the variable-size model are conceivable, e.g. one could neglect streams where the popping of an empty window occurs, or assume that the window size is always known to the algorithm. Then the statement of Lemma 2.5 no longer holds.

LEMMA 2.6. *Let Σ be a finite alphabet. For any function $s(n)$ and $\mathsf{X} \in \{\mathsf{F}, \mathsf{V}\}$, the class $\{L \subseteq \Sigma^* \mid \mathsf{X}_L(n) = O(s(n))\}$ forms a Boolean algebra.*

PROOF. Let $L \subseteq \Sigma^*$ be a language. Given a SW-algorithm for L for some fixed window size n or in the variable-size model, we can turn it into an algorithm for the complement $\Sigma^* \setminus L$ by negating its output. Clearly, it has the same space complexity as the original algorithm.

Let $L_1, L_2 \subseteq \Sigma^*$ be two languages. Let $\mathcal{P}_1, \mathcal{P}_2$ be SW-algorithms for L_1, L_2, respectively, either for some fixed window size n or in the variable-size model. Define $\mathcal{P}$ to be the product automaton of $\mathcal{P}_1$ and $\mathcal{P}_2$ which outputs the disjunction of the outputs of the $\mathcal{P}_i$.

In the case of a fixed window size n, $\mathcal{P}$ has $2^{s(\mathcal{P}_1)} \cdot 2^{s(\mathcal{P}_2)} = 2^{s(\mathcal{P}_1)+s(\mathcal{P}_2)}$ many states and hence $s(\mathcal{P}) = s(\mathcal{P}_1) + s(\mathcal{P}_2)$. This implies $\mathsf{F}_{L_1 \cup L_2}(n) \leqslant \mathsf{F}_{L_1}(n) + \mathsf{F}_{L_2}(n)$. For the variable-size model, notice that $2^{v(\mathcal{P},n)} \leqslant 2^{v(\mathcal{P}_1,n)} \cdot 2^{v(\mathcal{P}_2,n)} = 2^{v(\mathcal{P}_1,n)+v(\mathcal{P}_2,n)}$. Therefore, $\mathsf{V}_{L_1 \cup L_2}(n) \leqslant \mathsf{V}_{L_1}(n) + \mathsf{V}_{L_2}(n)$. ■

REMARK 2.7. Before we start our investigation of the space complexity of regular languages in the sliding window model, we would like to discuss an aspect of our definition of the space complexities $\mathsf{F}_L(n)$ and $\mathsf{V}_L(n)$. Both complexity measures do not include the space needed

3 In other words, for all $x_1, x_2 \in \Sigma_\downarrow^*$, if $\mathcal{P}(x_1) = \mathcal{P}(x_2)$ then $|\mathsf{wnd}(x_1)| = |\mathsf{wnd}(x_2)|$.

for internal computations, i.e., the space needed for computing the memory updates of the streaming algorithm. Let us explain this in more detail for the variable-size model (the same arguments apply to the fixed-size model). Take the space-optimal variable-size SW-algorithm $\mathcal{P}$ for a language L; see Lemma 2.3. The function $\mathsf{V}_L(n)$ measures the number of bits needed to encode the states in the set $M_{\leqslant n}$ (see the line after (3)). But the transition function of $\mathcal{P}$ may be difficult to compute. In other words: if we have two memory states $p, q \in M_{\leqslant n}$ (both encoded by bit strings of length $\mathsf{V}_L(n)$) and an a-labelled transition from p to q in $\mathcal{P}$ then additional memory is needed in general in order to compute q from p and a. This memory is what we mean by the space needed for internal computations. Our definition of $\mathsf{V}_L(n)$ does not include this space. One reason for this is that if we would include the space needed for internal computations in the total space bound, then it would be difficult to obtain lower bounds that match the upper bounds. In particular, techniques based on communication complexity that we use in the randomized setting (see Section 4) are not able to take space for internal calculations into account. In our setting, these techniques only allow to prove lower bounds on the number of memory states of a (randomized) streaming algorithm and therefore are not sensitive with respect to the space needed to go from one memory state to the next memory state.

3. Deterministic sliding window algorithms

In this section, we will show that the space complexity of every regular language in both sliding window models is either constant, logarithmic or linear. In Example 1.2 we have already seen prototypical languages with these three space complexities, namely $\Sigma^* a$ (constant), $\Sigma^* a \Sigma^*$ (logarithmic) and $a\Sigma^*$ (linear) for $\Sigma = \{a, b\}$. Intuitively, for languages of logarithmic space complexity it suffices to maintain a constant number of positions in the window. For languages of constant space complexity it suffices to maintain a constant-length suffix of the window. Moreover, we describe the languages with logarithmic and constant space complexity as finite Boolean combinations of simple atomic languages.

3.1 Right-deterministic finite automata

It turns out that the appropriate representation of a regular language for the analysis in the sliding window model are deterministic finite automata which read the input word, i.e., the window, from right to left. Such automata are called *right-deterministic finite automata (rDFA)* in this paper. The reason why we use rDFAs instead of DFAs can be explained intuitively for the variable-size sliding window model as follows. The variable-size model contains operations in both "directions": On the one hand a variable-size window can be extended on the right, and on the other hand the window can be shortened to an arbitrary suffix. For regular languages the extension to longer windows is "tame" because the Myhill–Nerode right congruences have finite index. Hence, it remains to control the structure of all suffixes with respect to the regular language, which is best captured by an rDFA for the language.

Formally, a *right-deterministic finite automaton (rDFA)* $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ consists of a finite state set Q, a finite alphabet Σ, a set of final states $F \subseteq Q$, a transition function $\delta \colon \Sigma \times Q \to Q$, and an initial state $q_0 \in Q$. The transition function δ extends to a left action $\cdot \colon \Sigma^* \times Q \to Q$ by $\varepsilon \cdot q = q$ and $(aw) \cdot q = \delta(a, w \cdot q)$ for all $a \in \Sigma$, $w \in \Sigma^*$, $q \in Q$. The language accepted by $\mathcal{B}$ is $\mathsf{L}(\mathcal{B}) = \{w \in \Sigma^* \mid w \cdot q_0 \in F\}$.

A *run* of $\mathcal{B}$ on a word $w = a_1 \cdots a_n \in \Sigma^*$ from p_n to p_0 is a finite sequence

$$\pi = p_0 a_1 p_1 a_2 p_2 \cdots p_{n-1} a_n p_n \in Q(\Sigma Q)^*$$

such that $p_{i-1} = a_i \cdot p_i$ for all $1 \leqslant i \leqslant n$. Quite often we write such a run in the following way

$$\pi : p_0 \xleftarrow{a_1} p_1 \xleftarrow{a_2} p_2 \cdots p_{n-1} \xleftarrow{a_n} p_n.$$

If the intermediate states $p_1, \dots, p_{n-1}$ are not important we write this run also as

$$\pi : p_0 \xleftarrow{a_1 a_2 \cdots a_n} p_n.$$

The state p_n is also called the *starting state* of the above run π. The run π is *accepting* if $p_0 \in F$ (note that we do not require $p_n = q_0$) and otherwise *rejecting*. Its *length* $|\rho|$ is the length $|w|$ of w. A run of length zero is called empty; note that it consists of a single state. A run of length one is also called a *transition*. If $\pi = p_0 a_1 p_1 \cdots a_n p_n$ and $\rho = r_0 b_1 r_1 \cdots b_\ell r_\ell$ are runs such that $p_n = r_0$ then their *composition* $\pi\rho$ is defined as $\pi\rho = p_0 a_1 p_1 \cdots a_n r_0 b_1 r_1 \cdots b_\ell r_\ell$; it is a run on $a_1 \cdots a_n b_1 \cdots b_\ell$. This definition allows us to factorize runs in Section 3.3. We call a run π a *P-run* for a subset $P \subseteq Q$ if all states occurring in π are contained in P.

A state $q \in Q$ is *reachable* from $p \in Q$ if there exists a run from p to q, in which case we write $q \preceq_{\mathcal{B}} p$. We say that q is *reachable* if it is reachable from the initial state q_0. A set of states $P \subseteq Q$ is reachable if all $p \in P$ are reachable. The reachability relation $\preceq_{\mathcal{B}}$ is a *preorder* on Q, i.e., it is reflexive and transitive. Two states $p, q \in Q$ are *strongly connected* if $p \preceq_{\mathcal{B}} q \preceq_{\mathcal{B}} p$. This yields an equivalence relation on Q whose equivalence classes are the *strongly connected components (SCCs)* of $\mathcal{B}$. A subset $P \subseteq Q$ is *strongly connected* if it is contained in a single SCC, i.e., all $p, q \in P$ are strongly connected.

3.2 Space trichotomy

In this section, we state two technical results which directly imply Theorem 1.3. Let $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ be an rDFA. A set of states $P \subseteq Q$ is *well-behaved* if for any two P-runs π_1, π_2 which start in the same state and have equal length, either both π_1 and π_2 are accepting or both are rejecting. If every reachable SCC in $\mathcal{B}$ is well-behaved then $\mathcal{B}$ is called *well-behaved*. A state $q \in Q$ is *transient* if $x \cdot q \neq q$ for all $x \in \Sigma^+$. Every transient state in $\mathcal{B}$ forms an SCC of size one (a *transient SCC*); however, not every SCC of size one is transient (there can be a loop at

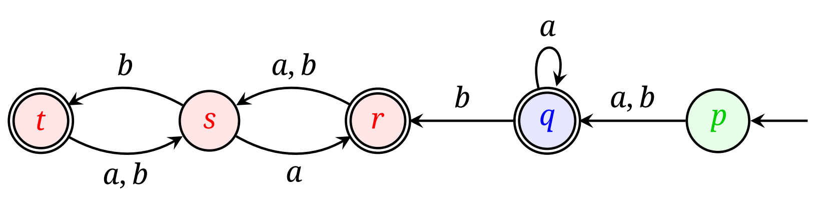

Figure 2. A well-behaved rDFA consisting of three SCCs.

the unique state of the SCC). Let $U(\mathcal{B}) \subseteq Q$ be the set of states $q \in Q$ for which there exists a nontransient state $p \in Q$ such that q is reachable from p and p is reachable from the initial state q_0. Notice that $q \in U(\mathcal{B})$ if and only if there exist runs of unbounded length from q_0 to q (hence the symbol U for unbounded). Moreover, if $U(\mathcal{B})$ is well-behaved then $\mathcal{B}$ must be well-behaved. This follows directly from the above definition and it is also a consequence of Theorem 3.2 below.

EXAMPLE 3.1. Consider the rDFA $\mathcal{A}$ in Figure 2. It consists of three SCCs, namely the green SCC $\{p\}$, the blue SCC $\{q\}$ and the red SCC $\{r, s, t\}$. The red SCC is well-behaved since any run starting in r ends in a final state if and only if its length is even. The other SCCs are also well-behaved and therefore, the entire automaton is well-behaved. State p is a transient state and $U(\mathcal{A}) = \{q, r, s, t\}$. ◆

THEOREM 3.2. *Let $L \subseteq \Sigma^*$ be regular and $\mathcal{B}$ be any rDFA for L.*

(1) If $\mathcal{B}$ is well-behaved then $\mathsf{V}_L(n) = \mathcal{O}(\log n)$ and $\mathsf{F}_L(n) = \mathcal{O}(\log n)$.
(2) If $\mathcal{B}$ is not well-behaved then $\mathsf{V}_L(n) = \Omega(n)$ and $\mathsf{F}_L(n) = \Omega^\infty(n)$.
(3) If $U(\mathcal{B})$ is well-behaved then $\mathsf{F}_L(n) = \mathcal{O}(1)$.
(4) If $U(\mathcal{B})$ is not well-behaved then $\mathsf{F}_L(n) = \Omega^\infty(\log n)$.
(5) If $L \in \{\emptyset, \Sigma^*\}$ then $\mathsf{V}_L(n) = \mathcal{O}(1)$.
(6) If $L \notin \{\emptyset, \Sigma^*\}$ then $\mathsf{V}_L(n) = \Omega(\log n)$.

Theorem 3.2 implies that $\mathsf{F}_L(n)$ is either $\Theta(1)$, $\Theta^\infty(\log n)$, or $\Theta^\infty(n)$, and $\mathsf{V}_L(n)$ is either $\Theta(1)$, $\Theta(\log n)$, or $\Theta(n)$. For the characterizations in Theorem 1.3 it remains to prove:

THEOREM 3.3. *Let $L \subseteq \Sigma^*$ be regular.*

(i) $\mathsf{F}_L(n) = \mathcal{O}(1) \iff L \in \langle \mathbf{ST}, \mathbf{Len} \rangle$.
(ii) $\mathsf{F}_L(n) = \mathcal{O}(\log n) \iff L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$.

In the rest of Section 3 we prove Theorem 3.2 and Theorem 3.3. We start with the path summary algorithm, which is our main deterministic SW-algorithm for the variable-size model.

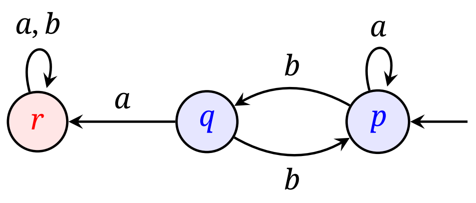

Figure 3. Another rDFA partitioned into two SCCs.

3.3 The path summary algorithm

In the following, let $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ be a right-deterministic finite automaton. We call a run π *internal* if π is a P-run for some SCC P. The *SCC-factorization* of π is the unique factorization $\pi = \pi_k \tau_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$, where every π_i is an internal (possibly empty) run but cannot be extended to an internal run. The τ_i are single transitions (i.e., runs from $Q\Sigma Q$) connecting distinct SCCs. Let $p_k, \dots, p_1 \in Q$ be the starting states of the runs $\pi_k, \dots, \pi_1$. Then, the *path summary* of π is defined as

$$\mathrm{ps}(\pi) = (|\pi_k|, p_k)(|\tau_{k-1}\pi_{k-1}|, p_{k-1}) \cdots (|\tau_2\pi_2|, p_2)(|\tau_1\pi_1|, p_1),$$

which is a sequence of pairs from $\mathbb{N} \times Q$. It specifies the first state that is visited in an SCC, and the length of the run until reaching the next SCC or the end of the word, respectively. The leftmost length $|\pi_k|$ can be zero but all other lengths $|\tau_i\pi_i| = 1 + |\pi_i|$ are strictly positive. We define $\pi_{w,q}$ to be the unique run of $\mathcal{B}$ on a word $w \in \Sigma^*$ starting from q, and $\mathrm{PS}_{\mathcal{B}}(w) = \{\mathrm{ps}(\pi_{w,q}) \mid q \in Q\}$.

EXAMPLE 3.4. Consider the rDFA $\mathcal{B}$ in Figure 3. For the moment, the final states are irrelevant. It consists of two SCCs, namely the blue SCC $\{p, q\}$ and the red SCC $\{r\}$. All its runs on the word $w = aababb$ are listed here:

$$\begin{array}{c}
r \xleftarrow{a} r \xleftarrow{a} q \xleftarrow{b} p \xleftarrow{a} p \xleftarrow{b} q \xleftarrow{b} p \\
r \xleftarrow{a} r \xleftarrow{a} r \xleftarrow{b} r \xleftarrow{a} q \xleftarrow{b} p \xleftarrow{b} q \\
r \xleftarrow{a} r \xleftarrow{a} r \xleftarrow{b} r \xleftarrow{a} r \xleftarrow{b} r \xleftarrow{b} r
\end{array}$$

Then, $\mathrm{PS}_{\mathcal{B}}(w)$ contains the path summaries $(1, r)(5, p)$, $(3, r)(3, q)$ and $(6, r)$. ◆

The *path summary algorithm for* $\mathcal{B}$ is a streaming algorithm over $\Sigma_{\downarrow}$ described in Algorithm 1. The data structure at time instant t is denoted by S_t. The acceptance condition will be defined later.

LEMMA 3.5. *Algorithm 1 correctly maintains* $\mathrm{PS}_{\mathcal{B}}(w)$ *for the active window* $w \in \Sigma^*$.

PROOF. Initially $\mathrm{PS}_{\mathcal{B}}(\varepsilon)$ contains the path summary of every empty run from every state, which is formally $\{0\} \times Q$.

Assume $S_{t-1} = \mathrm{PS}_{\mathcal{B}}(w)$ for some window $w \in \Sigma^*$ and that $a \in \Sigma$ is the incoming symbol. The claim is that the algorithm computes $S_t = \mathrm{PS}_{\mathcal{B}}(wa)$ from S_{t-1}. Suppose that π' is a run in $\mathcal{B}$ on wa.

```
Input:  sequence of operations $a_1 a_2 a_3 \cdots \in \Sigma_\downarrow^\omega$
$S_0 = \{0\} \times Q$
foreach $t \geqslant 1$ do
    $S_t = \emptyset$
    if $a_t \in \Sigma$ then
        for $p_0 \in Q$ do
            let $p_1 = a_t \cdot p_0$ and $(\ell_k, p_k) \cdots (\ell_1, p_1) \in S_{t-1}$
            if $p_0$ and $p_1$ are strongly connected then
                add $(\ell_k, p_k) \cdots (\ell_2, p_2)(\ell_1 + 1, p_0)$ to $S_t$
            else
                add $(\ell_k, p_k) \cdots (\ell_1, p_1)(1, p_0)$ to $S_t$
    if $a_t = \downarrow$ then
        if $S_{t-1} = \{0\} \times Q$ then
            $S_t = S_{t-1}$
        else
            for $(\ell_k, p_k) \cdots (\ell_1, p_1) \in S_{t-1}$ do
                if $\ell_k \geqslant 1$ then
                    add $(\ell_k - 1, p_k)(\ell_{k-1}, p_{k-1}) \cdots (\ell_1, p_1)$ to $S_t$
                else
                    add $(\ell_{k-1} - 1, p_{k-1})(\ell_{k-2}, p_{k-2}) \cdots (\ell_1, p_1)$ to $S_t$
```

Algorithm 1. The path summary algorithm

It can be factorized as $\pi' = \pi\, p_1 a\, p_0$ with $\mathrm{ps}(\pi) \in S_{t-1}$. Let $\pi = \pi_k \tau_{k-1} \pi_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$ be the SCC-factorization of π. If p_0 and p_1 are strongly connected then the SCC-factorization of π' is $\pi' = \pi_k \tau_{k-1} \pi_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1'$ where $\pi_1' = \pi_1\, p_1 a\, p_0$, and otherwise $\pi' = \pi_k \tau_{k-1} \pi_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1\, p_1 a\, p_0$. In this way the algorithm computes $\mathrm{ps}(\pi')$ from $\mathrm{ps}(\pi)$.

Now, consider the case $a = \downarrow$. We have $w = \varepsilon$ if and only if $\mathrm{PS}_{\mathcal{B}}(w) = \{0\} \times Q$, and in this case the set of path summaries is unchanged, i.e., we set $S_t = S_{t-1}$. Otherwise assume $w = bv$ for some $b \in \Sigma$. We claim that the algorithm computes $S_t = \mathrm{PS}_{\mathcal{B}}(v)$ from S_{t-1}. Suppose that π' is a run in $\mathcal{B}$ on v which ends in state $p \in Q$. If $q = \delta(b, p)$ in $\mathcal{B}$ then let $\pi = q\, b\, p\, \pi'$. We have $\mathrm{ps}(\pi) \in S_{t-1}$. Let $\pi = \pi_k \tau_{k-1} \pi_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$ be the SCC-factorization of π. If $|\pi_k| \geqslant 1$ then $\pi' = \pi_k' \tau_{k-1} \pi_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$ is the SCC-factorization of π' where $\pi_k = q\, b\, p\, \pi_k'$. Otherwise π_k is empty and $\tau_{k-1} = q\, b\, p$. Therefore, $\pi' = \pi_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$ is the SCC-factorization of π'. In this way the algorithm computes $\mathrm{ps}(\pi')$ from $\mathrm{ps}(\pi)$. ∎

Observe that Algorithm 1 cannot be directly adapted to work for (left-)DFA: For a pop operation, one would need to remove the first transition from each path summary, which is generally not possible since a path summary does not store its second state.

3.4 Proof of Theorem 3.2(1)

Using the path summary algorithm we can prove Theorem 3.2(1):

PROPOSITION 3.6. *If $\mathcal{B}$ is well-behaved then the regular language $L = \mathsf{L}(\mathcal{B})$ has space complexity $\mathsf{V}_L(n) = O(|\mathcal{B}|^2 \cdot \log n)$, which is $O(\log n)$ for a fixed $\mathcal{B}$.*

PROOF. Let $\mathcal{B}$ be well-behaved. Call a path summary *accepting* if it is the path summary of some accepting run. The variable-size sliding window algorithm for L is the path summary algorithm for $\mathcal{B}$ where the algorithm accepts if the path summary starting in q_0 is accepting.

For the correctness of the algorithm it suffices to show that any run π starting in q_0 is accepting if and only if $\mathrm{ps}(\pi)$ is accepting. The direction from left to right is immediate by definition. For the other direction, consider the path summary $\mathrm{ps}(\pi) = (\ell_k, p_k) \cdots (\ell_1, p_1)$ and the SCC-factorization $\pi = \pi_k \tau_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$. Since $\mathrm{ps}(\pi)$ is accepting, there is an accepting run π'_k that starts in p_k and has length ℓ_k. Since $\mathcal{B}$ is well-behaved, the SCC of p_k is well-behaved. Therefore, since π'_k is accepting and $|\pi_k| = |\pi'_k| = \ell_k$, π_k must also be accepting and thus π is accepting.

We claim that the space complexity of the path summary algorithm is bounded by $O(|\mathcal{B}|^2 \cdot (\log n + \log |\mathcal{B}|))$. Observe that $\mathrm{PS}_{\mathcal{B}}(w)$ contains $|\mathcal{B}|$ path summaries, and a single path summary $\mathrm{ps}(\pi)$ consists of a sequence of at most $|\mathcal{B}|$ states and a sequence $(\ell_k, \dots, \ell_1)$ of $k \leqslant |\mathcal{B}|$ numbers up to $|\pi|$. Hence, the path summary $\mathrm{ps}(\pi)$ can be encoded using $O(|\mathcal{B}| \cdot (\log |\mathcal{B}| + \log |\pi|))$ bits, which yields the total space complexity $O(|\mathcal{B}|^2 \cdot (\log n + \log |\mathcal{B}|))$.

To reduce the space complexity to $O(|\mathcal{B}|^2 \cdot \log n)$ we need to make a case distinction. The algorithm maintains the window size $n \in \mathbb{N}$ and the maximal suffix of the window of length up to $|\mathcal{B}|$ (explicitly) using $O(\log n + |\mathcal{B}|)$ bits. If $n \leqslant |\mathcal{B}|$ then this information suffices to test membership of the window to L. As soon as n exceeds $|\mathcal{B}|$ we initialize $\mathrm{PS}_{\mathcal{B}}(w)$ and use the path summary algorithm as described above. If $n > |\mathcal{B}|$ then its space complexity is $O(|\mathcal{B}|^2 \cdot (\log n + \log |\mathcal{B}|)) \subseteq O(|\mathcal{B}|^2 \cdot \log n)$. ■

Observe that the path summary algorithm only stores $O(\log n)$ bits where n is the *current* (not the *maximum*) window size.

Before we continue with the proof of the other points from Theorem 3.2, we discuss some implementation details for our logspace SW-algorithm.[4] To implement the path summary algorithm on a realistic computation model, we have to be able to efficiently determine whether

4 These details are not needed for the proof of Proposition 3.6 since we abstract from internal computations in our sliding window model; see Remark 2.7.

a path summary is accepting. Given a number $d \geqslant 1$, a set of natural numbers $X \subseteq \mathbb{N}$ is *d-periodic* if we have $x \in X$ if and only if $x + d \in X$.

LEMMA 3.7. *Let $P \subseteq Q$ be a well-behaved subset in $\mathcal{B}$ and $p_0 \in P$ be nontransient. Then* $\mathrm{Acc}(P, p_0) := \{|\pi| : \pi$ *is an accepting P-run starting in* $p_0\}$ *is d-periodic for some* $d \leqslant |Q|$.

PROOF. Let π_0 be any nonempty run from p_0 to p_0, which exists because p_0 is nontransient. Furthermore, we can choose π_0 such that its length $d := |\pi_0|$ is at most $|Q|$.

If $\ell \in \mathrm{Acc}(P, p_0)$, then there exists an accepting P-run π starting in p_0 of length ℓ. Then $\pi\pi_0$ is also an accepting P-run and we conclude $|\pi\pi_0| = \ell + d \in \mathrm{Acc}(P, p_0)$.

Now we need to show that $\ell \notin \mathrm{Acc}(P, p_0)$ implies $\ell + d \notin \mathrm{Acc}(P, p_0)$. Towards a contradiction assume that $\ell \notin \mathrm{Acc}(P, p_0)$ and $\ell + d \in \mathrm{Acc}(P, p_0)$, i.e., there exists an accepting P-run π starting in p_0 of length $\ell + d$. Factorize $\pi = \pi_1\pi_2$ where $|\pi_2| = \ell$. Now π_2 must be rejecting since $\ell \notin \mathrm{Acc}(P, p_0)$. But then $\pi_2\pi_0$ is a rejecting P-run of length $\ell + d$, which contradicts the well-behavedness of P since $\ell + d \in \mathrm{Acc}(P, p_0)$. ■

In the following we describe how to implement the algorithm from Proposition 3.6. We do the following preprocessing on the well-behaved rDFA $\mathcal{B}$. Using depth-first search we compute all SCCs in $\mathcal{B}$. For every SCC P we pick a state $p \in P$ and compute the distance $\mathrm{dist}(p, q)$ from p to all states $q \in P$ using any shortest path algorithm. Furthermore let d be the minimal length of a nonempty run from p to p itself, which is the period d from Lemma 3.7. If no such run exists then we store the information that p is transient. Otherwise we assign to each state $q \in P$ the distance from p modulo d. By traversing an arbitrary P-run of length d from p we can compute a bit vector of length d which represents $\mathrm{Acc}(P, p_0)$. Using this information we can easily answer whether a path summary $(\ell_k, p_k) \cdots (\ell_1, p_1)$ is accepting: it is accepting if and only if either p_k is transient, $\ell_k = 0$ and $p_k \in F$, or $\mathrm{dist}(p_0, p_k) + \ell_k \bmod d$ belongs to $\mathrm{Acc}(P, p_0)$ where P is the SCC of p_k and p_0 is the picked state in P.

3.5 Proof of Theorem 3.2(2)

We continue with proving a linear lower bound for rDFA which are not well-behaved.

LEMMA 3.8. *If $\mathcal{B}$ is not well-behaved then there exist words $u_1, u_2, v_1, v_2, z \in \Sigma^*$ where $|u_i| = |v_i|$ for $i \in \{1, 2\}$ such that $L = \mathsf{L}(\mathcal{B})$ separates $u_2\{u_1u_2, v_1v_2\}^*z$ and $v_2\{u_1u_2, v_1v_2\}^*z$.*

PROOF. The automaton structure is illustrated in Figure 4. Since $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ is not well-behaved, there is a reachable SCC S that is not well-behaved. Take a state $p \in S$ and a word $z \in \Sigma^*$ with

$$p \overset{z}{\leftarrow} q_0.$$

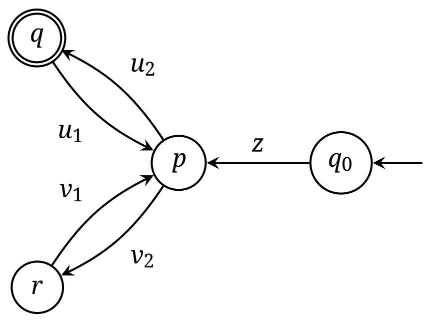

Figure 4. Forbidden pattern for well-behaved rDFAs where $|u_1| = |v_1|$ and $|u_2| = |v_2|$.

Moreover, since S is strongly connected and not well-behaved there are states $q \in S \cap F$, $r \in S \setminus F$ and nonempty words $u_2, v_2 \in \Sigma^*$ such that $|u_2| = |v_2|$ and

$$q \xleftarrow{u_2} p \xleftarrow{z} q_0 \quad \text{and} \quad r \xleftarrow{v_2} p \xleftarrow{z} q_0.$$

Finally, since S is strongly connected, there are words $u_1, v_1 \in \Sigma^*$ such that

$$p \xleftarrow{u_1} q \xleftarrow{u_2} p \xleftarrow{z} q_0 \quad \text{and} \quad p \xleftarrow{v_1} r \xleftarrow{v_2} p \xleftarrow{z} q_0.$$

We can ensure that $|u_1| = |v_1|$ and hence also $|u| = |v|$ for $u = u_1u_2, v = v_1v_2$. If $k = |u|$ and $\ell = |v|$ we replace u_1 by $u^{\ell-1}u_1$ and v_1 by $v^{k-1}v_1$ (note that $k > 0$ and $\ell > 0$ since u and v are nonempty), which preserves all properties above. Then, $u_2\{u, v\}^*z$ and $v_2\{u, v\}^*z$ are separated by L. ∎

We can now show Theorem 3.2(2):

PROPOSITION 3.9. *If $\mathcal{B}$ is not well-behaved then the language $L = \mathsf{L}(\mathcal{B})$ satisfies $\mathsf{F}_L(n) = \Omega^\infty(n)$ and $\mathsf{V}_L(n) = \Omega(n)$.*

PROOF. Let $u_1, u_2, v_1, v_2, z \in \Sigma^*$ be the words from Lemma 3.8 and let $u = u_1u_2$ and $v = v_1v_2$. Now, consider an SW-algorithm $\mathcal{P}_n$ for L and window size $n = |u_2| + |u| \cdot (m-1) + |z|$ for some $m \geqslant 1$. We prove that $\mathcal{P}_n$ has at least 2^m many states by showing that $\mathcal{P}_n(x) \neq \mathcal{P}_n(y)$ for any $x, y \in \{u, v\}^m$ with $x \neq y$. Notice that $|\{u, v\}^m| = 2^m$ since $u \neq v$ and $|u| = |v|$.

Read two distinct words $x, y \in \{u, v\}^m$ into two instances of $\mathcal{P}_n$. Consider the right most $\{u, v\}$-block where x and y differ. Without loss of generality assume $x = x'us$ and $y = y'vs$ for some $x', y', s \in \{u, v\}^*$ with $|x'| = |y'|$. By reading $x'z$ into both instances the window of the x-instance becomes $\mathsf{last}_n(xx'z) = u_2sx'z$ and the window of the y-instance becomes $\mathsf{last}_n(yx'z) = v_2sx'z$. By Lemma 3.8 the two windows are separated by L, and therefore the algorithm $\mathcal{P}_n$ must accept one of the streams $xx'z$ and $yx'z$, and reject the other. In conclusion $\mathcal{P}_n(x) \neq \mathcal{P}_n(y)$ and hence $\mathcal{P}_n$ must use at least $m = \Omega(n)$ bits. This holds for infinitely many n, namely all n of the form $|u_2| + |z| + |u| \cdot (m-1)$ for some $m \geqslant 1$.

The argument above shows that there exist numbers $c, d \in \mathbb{N}$ such that for all $m \geqslant 1$ we have $\mathsf{V}_L(cm+d) \geqslant \mathsf{F}_L(cm+d) = \Omega(m)$. If $n \geqslant d$ then $m = \lfloor (n-d)/c \rfloor = \Omega(n)$ satisfies $cm+d \leqslant n$. Therefore, $\mathsf{V}_L(n) \geqslant \mathsf{V}_L(cm+d) = \Omega(m)$ by monotonicity of V_L and hence $\mathsf{V}_L(n) = \Omega(n)$. ∎

From Proposition 3.6 and Proposition 3.9 we obtain:

COROLLARY 3.10. *Let $\mathsf{X} \in \{\mathsf{F}, \mathsf{V}\}$. A regular language $L \subseteq \Sigma^*$ satisfies $\mathsf{X}_L(n) = \mathcal{O}(\log n)$ if and only if L is recognized by a well-behaved rDFA.*

3.6 Proof of Theorem 3.2(3)–(6)

Next, we study which regular languages have sublogarithmic complexity. Recall that in the variable-size model any such language must be empty or universal because the algorithm must at least maintain the current window size by Lemma 2.5.

COROLLARY 3.11. *The empty language $L = \emptyset$ and the universal language $L = \Sigma^*$ satisfy $\mathsf{V}_L(n) = \mathcal{O}(1)$. All other languages satisfy $\mathsf{V}_L(n) = \Omega(\log n)$.*

This proves points (5) and (6) in Theorem 3.2. Now, we can turn to the fixed-size model and prove the points (3) and (4). Point (3) follows from:

PROPOSITION 3.12. *If $U(\mathcal{B})$ is well-behaved then $L = \mathsf{L}(\mathcal{B})$ has space complexity $\mathsf{F}_L(n) = \mathcal{O}(|\mathcal{B}|)$, which is $\mathcal{O}(1)$ when $\mathcal{B}$ is fixed.*

PROOF. Let $k = |\mathcal{B}|$. The SW-algorithm $\mathcal{P}_n$ for $\mathsf{SW}_n(L)$ maintains $\mathsf{last}_k(x)$ for an input stream $x \in \Sigma^*$ using $\mathcal{O}(k)$ bits. If $n \leqslant k$ then $\mathsf{last}_n(x)$ is a suffix of $\mathsf{last}_k(x)$ and hence $\mathcal{P}_n$ can determine whether $\mathsf{last}_n(x) \in L$. If $n > k$ then $\mathsf{last}_k(x)$ is a suffix of $\mathsf{last}_n(x)$, say $\mathsf{last}_n(x) = s\,\mathsf{last}_k(x)$. We can decide if $\mathsf{last}_n(x) \in L$ as follows: Consider the run of $\mathcal{B}$ on $\mathsf{last}_n(x)$ starting from the initial state:

$$r \xleftarrow{s} q \xleftarrow{\mathsf{last}_k(x)} q_0.$$

By the choice of k some state $p \in Q$ must occur twice in the run $q \xleftarrow{\mathsf{last}_k(x)} q_0$. Therefore, p is nontransient and all states in the run $r \xleftarrow{s} q$ belong to $U(\mathcal{B})$. Since $U(\mathcal{B})$ is well-behaved, r is final if and only if some run of length $|s| = n - k$ starting in q is accepting. This information can be precomputed for each state q in the fixed-size model. ■

For the lower bound in Theorem 3.2(4) we need the following lemma:

LEMMA 3.13. *If $U(\mathcal{B})$ is not well-behaved then there exist words $x, y, z \in \Sigma^*$ where $|x| = |y|$ such that $L = \mathsf{L}(\mathcal{B})$ separates xy^*z and y^*z.*

PROOF. Since $U(\mathcal{B})$ is not well-behaved, there are $U(\mathcal{B})$-runs π and ρ from the same starting state $q \in U(\mathcal{B})$ such that $|\pi| = |\rho|$ and exactly one of the runs π and ρ is accepting. By definition of $U(\mathcal{B})$ the state q is reachable from a nontransient state p via some run σ such that p is reachable from the initial state q_0, say $p \xleftarrow{z_0} q_0$. We can replace π by $\pi\sigma$ and ρ by $\rho\sigma$ preserving the properties of being $U(\mathcal{B})$-runs and $|\pi| = |\rho|$. Assume that π and ρ are runs on words $v \in \Sigma^*$ and $w \in \Sigma^*$, respectively. Since p is nontransient, we can construct runs from p to p of unbounded lengths. Consider such a run $p \xleftarrow{u} p$ of length $|u| \geqslant |v| = |w|$. Then, L separates

vu^*z_0 and wu^*z_0. Factorize $u = u_1u_2$ so that $|u_2| = |v| = |w|$. Notice that all words in $u_2u^*z_0$ reach the same state in $\mathcal{B}$ and hence $u_2u^*z_0$ is either contained in L or disjoint from L. Then, L separates either $u_2u^*z_0$ and vu^*z_0, or $u_2u^*z_0$ and wu^*z_0. Hence, L also separates $(u_2u_1)^*u_2z_0$ from either $vu_1(u_2u_1)^*u_2z_0$ or from $wu_1(u_2u_1)^*u_2z_0$. This yields the words $z = u_2z_0$, $y = u_2u_1$ and $x = vu_1$ or $x = wu_1$ with the claimed properties. ■

PROPOSITION 3.14. *If $U(\mathcal{B})$ is not well-behaved then $L = \mathsf{L}(\mathcal{B})$ satisfies $\mathsf{F}_L(n) \geqslant \log n - \mathcal{O}(1)$ for infinitely many n. In particular, $\mathsf{F}_L(n) = \Omega^\infty(\log n)$.*

PROOF. Let $x, y, z \in \Sigma^*$ be the words from Lemma 3.13. Consider an SW-algorithm $\mathcal{P}_n$ for L and window size $n = |x| + |y| \cdot m + |z|$ for some $m \geqslant 1$. We prove that $\mathcal{P}_n$ has at least m many states by showing that $\mathcal{P}_n(xy^i) \neq \mathcal{P}_n(xy^j)$ for any $1 \leqslant i < j \leqslant m$. Let $1 \leqslant i < j \leqslant m$. Then, we have

$$\mathsf{last}_n(xy^iy^{m-i}z) = \mathsf{last}_n(xy^mz) = xy^mz$$

and

$$\mathsf{last}_n(xy^jy^{m-i}z) = \mathsf{last}_n(xy^{m+j-i}z) = y^{m+1}z.$$

Since exactly one of the words xy^mz and $y^{m+1}z$ belongs to L, it also holds that exactly one of the streams $xy^iy^{m-i}z$ and $xy^jy^{m-i}z$ is accepted by $\mathcal{P}_n$. This proves that $\mathcal{P}_n$ must reach different memory states on inputs xy^i and xy^j. In conclusion $\mathcal{P}_n$ must use $\log m \geqslant \log n - \mathcal{O}(1)$ bits, and this holds for infinitely many n. ■

3.7 Characterization of constant space

Next, we prove that a regular language L has constant space complexity $\mathsf{F}_L(n)$ if and only if it is a Boolean combination of suffix testable languages and regular length languages (Theorem 3.3(i)).

The language L is called *k-suffix testable* if for all $x, y \in \Sigma^*$ and $z \in \Sigma^k$ we have $xz \in L$ if and only if $yz \in L$. Equivalently, L is a Boolean combination of languages of the form Σ^*w where $w \in \Sigma^{\leqslant k}$. Clearly, a language is suffix testable if and only if it is k-suffix testable for some $k \in \mathbb{N}$. Let us remark that the class of suffix testable languages corresponds to the variety $\mathbf{D}$ of definite monoids [87]. Clearly, every finite language is suffix testable: If k is the maximum length of a word in $L \subseteq \Sigma^*$ then L is $(k+1)$-suffix testable since $L = \bigcup_{w \in L}\{w\}$ and $\{w\} = \Sigma^*w \setminus \bigcup_{a \in \Sigma} \Sigma^*aw$.

We will utilize a distance notion between states in a DFA, which is also studied in [51]. The *symmetric difference* of two sets A and B is $A \triangle B = (A \cup B) \setminus (A \cap B)$. We define the distance $d(K, L)$ of two languages $K, L \subseteq \Sigma^*$ by

$$d(K,L) = \begin{cases} \sup_{u \in K \triangle L} |u| + 1, & \text{if } K \neq L, \\ 0, & \text{if } K = L. \end{cases}$$

Notice that $d(K, L) < \infty$ if and only if $K \triangle L$ is finite. For a DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ and a state $p \in Q$, we define $\mathcal{A}_p = (Q, \Sigma, p, \delta, F)$. Moreover, for two states $p, q \in Q$, we define the distance

$d(p,q) = d(\mathsf{L}(\mathcal{A}_p), \mathsf{L}(\mathcal{A}_q))$. If we have two runs $p \xrightarrow{u} p'$ and $q \xrightarrow{u} q'$ where $p' \in F$, $q' \notin F$ and $|u| \geqslant |Q|^2$ then some state pair occurs twice in the runs and we can pump the runs to unbounded lengths. Therefore, $d(p,q) < \infty$ implies $d(p,q) \leqslant |Q|^2$. In fact $d(p,q) < \infty$ implies $d(p,q) \leqslant |Q|$ by [51, Lemma 1].

LEMMA 3.15. *Let $L \subseteq \Sigma^*$ be regular and $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be its minimal DFA. We have:*

(i) for all $p, q \in Q$, $d(p,q) \leqslant k$ if and only if $\forall z \in \Sigma^k : p \cdot z = q \cdot z$,

(ii) L is k-suffix testable if and only if $d(p,q) \leqslant k$ for all $p, q \in Q$,

(iii) if there exists $k \geqslant 0$ such that L is k-suffix testable, then L is $|Q|$-suffix testable.

PROOF. The proof of (i) is an easy induction. If $k = 0$, the statement is $d(p,q) = 0$ if and only if $p = q$, which is true because $\mathcal{A}$ is minimal. For the induction step, we have $d(p,q) \leqslant k+1$ if and only if $d(\delta(p,a), \delta(q,a)) \leqslant k$ for all $a \in \Sigma$ if and only if $\delta(p,a) \cdot z = \delta(q,a) \cdot z$ for all $z \in \Sigma^k$ if and only if $p \cdot z = q \cdot z$ for all $z \in \Sigma^{k+1}$.

For (ii), assume that L is k-suffix testable and consider two states $p = \mathcal{A}(x)$ and $q = \mathcal{A}(y)$. If $z \in \mathsf{L}(\mathcal{A}_p) \triangle \mathsf{L}(\mathcal{A}_q)$, then $|z| < k$ because $xz \in L$ if and only if $yz \notin L$ and L is k-suffix testable.

Now, assume that $d(p,q) \leqslant k$ for all $p, q \in Q$ and consider $x, y \in \Sigma^*$, $z \in \Sigma^k$. Since we have $d(\mathcal{A}(x), \mathcal{A}(y)) \leqslant k$, (i) implies $\mathcal{A}(xz) = \mathcal{A}(yz)$, and in particular $xz \in L$ if and only if $yz \in L$. Therefore, L is k-suffix testable.

Point (iii) follows from (ii) and from the above cited [51, Lemma 1]. ■

LEMMA 3.16. *For any $L \subseteq \Sigma^*$ and $n \geqslant 0$, the language $\mathsf{SW}_n(L)$ is $2^{\mathsf{F}_L(n)}$-suffix testable.*

PROOF. Let $\mathcal{P}_n$ be an SW-algorithm for L and window size n with space complexity $\mathsf{F}_L(n)$. Therefore, $\mathcal{P}_n$ has at most $2^{\mathsf{F}_L(n)}$ states. The definition of $\mathsf{SW}_n(L)$ directly implies that $\mathsf{SW}_n(L)$ is n-suffix testable. By Lemma 3.15(iii) $\mathsf{SW}_n(L)$ is $2^{\mathsf{F}_L(n)}$-suffix testable. ■

Note that Lemma 3.16 holds for arbitrary languages and not only for regular languages.

PROOF OF THEOREM 3.3(i). First, let $L \subseteq \Sigma^*$ be a regular language with $\mathsf{F}_L(n) = O(1)$ and let $k = \max_{n \in \mathbb{N}} 2^{\mathsf{F}_L(n)}$. By Lemma 3.16 the language $\mathsf{SW}_n(L)$ is k-suffix testable for all $n \geqslant 0$. We can express L as the Boolean combination

$$L = (L \cap \Sigma^{\leqslant k-1}) \cup \bigcup_{z \in \Sigma^k} (Lz^{-1})\, z = (L \cap \Sigma^{\leqslant k-1}) \cup \bigcup_{z \in \Sigma^k} ((Lz^{-1})\, \Sigma^k \cap \Sigma^* z)$$

where the right quotient $Lz^{-1} = \{x \in \Sigma^* \mid xz \in L\}$ is regular [15, Chapter 3, Example 5.7]. The set $L \cap \Sigma^{\leqslant k-1}$ is finite and hence suffix testable. It remains to show that each Lz^{-1} for $z \in \Sigma^k$ is a length language. Consider two words $x, y \in \Sigma^*$ of the same length $|x| = |y| = n$. Since $|xz| = |yz| = n + k$ and $\mathsf{SW}_{n+k}(L)$ is k-suffix testable, we have $xz \in L$ if and only if $yz \in L$, and hence $x \in Lz^{-1}$ if and only if $y \in Lz^{-1}$.

For the other direction note that (i) if L is a length language or a suffix testable language then clearly $\mathsf{F}_L(n) = \mathcal{O}(1)$, and (ii) $\{L \subseteq \Sigma^* \mid \mathsf{F}_L(n) = \mathcal{O}(1)\}$ is closed under Boolean operations by Lemma 2.6. This proves the theorem. ∎

3.8 Characterization of logarithmic space

Recall from Theorem 3.2 that well-behaved rDFAs precisely define those regular languages with logarithmic space complexity $\mathsf{F}_L(n)$ or equivalently $\mathsf{V}_L(n)$. In the following, we will show that well-behaved rDFAs recognize precisely the finite Boolean combinations of regular left ideals and regular length languages, which therefore are precisely the regular languages with logarithmic space complexity (Theorem 3.3(ii)). Let us start with the easy direction:

PROPOSITION 3.17. *Every language $L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$ is recognized by a well-behaved rDFA.*

PROOF. Let $\mathcal{B}$ be an rDFA for L. If L is a length language then for all reachable states q and all runs π, π' starting from q with $|\pi| = |\pi'|$ we have: π is accepting if and only if π' is accepting. If L is a left ideal, then whenever a final state p is reachable, and q is reachable from p, then q is also final. Hence, for every reachable SCC P in $\mathcal{B}$ either all states of P are final or all states of P are nonfinal. In particular, $\mathcal{B}$ is well-behaved.

It remains to show that the class of languages $L \subseteq \Sigma^*$ recognized by well-behaved rDFAs is closed under Boolean operations. If $\mathcal{B}$ is well-behaved then the complement automaton $\overline{\mathcal{B}}$ is also well-behaved. Given two well-behaved rDFAs $\mathcal{B}_1, \mathcal{B}_2$, we claim that the product automaton $\mathcal{B}_1 \times \mathcal{B}_2$ recognizing the intersection language is also well-behaved. Suppose that $\mathcal{B}_i = (Q_i, \Sigma, F_i, \delta_i, q_{0,i})$ for $i \in \{1,2\}$. The product automaton for the intersection language is defined by

$$\mathcal{B}_1 \times \mathcal{B}_2 = (Q_1 \times Q_2, \Sigma, F_1 \times F_2, \delta, (q_{0,1}, q_{0,2}))$$

where $\delta(a, (q_1, q_2)) = (\delta_1(a, q_1), \delta_2(a, q_2))$ for all $q_1 \in Q_1, q_2 \in Q_2$ and $a \in \Sigma$. Consider an SCC S of $\mathcal{B}_1 \times \mathcal{B}_2$ which is reachable from the initial state and let $(p_1, p_2), (q_1, q_2), (r_1, r_2) \in S$ such that

$$(q_1, q_2) \xleftarrow{u} (p_1, p_2) \text{ and } (r_1, r_2) \xleftarrow{v} (p_1, p_2)$$

for some words $u, v \in \Sigma^*$ with $|u| = |v|$. Since for $i \in \{1,2\}$ we have $q_i \xleftarrow{u} p_i$ and $r_i \xleftarrow{v} p_i$, and $\{p_i, r_i, q_i\}$ is contained in an SCC of $\mathcal{B}_i$ (which is also reachable from the initial state $q_{0,i}$), we have

$$\begin{aligned} (q_1, q_2) \text{ is final} \quad &\iff \quad q_1 \text{ and } q_2 \text{ are final} \\ &\iff \quad r_1 \text{ and } r_2 \text{ are final} \\ &\iff \quad (r_1, r_2) \text{ is final,} \end{aligned}$$

and therefore $\mathcal{B}_1 \times \mathcal{B}_2$ is well-behaved. ∎

It remains to prove that every well-behaved rDFA recognizes a finite Boolean combination of regular left ideals and regular length languages. With a right-deterministic finite automaton $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ we associate the directed graph (Q, E) with edge set $E = \{(p, a \cdot p) \mid p \in Q, a \in \Sigma\}$. The *period* $g(G)$ of a directed graph G is the greatest common divisor of all cycle lengths in G. If G is acyclic we define the period to be ∞. We will apply the following lemma from Alon et al. [2] to the nontransient SCCs of $\mathcal{B}$.

LEMMA 3.18 ([2]). *Let $G = (V, E)$ be a strongly connected directed graph with $E \neq \emptyset$ and finite period g. Then there exist a partition $V = \bigcup_{i=0}^{g-1} V_i$ and a constant $m(G) \leqslant 3|V|^2$ with the following properties:*

- *For every $0 \leqslant i, j \leqslant g-1$ and for every $u \in V_i$, $v \in V_j$ the length of every directed path from u to v in G is congruent to $j - i$ modulo g.*
- *For every $0 \leqslant i, j \leqslant g-1$, for every $u \in V_i$, $v \in V_j$ and every integer $r \geqslant m(G)$, if r is congruent to $j - i$ modulo g, then there exists a directed path from u to v in G of length r.*

LEMMA 3.19 (uniform period). *For every regular language there exists an rDFA $\mathcal{B}$ recognizing L and a number g such that every nontransient SCC C in $\mathcal{B}$ has period $g(C) = g$.*

PROOF. Let $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ be any rDFA for L. Let g be the product of all periods $g(C)$ over all nontransient SCCs C in $\mathcal{B}$. In the following, we compute in the additive group $\mathbb{Z}_g = \{0, \ldots, g-1\}$. We define

$$\mathcal{B} \times \mathbb{Z}_g = (Q \times \mathbb{Z}_g, \Sigma, F \times \mathbb{Z}_g, \delta', (q_0, 0)),$$

where for all $(p, i) \in Q \times \mathbb{Z}_g$ and $a \in \Sigma$ we set

$$\delta'(a, (p, i)) = \begin{cases} (\delta(a, p), i+1), & \text{if } p \text{ and } \delta(a, p) \text{ are strongly connected,} \\ (\delta(a, p), 0), & \text{otherwise.} \end{cases}$$

Clearly, $\mathsf{L}(\mathcal{B} \times \mathbb{Z}_g) = \mathsf{L}(\mathcal{B})$. We show that every nontransient SCC of $\mathcal{B} \times \mathbb{Z}_g$ has period g. Let D be a nontransient SCC of $\mathcal{B} \times \mathbb{Z}_g$. Clearly, every cycle length in D is a multiple of g. Take any state $(q, i) \in D$ and let C be the SCC of q in $\mathcal{B}$. Since D is nontransient, there exists a cycle in $\mathcal{B} \times \mathbb{Z}_g$ containing (q, i), which induces a cycle in $\mathcal{B}$ containing q. This implies that C is nontransient. Hence, we can apply Lemma 3.18 and obtain a cycle of length $k \cdot g(C)$ in C for every sufficiently large $k \in \mathbb{N}$ ($k \geqslant m(C)$ suffices). Since g is a multiple of $g(C)$, C also contains a cycle of length $k \cdot g$ for every sufficiently large k. But every such cycle induces a cycle of the same length $k \cdot g$ in D. Hence, there exists $k \in \mathbb{N}$ such that D contains cycles of length $k \cdot g$ and $(k+1) \cdot g$. It follows that the period of D divides $\gcd(k \cdot g, (k+1) \cdot g) = g$. This proves that the period of D is exactly g. ■

PROOF OF THEOREM 3.3(ii). It remains to show the direction from left to right. Consider a well-behaved rDFA $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ for a regular language $L \subseteq \Sigma^*$. We prove that L is a finite

Boolean combination of regular left ideals and regular length languages. By Lemma 3.19 we can ensure that all nontransient SCCs in $\mathcal{B}$ have the same period g. This new rDFA $\mathcal{B}$ is also well-behaved since in fact any rDFA for L must be well-behaved; this follows from Proposition 3.9 and Corollary 3.10. Alternatively, one can verify that the transformation from Lemma 3.19 preserves the well-behavedness of $\mathcal{B}$.

A *path description* P is a sequence

$$C_k, (q_k, a_{k-1}, p_{k-1}), C_{k-1}, \dots, (q_3, a_2, p_2), C_2, (q_2, a_1, p_1), C_1, q_1 \tag{4}$$

where $C_k, \dots, C_1$ are pairwise distinct SCCs of $\mathcal{B}$, $q_1 = q_0$, (q_{i+1}, a_i, p_i) is a transition in $\mathcal{B}$ for all $1 \leqslant i \leqslant k-1$, $p_i, q_i \in C_i$ for all $1 \leqslant i \leqslant k-1$, and $q_k \in C_k$. A run π in $\mathcal{B}$ *respects* the path description P if the SCC-factorization of π is $\pi = \pi_k \tau_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$, π_i is a C_i-internal run from q_i to p_i for all $1 \leqslant i \leqslant k-1$, $\tau_i = q_{i+1} a_i p_i$ for all $1 \leqslant i \leqslant k-1$, and π_k is a C_k-internal run starting in q_k. Let L_P be the set of words $w \in \Sigma^*$ such that the unique run of $\mathcal{B}$ on w starting in q_0 respects the path description P. We can write $L = \bigcup_P (L_P \cap L)$ where P ranges over all path descriptions. Notice that the number of path descriptions is finite.

Let us fix a path description P as in (4). We prove that $L_P \cap L$ is a finite Boolean combination of regular left ideals and regular length languages. First, we claim that L_P is a finite Boolean combination of regular left ideals. Let $\Delta = \{(a \cdot p, a, p) \mid p \in Q, a \in \Sigma\}$ be the set of all transition triples and let $\Delta_P \subseteq \Delta$ be the set of transition triples contained in any of the SCCs $C_k, \dots, C_1$ together with the transition triples (q_{i+1}, a_i, p_i) for $1 \leqslant i \leqslant k-1$. A word $w \in \Sigma^*$ then belongs to L_P if and only if w belongs to the regular left ideal $\Sigma^* L_P$ and the run of $\mathcal{B}$ on w starting in q_0 does not use any transitions from $\Delta \setminus \Delta_P$. It is easy to construct for every $\tau \in \Delta \setminus \Delta_P$ an rDFA $\mathcal{D}_\tau$ which accepts all words w such that the run of $\mathcal{B}$ on w starting in q_0 uses the transition τ. Clearly, this language is a left ideal. In total we have $L_P = \Sigma^* L_P \setminus \bigcup_{\tau \in \Delta \setminus \Delta_P} \mathsf{L}(\mathcal{D}_\tau)$, which proves the claim.

If C_k is a transient SCC then $L_P \cap L$ is either empty or L_P, and we are done. For the rest of the proof we assume that C_k is nontransient. Recall that all nontransient SCCs in $\mathcal{B}$ have period g. Furthermore, C_k is well-behaved since it is reachable from q_0 according to the path description P. Let $C_k = \bigcup_{i=0}^{g-1} V_i$ be the partition from Lemma 3.18. We claim that $F \cap C_k$ is a union of some of the V_i's. Towards a contradiction assume that there exist states $p, q \in V_i$ where $p \in F$ and $q \notin F$. Let $r \geqslant m(C)$ be any number divisible by g. Then, by Lemma 3.18 there exist runs from p to p and from p to q, both of length r. This contradicts the fact that C_k is well-behaved.

Let π, π' be two runs of $\mathcal{B}$ starting from q_0 which respect P. We claim that $|\pi| \equiv |\pi'| \pmod{g}$ if and only if π and π' end in the same part V_i of C_k. Consider the SCC-factorizations $\pi = \pi_k \tau_{k-1} \pi_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$ and $\pi' = \pi'_k \tau_{k-1} \pi'_{k-1} \cdots \tau_2 \pi'_2 \tau_1 \pi'_1$. For all $1 \leqslant i \leqslant k-1$ the subruns π_i and π'_i start in q_i and end in p_i. If C_i is nontransient then $|\pi_i| \equiv |\pi'_i| \pmod{g}$ by Lemma 3.18, and otherwise $|\pi_i| = |\pi'_i| = 0$. This implies $|\tau_{k-1} \pi_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1| \equiv |\tau_{k-1} \pi'_{k-1} \cdots \tau_2 \pi'_2 \tau_1 \pi'_1| \pmod{g}$.

Also by Lemma 3.18 we know that $|\pi_k| \equiv |\pi'_k| \pmod{g}$ if and only if π and π' end in the same part V_i. This proves the claim.

It follows that we can write $L_P \cap L = L_P \cap K$ where

$$K = \{w \in \Sigma^* \mid \exists r \in R \colon |w| \equiv r \pmod{g}\}$$

for some $R \subseteq \{0, \ldots, g-1\}$. Since K is a regular length language, we have proved the claim that L is a Boolean combination of regular left ideals and regular length languages. This concludes the proof of Theorem 3.3(ii). ■

4. Randomized sliding window algorithms

Most of the work in the context of streaming uses randomness and/or approximation to design space- and time-efficient algorithms. For example, the AMS-algorithm [3] approximates the number of distinct elements in a stream with high probability in $O(\log m)$ space where m is the size of the universe. Furthermore, it is proved that any deterministic approximation algorithm and any randomized exact algorithm must use $\Omega(n)$ space [3]. On the other hand, the exponential histogram algorithm by Datar et al. [27] for approximating the number of 1's in a sliding window is a *deterministic* sliding window approximation algorithm that uses $O(\frac{1}{\epsilon}\log^2 n)$ bits. It is proven in [27] that $\Omega(\frac{1}{\epsilon}\log^2 n)$ bits are necessary even for randomized (Monte Carlo or Las Vegas) sliding window algorithms.

In this section, we will study if and how randomness helps for testing membership to regular languages over sliding windows. The main result of this section is a space tetrachotomy in the fixed-size sliding window model, stating that every regular language has optimal space complexity $\Theta(1)$, $\Theta^\infty(\log\log n)$, $\Theta^\infty(\log n)$ or $\Theta^\infty(n)$ if the streaming algorithms are randomized with two-sided error.

4.1 Randomized streaming algorithms

In the following, we will introduce probabilistic automata [78, 81] as a model of randomized streaming algorithms. With $[0,1]$ we denote the set of all real numbers r with $0 \leqslant r \leqslant 1$. A *probabilistic automaton* $\mathcal{P} = (Q, \Sigma, \iota, \rho, F)$ consists of a nonempty countable set of states Q, a finite alphabet Σ, an initial state distribution $\iota \colon Q \to [0,1]$, a transition probability function $\rho \colon Q \times \Sigma \times Q \to [0,1]$, and a set of final states $F \subseteq Q$, such that

(i) $\sum_{q \in Q} \iota(q) = 1$,

(ii) $\sum_{q \in Q} \rho(p, a, q) = 1$ for all $p \in Q$, $a \in \Sigma$.

If Q is infinite then this means of course that the above infinite sums converge to 1. This implies that these sums are absolutely convergent (all $\iota(q)$ and $\rho(p, a, q)$ are non-negative) and therefore the order of summation is not relevant.

If ι and ρ map into $\{0,1\}$, then $\mathcal{P}$ can be viewed as a deterministic automaton. We will refer to probabilistic automata also as *randomized streaming algorithms*.

For a word $w \in \Sigma^*$ and a state q we define the probability $\mathcal{P}(w,q)$ that $\mathcal{P}$ after reading the word w arrives in state q inductively over the length of w as follows, where $q \in Q$, $v \in \Sigma^*$ and $a \in \Sigma$:

— $\mathcal{P}(\varepsilon, q) = \iota(q)$ and

— $\mathcal{P}(va, q) = \sum_{p \in Q} \mathcal{P}(v,p) \cdot \rho(p,a,q)$.

Then the probability that $\mathcal{P}$ accepts the word w is

$$\Pr[\mathcal{P} \text{ accepts } w] = \sum_{q \in F} \mathcal{P}(w,q).$$

and the probability that $\mathcal{P}$ rejects the word w is

$$\Pr[\mathcal{P} \text{ rejects } w] = \sum_{q \in Q \setminus F} \mathcal{P}(w,q).$$

The *space* of $\mathcal{P}$ (or *number of bits used by* $\mathcal{P}$) is given by $s(\mathcal{P}) = \log |Q| \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$. We say that $\mathcal{P}$ is a randomized streaming algorithm for $L \subseteq \Sigma^*$ with error probability $0 \leqslant \lambda \leqslant 1$ if

— $\Pr[\mathcal{P} \text{ accepts } w] \geqslant 1 - \lambda$ for all $w \in L$,

— $\Pr[\mathcal{P} \text{ rejects } w] \geqslant 1 - \lambda$ for all $w \notin L$.

The error probability λ is also called a *two-sided error*. If we omit λ we choose $\lambda = 1/3$.

For a randomized streaming algorithm $\mathcal{P} = (Q, \Sigma, \iota, \rho, F)$ and a number $k \geqslant 1$ let $\mathcal{P}^{(k)}$ be the randomized streaming algorithm which simulates k instances of $\mathcal{P}$ in parallel with independent random choices and outputs the majority vote. Formally the states of $\mathcal{P}^{(k)}$ are multisets of size k over Q (using multisets instead of ordered tuples will yield a better space bound in Section 4.3). Therefore, $s(\mathcal{P}^{(k)}) \leqslant k \cdot s(\mathcal{P})$.

LEMMA 4.1 (probability amplification). *For all $0 < \lambda' < \lambda < 1/2$ there exists a number $k = \mathcal{O}(\log\left(\frac{1}{\lambda'}\right) \cdot \left(\frac{1}{2} - \lambda\right)^{-2})$ such that for all randomized streaming algorithms $\mathcal{P}$ and all $w \in \Sigma^*$ we have:*

(i) If $\Pr[\mathcal{P}$ accepts $w] \geqslant 1 - \lambda$ then $\Pr[\mathcal{P}^{(k)}$ accepts $w] \geqslant 1 - \lambda'$.

(ii) If $\Pr[\mathcal{P}$ rejects $w] \geqslant 1 - \lambda$ then $\Pr[\mathcal{P}^{(k)}$ rejects $w] \leqslant 1 - \lambda'$.

PROOF. We will choose k later. Let $X_1, \dots, X_k$ be independent Bernoulli random variables with $\Pr[X_i = 0] = \lambda$ and $\Pr[X_i = 1] = 1 - \lambda$. Let $X = \sum_{i=1}^k X_i$ with expectation $\mu = k(1-\lambda)$. Suppose that $\mathcal{P}$ accepts w with probability $\geqslant 1 - \lambda$, i.e., $\mathcal{P}$ rejects w with probability at most λ. Then, $\mathcal{P}^{(k)}$ rejects w with probability at most $\Pr[X \leqslant k/2]$. By the Chernoff bound [75, Theorem 4.5], for any $0 < \delta < 1$ we have

$$\Pr[X \leqslant (1-\delta)\mu] \leqslant \exp\left(-\frac{\mu\delta^2}{2}\right). \tag{5}$$

By choosing $\delta = 1 - \frac{1}{2(1-\lambda)}$ we get $(1-\delta)\mu = k/2$. Then, (5) gives the following estimate:

$$\begin{aligned}\Pr[X \leqslant k/2] &\leqslant \exp\left(-\frac{k(1-\lambda)(1-\frac{1}{2(1-\lambda)})^2}{2}\right)\\ &= \exp\left(-\frac{k}{2}\cdot\frac{(\frac{1}{2}-\lambda)^2}{1-\lambda}\right)\\ &\leqslant \exp\left(-\frac{k}{2}\cdot\left(\frac{1}{2}-\lambda\right)^2\right).\end{aligned}$$

By choosing

$$k \geqslant 2\cdot\ln\left(\frac{1}{\lambda'}\right)\cdot\left(\frac{1}{2}-\lambda\right)^{-2}$$

we can bound the probability that $\mathcal{P}^{(k)}$ rejects w by λ'. Statement (ii) can be shown analogously. ∎

4.2 Space tetrachotomy

A *randomized sliding window algorithm* for a language L and window size n is a randomized streaming algorithm for $\mathsf{SW}_n(L)$. The *randomized space complexity* $\mathsf{F}^{\mathsf{r}}_L(n)$ of L in the fixed-size sliding window model is the minimal space complexity $s(\mathcal{P}_n)$ of a randomized sliding window algorithm $\mathcal{P}_n$ for L and window size n. For this to be well-defined it is important that we require the error probability to be at most $1/3$.

Before we investigate randomized streaming algorithms in more detail, let us first comment on the fact that in our definition of randomized SW-algorithms we allow arbitrary (even irrational) probabilities in the state transitions. On the other hand, in all correctness proofs for our randomized SW-algorithms we only need the fact that the probabilities are from a certain interval $I \subseteq [0,1]$. Therefore, if d is the length of the interval I, we can always choose a probability $p \in I$ with $O(\log_2(1/d))$ bits such that the algorithm still achieves an error probability of at most $1/3$. However, the size of the interval I may depend on the window size n; more precisely it may shrink when n grows. In particular, the number of bits needed to write down the probabilities used in $\mathcal{P}_n$ (the algorithm for window size n) may grow with n. One might argue that these bits should also enter the definition of the space used by the algorithm. The reason why we do not take these bits into account is the same as why we do not consider the space for internal calculations; see Remark 2.7. Assume for instance that we need to implement a randomized branching with probabilities p and $1-p$ and let m be the number of bits of p. Let us moreover assume that we have a randomized machine model that apart from deterministic commands can only toss fair coins. It is not difficult to see that one can implement a biased coin with probabilities p and $1-p$ using m many fair coins. For this, one also needs some additional control structure for which $O(\log m)$ bits are needed (basically to store the program counter).

But this is internal space that we do not take into account in our definition of space (as justified in Remark 2.7).[5]

Clearly we have $\mathsf{F}^{\mathsf{r}}_L(n) \leqslant \mathsf{F}_L(n)$. Furthermore, we prove that randomness can reduce the space complexity at most exponentially:

LEMMA 4.2. *For any language L we have $\mathsf{F}_L(n) = 2^{O(\mathsf{F}^{\mathsf{r}}_L(n))}$.*

PROOF. Rabin proved that any probabilistic finite automaton with a so-called isolated cut-point can be made deterministic with an exponential size increase [81]. Let $\mathcal{P} = (Q, \Sigma, \iota, \rho, F)$ be a probabilistic finite automaton with m states. Suppose that $\lambda \in [0, 1]$ is an *isolated cut-point* with radius $\delta > 0$, i.e., $|\Pr[\mathcal{P} \text{ accepts } w] - \lambda| \geqslant \delta$ for all $w \in \Sigma^*$. Then, $L = \{w \in \Sigma^* \mid \Pr[\mathcal{P} \text{ accepts } w] \geqslant \lambda\}$ is recognized by a DFA $\mathcal{A}$ with at most $(1 + m/\delta)^{m-1} = 2^{O(m \log m)}$ states [81, Theorem 3].

Now, let $\mathcal{P}_n$ be a minimal probabilistic finite automaton for $\mathsf{SW}_n(L)$ with m states and error probability $\leqslant 1/3$. Since $\mathcal{P}_n$ has $1/2$ as an isolated cutpoint with radius $1/2 - 1/3 = 1/6$, there exists an equivalent DFA Q_n with $|Q_n| \leqslant 2^{O(m \log m)}$ states. The statement follows from $\mathsf{F}_L(n) \leqslant \log |Q_n| = O(m \log m) = O(2^{\mathsf{F}^{\mathsf{r}}_L(n)} \cdot \mathsf{F}^{\mathsf{r}}_L(n))$, which is bounded by $2^{O(\mathsf{F}^{\mathsf{r}}_L(n))}$. ■

In this section, we will prove Theorem 1.4, which is a tetrachotomy for the randomized space complexity of regular languages in the fixed-size sliding window model. Let us rephrase Theorem 1.4 and split it into three upper bounds and three lower bounds.

THEOREM 4.3. *Let $L \subseteq \Sigma^*$ be a regular language.*

(1) If $L \in \langle \mathbf{ST}, \mathbf{Len} \rangle$ then $\mathsf{F}^{\mathsf{r}}_L(n) = O(1)$.
(2) If $L \notin \langle \mathbf{ST}, \mathbf{Len} \rangle$ then $\mathsf{F}^{\mathsf{r}}_L(n) = \Omega^{\infty}(\log \log n)$.
(3) If $L \in \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$ then $\mathsf{F}^{\mathsf{r}}_L(n) = O(\log \log n)$.
(4) If $L \notin \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$ then $\mathsf{F}^{\mathsf{r}}_L(n) = \Omega^{\infty}(\log n)$.
(5) If $L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$ then $\mathsf{F}^{\mathsf{r}}_L(n) = O(\log n)$.
(6) If $L \notin \langle \mathbf{LI}, \mathbf{Len} \rangle$ then $\mathsf{F}^{\mathsf{r}}_L(n) = \Omega^{\infty}(n)$.

Points (1) and (5) already hold in the deterministic setting, see Theorem 3.3. In the next sections we prove points (2), (3), (4), and (6).

We first transfer Lemma 2.6 to the fixed-size model in the randomized setting:

LEMMA 4.4. *Let Σ be a finite alphabet. For any function $s(n)$, the class $\{L \subseteq \Sigma^* \mid \mathsf{F}^{\mathsf{r}}_L(n) = O(s(n))\}$ forms a Boolean algebra.*

PROOF. Let $L \subseteq \Sigma^*$ be a language and $n \in \mathbb{N}$ a window size. If $\mathcal{P}_n$ is a randomized SW-algorithm for L and window size n then $\overline{\mathcal{P}}_n$ is a randomized SW-algorithm for $\Sigma^* \setminus L$ and window size

5 The reader may view this control structure as additional ε-transitions in a probabilistic automaton that are taken with probability $1/2$.

n, where $\overline{\mathcal{P}}_n$ simulates $\mathcal{P}_n$ and returns the negated output. Let $\mathcal{P}_n$ and $\mathcal{Q}_n$ be randomized SW-algorithms for K and L, respectively, and window size n. By Lemma 4.1 we can reduce their error probabilities to 1/6 with a constant space increase. Then, the algorithm which simulates $\mathcal{P}_n$ and $\mathcal{Q}_n$ in parallel and returns the disjunction of the outputs is a randomized SW-algorithm for $K \cup L$ and window size n. Its error probability is at most 1/3 by the union bound. ■

4.3 The Bernoulli counter

The crucial algorithmic tool for the proof of Theorem 4.3(3) is a simple probabilistic counter. It is inspired by the approximate counter by Morris [36, 76], which uses $O(\log\log n)$ bits. For our purposes, it suffices to detect whether the counter has exceeded a certain threshold, which can be accomplished using only $O(1)$ bits.

Formally, a *probabilistic counter* is a probabilistic automaton

$$\mathcal{Z} = (C, \{\mathtt{inc}\}, \iota, \rho, F)$$

over the unary alphabet $\{\mathtt{inc}\}$. States in F are called *high* and states in $C \setminus F$ are called *low*. We make the restriction that there is a low state $c_0 \in C$ such that $\iota(c_0) = 1$ (and hence $\iota(c) = 0$ for all $c \in C \setminus \{c_0\}$); thus $\mathcal{Z}$ has a unique initial state c_0 (which must be low) and we write $\mathcal{Z} = (C, \{\mathtt{inc}\}, c_0, \rho, F)$. This restriction is not really important (and can in fact be achieved for every probabilistic automaton by adding a new state), but it will simplify our constructions.

In the following we write $\mathcal{Z}(k, c)$ for $\mathcal{Z}(\mathtt{inc}^k, c)$ ($k \geqslant 0$, $c \in C$), which is the probability that $\mathcal{Z}$ arrives in state c after k increments. Moreover, $\mathcal{Z}^{\mathsf{hi}}(k)$ is the probability that $\mathcal{Z}$ is in a high state after k increments (this is the same as $\Pr[\mathcal{Z} \text{ accepts } \mathtt{inc}^k]$). Given numbers $0 \leqslant \ell < h$ we say that $\mathcal{Z}$ is an (h, ℓ)*-counter with error probability* $\lambda < \frac{1}{2}$ if for all $k \in \mathbb{N}$ we have:

— If $k \leqslant \ell$, then $\mathcal{Z}^{\mathsf{hi}}(k) \leqslant \lambda$.

— If $k \geqslant h$, then $\mathcal{Z}^{\mathsf{hi}}(k) \geqslant 1 - \lambda$.

In other words, a probabilistic counter can distinguish with high probability between values below ℓ and values above h but does not give any guarantees for counter values strictly between ℓ and h. A *Bernoulli counter* is a probabilistic counter $\mathcal{Z}_p$ that is parameterized by a probability $0 < p < 1$ and that has the state set $\{0, 1\}$, where 0 is a low state and 1 is a high state. Initially the counter is in the state $x = 0$. On every increment we set $x = 1$ with probability p; the state remains unchanged with probability $1 - p$. We have

$$\mathcal{Z}_p^{\mathsf{hi}}(k) = \mathcal{Z}_p(k, 1) = 1 - (1 - p)^k.$$

Let us first show the following claim.

LEMMA 4.5. *For all $h > 0$, $0 < \xi < 1$ and $0 < \ell \leqslant (1 - \xi)h$ there exists $0 < p < 1$ such that $\mathcal{Z}_p$ is an (h, ℓ)-counter with error probability $1/2 - \xi/8$.*

PROOF. We need to choose $0 < p < 1$ such that

(i) $1-(1-p)^{(1-\xi)h} \leqslant 1/2 - \xi/8$, or equivalently, $1/2+\xi/8 \leqslant (1-p)^{(1-\xi)h}$, and

(ii) $(1-p)^h \leqslant 1/2-\xi/8$, or equivalently, $(1-p)^{(1-\xi)h} \leqslant (1/2-\xi/8)^{1-\xi}$.

It suffices to show

$$\frac{1}{2}+\frac{\xi}{8} \leqslant \left(\frac{1}{2}-\frac{\xi}{8}\right)^{1-\xi}. \tag{6}$$

Then one can take for instance $p = 1-(1/2-\xi/8)^{1/h} \in (0,1)$, which satisfies (ii). Moreover, (i) is satisfied due to (6).

Taking logarithms shows that (6) is equivalent to

$$\ln(4+\xi)-\ln 8 \leqslant (1-\xi)\cdot(\ln(4-\xi)-\ln 8),$$

which can be rearranged to $\ln(4+\xi) \leqslant \ln(4-\xi)+\xi(\ln 8-\ln(4-\xi))$. Since $\ln 8-\ln(4-\xi) \geqslant \ln 8-\ln 4 = \ln 2$, it suffices to prove

$$\ln(4+\xi) \leqslant \ln(4-\xi)+\xi\ln 2. \tag{7}$$

One can verify $3\ln 2 \approx 2.0794 \geqslant 2$. We have

$$\begin{aligned} 4+\xi &\leqslant 4+(3\ln 2-1)\xi \\ &= 4+(4\ln 2-1)\xi-\xi\ln 2 \\ &\leqslant 4+(4\ln 2-1)\xi-\xi^2\ln 2 \\ &= (4-\xi)(\xi\ln 2+1). \end{aligned}$$

By taking logarithms and plugging in $\ln x \leqslant x-1$ for all $x > 0$, we obtain

$$\ln(4+\xi) \leqslant \ln(4-\xi)+\ln(\xi\ln 2+1) \leqslant \ln(4-\xi)+\xi\ln 2.$$

This proves (7) and hence (6), and thus the lemma. ■

PROPOSITION 4.6. *For all $h > 0$, $0 < \xi < 1$, $0 < \ell \leqslant (1-\xi)h$ and $0 < \lambda' < 1/2$ there exists an (h,ℓ)-counter $\mathcal{Z}$ with error probability λ' which uses $O(\log\log(1/\lambda')+\log(1/\xi))$ bits.*

PROOF. Take the (h,ℓ)-counter $\mathcal{Z}_p$ from Lemma 4.5, whose error probability is $\lambda := 1/2-\xi/8$. To $\mathcal{Z}_p$ we apply Lemma 4.1, which states that we need to run $k = O(\log(\frac{1}{\lambda'})\cdot\frac{1}{\xi^2})$ independent copies to reduce the error probability to λ'. The states of $\mathcal{Z}_p^{(k)}$ are multisets over $\{0,1\}$ of size k, which can be encoded with $O(\log k) = O(\log\log\frac{1}{\lambda'}+\log\frac{1}{\xi})$ bits by specifying the number of 1-bits in the multiset. Note that the unique initial state of $\mathcal{Z}_p^{(k)}$ is the multiset with k occurrences of 0. ■

4.4 Suffix-free languages

In this section, we prove Theorem 4.3(3). Since languages in **ST** $\cup$ **Len** have constant space (deterministic) SW-algorithms it suffices by Lemma 4.4 to show:

THEOREM 4.7. *If L is regular and suffix-free then $\mathsf{F}^{\mathsf{r}}_L(n) = O(\log\log n)$.*

Fix a regular suffix-free language $L \subseteq \Sigma^*$ and let $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ be an rDFA for L where all states are reachable. Excluding the trivial case $L = \emptyset$, we assume that $\mathcal{B}$ contains at least one final state. Furthermore, since L is suffix-free, any run in $\mathcal{B}$ contains at most one final state. Therefore, we can assume that F contains exactly one final state q_F, and all outgoing transitions from q_F lead to a sink state. For a stream $w \in \Sigma^*$ define the function $\ell_w \colon Q \to \mathbb{N} \cup \{\infty\}$ by

$$\ell_w(q) = \inf\{k \in \mathbb{N} \mid \mathsf{last}_k(w) \cdot q = q_F\}, \tag{8}$$

where we set $\inf(\emptyset) = \infty$ (note that $\{k \in \mathbb{N} \mid \mathsf{last}_k(w) \cdot q = q_F\}$ is either empty or a singleton set). Notice that $\mathsf{last}_n(w) \in L$ if and only if $\ell_w(q_0) = n$. Also, it holds $\ell_w(q_F) = 0$ for every $w \in \Sigma^*$. A *deterministic* streaming algorithm can maintain the function ℓ_w where $w \in \Sigma^*$ is the stream prefix read so far: If a symbol $a \in \Sigma$ is read, we can determine

$$\ell_{wa}(q) = \begin{cases} 0, & \text{if } q = q_F, \\ 1 + \ell_w(a \cdot q), & \text{otherwise,} \end{cases} \tag{9}$$

where $1 + \infty = \infty$. Storing $\ell_w(q)$ may require up to $\log |w|$ bits. Therefore, if an SW-algorithm for window size n wants to store all $\ell_w(q)$ for $q \in Q$ (w is the input stream and not just the sliding window), then the space is not bounded in the window size. The solution is to use probabilistic counters with suitable threshold values ℓ and h.

Let $n \in \mathbb{N}$ be a window size. The randomized sliding window algorithm $\mathcal{P}_n$ for L consists of two parts: a constant-space threshold algorithm $\mathcal{T}_n$, which rejects with high probability whenever $\ell_w(q_0) \geqslant 2n$, and a modulo counting algorithm $\mathcal{M}_n$, which maintains ℓ_w modulo a random prime number with $O(\log\log n)$ bits.

LEMMA 4.8 (threshold counting). *There exists a randomized streaming algorithm $\mathcal{T}_n$ with $O(1)$ bits such that for all $w \in \Sigma^*$ we have:*

- $\Pr[\mathcal{T}_n$ *accepts* $w] \geqslant 2/3$, *if* $\ell_w(q_0) \leqslant n$, *and*
- $\Pr[\mathcal{T}_n$ *rejects* $w] \geqslant 2/3$, *if* $\ell_w(q_0) \geqslant 2n$.

PROOF. By Proposition 4.6 there is a $(2n, n)$-counter $\mathcal{Z} = (C, \{\mathtt{inc}\}, c_0, \rho, F)$ with error probability 1/3 which uses $O(1)$ space. Let $c_\infty \in F$ be an arbitrary high state. The algorithm $\mathcal{T}_n$ maintains for every $q \in Q$ an instance $\mathcal{Z}_q$ of the $(2n, n)$-counter $\mathcal{Z}$. The input alphabet of $\mathcal{Z}_q$ is Σ (instead of $\{\mathtt{inc}\}$) and the probability $\mathcal{Z}_q(w, c)$ of reaching $c \in C$ after reading the word

$w \in \Sigma^*$ will satisfy

$$\mathcal{Z}_q(w, c) = \mathcal{Z}(\ell_w(q), c), \tag{10}$$

where we set $\mathcal{Z}(\infty, c_\infty) = 1$ (and $\mathcal{Z}(\infty, c) = 0$ for all states $c \neq c_\infty$) We initialize $\mathcal{Z}_q$ in order to get (10) for $w = \varepsilon$. To this end, we distinguish whether state q has finite or infinite value $\ell_\varepsilon(q)$. Notice that $\ell_\varepsilon(q)$ is finite if and only if the final state q_F can be reached from state q by only reading the padding symbol $\square$. If $\ell_\varepsilon(q) < \infty$, then we initialize $\mathcal{Z}_q$ in its initial state c_0 and then execute $\ell_\varepsilon(q)$ increments. If $\ell_\varepsilon(q) = \infty$, we set $\mathcal{Z}_q$ to state c_∞ (with probability one). Given an input symbol $a \in \Sigma$, we compute the new states of the counters $\mathcal{Z}_q$ as follows: Assume that c_q is the current state of $\mathcal{Z}_q$. First we set $\mathcal{Z}_{q_F}$ to the initial state c_0. This ensures (10) for q_F since $\ell_{wa}(q_F) = 0$ and $\mathcal{Z}(0, c_0) = 1$. For $q \in Q \setminus \{q_F\}$ we set the new state of $\mathcal{Z}_q$ with probability $\rho(c_{a\cdot q}, \mathtt{inc}, c)$ to c. This ensures again (10):

$$\begin{aligned} \mathcal{Z}_q(wa, c) &= \sum_{c' \in C} \mathcal{Z}_{a\cdot q}(w, c') \cdot \rho(c', \mathtt{inc}, c) \\ &= \sum_{c' \in C} \mathcal{Z}(\ell_w(a \cdot q), c') \cdot \rho(c', \mathtt{inc}, c) \\ &= \mathcal{Z}(\ell_w(a \cdot q) + 1, c) = \mathcal{Z}(\ell_{wa}(q), c). \end{aligned}$$

The algorithm $\mathcal{T}_n$ accepts the word w if and only if $\mathcal{Z}_{q_0}$ is in a low state after reading w. Note that this happens with probability $1 - \mathcal{Z}_{q_0}^{\mathsf{hi}}(w) = 1 - \mathcal{Z}^{\mathsf{hi}}(\ell_w(q_0))$ ($\mathcal{Z}^{\mathsf{hi}}(k)$ is the probability that $\mathcal{Z}$ is in a high state after k increments). Correctness follows from the fact that $\mathcal{Z}$ is a $(2n, n)$-counter with error probability 1/3:

$$\Pr[\mathcal{T}_n \text{ accepts } w] = 1 - \mathcal{Z}^{\mathsf{hi}}(\ell_w(q_0)) \begin{cases} \geqslant 2/3 \text{ if } \ell_w(q_0) \leqslant n, \\ \leqslant 1/3 \text{ if } \ell_w(q_0) \geqslant 2n. \end{cases}$$

This proves the lemma. ■

Also note that the randomized SW-algorithm from the previous proof uses several probabilistic counters $\mathcal{Z}_q$ (one for each state $q \in Q$) and they all have the same parameters $\ell = n$ and $h = 2n$. For each new input symbol, a subset of these counters have to be incremented. These increments are not needed to be independent. Hence, in each step, only the random bits for incrementing a single $(2n, n)$-counter are needed. These random bits can be used for all $\mathcal{Z}_q$ that have to be incremented.

We now come to the modulo counting algorithm, for which we use the following simple fact on prime numbers.

LEMMA 4.9. *There is a constant c such that for every large enough $m \in \mathbb{N}$ and all $0 \leqslant a, b \leqslant m$ with $a \neq b$ the following holds: If the prime number p is picked uniformly at random among all prime numbers that are no greater than $c \log m \log\log m$, then* $\Pr[a \equiv b \pmod{p}] \leqslant 1/3$.

PROOF. Let p_i be the i-th prime number. It is known that $p_i < i \cdot (\ln i + \ln\ln i)$ for $i \geqslant 6$ [83, page 3.13]. Fix an m and let k be the first natural number such that $\prod_{i=1}^{k} p_i \geqslant m$. Since $\prod_{i=1}^{k} p_i \geqslant 2^k$, we have $k \leqslant \log m$ and hence $p_{3k} \leqslant 3 \log m \cdot (\ln(3\log m) + \ln\ln(3\log m)) \leqslant c \log m \log\log m$ for some constant c and all large enough m.

Since $-m \leqslant a - b \leqslant m$ and any product of at least $k+1$ pairwise distinct primes exceeds m, the integer $a - b \neq 0$ has at most k prime factors. Hence, for a randomly chosen prime $p \in \{p_1, \dots, p_{3k}\}$ we have $\Pr[a \equiv b \pmod p] \leqslant 1/3$. ■

LEMMA 4.10 (modulo counting). *There exists a randomized streaming algorithm $\mathcal{M}_n$ with $O(\log\log n)$ bits such that for all $w \in \Sigma^*$ we have:*

— $\Pr[\mathcal{M}_n$ *accepts* $w] = 1$, *if* $\ell_w(q_0) = n$, *and*

— $\Pr[\mathcal{M}_n$ *rejects* $w] \geqslant 2/3$, *if* $\ell_w(q_0) < 2n$ *and* $\ell_w(q_0) \neq n$.

PROOF. Let c be the constant from Lemma 4.9 which is applied with $m = 2n$. The algorithm $\mathcal{M}_n$ initially picks a random prime $p \leqslant c \log(2n) \log\log(2n)$ which is stored throughout the run using $O(\log\log n)$ bits. Then, after reading $w \in \Sigma^*$, $\mathcal{M}_n$ stores for every $q \in Q$ a bit telling whether $\ell_w(q) < \infty$ and, if the latter holds, the value $\ell_w(q) \bmod p$ using in total $O(|Q| \cdot \log\log n)$ bits. These numbers can be maintained according to (9). The algorithm accepts if and only if $\ell_w(q_0) \equiv n \pmod p$.

If $\ell_w(q_0) = n$ then the algorithm always accepts. Now, assume $\ell_w(q_0) < 2n$ and $\ell_w(q_0) \neq n$. Then Lemma 4.9 with $a = \ell_w(q_0)$ and $b = n$ yields $\Pr[\ell_w(q_0) \equiv n \pmod p] \leqslant 1/3$. Therefore, $\mathcal{M}_n$ rejects with probability at least $2/3$. ■

It is worth mentioning that in the above modulo counting algorithm the errors after reading different prefixes of an input stream w are not independent. If for instance $w = uu$ with $|u|$ the window size, then the algorithm will make an error after reading u if and only if it makes an error after reading uu. This is of course due to the fact that the only random choice is made at the very beginning. After this random choice, the algorithm proceeds deterministically.

By combining the algorithms from Lemma 4.8 and Lemma 4.10 we can prove Theorem 4.7. The algorithm $\mathcal{P}_n$ is the conjunction of the threshold algorithm $\mathcal{T}_n$ and the modulo counting algorithm $\mathcal{M}_n$. Recall that $\mathrm{last}_n(w) \in L$ if and only if $\ell_w(q_0) = n$. If $\ell_w(q_0) = n$ then $\mathcal{T}_n$ accepts with probability at least $2/3$ and $\mathcal{M}_n$ accepts with probability 1; hence $\mathcal{P}_n$ accepts with probability at least $2/3$. If $\ell_w(q_0) \neq n$ then $\mathcal{M}_n$ or $\mathcal{T}_n$ rejects with probability at least $2/3$. Hence, $\mathcal{P}_n$ rejects with probability at least $2/3$.

4.5 Lower bounds

In this section, we prove the lower bounds from Theorem 4.3. Point (2) from Theorem 4.3 follows easily from the relation $\mathsf{F}_L(n) = 2^{O(\mathsf{F}_L^{\mathsf{r}}(n))}$ (Lemma 4.2). Since every language $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$ satisfies $\mathsf{F}_L(n) = \Omega^\infty(\log n)$ (Theorem 3.2 and 3.3), it also satisfies $\mathsf{F}_L^{\mathsf{r}}(n) = \Omega^\infty(\log\log n)$.

For (4) and (6) we apply known lower bounds from communication complexity by deriving a randomized communication protocol from a randomized SW-algorithm. This is in fact a standard technique for obtaining lower bounds for streaming algorithms; see e.g. [84].

We present the necessary background from communication complexity; see [71] for a detailed introduction. We only need the one-way setting where Alice sends a single message to Bob. Consider a function $f\colon X \times Y \to \{0,1\}$ for some finite sets X and Y. A *randomized one-way (communication) protocol* $P = (a, b)$ consists of functions $a\colon X \times R_a \to \{0,1\}^*$ and $b\colon \{0,1\}^* \times Y \times R_b \to \{0,1\}$, where R_a and R_b are finite sets of random choices of Alice and Bob, respectively. The *cost* of P is the maximum number of bits transmitted by Alice, i.e.

$$\mathrm{cost}(P) = \max_{x\in X, r_a\in R_a} |a(x, r_a)|.$$

Moreover, probability distributions are given on R_a and R_b. Alice computes from her input $x \in X$ and a random choice $r_a \in R_a$ the value $a(x, r_a)$ and sends it to Bob. Using this value, his input $y \in Y$ and a random choice $r_b \in R_b$ he outputs $b(a(x, r_a), y, r_b)$. The random choices $r_a \in R_a, r_b \in R_b$ are chosen independently of each other. The protocol P *computes* f if for all $(x, y) \in X \times Y$ we have

$$\Pr_{r_a\in R_a, r_b\in R_b}[P(x, y) \neq f(x, y)] \leqslant \frac{1}{3}, \tag{11}$$

where $P(x, y)$ is the random variable $b(a(x, r_a), y, r_b)$. The *randomized one-way communication complexity* of f is the minimal cost among all one-way randomized protocols that compute f (with an arbitrary number of random bits). The choice of the constant $1/3$ in (11) is arbitrary in the sense that changing the constant to any $\lambda < 1/2$ only changes the communication complexity by a constant (depending on λ), see [71, p. 30]. We will use established lower bounds on the randomized one-way communication complexity of some functions.

THEOREM 4.11 ([70, Theorem 3.7 and 3.8]). *Let $n \in \mathbb{N}$.*

— *The* index function

$$\mathrm{IDX}_n\colon \{0,1\}^n \times \{1, \ldots, n\} \to \{0,1\}$$

with $\mathrm{IDX}_n(a_1 \cdots a_n, i) = a_i$ *has randomized one-way communication complexity* $\Theta(n)$.

— *The* greater-than function

$$\mathrm{GT}_n\colon \{1, \ldots, n\} \times \{1, \ldots, n\} \to \{0,1\}$$

with $\mathrm{GT}_n(i, j) = 1$ *if and only if* $i > j$ *has randomized one-way communication complexity* $\Theta(\log n)$.

The upper bounds from these statements also hold for the deterministic one-way communication complexity as witnessed by the trivial deterministic protocols. We also define the *equality function*

$$\mathrm{EQ}_n\colon \{1, \ldots, n\} \times \{1, \ldots, n\} \to \{0,1\}$$

by $\mathrm{EQ}_n(i,j) = 1$ if and only if $i = j$. Its randomized one-way communication complexity is $\Theta(\log\log n)$ whereas its deterministic one-way communication complexity is $\Theta(\log n)$ [71].

We start with the proof of (6) from Theorem 4.3, which extends our linear space lower bound from the deterministic setting to the randomized setting.

PROPOSITION 4.12. *If $L \in \mathbf{Reg} \setminus \langle \mathbf{LI}, \mathbf{Len}\rangle$ then $\mathsf{F}^{\mathsf{r}}_L(n) = \Omega^\infty(n)$.*

PROOF. By Theorem 3.3(ii) any rDFA for L is not well-behaved and by Lemma 3.8 there exist words $u = u_1u_2, v = v_1v_2, z \in \Sigma^*$ such that $|u_1| = |v_1|$, $|u_2| = |v_2|$ and L separates $u_2\{u,v\}^*z$ and $v_2\{u,v\}^*z$. Let $\eta\colon \{0,1\}^* \to \{u,v\}^*$ be the injective homomorphism defined by $\eta(0) = u$ and $\eta(1) = v$.

Now, consider a randomized SW-algorithm $\mathcal{P}_n$ for L and window size $n = |u_2| + |u| \cdot m + |z|$ for some $m \geqslant 1$. We describe a randomized one-way communication protocol for IDX_m.

Let $\alpha = \alpha_1 \cdots \alpha_m \in \{0,1\}^m$ be Alice's input and $i \in \{1,\dots,m\}$ be Bob's input. Alice reads $\eta(\alpha)$ into $\mathcal{P}_n$ (using here random choices in order to select the outgoing transitions in $\mathcal{P}_n$) and sends the memory state using $O(s(\mathcal{P}_n))$ bits to Bob. Continuing from the received state, Bob reads u^iz into $\mathcal{P}_n$. Then, the active window is

$$\mathsf{last}_n(\eta(\alpha)u^iz) = s\,\eta(\alpha_{i+1}\cdots\alpha_m)u^iz \in \{u_2, v_2\}\{u,v\}^*z$$

where $s = u_2$ if $\alpha_i = 0$ and $s = v_2$ if $\alpha_i = 1$. Hence, from the output of $\mathcal{P}_n$ Bob can determine whether $\alpha_i = 1$. The cost of the protocol is bounded by $O(s(\mathcal{P}_n))$ and must be at least $\Omega(m) = \Omega(n)$ by Theorem 4.11. We conclude that $s(\mathcal{P}_n) = \Omega(n)$ for infinitely many n and therefore $\mathsf{F}^{\mathsf{r}}_L(n) = \Omega^\infty(n)$. ∎

Next, we prove point (4) from Theorem 4.3. For that, we need the following automaton property, where $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ is an rDFA.

A pair $(p,q) \in Q \times Q$ of states is called *synchronized* if there exist words $x,y,z \in \Sigma^*$ with $|x| = |y| = |z| \geqslant 1$ such that

$$q \xleftarrow{x} q \xleftarrow{y} p \xleftarrow{z} p.$$

A pair $(p,q) \in Q \times Q$ is called *reachable* if p and q are reachable from q_0 and (p,q) is called *F-consistent* if either $\{p,q\} \cap F = \emptyset$ or $\{p,q\} \subseteq F$. We remark that synchronized state pairs have no connection to the notion of synchronizing words.

Our main technical result for synchronized pairs is the following:

LEMMA 4.13. *Assume that every reachable synchronized pair in $\mathcal{B}$ is F-consistent. Then, $\mathsf{L}(\mathcal{B})$ belongs to $\langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len}\rangle$.*

For the proof of Lemma 4.13 we need two lemmas.

LEMMA 4.14. *A state pair (p,q) is synchronized if and only if p and q are nontransient and there exists a nonempty run from p to q whose length is a multiple of $|Q|!$.*

PROOF. First assume that (p, q) is synchronized. Let $x, y, z \in \Sigma^+$ with $|x| = |y| = |z| = k$ such that $q \xleftarrow{x} q \xleftarrow{y} p \xleftarrow{z} p$. Then, p and q are nontransient and we have

$$q \xleftarrow{x^{|Q|!-1}y} p,$$

where $x^{|Q|!-1}y$ has length $(|Q|! - 1) \cdot k + k = |Q|! \cdot k$.

Conversely, assume that p and q are nontransient and there exists a nonempty run $q \xleftarrow{y} p$ whose length is divided by $|Q|!$. Since the states p and q are nontransient, there are words x and z of length at most $|Q|$ with $q \xleftarrow{x} q$ and $p \xleftarrow{z} p$. These words can be pumped up to have length $|y|$. ■

Let $Q = T \cup N$ be the partition of the state set into the set T of transient states and the set N of nontransient states. A function $\beta \colon \mathbb{N} \to \{0, 1\}$ is *k-periodic* if $\beta(i) = \beta(i + k)$ for all $i \in \mathbb{N}$.

LEMMA 4.15. *Assume that every reachable synchronized pair in $\mathcal{B}$ is F-consistent. Then, for every word $v \in \Sigma^*$ of length at least $|Q|! \cdot (|T|+1)$ there exists a $|Q|!$-periodic function $\beta_v \colon \mathbb{N} \to \{0, 1\}$ such that the following holds: If $w \in \Sigma^* v$ and $w \cdot q_0 \in N$, then we have $w \in \mathsf{L}(\mathcal{B})$ if and only if $\beta(|w|) = 1$.*

PROOF. Let $v = a_k \cdots a_2 a_1$ with $k \geqslant |Q|! \cdot (|T| + 1)$, and consider the run

$$q_k \xleftarrow{a_k} \cdots \xleftarrow{a_2} q_1 \xleftarrow{a_1} q_0 \tag{12}$$

of $\mathcal{B}$ on v. Clearly, each transient state can occur at most once in the run. First notice that for each $0 \leqslant i \leqslant |Q|! - 1$ at least one of the states in

$$Q_i = \{q_{i+j|Q|!} \mid 0 \leqslant j \leqslant |T|\}$$

is nontransient because otherwise the set would contain $|T| + 1$ pairwise distinct transient states. Furthermore, we claim that the nontransient states in Q_i are either all final or all nonfinal: Take two nontransient states $q_{i+j_1|Q|!}$ and $q_{i+j_2|Q|!}$ with $j_1 < j_2$. Since we have a run of length $(j_2 - j_1)|Q|!$ from $q_{i+j_1|Q|!}$ to $q_{i+j_2|Q|!}$, these two states form a synchronized pair by Lemma 4.14, which by assumption must be F-consistent.

Now, define $\beta_v \colon \mathbb{N} \to \{0, 1\}$ by

$$\beta_v(m) = \begin{cases} 1, & \text{if the states in } Q_{m \bmod |Q|!} \cap N \text{ are final,} \\ 0, & \text{if the states in } Q_{m \bmod |Q|!} \cap N \text{ are nonfinal,} \end{cases}$$

which is well-defined by the remarks above. Clearly β_v is $|Q|!$-periodic.

Let $w = a_m \cdots a_2 a_1 \in \Sigma^* v$ be a word of length $m \geqslant k$. The run of $\mathcal{B}$ on w starting from the initial state prolongs the run in (12):

$$q_m \xleftarrow{a_m} \cdots \xleftarrow{a_{k+2}} q_{k+1} \xleftarrow{a_{k+1}} q_k \xleftarrow{a_k} \cdots \xleftarrow{a_2} q_1 \xleftarrow{a_1} q_0$$

Assume that $q_m \in N$. As argued above, there is a position $0 \leqslant i < k$ such that $i \equiv m \pmod{|Q|!}$ and $q_i \in N$. Therefore, there exists a nonempty run from q_i to q_m whose length is a multiple of $|Q|!$. Hence, (q_i, q_m) is a synchronized pair by Lemma 4.14, which is F-consistent by assumption. Therefore, $w \in L$ if and only if $q_m \in F$ if and only if $q_i \in F$ if and only if $\beta_v(|w|) = 1$. ■

We can now prove Lemma 4.13.

PROOF OF LEMMA 4.13. Given a subset $P \subseteq Q$ let $\mathsf{L}(\mathcal{B}, P) := \mathsf{L}(Q, \Sigma, P, \delta, q_0)$. Let $F_N = N \cap F$ and $F_T = T \cap F$. We disjointly decompose L into

$$L = \mathsf{L}(\mathcal{B}, F_N) \cup \bigcup_{q \in F_T} \mathsf{L}(\mathcal{B}, \{q\}).$$

First observe that $\mathsf{L}(\mathcal{B}, \{q\}) \in \mathbf{SF}$ for all $q \in F_T$ because a transient state q can occur at most once in a run of $\mathcal{B}$.

It remains to show that $\mathsf{L}(\mathcal{B}, F_N)$ belongs to $\langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$. Using the threshold $k = |Q|! \cdot (|T| + 1)$, we distinguish between words of length at most $k - 1$ and words of length at least k, and group the latter set by their suffixes of length k:

$$\mathsf{L}(\mathcal{B}, F_N) = (\mathsf{L}(\mathcal{B}, F_N) \cap \Sigma^{\leqslant k-1}) \cup \bigcup_{v \in \Sigma^k} (\mathsf{L}(\mathcal{B}, F_N) \cap \Sigma^* v).$$

The first part $\mathsf{L}(\mathcal{B}, F_N) \cap \Sigma^{\leqslant k-1}$ is finite and thus suffix testable. To finish the proof, we will show that $\mathsf{L}(\mathcal{B}, F_N) \cap \Sigma^* v \in \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$ for each $v \in \Sigma^k$. Let $v \in \Sigma^k$ and let $\beta_v \colon \mathbb{N} \to \{0, 1\}$ be the $|Q|!$-periodic function from Lemma 4.15. The lemma implies that

$$\mathsf{L}(\mathcal{B}, F_N) \cap \Sigma^* v = (\Sigma^* v \cap \{w \in \Sigma^* \mid \beta(|w|) = 1\}) \setminus \mathsf{L}(\mathcal{B}, T).$$

The language $\{w \in \Sigma^* \mid \beta(|w|) = 1\}$ is a regular length language, $\Sigma^* v$ is suffix testable and $\mathsf{L}(\mathcal{B}, T)$ is a finite union of regular suffix-free languages. ■

The following lemma is an immediate consequence of Lemma 4.13.

LEMMA 4.16. *If $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$ then there exist $u, x, y, z \in \Sigma^*$ with $|x| = |y| = |z| \geqslant 1$ such that L separates $x^* y z^* u$ and $z^* u$.*

Now, we can finally prove point (4) from Theorem 4.3.

PROPOSITION 4.17. *If $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$ then $\mathsf{F}^{\mathsf{r}}_L(n) = \Omega^\infty(\log n)$.*

PROOF. Consider the words $u, x, y, z \in \Sigma^*$ described in Lemma 4.16. Let $n = |z| \cdot m + |u|$ for some $m \geqslant 1$ and let $\mathcal{P}_n$ be a randomized SW-algorithm for L. We describe a randomized one-way protocol for GT_m: Let $1 \leqslant i \leqslant m$ be the input of Alice and $1 \leqslant j \leqslant m$ be the input of Bob. Alice starts with reading $x^m y z^{m-i}$ into $\mathcal{P}_n$. Then she sends the reached state to Bob using $O(s(\mathcal{P}_n))$

bits. Bob then continues the run of $\mathcal{P}_n$ from the transmitted state with the word $z^j u$. Hence, $\mathcal{P}_n$ is simulated on the word $w := x^m y z^{m-i} z^j u = x^m y z^{m-i+j} u$. We have

$$\mathsf{last}_n(w) = \begin{cases} x^{i-1-j} y z^{m-i+j} u, & \text{if } i > j, \\ z^m u, & \text{if } i \leqslant j. \end{cases}$$

By Lemma 4.16, $\mathsf{last}_n(w)$ belongs to L in exactly one of the two cases $i > j$ and $i \leqslant j$. Hence, Bob can distinguish these two cases with probability at least 2/3. It follows that the protocol computes GT_m and its cost is bounded by $s(\mathcal{P}_n)$. By Theorem 4.11 we can conclude that $s(\mathcal{P}_n) = \Omega(\log m) = \Omega(\log n)$, and therefore $\mathsf{F}^{\mathsf{r}}_L(n) = \Omega^\infty(\log n)$. ■

4.6 Sliding window algorithms with one-sided error

So far, we have only considered randomized SW-algorithms with two-sided error (analogously to the complexity class BPP). Randomized SW-algorithms with one-sided error (analogously to the classes RP and coRP) can be motivated by applications where all "yes"-outputs or all "no"-outputs, respectively, have to be correct. We distinguish between true-biased and false-biased algorithms. A *true-biased (randomized) streaming algorithm* $\mathcal{P}$ for a language L satisfies the following properties:

— If $w \in L$ then $\Pr[\mathcal{P} \text{ accepts } w] \geqslant 2/3$.

— If $w \notin L$ then $\Pr[\mathcal{P} \text{ rejects } w] = 1$.

A *false-biased (randomized) streaming algorithm* $\mathcal{P}$ for a language L satisfies the following properties:

— If $w \in L$ then $\Pr[\mathcal{P} \text{ accepts } w] = 1$.

— If $w \notin L$ then $\Pr[\mathcal{P} \text{ rejects } w] \geqslant 2/3$.

Let $F^0_L(n)$ (resp., $F^1_L(n)$) be the minimal space complexity $s(\mathcal{P}_n)$ of any true-biased (resp., false-biased) SW-algorithm $\mathcal{P}_n$ for L and window size n. We have the relations $F^{\mathsf{r}}_L(n) \leqslant F^i_L(n) \leqslant F_L(n)$ for $i \in \{0,1\}$, and $F^0_L(n) = F^1_{\Sigma^*\setminus L}(n)$.

For $F^0_L(n)$ and $F^1_L(n)$ a statement analogous to Lemma 4.4 does not hold, i.e., the classes $\{L \subseteq \Sigma^* \mid \mathsf{F}^{\mathsf{i}}_L(n) = \mathcal{O}(s(n))\}$ for i $\in \{0,1\}$ and a function $s(n)$ do not form a Boolean algebra. To see this, consider the language $L = \{\$w\#w : w \in \{0,1\}^*\}$. It is easy to see that $F^1_L(n) = \mathcal{O}(\log n)$. On the other hand, every true-biased randomized (in fact, every nondeterministic) communication protocol for EQ_n (over the domain $\{1,\dots,n\}$) has cost $\Omega(\log n)$ [84, Chapter 5]. This implies $F^1_{\Sigma^*\setminus L}(n) = F^0_L(n) = \Omega^\infty(n)$, where $\Sigma = \{0,1,\$,\#\}$.

We show that for all regular languages SW-algorithms with one-sided error have no advantage over their deterministic counterparts:

THEOREM 4.18 (one-sided error). *Let L be regular.*

(i) If $L \in \langle \mathbf{ST}, \mathbf{Len} \rangle$ then $F^0_L(n)$ and $F^1_L(n)$ are $\mathcal{O}(1)$.

(ii) If $L \notin \langle \mathbf{ST}, \mathbf{Len} \rangle$ then $F^0_L(n)$ and $F^1_L(n)$ are $\Omega^\infty(\log n)$.
(iii) If $L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$ then $F^0_L(n)$ and $F^1_L(n)$ are $O(\log n)$.
(iv) If $L \notin \langle \mathbf{LI}, \mathbf{Len} \rangle$ then $F^0_L(n)$ and $F^1_L(n)$ are $\Omega^\infty(n)$.

The upper bounds in (i) and (iii) already hold for deterministic SW-algorithms (Theorem 1.3). Moreover, the lower bound in (iv) already holds for SW-algorithms with two-sided error (Theorem 4.3(6)). It remains to prove point (ii) of the theorem.[6] In fact we show that any *nondeterministic* SW-algorithm for a regular language $L \notin \langle \mathbf{ST}, \mathbf{Len} \rangle$ requires space $\Omega^\infty(\log n)$ (this generalizes the lower bound in the second equivalence of Theorem 1.3). A nondeterministic SW-algorithm for a language L and window size n is an NFA $\mathcal{P}_n$ with $\mathsf{L}(\mathcal{P}_n) = \mathsf{SW}_n(L)$, and its space complexity is $s(\mathcal{P}_n) = \log |\mathcal{P}_n|$. If we have a true-biased randomized SW-algorithm for L we can turn it into a nondeterministic SW-algorithm by keeping only those transitions with nonzero probabilities and making all states q initial which have a positive initial probability $\iota(q) > 0$. Therefore, it suffices to show the following statement:

PROPOSITION 4.19. *Let $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$. Then, for infinitely many n every nondeterministic SW-algorithm $\mathcal{P}_n$ for L has $\Omega(\sqrt{n})$ many states.*

For the proof of Proposition 4.19 we need the following lemma.

LEMMA 4.20. *Let $L \subseteq a^*$ and $n \in \mathbb{N}$ such that L separates $\{a^n\}$ and $\{a^k \mid k > n\}$. Then, every NFA for L has at least $\sqrt{n}$ many states.*

PROOF. The easy case is $a^n \in L$ and $a^k \notin L$ for all $k > n$. If an NFA for L has at most n states then any successful run on a^n must have a state repetition. By pumping one can construct a successful run on a^k for some $k > n$, which is a contradiction.

Now, assume $a^n \notin L$ and $a^k \in L$ for all $k > n$. The proof is essentially the same as for [64, Lemma 6], where the statement of the lemma is shown for $L = a^* \setminus \{a^n\}$. Let us give the proof for completeness. It is known that every unary NFA has an equivalent NFA in so-called Chrobak normal form. A unary NFA in Chrobak normal form consists of a simple path (called the initial path in the following) whose starting state is the unique initial state of the NFA. From the last state of the initial path, edges go to a collection of disjoint cycles. In [50] it is shown that an m-state unary NFA has an equivalent NFA in Chrobak normal form whose initial path consists of $m^2 - m$ states. Now, assume that L is accepted by an NFA with m states and let $\mathcal{A}$ be the equivalent NFA in Chrobak normal form, whose initial path consists of $m^2 - m$ states. If $n \geqslant m^2 - m$ then all states that are reached in $\mathcal{A}$ from the initial state via a^n belong to a cycle and every cycle contains such a state. Since $a^n \notin L$, all these states are rejecting. Hence,

6 Note that Theorem 4.18((ii)) generalizes the lower bound $F_L(n) = \Omega^\infty(\log n)$ for languages $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$; see Theorem 1.3.

$a^{n+x\cdot d} \notin L$ for all $x \geqslant 0$, where d is the product of all cycle lengths. This contradicts the fact that $a^k \in L$ for all $k > n$. Hence, we must have $n < m^2 - m$ and therefore $m > \sqrt{n}$. ■

PROOF OF PROPOSITION 4.19. Let $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$. By Lemma 3.13 and the results from Section 3.7 there are words $x, y, z \in \Sigma^*$ such that $|x| = |y|$ and L separates xy^*z and y^*z. Note that we must have $x \neq y$.

Fix $m \geqslant 0$ and consider the window size $n = |x| + m|y| + |z|$. Let $\mathcal{P}_n = (Q, \Sigma, I, \Delta, F)$ be a nondeterministic SW-algorithm for L and window size n, i.e., it is an NFA for $\mathsf{SW}_n(L)$. Notice that $\mathcal{P}_n$ separates $\{xy^mz\}$ and $\{xy^kz \mid k > m\}$. We define an NFA $\mathcal{A}$ over the unary alphabet $\{a\}$ as follows:

- The state set of $\mathcal{A}$ is Q.
- The set of initial states of $\mathcal{A}$ is $\{q \in Q \mid \exists p \in I \colon p \xrightarrow{x} q \text{ in } \mathcal{P}_n\}$.
- The set of final states of $\mathcal{A}$ is $\{p \in Q \mid \exists q \in F \colon p \xrightarrow{z} q \text{ in } \mathcal{P}_n\}$.
- The set of transitions of $\mathcal{A}$ is $\{(p, a, q) \mid p \xrightarrow{y} q \text{ in } \mathcal{P}_n\}$.

It recognizes the language $\mathsf{L}(\mathcal{A}) = \{a^k \mid xy^kz \in \mathsf{SW}_n(L)\}$, and therefore $\mathsf{L}(\mathcal{A})$ separates $\{a^m\}$ and $\{a^k \mid k > m\}$. By Lemma 4.20, $\mathcal{A}$ has at least $\sqrt{m} = \Omega(\sqrt{n})$ states. Hence, also the number of states of $\mathcal{P}_n$ is in $\Omega(\sqrt{n})$. ■

Proposition 4.19 implies $F^0_L(n) \geqslant 1/2 \log n - O(1)$ on infinitely many n for all $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$. Since $\mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$ is closed under complement, this implies $F^1_L(n) = F^0_{\Sigma^* \setminus L}(n) \geqslant 1/2 \log n - O(1)$ on infinitely many n for all $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$.

4.7 Randomized variable-size model

In this section, we briefly look at randomized algorithms in the variable-size model. First we transfer the definitions from Section 2.5 in a straightforward way. A *randomized variable-size sliding window algorithm* $\mathcal{P}$ for $L \subseteq \Sigma^*$ is a randomized streaming algorithm for $\mathsf{SW}(L)$ (defined in (2) on page 14). Its *space complexity* is $v(\mathcal{P}, n) = \log |M_{\leqslant n}| \in \mathbb{N} \cup \{\infty\}$ where $M_{\leqslant n}$ contains all memory states in $\mathcal{P}$ which are reachable with nonzero probability in $\mathcal{P}$ on inputs $w \in \Sigma^*_\downarrow$ with $\mathsf{mwl}(w) \leqslant n$. Since the variable-size sliding window model subsumes the fixed-size model, we have $\mathsf{F}^{\mathsf{r}}_L(n) \leqslant v(\mathcal{P}, n)$ for every randomized variable-size sliding window algorithm $\mathcal{P}$ for L.

Again we raise the question if randomness can improve the space complexity in the variable-size model. We claim that, in contrast to the fixed-size model, randomness does not allow more space efficient algorithms in the variable-size setting. Clearly, all upper bounds for the deterministic variable-size setting transfer to the randomized variable-size setting, i.e., languages in $\langle \mathbf{LI}, \mathbf{Len} \rangle$ have $O(\log n)$ space complexity, and empty and universal languages have $O(1)$ space complexity. For every regular language L which is not contained in $\langle \mathbf{LI}, \mathbf{Len} \rangle$ we proved a linear lower bound on $\mathsf{F}^{\mathsf{r}}_L(n)$ (Proposition 4.12), which is also a lower bound on the space complexity of any randomized variable-size sliding window algorithm for L. It remains to look at languages $\emptyset \subsetneq L \subsetneq \Sigma^*$, for which we have proved a logarithmic lower bound in the deterministic setting (Lemma 2.5).

LEMMA 4.21. *If $\mathcal{P}$ is a randomized variable-size SW-algorithm for a language $\emptyset \subsetneq L \subsetneq \Sigma^*$ then $v(\mathcal{P}, n) = \Omega(\log n)$.*

PROOF. Let $\emptyset \subsetneq L \subsetneq \Sigma^*$ be a language. There must be a length-minimal nonempty word $a_1 \cdots a_k \in \Sigma^+$ such that $|\{\varepsilon, a_1 \cdots a_k\} \cap L| = 1$ and we fix such a word $a_1 \cdots a_k$. By minimality we also have $|\{a_1 \cdots a_k, a_2 \cdots a_k\} \cap L| = 1$. Let $\mathcal{P}$ be a randomized variable-size SW-algorithm for L. By Lemma 4.1 we can assume that the error probability of $\mathcal{P}$ is at most 1/6, which increases its space complexity $v(\mathcal{P}, n)$ by a constant factor.

For every $n \in \mathbb{N}$ we construct a protocol for GT_n with cost $O(v(\mathcal{P}, n))$. With Theorem 4.11 this implies that $v(\mathcal{P}, n) = \Omega(\log n)$. Let $1 \leqslant i \leqslant n$ be the input of Alice and $1 \leqslant j \leqslant n$ be the input of Bob. Alice starts two instances of $\mathcal{P}$ (using independent random bits) and reads a_1^i into both of them. She sends the memory states to Bob using $O(v(\mathcal{P}, i)) \leqslant O(v(\mathcal{P}, n))$ bits. Bob then continues from both states, and reads $\downarrow^j a_2 \cdots a_k$ into the first instance and $\downarrow^{j+1} a_1 a_2 \cdots a_k$ into the second instance. Let $y_1, y_2 \in \{0, 1\}$ be the outputs of the two instances of $\mathcal{P}$. With high probability, namely $(1 - 1/6)^2 \geqslant 2/3$, both answers are correct, i.e.

$$y_1 = 1 \iff \mathsf{wnd}(a_1^i \downarrow^j a_2 \cdots a_k) \in L$$

and

$$y_2 = 1 \iff \mathsf{wnd}(a_1^i \downarrow^{j+1} a_1 a_2 \cdots a_k) \in L.$$

Bob returns true, i.e., he claims $i > j$, if and only if $y_1 = y_2$.

Let us prove the correctness. If $i > j$ then

$$\mathsf{wnd}(a_1^i \downarrow^j a_2 \cdots a_k) = a_1^{i-j} a_2 \cdots a_k = \mathsf{wnd}(a_1^i \downarrow^{j+1} a_1 a_2 \cdots a_k)$$

and hence Bob returns true with probability at least 2/3. If $i \leqslant j$ then

$$\mathsf{wnd}(a_1^i \downarrow^j a_2 \cdots a_k) = a_2 \cdots a_k$$

and

$$\mathsf{wnd}(a_1^i \downarrow^{j+1} a_1 a_2 \cdots a_k) = a_1 a_2 \cdots a_k.$$

By assumption, exactly one of the words $a_1 \cdots a_k$, $a_2 \cdots a_k$ belongs to L, and therefore Bob returns false with probability at least 2/3. ■

The lower bound from Lemma 4.21 also holds for variable-size SW-algorithms with one-sided error since they are more restricted than algorithms with two-sided error. In fact, Lemma 4.21 also holds for nondeterministic and co-nondeterministic SW-algorithms since the (co-)nondeterministic communication complexity of GT_n is $\Theta(\log n)$ [84, Chapter 5].

5. Property testing in the sliding window model

In all settings discussed so far, there are some regular languages for which testing membership in the sliding window model requires linear space. To be more specific, for any language $L \in \mathbf{Reg} \setminus \langle \mathbf{LI}, \mathbf{Len} \rangle$ it requires linear space to test membership even for randomized sliding window algorithms with two-sided error. In order to achieve space-efficient sliding window algorithms for all regular languages, we have to allow randomized sliding window algorithms that are allowed to err with unbounded probability on some specific inputs. We formalize this in the context of the property testing framework. More precisely, we introduce in this section *sliding window (property) testers*, which must accept if the active window belongs to a language L and reject if it has large Hamming distance from L.

For words that are not in L but that have small Hamming distance from L the algorithm is allowed to give any answer. We consider deterministic sliding window property testers and randomized sliding window property testers.

While at first sight the only connection between property testers and sliding window property testers is that we must accept the input if it satisfies a property P and reject if it is far from satisfying P, there is, in fact, a deeper link. In particular, the property tester for regular languages due to Alon et al. [2] combined with an optimal sampling algorithm for sliding windows [18] immediately yields $\mathcal{O}(\log n)$-space, two-sided error sliding window property testers with Hamming gap $\gamma(n) = \epsilon n$ for all regular languages. We will improve on this observation.

5.1 Sliding window testers

The *Hamming distance* between two words $u = a_1 \cdots a_n$ and $v = b_1 \cdots b_n$ of equal length is the number of positions where u and v differ, i.e., $\mathrm{dist}(u, v) = |\{i \mid a_i \neq b_i\}|$. If $|u| \neq |v|$ we set $\mathrm{dist}(u, v) = \infty$. The distance of a word u to a language L is defined as

$$\mathrm{dist}(u, L) = \inf\{\mathrm{dist}(u, v) \mid v \in L\} \in \mathbb{N} \cup \{\infty\}.$$

Additionally, we define the *prefix distance* between equal-length words $u = a_1 \cdots a_n$ and $v = b_1 \cdots b_n$ by $\mathrm{pdist}(u, v) = \min\{i \in \{0, \dots, n\} \mid a_{i+1} \cdots a_n = b_{i+1} \cdots b_n\}$. For instance, we have $\mathrm{pdist}(abbaca, abcaca) = 3$ and $\mathrm{pdist}(abccca, abcaca) = 4$ (whereas the Hamming distance in both cases is 1). Clearly, we have $\mathrm{dist}(u, v) \leqslant \mathrm{pdist}(u, v)$. The algorithms presented in this section satisfy the stronger property that windows whose prefix distance to the language L is large are rejected by the algorithm.

In this section, γ is always a function $\gamma \colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ such that $\gamma(n) < n$ for all n. A *deterministic sliding window (property) tester* with Hamming gap $\gamma(n)$ for a language $L \subseteq \Sigma^*$ and window size n is a deterministic streaming algorithm $\mathcal{P}_n$ over the alphabet Σ with the following properties:

— If $\mathsf{last}_n(w) \in L$, then $w \in \mathsf{L}(\mathcal{P}_n)$.

— If $\mathrm{dist}(\mathsf{last}_n(w), L) > \gamma(n)$, then $w \notin \mathsf{L}(\mathcal{P}_n)$.

If neither of the two cases hold, the behavior of $\mathcal{P}_n$ can be arbitrary. Recall that $s(\mathcal{P}_n)$ is the space used by $\mathcal{P}_n$ (see Section 2.3). A *randomized sliding window tester* with Hamming gap $\gamma(n)$ for a language $L \subseteq \Sigma^*$ and window size n is a randomized streaming algorithm $\mathcal{P}_n$ over the alphabet Σ with the following properties. It has *two-sided error* if for all $w \in \Sigma^*$ we have:

— If $\mathsf{last}_n(w) \in L$, then $\Pr[\mathcal{P}_n \text{ accepts } w] \geqslant 2/3$.
— If $\mathrm{dist}(\mathsf{last}_n(w), L) > \gamma(n)$, then $\Pr[\mathcal{P}_n \text{ rejects } w] \geqslant 2/3$.

It is *true-biased* if for all $w \in \Sigma^*$ we have:

— If $\mathsf{last}_n(w) \in L$, then $\Pr[\mathcal{P}_n \text{ accepts } w] \geqslant 2/3$.
— If $\mathrm{dist}(\mathsf{last}_n(w), L) > \gamma(n)$, then $\Pr[\mathcal{P}_n \text{ rejects } w] = 1$.

It is *false-biased* if for all $w \in \Sigma^*$ we have:

— If $\mathsf{last}_n(w) \in L$, then $\Pr[\mathcal{P}_n \text{ accepts } w] = 1$.
— If $\mathrm{dist}(\mathsf{last}_n(w), L) > \gamma(n)$, then $\Pr[\mathcal{P}_n \text{ rejects } w] \geqslant 2/3$.

True-biased and false-biased algorithms are algorithms with *one-sided error*. Again, the success probability $2/3$ is an arbitrary choice in light of Lemma 4.1.

Intuitively, the Hamming gap function γ should be a small function. Typical choices for $\gamma(n)$ are ϵn for some constant ϵ $(0 < \epsilon < 1)$ or $\gamma(n) = c$ for a constant c. We will also consider Hamming gap functions that are between these two cases. The case $\gamma(n) = 0$ for all n corresponds to exact membership testing to L, which was studied in the previous sections.

Let us also remark that we only consider the fixed-size sliding window model in this section. One might also consider variable-size sliding window testers, where the size of the window can grow and shrink. We leave this for future work. Moreover, we only consider the Hamming distance in this paper. One might also consider other distances on words, like for instance edit distance. We believe that Hamming distance is the most basic distance measure on strings. The upper bounds stated below also apply to edit distance (since the edit distance is always bounded by the Hamming distance). Whether our lower bounds can be extended to edit distance remains open.

5.2 Main results of this section

Let us now state and discuss the main results of this section. We start with our upper bounds:

THEOREM 5.1. *For every regular language L and window size n there exists a deterministic sliding window tester $\mathcal{P}_n$ with Hamming gap $O(1)$ and $s(\mathcal{P}_n) = O(\log n)$.*

We will later see that allowing a larger (but not too large) Hamming gap in Theorem 5.1 does not allow a better space bound. This changes if we allow randomized sliding window testers with a two-sided error:

THEOREM 5.2. *For every regular language L there is a constant c such that the following holds: If the function $\gamma(n)$ and the window size n satisfy $\gamma(n) \geqslant c$ then there is a randomized sliding window tester $\mathcal{P}_n$ for L and window size n with a two-sided error, Hamming gap $\gamma(n)$, and $s(\mathcal{P}_n) = O(\log(n/\gamma(n)))$.*

From Theorem 5.2 we will easily obtain the following corollary:

COROLLARY 5.3. *For every regular language L, every window size n and every $0 < \epsilon < 1$ there exists a randomized sliding window tester $\mathcal{P}_n$ with two-sided error, Hamming gap ϵn and $s(\mathcal{P}_n) = O(1/\epsilon)$.*

The upper bounds in Theorem 5.1 and Theorem 5.2 hold for all regular languages. We will also identify subclasses for which these upper bounds can be improved. Recall the definition of suffix-free languages from Section 1.2. Another important language class in the context of sliding-window testers is the class of *trivial* languages. A language $L \subseteq \Sigma^*$ is $\gamma(n)$-*trivial* (for a function $\gamma(n) < n$) if for all $n \in \mathbb{N}$ with $L \cap \Sigma^n \neq \emptyset$ and all $w \in \Sigma^n$ we have $\mathrm{dist}(w, L) \leqslant \gamma(n)$. If L is $O(1)$-trivial we say that L is *trivial*. Examples of trivial languages include all length languages, all suffix (resp., prefix) testable languages (in particular, $L = a\{a, b\}^*$ for which $\mathsf{F}_L(n) = \Theta(n)$ holds), and also the set of all words over $\{a, b\}$ which contain an even number of a's. Note that Alon et al. [2] call a language L trivial if L is $o(n)$-trivial according to our definition, i.e., $\gamma(n)$-trivial for some function $\gamma(n) = o(n)$. In fact, we will prove that both definitions coincide for regular languages (Theorem 5.24). With **Triv** we denote the set of all regular trivial languages.

We can achieve a Hamming gap of $\gamma(n)$ simply with a deterministic sliding window tester that accepts or rejects all input words depending on the input length. Moreover, the tester has only one state and hence uses space $\log(1) = 0$. It turns out that for finite unions of regular trivial languages and regular suffix-free languages, we can obtain a doubly logarithmic space bound if we allow false-biased randomized sliding window testers. Let us write $\bigcup(\mathbf{Triv}, \mathbf{SF})$ for the class of all finite unions of regular trivial languages and regular suffix-free languages.

THEOREM 5.4. *For every $L \in \bigcup(\mathbf{Triv}, \mathbf{SF})$ and window size n there exists a false-biased randomized sliding window tester $\mathcal{P}_n$ with Hamming gap $O(1)$ and $s(\mathcal{P}_n) = O(\log \log n)$.*

Let us now discuss our lower bounds. It turns out that the above upper bounds are sharp in most cases. First of all, the logarithmic space bound in Theorem 5.1 cannot be improved whenever L is a regular nontrivial language. This holds even for randomized true-biased algorithms and a Hamming gap ϵn (assuming $\epsilon < 1$ is not too big). Similarly, the doubly logarithmic space bound in Theorem 5.4 cannot be improved.

THEOREM 5.5. *For every language $L \in \mathbf{Reg} \setminus \mathbf{Triv}$ there exist $\epsilon > 0$ and infinitely many window sizes $n \in \mathbb{N}$ for which every true-biased (resp., false-biased) randomized sliding window tester for L with Hamming gap ϵn uses space at least $\log n - O(1)$ (resp. $\log \log n - O(1)$).*

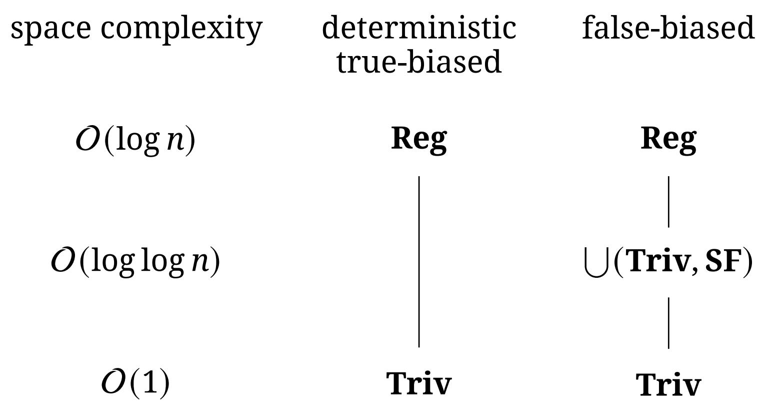

Figure 5. The space complexity of regular languages with respect to deterministic, true-biased and false-biased sliding window testers. As in Figure 1, only upper bounds are shown, and they hold for every Hamming gap function $\gamma(n)$ provided that $\gamma(n) \geqslant c$ for a constant c that depends on the language. All upper bounds can be matched with lower bounds that hold for every $\gamma(n) \leqslant \epsilon n$ for a constant ϵ that depends on the language.

Moreover, also for false-biased randomized sliding window testers the logarithmic space bound from Theorem 5.1 cannot be improved whenever $L \in \mathbf{Reg} \setminus \bigcup(\mathbf{Triv}, \mathbf{SF})$:

THEOREM 5.6. *If $L \in \mathbf{Reg} \setminus \bigcup(\mathbf{Triv}, \mathbf{SF})$ then there exist $\epsilon > 0$ and infinitely many window sizes $n \in \mathbb{N}$ for which every false-biased randomized sliding window tester for L with Hamming gap ϵn uses at least $\log n - O(1)$ space.*

The above results provide matching upper and lower space bounds for deterministic, true-biased and false-biased sliding window testers; see also Figure 5. Moreover, the upper bounds hold for a constant Hamming gap (Theorems 5.1 and 5.4) whereas the lower bounds hold for Hamming gap ϵn as long as ϵ is larger than a language-dependent constant (Theorems 5.5 and 5.6). Thus, in the deterministic, true-biased and false-biased settings, the space complexity is quite insensitive to the choice of the Hamming gap function $\gamma(n)$.

For randomized sliding window testers with a two-sided error, the situation is different. We have already discussed Theorem 5.2, where the Hamming gap $\gamma(n)$ is reflected in the space bound. It turns out that the upper bound in Theorem 5.2 is tight whenever L is not a finite union of regular trivial languages and regular suffix-free languages:

THEOREM 5.7. *If $L \in \mathbf{Reg} \setminus \bigcup(\mathbf{Triv}, \mathbf{SF})$ then there exist $\epsilon > 0$ and infinitely many window sizes $n \in \mathbb{N}$ for which every randomized sliding window tester with two-sided error for L and Hamming gap $\gamma(n) \leqslant \epsilon n$ needs space $\Omega(\log(n/\gamma(n)))$.*

If $L \in \bigcup(\mathbf{Triv}, \mathbf{SF})$ then the lower bound from Theorem 5.7 does not hold in general, since we have an upper bound of $O(\log\log n)$ from Theorem 5.4.[7] We do not know whether there is

7 Note that if $\gamma(n) = O(n/\log n)$ then the lower bound $\Omega^{\infty}(\log(n/\gamma(n)))$ becomes $\Omega^{\infty}(\log\log n)$.

a matching lower bound of $\Omega^\infty(\log\log n)$ for nontrivial languages. Currently, we can only show a slightly weaker lower bound in this case:

THEOREM 5.8. *If $L \in \mathbf{Reg} \setminus \mathbf{Triv}$ then there exist $\epsilon > 0$ and infinitely many window sizes $n \in \mathbb{N}$ for which every randomized sliding window tester with two-sided error for L and Hamming gap $\gamma(n) \leqslant \epsilon n$ needs space $\Omega(\log\log(n/\gamma(n)))$.*

Note that whenever $\gamma(n) = O(n^c)$ for some $c < 1$ then the lower bound $\Omega^\infty(\log\log(n/\gamma(n)))$ from Theorem 5.8 becomes $\Omega^\infty(\log\log n)$, which matches the upper bound from Theorem 5.4. It is left open to classify the space complexity for languages in $\bigcup(\mathbf{Triv}, \mathbf{SF}) \setminus \mathbf{Triv}$, e.g. $L = ab^*$, for sublinear Hamming gaps $\gamma(n)$ which are $\Omega(n^c)$ for all $c < 1$, e.g. $\gamma(n) = n/\log n$.

Let us also remark that Lemma 4.2 does not generalize to sliding window testers (with the obvious generalization of the space complexities $\mathsf{F}_L(n)$ and $\mathsf{F}_L^r(n)$ to sliding window testers). In the proof of Lemma 4.2 we used the fact that a randomized sliding window algorithm for a language L and a window size n is a probabilistic finite automaton with the isolated cut-point $1/2$. This is not necessarily true for randomized sliding window testers. If w is a word such that neither $\mathsf{last}_n(w) \in L$ nor $\mathrm{dist}(\mathsf{last}_n(w), L) > \gamma(n)$ holds then $\Pr[\mathcal{P}_n \text{ accepts } w] = 1/2$ is possible. Indeed, the generalization of Lemma 4.2 to sliding window testers would contradict Corollary 5.3 together with Theorem 5.5.

5.3 Upper bounds

In this section we prove Theorems 5.1, 5.2 and 5.4.

5.3.1 Deterministic sliding window testers

In this section, we prove Theorem 5.1: every regular language has a deterministic sliding window tester with constant Hamming gap which uses $O(\log n)$ space. It is based on the path summary algorithm from Section 3.3. In the following, we fix a regular language L and an rDFA $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ for L. By Lemma 3.19 we can assume that every nontransient SCC of $\mathcal{B}$ has the same period $g \geqslant 1$.

For a state $q \in Q$ we define $\mathrm{Acc}(q) = \{n \in \mathbb{N} \mid \exists w \in \Sigma^n \colon w \cdot q \in F\}$. The following lemma is the main tool to prove correctness of our sliding window testers. It states that if a word of length n is accepted from state p and ρ is any internal run (see Section 3.3 and page 20) of length at most n starting from state p, then after removing a bounded length run at the end of ρ, ρ can be extended to an accepting run of length n. Formally, a run π *t-simulates* a run ρ if one can factorize $\rho = \rho_1\rho_2$ and $\pi = \pi'\rho_2$ where $|\rho_1| \leqslant t$ for runs ρ_1, ρ_2, and π'; see also Figure 6. Note that this implies that ρ and π start in the same state. Also note that runs go from right to left (we work with an rDFA), so ρ_1 (resp., π') is the final part of ρ (resp., π).

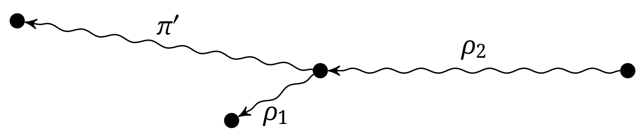

Figure 6. The run $\pi = \pi'\rho_2$ *t-simulates* the run $\rho = \rho_1\rho_2$. We have $|\rho_1| \leqslant t$.

LEMMA 5.9. *There exists a number $t \in \mathbb{N}$ (which only depends on $\mathcal{B}$) such that for every internal run ρ starting from a state p and every $n \in \mathrm{Acc}(p)$ with $n \geqslant |\rho|$, there exists an accepting run π of length n which t-simulates ρ.*

Note that the run π in this lemma is not necessarily internal.

Based on Lemma 5.9 we can prove Theorem 5.1. Afterwards we prove Lemma 5.9.

PROOF OF THEOREM 5.1. Let t be the constant from Lemma 5.9. We present a deterministic sliding window tester with constant Hamming gap t which uses $O(\log n)$ space. Let $n \in \mathbb{N}$ be the window size. By Lemma 3.5 we can maintain the set of all path summaries $\mathrm{PS}_{\mathcal{B}}(w) = \{\mathrm{ps}(\pi_{w,q}) \mid q \in Q\}$ for the active window $w \in \Sigma^n$, using $O(\log n)$ bits. In fact, the path summary algorithm works for variable-size windows but we do not need this here.

It remains to define the acceptance condition. Consider the SCC-factorization of π_{w,q_0}, say

$$\pi_{w,q_0} = \pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_1\pi_1$$

and its path summary $(\ell_m, q_m)\cdots(\ell_1, q_1)$. The algorithm accepts if and only if this path summary is accepting, i.e., $\ell_m = |\pi_m| \in \mathrm{Acc}(q_m)$. If $w \in L$ then clearly $|\pi_m| \in \mathrm{Acc}(q_m)$. On the other hand, if $|\pi_m| \in \mathrm{Acc}(q_m)$ then the internal run π_m can be t-simulated by an accepting run π'_m of equal length by Lemma 5.9. The run $\pi'_m\tau_{m-1}\pi_{m-1}\cdots\tau_1\pi_1$ is accepting and witnesses that $\mathrm{pdist}(w, L) \leqslant t$. We get $\mathrm{dist}(w, L) \leqslant \mathrm{pdist}(w, L) \leqslant t$. ■

To prove Lemma 5.9 we need to analyze the sets $\mathrm{Acc}(q)$ first. For $a \in \mathbb{N}$ and $X \subseteq \mathbb{N}$ we use the standard notation $X + a = \{a + x \mid x \in X\}$. A set $X \subseteq \mathbb{N}$ is *eventually d-periodic*, where $d \geqslant 1$ is an integer, if there exists a *threshold* $t \in \mathbb{N}$ such that for all $x \geqslant t$ we have $x \in X$ if and only if $x + d \in X$. If X is eventually d-periodic for some $d \geqslant 1$, then X is *eventually periodic*.

LEMMA 5.10. *For every $q \in Q$ the set $\mathrm{Acc}(q)$ is eventually g-periodic.*

PROOF. It suffices to show that for all $0 \leqslant r \leqslant g - 1$ the set $S_r = \{i \in \mathbb{N} \mid r + i \cdot g \in \mathrm{Acc}(q)\}$ is either finite or co-finite. Consider a remainder $0 \leqslant r \leqslant g - 1$ where S_r is infinite. We need to show that S_r is indeed co-finite. Let $i \in S_r$ with $i \geqslant |Q|$, i.e., there exists an accepting run π from q of length $r + i \cdot g$. Since π has length at least $|Q|$, it must traverse a state p in a nontransient SCC C. Choose j_0 such that $j_0 \cdot g \geqslant m(C)$ where $m(C)$ is the reachability constant from Lemma 3.18. By Lemma 3.18 for all $j \geqslant j_0$ there exists a cycle from p to p of length $j \cdot g$. Therefore, we can extend π to a longer accepting run by $j \cdot g$ symbols for any $j \geqslant j_0$. This proves that $x \in S_r$ for every $x \geqslant i + j_0$ and that S_r is co-finite. ■

Two sets $X, Y \subseteq \mathbb{N}$ are *equal up to a threshold* $t \in \mathbb{N}$, in symbol $X =_t Y$, if for all $x \geqslant t$: $x \in X$ if and only if $x \in Y$. Two sets $X, Y \subseteq \mathbb{N}$ are *almost equal* if they are equal up to some threshold $t \in \mathbb{N}$.

LEMMA 5.11. *A set $X \subseteq \mathbb{N}$ is eventually d-periodic if and only if X and $X + d$ are almost equal.*

PROOF. Let $t \in \mathbb{N}$ be such that for all $x \geqslant t$ we have $x \in X$ if and only if $x + d \in X$. Then, X and $X + d$ are equal up to threshold $t + d$. Conversely, if $X =_t X + d$, then for all $x \geqslant t$ we have $x + d \in X$ if and only if $x + d \in X + d$, which is true if and only if $x \in X$. ■

If the graph $G = (V, E)$ is strongly connected with $E \neq \emptyset$ and finite period g, and $V_0, \ldots, V_{g-1}$ satisfy the properties from Lemma 3.18, then we define the *shift* from $u \in V_i$ to $v \in V_j$ by

$$\text{shift}(u, v) = (j - i) \bmod g \in \{0, \ldots, g - 1\}. \tag{13}$$

Notice that shift(u, v) could be defined without referring to the partition $\bigcup_{i=0}^{g-1} V_i$ since the length of any path from u to v is congruent to shift(u, v) modulo g by Lemma 3.18. Also, note that $\text{shift}(u, v) + \text{shift}(v, u) \equiv 0 \pmod g$.

LEMMA 5.12. *Let C be a nontransient SCC in $\mathcal{B}$, $p, q \in C$ and $s = \text{shift}(p, q)$. Then, $\text{Acc}(p)$ and $\text{Acc}(q) + s$ are almost equal.*

PROOF. Let $k \in \mathbb{N}$ such that $k \cdot g \geqslant m(C)$ where $m(C)$ is the constant from Lemma 3.18. By Lemma 3.18 there exists a run from p to q of length $s + k \cdot g$, and a run from q to p of length $(k + 1) \cdot g - s$ (the latter number is congruent to shift(q, p) modulo g). By prolonging accepting runs we obtain

$$\text{Acc}(q) + s + k \cdot g \subseteq \text{Acc}(p) \text{ and } \text{Acc}(p) + (k + 1) \cdot g - s \subseteq \text{Acc}(q).$$

Adding $s + k \cdot g$ to both sides of the last inclusion yields

$$\text{Acc}(p) + (2k + 1) \cdot g \subseteq \text{Acc}(q) + s + k \cdot g \subseteq \text{Acc}(p).$$

By Lemma 5.10 and Lemma 5.11 the three sets above are almost equal. Also, $\text{Acc}(q) + s + k \cdot g$ is almost equal to $\text{Acc}(q) + s$ by Lemma 5.10 and Lemma 5.11. Since almost equality is a transitive relation, this proves the statement. ■

COROLLARY 5.13. *There exists a threshold $t \in \mathbb{N}$ such that*

(i) $\text{Acc}(q) =_t \text{Acc}(q) + g$ *for all* $q \in Q$, *and*

(ii) $\text{Acc}(p) =_t \text{Acc}(q) + \text{shift}(p, q)$ *for all nontransient SCCs C and all $p, q \in C$.*

Let us fix the threshold t from Corollary 5.13 in the following. We can now prove Lemma 5.9.

PROOF OF LEMMA 5.9. Let ρ be an internal run starting from p with $|\rho| \leqslant n \in \mathrm{Acc}(p)$. We have to find an accepting run π of length n starting from p and factorizations $\rho = \rho_1\rho_2$ and $\pi = \pi'\rho_2$ with $|\rho_1| \leqslant t$.

If $|\rho| \leqslant t$, then we can choose for π any accepting run from p of length $n \in \mathrm{Acc}(p)$. Otherwise, if $|\rho| > t$, then the internal run ρ is nonempty, which implies that the SCC C containing p is nontransient. Moreover, writing $\rho = \rho_1\rho_2$ where $|\rho_1| = t$, it is the case that ρ_2 leads from p to some state q of the same SCC. Set $s := \mathrm{shift}(q,p)$, which satisfies $s + |\rho_2| \equiv 0 \pmod{g}$ by the properties in Lemma 3.18 (see also the discussion before Lemma 5.12). Since $\mathrm{Acc}(q) =_t \mathrm{Acc}(p) + s$ by Corollary 5.13(ii), $n > t$ and $n \in \mathrm{Acc}(p)$, we have $n + s \in \mathrm{Acc}(q)$. Finally, since $n + s \equiv n - |\rho_2| \pmod{g}$ and $n - |\rho_2| = n - |\rho| + t \geqslant t$, we know $n - |\rho_2| \in \mathrm{Acc}(q)$ by Corollary 5.13(i). This yields an accepting run π' from q of length $n - |\rho_2|$. Then, ρ is t-simulated by $\pi = \pi'\rho_2$. ■

5.3.2 Sliding window testers with two-sided error

In this section, we will prove Theorem 5.2. We will construct for every regular language a randomized sliding window tester with two-sided error and Hamming gap $\gamma(n)$ that uses $O(\log(n/\gamma(n)))$ bits assuming the window size n satisfies $\gamma(n) \geqslant c$ for a suitably chosen constant. We still assume that the regular language L is recognized by an rDFA $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ whose nontransient SCCs have uniform period $g \geqslant 1$. Furthermore, we again use the constant t from Corollary 5.13.

We will set the constant c from Theorem 5.2 to $c = 4(t+1)$. Let us fix a window size n such that $\gamma(n) \geqslant 4(t+1)$. We define the parameters $h = n - t$ and $\ell = n - \gamma(n) + t + 1$, which satisfy

$$\begin{aligned} \frac{\ell}{h} &= \frac{n - \gamma(n) + t + 1}{n - t} \leqslant \frac{n - \gamma(n) + \frac{1}{4}\gamma(n)}{n - \frac{1}{4}\gamma(n)} \\ &= \frac{n - \frac{1}{4}\gamma(n) - \frac{1}{2}\gamma(n)}{n - \frac{1}{4}\gamma(n)} \leqslant 1 - \frac{\gamma(n)}{2n}. \end{aligned} \tag{14}$$

Let $\mathcal{Z} = (C, \{\texttt{inc}\}, c_0, \rho, F)$ be the (h,ℓ)-counter with error probability $1/(3|Q|)$ from Proposition 4.6, which uses $O(\log\log|Q| + \log(n/\gamma(n))) = O(\log(n/\gamma(n)))$ space by (14) (as usual, we consider $|Q|$ as a constant). The counter $\mathcal{Z}$ is used to define so-called compact summaries of runs.

A *compact summary* $\kappa = (q_m, r_m, c_m) \cdots (q_2, r_2, c_2)(q_1, r_1, c_1)$ is a sequence of triples, where each triple (q_i, r_i, c_i) consists of a state $q_i \in Q$, a remainder $0 \leqslant r_i \leqslant g - 1$, and a state $c_i \in C$ of the counter $\mathcal{Z}$. The state c_1 of the counter is always its initial state c_0 (and hence low) and $r_1 = 0$. We say that κ *represents* a run π if the SCC-factorization of π has the form $\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_1\pi_1$, and the following properties hold for all $1 \leqslant i \leqslant m$:

(C1) π_i starts in q_i;

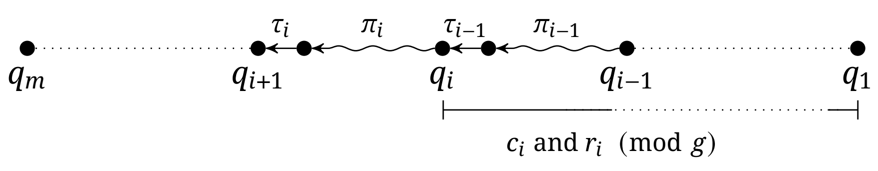

Figure 7. A compact summary of a run π.

(C2) $r_i = |\tau_{i-1}\pi_{i-1}\cdots\tau_1\pi_1| \bmod g$;

(C3) if $|\tau_{i-1}\pi_{i-1}\cdots\tau_1\pi_1| \leqslant n - \gamma(n) + t + 1$ then c_i is a low state;

(C4) if $|\tau_{i-1}\pi_{i-1}\cdots\tau_1\pi_1| \geqslant n - t$ then c_i is a high state.

Note that κ does not restrict π_m except that the latter must be an internal run starting in q_m.

The idea of a compact summary is visualized in Figure 7. If $m > |Q|$ then the above compact summary cannot represent a run. Therefore, we can assume that $m \leqslant |Q|$. For every triple (q_i, r_i, c_i), the entries q_i and r_i only depend on the rDFA $\mathcal{B}$, and hence can be stored with $O(1)$ bits. Each state c_i of the probabilistic counter $\mathcal{Z}$ needs $O(\log(n/\gamma(n)))$ bits. Hence, a compact summary can be stored in $O(\log(n/\gamma(n)))$ bits. In contrast to the deterministic sliding window tester, we maintain a set of compact summaries which represent all runs of $\mathcal{B}$ on the *complete* stream read so far (not only on the active window) with high probability.[8]

PROPOSITION 5.14. *For a given input stream $w \in \Sigma^*$, we can maintain a set of compact summaries $S = \{\kappa_w(q) \mid q \in Q\}$ such that for all $q \in Q$,*

— *$\kappa_w(q)$ starts in q, and*

— $\Pr[$*run $\pi_{w,q}$ is represented by $\kappa_w(q)$*$] \geqslant 2/3$.

PROOF. We maintain for the input word $w \in \Sigma^*$ a set of random compact summaries $S = \{\kappa_w(q) \mid q \in Q\}$ as follows.

For $w = \varepsilon$, we initialize $S = \{\kappa_\varepsilon(q) \mid q \in Q\}$ where $\kappa_\varepsilon(q) = (q, 0, c_0)$ for $q \in Q$. If $a \in \Sigma$ is the next input symbol in the stream, then S is updated to the new set S' of compact summaries by iterating over all transitions $q \xleftarrow{a} p$ in $\mathcal{B}$ and prolonging the compact summary starting in q by that transition. To prolong a compact summary

$$\kappa_w(q) = (q_m, r_m, c_m) \cdots (q_1, r_1, c_1) \tag{15}$$

we proceed similarly to Algorithm 1.

If p and $q = q_1$ are not in the same SCC then the new compact summary $\kappa_{wa}(p)$ is

$$(q_m, (r_m + 1) \bmod g, c'_m) \cdots (q_1, (r_1 + 1) \bmod g, c'_1)(p, 0, c_0),$$

where every counter state c'_i is chosen with probability $\rho(c_i, \mathtt{inc}, c'_i)$.

8 This is similar to the randomized SW-algorithm from Lemma 4.8 that stores in the probabilistic counters $\mathcal{Z}_q$ information about the complete input stream.

If p and $q = q_1$ belong to the same SCC, then $\kappa_{wa}(p)$ is

$$(q_m, (r_m + 1) \bmod g, c'_m) \cdots (q_2, (r_2 + 1) \bmod g, c'_2)(p, r_1, c_1),$$

where every counter state c'_i with $2 \leqslant i \leqslant m$ is chosen with probability $\rho(c_i, \mathtt{inc}, c'_i)$.

Note that the right-most triple of $\kappa_w(q)$ will be $(q, 0, c_0)$ with probability 1.

Finally we claim that for every $q \in Q$, the compact summary $\kappa_w(q)$ from (15) computed by the algorithm represents $\pi_{w,q}$ with probability $2/3$. Properties (C1) and (C2) are satisfied by construction. Furthermore, since the length of $\kappa_w(q)$ is bounded by $|Q|$ and each instance of $\mathcal{Z}$ has error probability $1/(3|Q|)$ the probability that property (C3) or (C4) is violated for some i is at most $1/3$ by the union bound. ■

For the randomized algorithm from the proof of Proposition 5.14 the same comment applies that was made after the proof of Lemma 4.8: the increments of the probabilistic counters do not have to be independent. Hence, in each step, only the random bits for incrementing a single (ℓ, h)-counter (with the above parameters ℓ and h) are needed. These random bits can be used for all counters that have to be incremented.

It remains to define an acceptance condition on compact summaries. For every $q \in Q$ we define

$$\mathrm{Acc}_{mod}(q) = \{\ell \bmod g \mid \ell \in \mathrm{Acc}(q) \text{ and } \ell \geqslant t\}.$$

Let $\kappa = (q_m, r_m, c_m) \cdots (q_1, r_1, c_1)$ be a compact summary. Since c_1 is the low initial state of the probabilistic counter, there exists a maximal index $i \in \{1, \dots, m\}$ such that c_i is low. We say that κ is *accepting* if $(n - r_i) \bmod g \in \mathrm{Acc}_{mod}(q_i)$.

PROPOSITION 5.15. *Let $w \in \Sigma^*$ with $|w| \geqslant n$ and let κ be a compact summary which represents π_{w,q_0}.*

(i) If $\mathsf{last}_n(w) \in L$, then κ is accepting.

(ii) If κ is accepting, then $\mathrm{pdist}(\mathsf{last}_n(w), L) \leqslant \gamma(n)$.

PROOF. Consider the SCC-factorization of $\pi = \pi_{w,q_0} = \pi_m \tau_{m-1} \cdots \tau_1 \pi_1$ and a compact summary $\kappa = (q_m, r_m, c_m) \cdots (q_1, r_1, c_1)$ representing π. Thus, $q_1 = q_0$ and $c_1 = c_0$. Consider the maximal index $1 \leqslant i \leqslant m$ where c_i is low, which means that $|\tau_{i-1}\pi_{i-1} \cdots \tau_1 \pi_1| < n - t$ by (C4). The run of $\mathcal{B}$ on $\mathsf{last}_n(w)$ has the form $\pi'_k \tau_{k-1} \pi_{k-1} \cdots \tau_1 \pi_1$ for some suffix π'_k of π_k and $k \geqslant i$. We have $|\pi'_k \tau_{k-1} \cdots \pi_i| = n - |\tau_{i-1}\pi_{i-1} \cdots \tau_1 \pi_1| > t$. By (C2) we know that

$$r_i = |\tau_{i-1}\pi_{i-1} \cdots \tau_1 \pi_1| \bmod g = n - |\pi'_k \tau_{k-1} \cdots \pi_i| \bmod g.$$

For (i) assume that $\mathsf{last}_n(w) \in L$. Thus, $\pi'_k \tau_{k-1} \pi_{k-1} \cdots \tau_1 \pi_1$ is an accepting run starting in q_0. By (C1) the run $\pi'_k \tau_{k-1} \cdots \pi_i$ starts in q_i. Hence, $\pi'_k \tau_{k-1} \cdots \pi_i$ is an accepting run from q_i of length at least t. By definition of $\mathrm{Acc}_{mod}(q_i)$ we have

$$n - r_i \bmod g = |\pi'_k \tau_{k-1} \cdots \pi_i| \bmod g \in \mathrm{Acc}_{mod}(q_i),$$

and therefore κ is accepting.

For (ii) assume that κ is accepting, i.e.

$$(n - r_i) \bmod g = |\pi'_k \tau_{k-1} \cdots \pi_i| \bmod g \in \mathrm{Acc}_{mod}(q_i).$$

Recall that $|\pi'_k \tau_{k-1} \cdots \pi_i| > t$. By definition of $\mathrm{Acc}_{mod}(q_i)$ there exists an accepting run from q_i whose length is congruent to $|\pi'_k \tau_{k-1} \cdots \pi_i| \bmod g$ and at least t. By point (i) from Corollary 5.13 we derive that $|\pi'_k \tau_{k-1} \cdots \pi_i| \in \mathrm{Acc}(q_i)$. We show that $|\pi_i \tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1| \geqslant n - \gamma(n) + t$ by a case distinction. If $i = m$, then clearly $|\pi_i \tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1| = |w| \geqslant n \geqslant n - \gamma(n) + t$. The latter inequality follows from our assumption $t + 1 \leqslant \gamma(n)/4$. If $i < m$, then c_{i+1} is high by maximality of i, which implies $|\tau_i \pi_i \cdots \tau_1 \pi_1| > n - \gamma(n) + t + 1$ by (C3). Since τ_i has length one, we have $|\pi_i \tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1| > n - \gamma(n) + t$.

Since $|\pi'_k \tau_{k-1} \cdots \pi_i| \in \mathrm{Acc}(q_i)$, we can apply Lemma 5.9 and obtain an accepting run ρ of length $|\pi'_k \tau_{k-1} \cdots \pi_i| \in \mathrm{Acc}(q_i)$ starting in q_i which t-simulates the internal run π_i. The prefix distance between ρ and $\pi'_k \tau_{k-1} \cdots \pi_i$ (which we define as the prefix distance between the words read along the two runs) is at most

$$|\pi'_k \tau_{k-1} \cdots \pi_{i+1} \tau_i| + t = n - |\pi_i \tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1| + t \leqslant n - n + \gamma(n) = \gamma(n).$$

Hence, the prefix distance from the accepting run $\rho \tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1$ to the run $\pi'_k \tau_{k-1} \pi_{k-1} \cdots \tau_1 \pi_1$ is also at most $\gamma(n)$. This implies $\mathrm{pdist}(\mathsf{last}_n(w), L) \leqslant \gamma(n)$. ■

We are now ready to prove Theorem 5.2.

PROOF OF THEOREM 5.2. Assume that the window size is such that $\gamma(n) \geqslant 4(t + 1)$ (recall that $4(t + 1)$ is our constant c from Theorem 5.2). We use the algorithm from Proposition 5.14, which is initialized by reading the initial window $\square^n$. It maintains a compact summary which represents π_{w,q_0} with probability at least 2/3 for the read stream prefix w. The algorithm accepts if that compact summary is accepting. From Proposition 5.15 we get:

- If $\mathsf{last}_n(w) \in L$, then the algorithm accepts with probability at least 2/3.
- If $\mathrm{pdist}(\mathsf{last}_n(w), L) > \gamma(n)$, then the algorithm rejects with probability at least 2/3.

This concludes the proof of Theorem 5.2. ■

From Theorem 5.2 we can easily deduce Corollary 5.3: Let $\gamma(n) = \epsilon n$ for some $0 < \epsilon < 1$ and let c be the constant from Theorem 5.2. Then the condition $\gamma(n) \geqslant c$ becomes $n \geqslant c/\epsilon$. Hence, for a window size $n \geqslant c/\epsilon$, Theorem 5.2 yields a randomized sliding window tester with two-sided error that uses space $O(\log(n/\gamma(n)) = O(\log(1/\epsilon))$. For $n < c/\epsilon$ we can use a trivial sliding window tester that stores the window content explicitly using $O(1/\epsilon)$ bits.

5.3.3 Sliding window testers with one-sided error

In the following, we turn to sliding window testers with one-sided error and prove Theorem 5.4, i.e., we present a false-biased sliding window tester for languages in $\bigcup(\mathbf{Triv}, \mathbf{SF})$ with constant Hamming gap using $O(\log\log n)$ space. By the following lemma, it suffices to consider the cases $L \in \mathbf{Triv}$ and $L \in \mathbf{SF}$.

LEMMA 5.16. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be randomized false-biased sliding window testers for L_1 and L_2, respectively, for window size n with Hamming gap $\gamma(n)$. Then, there exists a randomized false-biased sliding window tester for $L_1 \cup L_2$ for window size n with Hamming gap $\gamma(n)$ using space $O(s(\mathcal{P}_1) + s(\mathcal{P}_2))$.*

PROOF. First we reduce the error probability of $\mathcal{P}_i$ ($i \in \{1, 2\}$) from $1/3$ to $1/9$ by running 2 independent and parallel copies of $\mathcal{P}_i$ and reject if and only if one of the copies rejects. Then, we run both algorithms in parallel and accept if and only if one of them accepts. If the window belongs to $L_1 \cup L_2$ then either $\mathcal{P}_1$ or $\mathcal{P}_2$ accepts with probability 1. If the window w satisfies $\mathrm{dist}(w, L_1 \cup L_2) = \min(\mathrm{dist}(w, L_1), \mathrm{dist}(w, L_2)) > \gamma(n)$ then $\mathrm{dist}(w, L_i) > \gamma(n)$ for both $i \in \{1, 2\}$. Hence, both algorithms falsely accept with probability at most $1/9$ and the combined algorithm falsely accepts with probability at most $1/9 + 1/9 \leqslant 1/3$. ■

The case of a regular trivial language is covered by the following result:

THEOREM 5.17. *Let L be a language and $\gamma(n)$ be a function. The following statements are equivalent:*

- *L is $(\gamma(n) + c)$-trivial for some number $c \in \mathbb{N}$.*
- *There is a deterministic sliding window tester with Hamming gap $\gamma(n) + c'$ for L which uses constant space for some number $c' \in \mathbb{N}$.*

PROOF. Assume first that L is $(\gamma(n) + c)$-trivial. Let $n \in \mathbb{N}$ be a window size. If $L \cap \Sigma^n = \emptyset$, then the algorithm always rejects, which is obviously correct since any active window of size n has infinite Hamming distance to L. On the other hand, if $L \cap \Sigma^n \neq \emptyset$ then the Hamming distance between an arbitrary active window of size n and L is at most $\gamma(n) + c$. Hence, the algorithm that always accepts achieves a Hamming gap of $\gamma(n) + c$.

We now show the converse statement.[9] For each window size $n \in \mathbb{N}$ let $\mathcal{P}_n$ be a deterministic sliding window tester for L with Hamming gap $\gamma(n) + c'$ such that the number of states of $\mathcal{P}_n$ is constant. Assume that $\mathcal{P}_n$ has at most s states for every n. Let $N \subseteq \mathbb{N}$ be the set of all n such that $L \cap \Sigma^n \neq \emptyset$. Note that every $\mathcal{P}_n$ with $n \in N$ accepts a nonempty language. Every $\mathcal{P}_n$ is a DFA with a most s states over the fixed alphabet Σ. The number of pairwise nonisomorphic DFAs with at most s states over the input alphabet Σ is bounded by a fixed

9 The converse statement is not needed for the proof of Theorem 5.4 but we think it is of independent interest.

constant d. Hence, at most d nonisomorphic DFAs can appear in the list $(\mathcal{P}_n)_{n \in N}$. We therefore can choose numbers $n_1 < n_2 < \cdots < n_e$ from N with $e \leqslant d$ such that for every $n \in N$ there exists a unique $i \in \{1, 2, \ldots, e\}$ with $n_i \leqslant n$ and $\mathcal{P}_n = \mathcal{P}_{n_i}$ (here and in the following we do not distinguish between isomorphic DFAs). Let us choose for every $1 \leqslant i \leqslant e$ some word $u_i \in L$ of length n_i. Now, take any $n \in N$ and assume that $\mathcal{P}_n = \mathcal{P}_{n_i}$ where $n_i \leqslant n$. Consider any word $u \in \Sigma^* u_i$. Since $\mathsf{last}_{n_i}(u) = u_i \in L$, $\mathcal{P}_{n_i}$ has to accept u. Hence, $\mathcal{P}_n$ accepts all words from $\Sigma^* u_i$. In particular, for every word x of length $n - n_i$, $\mathcal{P}_n$ accepts $x u_i$. This implies that $\mathrm{dist}(x u_i, L) \leqslant \gamma(n) + c'$ for all $x \in \Sigma^{n-n_i}$. Recall that this holds for all $n \in N$ and that N is the set of all lengths realized by L. Hence, if we define $c'' := \max\{n_1, \ldots, n_e\}$, then every word w of length $n \in N$ has Hamming distance at most $\gamma(n) + c' + c''$ from a word in L. Therefore, L is $(\gamma(n) + c)$-trivial with $c = c' + c''$. ■

Let us now turn to the case of a regular suffix-free language L. We again consider an rDFA $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ for L whose nontransient SCCs have uniform period $g \geqslant 1$. Since L is suffix-free, $\mathcal{B}$ has the property that no final state can be reached from a final state by a nonempty run.

We adapt the definition of a path description from Section 3.8. In the following, a path description is a sequence

$$P = (q_k, a_k, p_{k-1}), C_{k-1}, \ldots, (q_2, a_2, p_1), C_1, (q_1, a_1, p_0), C_0, q_0. \tag{16}$$

where $C_{k-1}, \ldots, C_0$ is a chain (from right to left) in the SCC-ordering of $\mathcal{B}$, $p_i, q_i \in C_i$, $q_{i+1} \xleftarrow{a_{i+1}} p_i$ is a transition in $\mathcal{B}$ for all $0 \leqslant i \leqslant k-1$, and $q_k \in F$. Each path description defines a *partial rDFA* $\mathcal{B}_P = (Q_P, \Sigma, \{q_k\}, \delta_P, q_0)$ by restricting $\mathcal{B}$ to the state set $Q_P = \bigcup_{i=0}^{k-1} C_i \cup \{q_k\}$, restricting the transitions of $\mathcal{B}$ to internal transitions from the SCCs C_i and the transitions $q_{i+1} \xleftarrow{a_{i+1}} p_i$, and declaring q_k to be the only final state. This rDFA is partial since for every state p and every symbol $a \in \Sigma$ there exists at most one transition $q \xleftarrow{a} p$ in $\mathcal{B}_P$. Since the number of path descriptions P is finite and $\mathsf{L}(\mathcal{B}) = \bigcup_P \mathsf{L}(\mathcal{B}_P)$, we can fix a single path description P and provide a sliding window tester for $\mathsf{L}(\mathcal{B}_P)$ (we again use Lemma 5.16 here).

From now on, we fix a path description P as in (16). The acceptance sets $\mathrm{Acc}_P(q)$ are defined with respect to the restricted automaton $\mathcal{B}_P$. If all C_i are transient, then $\mathsf{L}(\mathcal{B}_P)$ is a singleton and we can use a trivial sliding window tester with space complexity $O(1)$. Now, assume the contrary and let $0 \leqslant e \leqslant k-1$ be maximal such that C_e is nontransient.

LEMMA 5.18. *There exist numbers $r_0, \ldots, r_{k-1}, s_0, \ldots, s_e \in \mathbb{N}$ such that the following holds:*

(i) For all $e+1 \leqslant i \leqslant k$, the set $\mathrm{Acc}_P(q_i)$ is a singleton.

(ii) For all $0 \leqslant i \leqslant k-1$, every run from q_i to q_{i+1} has length $r_i \pmod{g}$.

(iii) For all $0 \leqslant i \leqslant e$, $\mathrm{Acc}_P(q_i) =_{s_i} \sum_{j=i}^{k-1} r_j + g\mathbb{N}$.

PROOF. Point (i) follows immediately from the definition of transient SCCs. Let us now show (ii) and (iii). Let $0 \leqslant i \leqslant k-1$ and let N_i be the set of lengths of runs of the form $q_{i+1} \xleftarrow{a_{i+1}} p_i \xleftarrow{w} q_i$ in $\mathcal{B}_P$. If C_i is transient, then $N_i = \{1\}$. Otherwise, by Lemma 3.18 there exist a number $r_i \in \mathbb{N}$ and a cofinite set $D_i \subseteq \mathbb{N}$ such that $N_i = r_i + gD_i$. We can summarize both cases by saying that there exist a number $r_i \in \mathbb{N}$ and a set $D_i \subseteq \mathbb{N}$ which is either cofinite or $D_i = \{0\}$ such that $N_i = r_i + gD_i$. This implies point (ii). Moreover, the acceptance sets in $\mathcal{B}_P$ satisfy

$$\mathrm{Acc}_P(q_i) = \sum_{j=i}^{k-1} N_j = \sum_{j=i}^{k-1} (r_j + gD_j) = \sum_{j=i}^{k-1} r_j + g \sum_{j=i}^{k-1} D_j.$$

For all $0 \leqslant i \leqslant e$ we get $\mathrm{Acc}_P(q_i) =_{s_i} \sum_{j=i}^{k-1} r_j + g\mathbb{N}$ for some threshold $s_i \in \mathbb{N}$ (note that a nonempty sum of cofinite subsets of $\mathbb{N}$ is again cofinite). ■

Let us fix the numbers r_i and s_i from Lemma 5.18. Let p be a random prime with $\Theta(\log \log n)$ bits. By choosing the Θ-constant large enough and using Lemma 4.9 (where we set $m = n$, $a = \ell$ and $b = n$) we obtain for every $0 \leqslant \ell < n$ the inequality $\Pr[\ell \equiv n \pmod p] \leqslant 1/3$. Define the threshold

$$s = \max\{k, \sum_{j=0}^{k-1} r_j, s_0, \dots, s_e\}$$

and for a word $w \in \Sigma^*$ define the function $\ell_w \colon Q \to \mathbb{N} \cup \{\infty\}$ where

$$\ell_w(q) = \inf\{\ell \in \mathbb{N} \mid \delta_P(\mathsf{last}_\ell(w), q) = q_k\}$$

(we set $\inf \emptyset = \infty$). We now define an acceptance condition on $\ell_w(q)$. If $n \notin \mathrm{Acc}_P(q_0)$, we always reject. Otherwise, we accept w if and only if $\ell_w(q_0) \equiv n \pmod p$.

LEMMA 5.19. *Let $n \in \mathrm{Acc}_P(q_0)$ be a window size with $n \geqslant s + |Q_P|$ and $w \in \Sigma^*$ with $|w| \geqslant n$. There exists a constant $c > 0$ such that:*

(i) if $\mathsf{last}_n(w) \in \mathsf{L}(\mathcal{B}_P)$, then w is accepted (in the above sense) with probability 1, and

(ii) if $\mathrm{pdist}(\mathsf{last}_n(w), \mathsf{L}(\mathcal{B}_P)) > c$, then w is rejected with probability at least $2/3$.

PROOF. Consider a word $w \in \Sigma^*$ with $|w| \geqslant n$. We consider several cases.

Case 1: $\mathsf{last}_n(w) \in \mathsf{L}(\mathcal{B}_P)$. Since $\mathsf{L}(\mathcal{B}_P) \subseteq L$ is suffix-free, we have $\ell_w(q_0) \equiv n \pmod p$ and w is accepted with probability 1, which shows statement (i) from the lemma.

Case 2: $\mathsf{last}_n(w) \notin \mathsf{L}(\mathcal{B}_P)$. We then have $\ell_w(q_0) \neq n$, which yields the following two subcases.

Case 2.1: $\ell_w(q_0) < n$. Then, by the choice of p we have $\ell_w(q_0) \not\equiv n \pmod p$ with probability at least $2/3$. Hence, w is rejected with probability at least $2/3$.

Case 2.2: $\ell_w(q_0) > n$. We will show that this implies $\mathrm{pdist}(\mathsf{last}_n(w), \mathsf{L}(\mathcal{B}_P)) \leqslant c$ for a constant c. For this c, statement (ii) from the lemma then holds: if $\mathrm{pdist}(\mathsf{last}_n(w), \mathsf{L}(\mathcal{B}_P)) > c$, we must have $\ell_w(q_0) < n$, which by Case 2.1 implies that w is rejected with probability at least $2/3$.

Let π be the run of $\mathcal{B}_P$ on $\mathsf{last}_n(w)$ starting from the initial state, and let its SCC-factorization be $\pi = \pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_0 \pi_0$. We have $|\pi| = n$. Since $\ell_w(q_0) > n$, the run π can be strictly extended to a run to q_k and hence we must have $m < k$. For all $0 \leqslant i \leqslant m$, the run π_i is an internal run in the SCC C_i from q_i to p_i. For all $0 \leqslant i \leqslant m-1$ we have $\tau_i = q_{i+1}\, a_{i+1}\, p_i$ and $|\tau_i \pi_i| \equiv r_i \pmod g$ by point (ii) from Lemma 5.18. We claim that there exists an index $0 \leqslant i_0 \leqslant m$ such that the following three properties hold:

1. q_{i_0} is nontransient,
2. $|\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_{i_0} \pi_{i_0}| \geqslant s$,
3. $|\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_{i_0+1} \pi_{i_0+1}| \leqslant s + m$
   (note that $|\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_{i_0+1} \pi_{i_0+1}| = 0$ is possible).

Indeed, let $0 \leqslant i \leqslant m$ be the smallest integer such that q_i is nontransient (recall that $n \geqslant |Q_P|$ and hence π must traverse a nontransient SCC). Then, the run $\tau_{i-1} \pi_{i-1} \cdots \tau_0 \pi_0$ only passes transient states except for its last state q_i and hence its length is bounded by $|Q_P|$. Therefore, we have

$$|\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_i \pi_i| = n - |\tau_{i-1} \pi_{i-1} \cdots \tau_0 \pi_0| \geqslant n - |Q_P| \geqslant s.$$

Hence, there exists i satisfying properties (1) and (2) (with i_0 replaced by i). Let $0 \leqslant i_0 \leqslant m$ be the largest integer satisfying properties (1) and (2). We show that property (3) holds for i_0.

If the run $\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_{i_0+1} \pi_{i_0+1}$ only passes transient states, then its length is bounded by $m - i_0 \leqslant s + m$, and we are done. Otherwise, let $i_0 + 1 \leqslant j \leqslant m$ be the smallest integer such that q_j is nontransient. The run $\tau_{j-1} \pi_{j-1} \cdots \tau_{i_0+1} \pi_{i_0+1}$ only passes transient states except for its last state q_j and therefore it has length $j - i_0 - 1$. By maximality of i_0, we have $|\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_j \pi_j| < s$ and hence property (3) holds:

$$\begin{aligned} |\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_{i_0+1} \pi_{i_0+1}| &= |\pi_m \cdots \tau_j \pi_j| + |\tau_{j-1} \pi_{j-1} \cdots \tau_{i_0+1} \pi_{i_0+1}| \\ &< s + j - i_0 \leqslant s + m. \end{aligned}$$

In the rest of the proof let $0 \leqslant i_0 \leqslant m$ be the above index satisfying properties (1)-(3). Since q_{i_0} is nontransient, we have $i_0 \leqslant e$ and therefore

$$\mathrm{Acc}_P(q_{i_0}) =_s \sum_{j=i_0}^{k-1} r_j + g\mathbb{N} \tag{17}$$

by Lemma 5.18(iii) and $s \geqslant s_{i_0}$. We claim that

$$|\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_{i_0} \pi_{i_0}| \in \mathrm{Acc}_P(q_{i_0}). \tag{18}$$

Since $n \geqslant s$ and $n \in \mathrm{Acc}_P(q_0) =_s \sum_{j=0}^{k-1} r_j + g\mathbb{N}$ we have $n \in \sum_{j=0}^{k-1} r_j + g\mathbb{N}$. This implies

$$|\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_{i_0} \pi_{i_0}| = n - |\tau_{i_0-1} \pi_{i_0-1} \cdots \tau_0 \pi_0| \equiv n - \sum_{j=0}^{i_0-1} r_j \equiv \sum_{j=i_0}^{k-1} r_j \pmod g.$$

In addition, we have $|\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_{i_0}\pi_{i_0}| \geqslant s$ by property (2). Since $s \geqslant \sum_{j=i_0}^{k-1} r_j$, we get

$$|\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_{i_0}\pi_{i_0}| \in \sum_{j=i_0}^{k-1} r_j + g\mathbb{N}.$$

Finally, we obtain (18) from (17).

By Lemma 5.9 and (18), there is an accepting run π' from q_{i_0} which t-simulates the internal run π_{i_0} and has length $|\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_{i_0}\pi_{i_0}|$. Here, t is a constant only depending on $\mathcal{B}$. The prefix distance between the runs $\pi = \pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_0\pi_0$ and $\pi'\tau_{i_0-1}\pi_{i_0-1}\cdots\tau_0\pi_0$ is at most t in case $i_0 = m$, and at most

$$\begin{aligned} |\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_{i_0}| + t &= |\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_{i_0+1}\pi_{i_0+1}| + 1 + t \\ &\leqslant 1 + s + m + t =: c \end{aligned}$$

in case $i_0 < m$ due to property (3): Hence, the prefix distance between $\mathsf{last}_n(w)$ and $\mathsf{L}(\mathcal{B}_P)$ is bounded by the constant c. As explained before, statement (ii) of the lemma then holds. ■

PROOF OF THEOREM 5.4. Let $n \in \mathbb{N}$ be the window size. By the previous discussion, it suffices to give a randomized sliding window tester (with the properties stated in Theorem 5.4) for a fixed partial automaton $\mathcal{B}_P$. Assume $n \geqslant s + |Q|$, otherwise a trivial tester can be used. If $n \notin \mathrm{Acc}_P(q_0)$, the tester always rejects. Otherwise, the tester picks a random prime p with $\Theta(\log\log n)$ bits and maintains $\ell_w(q) \bmod p$ for all $q \in Q_P$, where w is the stream read so far, which requires $O(\log\log n)$ bits. When a symbol $a \in \Sigma$ is read, we can update ℓ_{wa} using ℓ_w: If $q = q_k$, then $\ell_{wa}(q) = 0$, otherwise $\ell_{wa}(q) = 1 + \ell_w(\delta_P(a, q)) \bmod p$ where $1 + \infty = \infty$. The tester accepts if $\ell_w(q_0) \equiv n \pmod p$. Lemma 5.19 guarantees that the tester is false-biased. ■

Note that in contrast to the randomized sliding window algorithm from Section 4.4, the randomized sliding window tester from this section only uses the modulo counting technique; Bernoulli counters are not needed. As a consequence, the tester only has to make a random choice at the beginning (where the prime p is chosen) before the first input symbol arrives. Then it continues deterministically.

5.4 Lower bounds

In this section we prove our lower bounds. In Section 5.4.1 we will prove Theorem 5.5 and Theorem 5.8. Theorem 5.6 and Theorem 5.7 will be shown in Section 5.4.2.

5.4.1 Regular nontrivial languages

In this section, we prove Theorem 5.5 and Theorem 5.8. For this, we first have to study regular trivial languages in more detail. We will also show a result of independent interest: every regular $o(n)$-trivial language L is already trivial (i.e., $O(1)$-trivial); see Theorem 5.24.

Given $i, j \geqslant 0$ and a word w of length at least $i+j$ we define $\mathrm{cut}_{i,j}(w) = y$ such that $w = xyz$, $|x| = i$ and $|z| = j$. If $|w| < i + j$, then $\mathrm{cut}_{i,j}(w)$ is undefined. For a language L we define the *cut-language* $\mathrm{cut}_{i,j}(L) = \{\mathrm{cut}_{i,j}(w) \mid w \in L, |w| \geqslant i + j\}$.

LEMMA 5.20. *If L is regular, then there are finitely many languages $\mathrm{cut}_{i,j}(L)$.*

PROOF. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be a DFA for L. Given $i, j \geqslant 0$, let I be the set of states reachable from q_0 via i symbols and let F' be the set of states from which F can be reached via j symbols. Then, the NFA $\mathcal{A}_{i,j} = (Q, \Sigma, I, \delta, F')$ recognizes $\mathrm{cut}_{i,j}(L)$. Since there are at most $2^{2|Q|}$ such choices for I and F', the number of languages of the form $\mathrm{cut}_{i,j}(L)$ must be finite. ■

LEMMA 5.21. *If $\mathrm{cut}_{i,j}(L)$ is a length language for some $i, j \geqslant 0$, then L is trivial.*

PROOF. Assume that $\mathrm{cut}_{i,j}(L)$ is a length language. Let $n \in \mathbb{N}$ such that $L \cap \Sigma^n \neq \emptyset$. We claim that $\mathrm{dist}(w, L) \leqslant i + j$ for all $w \in \Sigma^n$. If $n < i + j$ this is clear. So, assume that $n \geqslant i + j$. Let $w \in \Sigma^n$ and $w' \in L \cap \Sigma^n$. Then, $\mathrm{cut}_{i,j}(w') \in \mathrm{cut}_{i,j}(L)$ and hence also $\mathrm{cut}_{i,j}(w) \in \mathrm{cut}_{i,j}(L)$. Therefore, there exist $x \in \Sigma^i$ and $z \in \Sigma^j$ such that $x \ \mathrm{cut}_{i,j}(w)\ z \in L$ satisfies $\mathrm{dist}(w, x\ \mathrm{cut}_{i,j}(w)\ z) \leqslant i + j$. ■

The *restriction* of a language L to a set of lengths $N \subseteq \mathbb{N}$ is $L|_N = \{w \in L \mid |w| \in N\}$. A language L *excludes a word w as a factor* if w is not a factor of any word in L. A simple but important observation is that if L excludes w as a factor and v contains k disjoint occurrences of w, then $\mathrm{dist}(v, L) \geqslant k$: If we change at most $k - 1$ many symbols in v, then the resulting word v' must still contain w as a factor and hence $v' \notin L$.

LEMMA 5.22. *Let L be regular. If for all $i, j \geqslant 0$, $\mathrm{cut}_{i,j}(L)$ is not a length language, then there exists an arithmetic progression $N = d + e\mathbb{N}$ such that the restriction $L|_N$ is infinite and excludes a factor.*

PROOF. First notice that $\mathrm{cut}_{i,j}(L)$ determines $\mathrm{cut}_{i+1,j}(L)$ and $\mathrm{cut}_{i,j+1}(L)$: we have $\mathrm{cut}_{i+1,j}(L) = \mathrm{cut}_{1,0}(\mathrm{cut}_{i,j}(L))$ and similarly for $\mathrm{cut}_{i,j+1}(L)$. Since the number of cut-languages $\mathrm{cut}_{i,j}(L)$ is finite by Lemma 5.20, there exist numbers $i \geqslant 0$ and $d > 0$ such that $\mathrm{cut}_{i,0}(L) = \mathrm{cut}_{i+d,0}(L)$. Hence, we have $\mathrm{cut}_{i,j}(L) = \mathrm{cut}_{i+d,j}(L)$ for all $j \geqslant 0$. By the same argument, there exist numbers $j \geqslant 0$ and $e > 0$ such that $\mathrm{cut}_{i,j}(L) = \mathrm{cut}_{i,j+e}(L) = \mathrm{cut}_{i+d,j}(L) = \mathrm{cut}_{i+d,j+e}(L)$, which implies $\mathrm{cut}_{i,j}(L) = \mathrm{cut}_{i,j+h}(L) = \mathrm{cut}_{i+h,j}(L) = \mathrm{cut}_{i+h,j+h}(L)$ for some $h > 0$ (we can take $h = ed$). This implies that $\mathrm{cut}_{i,j}(L)$ is closed under removing prefixes and suffixes of length h.

By assumption $\mathrm{cut}_{i,j}(L)$ is not a length language, i.e., there exist words $y' \in \mathrm{cut}_{i,j}(L)$ and $y \notin \mathrm{cut}_{i,j}(L)$ of the same length k. Let $N = \{k + i + j + hn \mid n \in \mathbb{N}\}$. For any $n \in \mathbb{N}$ the restriction $L|_N$ contains a word of length $k + i + j + hn$ because $y' \in \mathrm{cut}_{i,j}(L) = \mathrm{cut}_{i+hn,j}(L)$. This proves that $L|_N$ is infinite.

Let u be an arbitrary word which contains for every remainder $0 \leqslant r \leqslant h-1$ an occurrence of y as a factor starting at a position which is congruent to r mod h (these occurrences do not have to be disjoint). We claim that $L|_N$ excludes $a^i u a^j$ as a factor where a is an arbitrary symbol. Assume that there exists a word $w \in L|_N$ which contains $a^i u a^j$ as a factor. Then, $\mathrm{cut}_{i,j}(w)$ contains u as a factor, has length $k + hn$ for some $n \geqslant 0$, and belongs to $\mathrm{cut}_{i,j}(L)$. Therefore, $\mathrm{cut}_{i,j}(w)$ also contains h many occurrences of y, one per remainder $0 \leqslant r \leqslant h-1$. Consider the occurrence of y in $\mathrm{cut}_{i,j}(w)$ which starts at a position that is divisible by h, i.e., we can factorize $\mathrm{cut}_{i,j}(w) = xyz$ such that $|x|$ is a multiple of h. Since $|\mathrm{cut}_{i,j}(w)| = k + hn$ and $|y| = k$, then $|z|$ is also a multiple of h. Therefore, $y \in \mathrm{cut}_{i+|x|,j+|z|}(L) = \mathrm{cut}_{i,j}(L)$, which is a contradiction. ■

LEMMA 5.23. *If $L \in \mathbf{Reg} \setminus \mathbf{Triv}$ then there are a restriction $L|_N$ that excludes some factor w_f and words x, y, z such that $|y| > 0$ and $xy^*z \subseteq L|_N$.*

PROOF. By Lemma 5.21, $\mathrm{cut}_{i,j}(L)$ is not a length language for all $i, j \geqslant 0$. Let N be the set of lengths from Lemma 5.22 such that $L|_N$ is infinite and excludes some factor w_f. Since N is an arithmetic progression, $L|_N$ is regular. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be a DFA for $L|_N$. Since $\mathsf{L}(\mathcal{A})$ is infinite, there must exist words x, y, z such that $y \neq \varepsilon$ and for $\delta(q_0, x) = q$ we have $\delta(q, y) = q$ and $\delta(q, z) \in F$. ■

Before we prove Theorem 5.5 let us first show the following result of independent interest:

THEOREM 5.24. *For every regular language L, the following statements are equivalent:*

(i) *L is trivial.*

(ii) *L is $o(n)$-trivial.*

(iii) $\mathrm{cut}_{i,j}(L)$ *is a length language for some $i, j \geqslant 0$.*

PROOF. If $\mathrm{cut}_{i,j}(L)$ is a length language then L is trivial by Lemma 5.21, and thus also $o(n)$-trivial. It remains to show the direction (ii) to (iii). Assume that L is $o(n)$-trivial. If (iii) would not hold then some infinite restriction $L|_N$ of L excludes a factor w_f by Lemma 5.22. Hence, if $n \in N$ is a length with $L|_N \cap \Sigma^n \neq \emptyset$, then any word v of length n which contains at least $\lfloor n/|w_f| \rfloor$ many disjoint occurrences of w_f, has distance $\mathrm{dist}(v, L) \geqslant \lfloor n/|w_f| \rfloor$ to L. Then, L is not $o(n)$-trivial, which is a contradiction. ■

PROOF OF THEOREM 5.5. We will prove the two lower bounds in Theorem 5.5 for the more general class of nondeterministic and co-nondeterministic sliding window testers. A *nondeterministic* sliding window tester for a language L and window size $n \in \mathbb{N}$ with Hamming gap $\gamma(n)$ is a nondeterministic finite automaton $\mathcal{P}_n$ such that for all input words $w \in \Sigma^*$ we have the following (recall from Section 2.2 that a successful run is a run from an initial state to a final state):

— If $\mathsf{last}_n(w) \in L$, then there is at least one successful run of $\mathcal{P}_n$ on w.

— If $\mathrm{dist}(\mathsf{last}_n(w), L) > \gamma(n)$, then there is no successful run of $\mathcal{P}_n$ on w.

In contrast, $\mathcal{P}_n$ is co-nondeterministic if for all $w \in \Sigma^*$ we have:

— If $\mathsf{last}_n(w) \in L$, then all runs of $\mathcal{P}_n$ on w that start in an initial state are successful.
— If $\mathrm{dist}(\mathsf{last}_n(w), L) > \gamma(n)$, then there is a nonsuccessful run of $\mathcal{P}_n$ on w that starts in an initial state.

The space complexity of $\mathcal{P}_n$ is $\log |\mathcal{P}_n|$. Clearly, every true-biased (resp., false-biased) sliding window tester is a nondeterministic (resp., co-nondeterministic) one.

Assume that $L \in \mathbf{Reg}\backslash\mathbf{Triv}$. We will prove an $\log n - O(1)$ lower bound for nondeterministic sliding window testers, and hence also for true-biased and deterministic sliding window testers. By the power set construction one can transform a co-nondeterministic sliding window tester with m memory states into an equivalent nondeterministic (and in fact, even deterministic) sliding window tester with 2^m memory states. Hence, an $\log n - O(1)$ lower bound for nondeterministic sliding window testers immediately yields an $\log(\log n - O(1)) \geqslant \log\log n - O(1)$ lower bound for co-nondeterministic sliding window testers, and hence also for false-biased sliding window testers.

By Lemma 5.23 there are a restriction $L|_N$ that excludes some factor w_f and words x, y, z such that $|y| > 0$ and $xy^*z \subseteq L|_N$. Let $c = |w_f| > 0$ and choose $0 < \epsilon < 1/c$. Moreover, let $d = |xz|$ and $e = |y| > 0$, which satisfy $d + e\mathbb{N} \subseteq N$. Recall that every word v that contains k disjoint occurrences of w_f has Hamming distance at least k from any word in $L|_N$.

Fix a window size $n \in N$ and consider a nondeterministic sliding window tester $\mathcal{P}_n$ for L and window size n with Hamming gap ϵn. Define for $k \geqslant 0$ the input stream

$$v_k = w_f^n x y^k z.$$

Let $\alpha = c\epsilon < 1$. If $0 \leqslant k \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$, then the suffix of v_k of length n contains at least

$$\left\lfloor \frac{n-d-ek}{c} \right\rfloor \geqslant \left\lfloor \frac{n-d-(1-\alpha)n+c+d}{c} \right\rfloor = \left\lfloor \frac{\alpha n + c}{c} \right\rfloor = \lfloor \epsilon n + 1 \rfloor > \epsilon n$$

many disjoint occurrences of w_f. Hence, $\mathcal{P}_n$ has no successful run on an input stream v_k with $0 \leqslant k \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$.

Assume now that the window size n satisfies $n \geqslant d$ and $n \equiv d \pmod{e}$. Write $n = d + le$ for some $l \geqslant 0$. We have $l = \frac{n-d}{e} > \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$. The suffix of $v_l = w_f^n x y^l z$ of length n is $xy^l z \in L|_N$. Therefore, there exists a successful run π of $\mathcal{P}_n$ on v_l. Let m be the number of states of $\mathcal{P}_n$. For $0 \leqslant i \leqslant l$ let p_i be the state on the run π that is reached after the prefix $w_f^n x y^i$ of v_l.

In order to deduce a contradiction, let us assume that $m \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$. Then, there must exist numbers i and j with $0 \leqslant i < j \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$ such that $p_i = p_j =: p$. By cutting off cycles at p from the run π and repeating this, we finally obtain a run of $\mathcal{P}_n$ on an input stream $v_k = w_f^n x y^k z$ with $k \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$. This run is still successful. But this contradicts our previous observation that $\mathcal{P}_n$ has no successful run on an input stream v_k with $0 \leqslant k \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$. We conclude that $\mathcal{P}_n$ must have more than $\lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$ states. This implies

$$s(\mathcal{P}_n) \geqslant \log\left(\frac{(1-\alpha)n-c-d}{e}\right) \geqslant \log n - O(1),$$

which proves the theorem. ■

For the proof of Theorem 5.8 we need a promise variant of the communication problem EQ_m (see Section 4.5). With $\mathrm{EQ}_m^{\geqslant}$ we denote the following promise communication problem: Alice's (resp., Bob's) input is a number $i \in \{1,\dots,m\}$ (resp., $j \in \{1,\dots,m\}$) and the promise is that $i \geqslant j$ (i.e., we do not care about the output of the protocol in case $i < j$). If $i = j$ then Bob's final output must be 1 and if $i > j$ then Bob's final output must be 0. We claim that the randomized one-way communication complexity of $\mathrm{EQ}_m^{\geqslant}$ is $\Omega(\log\log m)$.

Since the randomized one-way communication complexity of EQ_m is $\Omega(\log\log m)$ by Theorem 4.11, it suffices to show that a randomized one-way protocol for $\mathrm{EQ}_m^{\geqslant}$ with cost $c(m)$ and error probability λ yields a randomized one-way protocol for EQ_m with cost $2c(m)$ and error probability 2λ. This is easy to see: Assume that P_m is a randomized one-way protocol for $\mathrm{EQ}_m^{\geqslant}$ with cost $c(m)$. To get a randomized one-way protocol for EQ_m, Alice and Bob run two copies of P_m, one on inputs i, j and the other one on inputs $m-i, m-j$ in parallel and with independent random bits. If both copies of P_m yield output 1 then Bob returns 1. In all other cases, Bob returns 0. If $i = j$ then this combined protocol returns 1 with probability at least $1-2\lambda$. On the other hand if $i > j$ or $i < j$ then the protocol returns 0 with probability at least $1-\lambda$.

PROOF OF LEMMA 5.8. Let L be a language in $L \in \mathbf{Reg} \setminus \mathbf{Triv}$. By Lemma 5.23 there are a restriction $L|_N$ that excludes some factor w_f and words x, y, z such that $|y| > 0$ and $xy^*z \subseteq L|_N$.

Let $b = |y|$ and $c = |xz|$. Note in the following that b, c, and $|w_f|$ are constants. Fix a window size $n \in N$ such that $n - c$ is a multiple of b. Since $xy^*z \subseteq L|_N$, we have $n \in N$. Define the word $v = uw_f^k$ where $k = \lfloor \gamma(n) \rfloor + 1 > \gamma(n)$ and u is a word of length at most $b-1$ such that $|v|$ is a multiple of b. Let $l \in \mathbb{N}$ be such that $|v| = b \cdot l$. Since b divides $n - c$ we can write $n - c = (m \cdot l + r) \cdot b$ for $m \in \mathbb{N}$ and $0 \leqslant r \leqslant l-1$. Choose the constant ϵ from the theorem statement such that $0 < \epsilon < \frac{1}{|w_f|}$ and hence $\gamma(n) \leqslant \epsilon n$. Assuming n is large enough, we obtain $k = \lfloor \gamma(n) \rfloor + 1 \leqslant \frac{1}{|w_f|}(n-b-c)$, i.e., $b \cdot l = |v| \leqslant b + k \cdot |w_f| \leqslant n - c$ and hence $m \geqslant 1$. Moreover, we have $l = |v|/b = \Theta(\gamma(n))$ and therefore

$$m = \frac{n-c}{b \cdot l} - \frac{r}{l} = \Theta(n/\gamma(n)).$$

Consider now a randomized sliding window tester $\mathcal{P}_n$ with two-sided error for L and window size n with Hamming gap $\gamma(n)$. We show that from $\mathcal{P}_n$ we can obtain a randomized one-way protocol for $\mathrm{EQ}_m^{\geqslant}$.

Alice produces from her input $i \in \{1,\dots,m\}$ the word $vxy^{r+(m-i)l}$. She then runs $\mathcal{P}_n$ on this word and sends the memory state to Bob. Bob continues the run of the randomized sliding window tester, starting from the transferred memory state, with the input stream $y^{jl}z$, where

$j \in \{1, \dots, m\}$ is his input. He obtains the memory state reached after the input $vxy^{r+(m-i+j)l}z$. Finally, Bob outputs the answer given by the randomized sliding window tester. If $i = j$, then $\mathsf{last}_n(vxy^{r+(m-i+j)l}z) = xy^{m\cdot l+r}z \in L$ and Bob accepts with high probability. On the other hand, if $i > j$, then $\mathsf{last}_n(vxy^{r+(m-i+j)l}z)$ contains v (recall that $|v| = b\cdot l$ and hence $|xy^{r+(m-i+j)l}z| \leqslant n-|v|$). Since v contains strictly more than $\gamma(n)$ disjoint occurrences of w_f (an excluded factor of $L|_N$), we have $\mathrm{dist}(\mathsf{last}_n(w), L) > \gamma(n)$ (here it is important that $n \in N$). Thus, Bob rejects with high probability. We therefore have a correct protocol for $\mathrm{EQ}_m^{\geqslant}$.

Since the randomized one-way communication complexity of $\mathrm{EQ}_m^{\geqslant}$ is $\Omega(\log\log m)$ we finally obtain $s(\mathcal{P}_n) = \Omega(\log\log m) = \Omega(\log\log(n/\gamma(n)))$. ■

5.4.2 Regular languages that are not finite unions of suffix-free and trivial languages

In this section, we show the lower bounds from Theorem 5.6 and Theorem 5.7. We start with the following observation.

LEMMA 5.25. *Every suffix-free language excludes a factor.*

PROOF. Let $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ be an rDFA for L. Since L is suffix-free, we can assume that there exists a unique sink state $q_{fail} \notin F$, i.e., $\delta(a, q_{fail}) = q_{fail}$ for all $a \in \Sigma$, which is reachable from all states. We construct a word $w_f \in \Sigma^*$ such that $\delta(p, w_f) = q_{fail}$ for all $p \in Q$. Let $p_1, \dots, p_m$ be an enumeration of all states in $Q \setminus \{q_{fail}\}$. We then construct inductively words $w_0, w_1, \dots, w_m \in \Sigma^*$ such that for all $0 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant i$: $\delta(w_i, p_j) = q_{fail}$. We start with $w_0 = \varepsilon$. Assume that w_i has been constructed for some $i < m$. There is a word x such that that $\delta(x, \delta(w_i, p_{i+1})) = q_{fail}$. We set $w_{i+1} = xw_i$. Then, $\delta(w_{i+1}, p_{i+1}) = \delta(xw_i, p_{i+1}) = q_{fail}$ and $\delta(w_{i+1}, p_j) = \delta(xw_i, p_j) = \delta(x, q_{fail}) = q_{fail}$ for $1 \leqslant j \leqslant i$. Finally, we define $w_f = w_m$. ■

LEMMA 5.26. *Every regular language L satisfies one of the following properties:*[10]

- $L \in \bigcup(\mathbf{Triv}, \mathbf{SF})$
- *L has a restriction $L|_N$ which excludes some factor and contains y^*z for some $y, z \in \Sigma^*$, $|y| > 0$.*

PROOF. Let $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ be an rDFA for L. Let $\mathcal{B}_r = (Q, \Sigma, F_r, \delta, q_0)$ where F_r is the set of nontransient final states and $\mathcal{B}_q = (Q, \Sigma, \{q\}, \delta, q_0)$ for $q \in Q$. We can decompose L as a union of $L_r = \mathsf{L}(\mathcal{B}_r)$ and all languages $\mathsf{L}(\mathcal{B}_q)$ over all transient states $q \in F$. Notice that $\mathsf{L}(\mathcal{B}_q)$ is suffix-free for all transient $q \in F$ since any run to q cannot be prolonged to another run to q. If L_r is trivial, then L satisfies the first property. If L_r is nontrivial, then by Lemma 5.21 and Lemma 5.22 there exists an arithmetic progression $N = a + b\mathbb{N}$ such that $L_r|_N$ is infinite and excludes some word $w \in \Sigma^*$ as a factor. Let $z \in L_r|_N$ be any word. Since some nontransient final state p is reached in

10 It is not hard to see that the two properties exclude each other, but this is not needed for our further consideration.

$\mathcal{B}_r$ on input z, there exists a word y which leads from p back to p. We can ensure that $|y|$ is a multiple of b by replacing y by y^b. Then, $y^*z \subseteq L_r|_N \subseteq L|_N$. Furthermore, since each language $\mathsf{L}(\mathcal{B}_q)$ excludes some factor w_q by Lemma 5.25, the language $L|_N \subseteq L_r|_N \cup \bigcup_q \mathsf{L}(\mathcal{B}_q)$ excludes any concatenation of w and all words w_q as a factor. ■

PROOF OF THEOREM 5.6. Let $L \in \mathbf{Reg} \setminus \bigcup(\mathbf{Triv}, \mathbf{SF})$. By Lemma 5.26, L has a restriction $L|_N$ which excludes some factor w_f and contains y^*z for some $y, z \in \Sigma^*$, $|y| > 0$. Let $c = |w_f| \geqslant 1$. We choose $0 < \epsilon < 1/c$. Let $d = |z|$ and $e = |y|$. Fix a window size $n \in N$ and define for $k \geqslant 0$ the input stream $v_k = w_f^n y^k z$.

We show the lower bound of $\log n - O(1)$ for co-nondeterministic sliding window testers. Consider a co-nondeterministic sliding window tester $\mathcal{P}_n$ for L and window size n with Hamming gap ϵn. Let $\alpha = c\epsilon < 1$ and $r = \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$. If $0 \leqslant k \leqslant r$, then the suffix of v_k of length n contains at least

$$\left\lfloor \frac{n-d-ek}{c} \right\rfloor \geqslant \left\lfloor \frac{n-d-(1-\alpha)n+c+d}{c} \right\rfloor = \left\lfloor \frac{\alpha n + c}{c} \right\rfloor = \lfloor \epsilon n + 1 \rfloor > \epsilon n$$

many disjoint occurrences of w_f. Hence, $\mathcal{P}_n$ must reject the input stream v_k for $0 \leqslant k \leqslant r$, i.e., there is a run of $\mathcal{P}_n$ on v_k that starts in an initial state and ends in a nonfinal state. Consider such a run π for v_r. For $0 \leqslant i \leqslant r$ let p_i be the state in π that is reached after the prefix $w_f^n y^i$ of v_r. Let now m be the number of states of $\mathcal{P}_n$ and assume $m \leqslant r$. There must exist numbers i and j with $0 \leqslant i < j \leqslant r$ such that $p_i = p_j =: p$. It follows that there is a $\mathcal{P}_n$-run on y^{j-i} that starts and ends in state p. Using that cycle we can now prolong the run π, i.e., for all $t \geqslant 0$ there is a run of $\mathcal{P}_n$ on $v_{r+(j-i)\cdot t} = w_f^n y^{r+(j-i)\cdot t} z$ that starts in an initial state and ends in a nonfinal state.

Assume now that the window size satisfies $n \geqslant d$ and $n \equiv d \pmod e$. Write $n = d + le$ for some $l \geqslant 0$. Each n with this property satisfies $n \in N$ since the word $y^l z$ belongs to $L|_N$. We have $l = \frac{n-d}{e} > \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor = r$. For every $k \geqslant l$, the suffix of $v_k = w_f^n y^k z$ of length n is $y^l z \in L$. Therefore, $\mathcal{P}_n$ accepts v_k, i.e., for all $k \geqslant l$, every run of $\mathcal{P}_n$ on v_k that starts in an initial state has to end in a final state. This contradicts our observation that for all $t \geqslant 0$ there is a run of $\mathcal{P}_n$ on $v_{r+(j-i)\cdot t}$ that goes from an initial state to a nonfinal state. We conclude that $\mathcal{P}_n$ has at least $r + 1 \geqslant \frac{(1-\alpha)n-c-d}{e}$ states. It follows that

$$s(\mathcal{P}_n) \geqslant \log\left(\frac{(1-\alpha)n-c-d}{e}\right) \geqslant \log n - O(1).$$

This proves the theorem. ■

Finally, we prove Theorem 5.7.

PROOF OF LEMMA 5.7. Let L be a language in $\mathbf{Reg} \setminus \bigcup(\mathbf{Triv}, \mathbf{SF})$. By Lemma 5.26, L has a restriction $L|_N$ which excludes some factor w_f and contains y^*z for some $y, z \in \Sigma^*$, $|y| > 0$. Let $b = |y|$ and $c = |z|$. Note in the following that b, c, and $|w_f|$ are constants. Choose a window size $n \geqslant c$ such that $n - c$ is a multiple of b. Since $y^*z \subseteq L|_N$, we have $n \in N$. Define the word $v = uw_f^k$ where $k = \lfloor \gamma(n) \rfloor + 1 > \gamma(n)$ and u is any word of length at most $b-1$ such that $|v|$ is a multiple of b. Let $l \in \mathbb{N}$ be such that $|v| = b \cdot l$. Since b divides $n - c$ we can write $n - c = (m \cdot l + r) \cdot b$ for $m \in \mathbb{N}$ and $0 \leqslant r \leqslant l - 1$. Choose the constant ϵ from the theorem statement such that $0 < \epsilon < \frac{1}{|w_f|}$ and hence $\gamma(n) \leqslant \epsilon n$. Assuming n is large enough, we obtain $k = \lfloor \gamma(n) \rfloor + 1 \leqslant \frac{1}{|w_f|}(n - b - c)$, i.e., $b \cdot l = |v| \leqslant b + k \cdot |w_f| \leqslant n - c$ and hence $m \geqslant 1$. Moreover, $l = |v|/b = \Theta(\gamma(n))$ (b and $|w_f|$ are constants) and therefore

$$m = \frac{n-c}{b \cdot l} - \frac{r}{l} = \Theta(n/\gamma(n)).$$

Consider now a randomized sliding window tester $\mathcal{P}_n$ with two-sided error for L and window size n with Hamming gap $\gamma(n)$. We show that from $\mathcal{P}_n$ we can obtain a randomized one-way protocol for GT_m (the greater-than-function on the interval $\{1, \dots, m\}$). Recall that $m \geqslant 1$.

Alice produces from her input $i \in \{1, \dots, m\}$ the word $vy^{r+(m-i)l}$. She then runs $\mathcal{P}_n$ on this word and sends the memory state to Bob. Bob continues the run of the randomized sliding window tester, starting from the transferred memory state, with the input stream $y^{jl}z$, where $j \in \{1, \dots, m\}$ is his input. He obtains the memory state reached after the input $vy^{r+(m-i+j)l}z$. Finally, Bob outputs the negated answer given by the randomized sliding window tester. If $i \leqslant j$, then $\mathsf{last}_n(vy^{r+(m-i+j)l}z) = y^{m \cdot l + r}z \in L$ and Bob rejects with high probability. On the other hand, if $i > j$, then $\mathsf{last}_n(vy^{r+(m-i+j)l}z)$ contains v (recall that $|v| = b \cdot l$ and hence $|y^{r+(m-i+j)l}z| \leqslant n - |v|$). Since v contains strictly more than $\gamma(n)$ disjoint occurrences of w_f (an excluded factor of $L|_N$), we have $\mathrm{dist}(\mathsf{last}_n(w), L) > \gamma(n)$ (here it is important that $n \in N$). Thus, Bob accepts with high probability. We therefore have a correct protocol for GT_m.

Since the randomized one-way communication complexity of GT_m is $\Omega(\log m)$ (Theorem 4.11) we finally obtain $s(\mathcal{P}_n) = \Omega(\log m) = \Omega(\log(n/\gamma(n)))$. ■

For example, if $\gamma(n) \leqslant n^c$ for some $0 < c < 1$ then the lower bound in Theorem 5.7 is $\Omega(\log n)$. Note that if $L \in \bigcup(\mathbf{Triv}, \mathbf{SF})$ then the lower bound of Theorem 5.7 does not hold anymore; this follows from Theorem 5.4.

Note the similarity between the proofs of Theorem 5.7 and Theorem 5.8. The difference is that the word x in the proof of Theorem 5.8 is not present in the proof of Theorem 5.7. The possibly non-empty x in the proof of Theorem 5.8 only allows a reduction from $\mathrm{EQ}_m^{\geqslant}$, whereas the empty x in the proof of Theorem 5.7 allows a reduction from GT_m, which yields a larger lower bound.

6. Conclusion and future work

In this paper we precisely determined the space complexity of regular languages in the sliding window model in the following settings: deterministic, randomized, deterministic property testing, and randomized property testing. Two important restrictions that made our results possible but that also limit their applicability are the following:

- Our sliding window algorithms only answer Boolean queries (does the window content belong to a language or not?). In many applications one wants to compute a certain non-Boolean value, e.g. the number of 1's in the window. This leads to the question whether our automata theoretic framework for sliding window problems can be extended to non-Boolean queries. Weighted automata [29] or cost register automata [4] could be a suitable framework for such an endeavor. Another interesting problem in this context is to maintain the distance (e.g. the Hamming distance or edit distance) between the sliding window and a fixed language L.
- The incoming data values in our model are from a fixed finite alphabet. In many practical situations the incoming data values are from an infinite domain (at least on an abstract level) like the natural numbers or real numbers. Again, the question arises, whether our automata theoretic approach can be extended to such a setting. A popular automata model for words over an infinite alphabet (which are known as data words in this context) are *register automata*, which are also known as finite memory automata [65, 77]. In the context of sliding window streaming, deterministic register automata (DRA for short) [37] might be a good starting point. Benedikt, Ley, and Puppis [14] proved a Myhill-Nerode-like theorem that characterizes the class of data languages recognized by DRA for the case that the underlying relational structure $\mathcal{A}$ on the data values is either $(D, =)$ (where $=$ denotes the equality relation) or $(D, <)$ for a strict linear order $<$. As a byproduct of this characterization, they obtain a minimal DRA for any DRA-recognizable language. This DRA is minimal in a very strong sense: at the same time it has the minimal number of states and the minimal number of registers among all equivalent DRA. Using these minimal DRA, one can define space-optimal streaming algorithms for data languages analogously to the case of words over a finite alphabet. This yields a starting point for studying space complexity classes for streaming algorithms over infinite domains.

Acknowledgement.

We thank the two referees for their helpful comments. The first and third author were partly supported by the DFG project LO 748/13-1/2 (Streaming Automata Theory).

References

[1] **Charu C. Aggarwal**, editor. Data Streams - Models and Algorithms, volume 31 of *Advances in Database Systems*. Springer, 2007. DOI (2)

[2] **Noga Alon**, **Michael Krivelevich**, **Ilan Newman**, and **Mario Szegedy**. Regular languages are testable with a constant number of queries. *SIAM J. Comput.* 30(6):1842–1862, 2000. DOI (6, 29, 48, 50)

[3] **Noga Alon**, **Yossi Matias**, and **Mario Szegedy**. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.* 58(1):137–147, 1999. DOI (5, 31)

[4] **Rajeev Alur**, **Loris D'Antoni**, **Jyotirmoy V. Deshmukh**, **Mukund Raghothaman**, and **Yifei Yuan**. Regular functions and cost register automata. *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013*, pages 13–22. IEEE Computer Society, 2013. DOI (71)

[5] **Rajeev Alur** and **P. Madhusudan**. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009. DOI (8)

[6] **Antoine Amarilli**, **Louis Jachiet**, and **Charles Paperman**. Dynamic membership for regular languages. *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, 116:1–116:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. DOI (10)

[7] **Arvind Arasu** and **Gurmeet Singh Manku**. Approximate counts and quantiles over sliding windows. *Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2004*, pages 286–296. ACM, 2004. DOI (2, 9)

[8] **Brian Babcock**, **Mayur Datar**, and **Rajeev Motwani**. Sampling from a moving window over streaming data. *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002*, pages 633–634. ACM/SIAM, 2002. URL (9)

[9] **Brian Babcock**, **Mayur Datar**, **Rajeev Motwani**, and **Liadan O'Callaghan**. Maintaining variance and k-medians over data stream windows. *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2003*, pages 234–243. ACM, 2003. DOI (9)

[10] **Ajesh Babu**, **Nutan Limaye**, **Jaikumar Radhakrishnan**, and **Girish Varma**. Streaming algorithms for language recognition problems. *Theor. Comput. Sci.* 494:13–23, 2013. DOI (9)

[11] **Corentin Barloy**, **Filip Murlak**, and **Charles Paperman**. Stackless processing of streamed trees. *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS'21*, pages 109–125. ACM, 2021. DOI (9)

[12] **Ran Ben Basat**, **Gil Einziger**, and **Roy Friedman**. Give me some slack: efficient network measurements. *Theor. Comput. Sci.* 791:87–108, 2019. DOI (9)

[13] **Ran Ben-Basat**, **Gil Einziger**, **Roy Friedman**, and **Yaron Kassner**. Efficient summing over sliding windows. *Proceedings of the 15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016*, volume 53 of *LIPIcs*, 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. DOI (8, 9)

[14] **Michael Benedikt**, **Clemens Ley**, and **Gabriele Puppis**. What you must remember when processing data words. *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. (71)

[15] **Jean Berstel**. Transductions and context-free languages, volume 38 of *Teubner Studienbücher : Informatik*. Teubner, 1979. URL (10, 27)

[16] **Vladimir Braverman**, **Elena Grigorescu**, **Harry Lang**, **David P. Woodruff**, and **Samson Zhou**. Nearly optimal distinct elements and heavy hitters on sliding windows. *Proceedings of the 21st International Conference on Approximation Algorithms for Combinatorial Optimization Problems, and the 22nd International Conference on Randomization and Computation, APPROX/RANDOM 2018*, volume 116 of *LIPIcs*, 7:1–7:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. DOI (9)

[17] **Vladimir Braverman** and **Rafail Ostrovsky**. Smooth histograms for sliding windows. *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2007*, pages 283–293. IEEE Computer Society, 2007. DOI (9)

[18] **Vladimir Braverman**, **Rafail Ostrovsky**, and **Carlo Zaniolo**. Optimal sampling from sliding windows. *J. Comput. Syst. Sci.* 78(1):260–272, 2012. DOI (9, 48)

[19] **Dany Breslauer** and **Zvi Galil**. Real-time streaming string-matching. *ACM Trans. Algorithms*, 10(4):22:1–22:12, 2014. DOI (9)

[20] **Nicolaas G. de Bruijn**. A combinatorial problem. English. *Nederl. Akad. Wetensch. Proc.* 49(7):758–764, 1946. (13)

[21] **Raphaël Clifford**, **Allyx Fontaine**, **Ely Porat**, **Benjamin Sach**, and **Tatiana Starikovskaya**. Dictionary matching in a stream. *Proceedings of the 23rd Annual European Symposium, ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2015. DOI (9)

[22] **Raphaël Clifford**, **Allyx Fontaine**, **Ely Porat**, **Benjamin Sach**, and **Tatiana Starikovskaya**. The *k*-mismatch problem revisited. *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 2039–2052. SIAM, 2016. DOI (9)

[23] **Raphaël Clifford, Tomasz Kociumaka, and Ely Porat**. The streaming k-mismatch problem. *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1106–1125. SIAM, 2019. DOI (9)

[24] **Raphaël Clifford and Tatiana Starikovskaya**. Approximate Hamming distance in a stream. *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *LIPIcs*, 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. DOI (9)

[25] **Edith Cohen and Martin J. Strauss**. Maintaining time-decaying stream aggregates. *J. Algorithms*, 59(1):19–36, 2006. DOI (9)

[26] **Gianpaolo Cugola and Alessandro Margara**. Processing flows of information: from data stream to complex event processing. *ACM Computing Surveys*, 44(3), 2012. DOI (2)

[27] **Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani**. Maintaining stream statistics over sliding windows. *SIAM J. Comput.* 31(6):1794–1813, 2002. DOI (2, 8, 9, 31)

[28] **Mayur Datar and Shan Muthukrishnan**. Estimating rarity and similarity over data stream windows. *Proceedings of the 10th European Symposium on Algorithms, ESA 2002*, volume 2461 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 2002. DOI (9)

[29] **Manfred Droste, Werner Kuich, and Heiko Vogler**. Handbook of Weighted Automata. Springer, 2009. (71)

[30] **Samuel Eilenberg**. Automata, languages, and machines, volume A of *Pure and applied mathematics*. Academic Press, 1974. (11)

[31] **Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou**. Periodicity in data streams with wildcards. *Proceedings of the 13th International Computer Science Symposium in Russia, CSR 2018*, volume 10846 of *Lecture Notes in Computer Science*, pages 90–105. Springer, 2018. DOI (9)

[32] **Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou**. Streaming periodicity with mismatches. *Proceedings of Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017*, volume 81 of *LIPIcs*, 42:1–42:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. DOI (9)

[33] **Funda Ergün, Hossein Jowhari, and Mert Saglam**. Periodicity in streams. *Proceeding of the 14th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX-RANDOM 2010*, volume 6302 of *Lecture Notes in Computer Science*, pages 545–559. Springer, 2010. DOI (9)

[34] **Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan**. Testing and spot-checking of data streams. *Algorithmica*, 34(1):67–80, 2002. DOI (6)

[35] **Joan Feigenbaum, Sampath Kannan, and Jian Zhang**. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2005. DOI (9)

[36] **Philippe Flajolet**. Approximate counting: A detailed analysis. *BIT*, 25(1):113–134, 1985. (35)

[37] **Nissim Francez and Michael Kaminski**. An algebraic characterization of deterministic regular languages over infinite alphabets. *Theor. Comput. Sci.* 306(1-3):155–175, 2003. (71)

[38] **Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre**. Streaming property testing of visibly pushdown languages. *Proceedings of the 24th Annual European Symposium on Algorithms, ESA 2016*, volume 57 of *LIPIcs*, 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. DOI (6, 9)

[39] **Gudmund Skovbjerg Frandsen, Thore Husfeldt, Peter Bro Miltersen, Theis Rauhe, and Søren Skyum**. Dynamic algorithms for the Dyck languages. *Proceedings of the 4th International Workshop on Algorithms and Data Structures, WADS '95*, volume 955 of *Lecture Notes in Computer Science*, pages 98–108. Springer, 1995. DOI (10)

[40] **Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum**. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997. DOI (10)

[41] **Moses Ganardi**. Language recognition in the sliding window model. PhD thesis, University of Siegen, Germany, 2019. URL (7)

[42] **Moses Ganardi**. Visibly pushdown languages over sliding windows. *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPIcs*, 29:1–29:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. DOI (8)

[43] **Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras**. Automata theory on sliding windows. *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, volume 96 of *LIPIcs*, 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. DOI (6, 7)

[44] **Moses Ganardi, Danny Hucke, and Markus Lohrey**. Derandomization for sliding window algorithms with strict correctness*. *Theory Comput. Syst.* 65(3):1–18, 2021. DOI (8)

[45] **Moses Ganardi, Danny Hucke, and Markus Lohrey**. Querying regular languages over sliding windows. *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, volume 65 of *LIPIcs*, 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. DOI (6)

[46] **Moses Ganardi, Danny Hucke, and Markus Lohrey**. Randomized sliding window algorithms for regular languages. *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, 127:1–127:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. DOI (6)

[47] **Moses Ganardi**, **Danny Hucke**, **Markus Lohrey**, and **Tatiana Starikovskaya**. Sliding window property testing for regular languages. *Proceedings of the 30th International Symposium on Algorithms and Computation, ISAAC 2019*, volume 149 of *LIPIcs*, 6:1–6:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. DOI (6)

[48] **Moses Ganardi**, **Louis Jachiet**, **Markus Lohrey**, and **Thomas Schwentick**. Low-latency sliding window algorithms for formal languages. *Proceedings of the 42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022*, volume 250 of *LIPIcs*, 38:1–38:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. DOI (9)

[49] **Moses Ganardi**, **Artur Jeż**, and **Markus Lohrey**. Sliding windows over context-free languages. *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018*, volume 117 of *LIPIcs*, 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. DOI (8)

[50] **Pawel Gawrychowski**. Chrobak normal form revisited, with applications. *Proceedings of the 16th International Conference on Implementation and Application of Automata, CIAA 2011*, volume 6807 of *Lecture Notes in Computer Science*, pages 142–153. Springer, 2011. DOI (45)

[51] **Pawel Gawrychowski and Artur Jeż**. Hyper-minimisation made efficient. *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science 2009, MFCS 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 356–368. Springer, 2009. DOI (26, 27)

[52] **Paweł Gawrychowski**, **Oleg Merkurev**, **Arseny M. Shur**, and **Przemyslaw Uznański**. Tight tradeoffs for real-time approximation of longest palindromes in streams. *Algorithmica*, 81(9):3630–3654, 2019. DOI (9)

[53] **Paweł Gawrychowski**, **Jakub Radoszewski**, and **Tatiana Starikovskaya**. Quasi-periodicity in streams. *Proceedings of the 30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, volume 128 of *LIPIcs*, 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. DOI (9)

[54] **Pawel Gawrychowski and Tatiana Starikovskaya**. Streaming dictionary matching with mismatches. *Algorithmica*, 84(4):896–916, 2022. DOI (9)

[55] **Phillip B. Gibbons and Srikanta Tirthapura**. Distributed streams algorithms for sliding windows. *Theory Comput. Syst.* 37(3):457–478, 2004. DOI (9)

[56] **Lukasz Golab**, **David DeHaan**, **Erik D. Demaine**, **Alejandro López-Ortiz**, and **J. Ian Munro**. Identifying frequent items in sliding windows over on-line packet streams. *Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference, IMC 2003*, pages 173–178. ACM, 2003. DOI (9)

[57] **Shay Golan**, **Tomasz Kociumaka**, **Tsvi Kopelowitz**, and **Ely Porat**. The streaming k-mismatch problem: tradeoffs between space and total time. *Proceedings of the 31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, volume 161 of *LIPIcs*, 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. DOI (9)

[58] **Shay Golan**, **Tsvi Kopelowitz**, and **Ely Porat**. Streaming pattern matching with d wildcards. *Algorithmica*, 81(5):1988–2015, 2019. DOI (9)

[59] **Shay Golan**, **Tsvi Kopelowitz**, and **Ely Porat**. Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams. *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, 65:1–65:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. DOI (9)

[60] **Shay Golan and Ely Porat**. Real-time streaming multi-pattern search for constant alphabet. *Proceedings of the 25th Annual European Symposium on Algorithms, ESA 2017*, volume 87 of *LIPIcs*, 41:1–41:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. DOI (9)

[61] **Oded Goldreich**, **Shafi Goldwasser**, and **Dana Ron**. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. DOI (6)

[62] **John E. Hopcroft and Jeffrey D. Ullman**. Introduction to Automata Theory, Languages and Computation. Addison-Wesley, 1979. (10)

[63] **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, October 2014. DOI (9)

[64] **Galina Jirásková and Peter Mlynárcik**. Complement on prefix-free, suffix-free, and non-returning NFA languages. *Proceedings of the 16th International Workshop on Descriptional Complexity of Formal Systems, DCFS 2014*, volume 8614 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2014. DOI (45)

[65] **Michael Kaminski and Nissim Francez**. Finite-memory automata. *Theor. Comput. Sci.* 134(2):329–363, 1994. DOI (71)

[66] **Tomasz Kociumaka**, **Ely Porat**, and **Tatiana Starikovskaya**. Small-space and streaming pattern matching with k edits. *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 885–896. IEEE, 2021. DOI (9)

[67] **Christian Konrad and Frédéric Magniez**. Validating XML documents in the streaming model with external memory. *ACM Trans. Database Syst.* 38(4):27:1–27:36, 2013. DOI (9)

[68] **Dexter Kozen**. Automata and computability. Undergraduate texts in computer science. Springer, 1997. (10)

[69] **Andreas Krebs**, **Nutan Limaye**, and **Srikanth Srinivasan**. Streaming algorithms for recognizing nearly well-parenthesized expressions. *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science, MFCS 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 412–423. Springer, 2011. DOI (9)

[70] **Ilan Kremer**, **Noam Nisan**, and **Dana Ron**. On randomized one-round communication complexity. *Comput. Complex.* 8(1):21–49, 1999. DOI (40)

[71] **Eyal Kushilevitz** and **Noam Nisan**. Communication complexity. Cambridge University Press, 1997. (40, 41)

[72] **Frédéric Magniez**, **Claire Mathieu**, and **Ashwin Nayak**. Recognizing well-parenthesized expressions in the streaming model. *SIAM J. Comput.* 43(6):1880–1905, 2014. DOI (9)

[73] **Oleg Merkurev** and **Arseny M. Shur**. Searching long repeats in streams. *Proceedings of the 30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, volume 128 of *LIPIcs*, 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (9)

[74] **Oleg Merkurev** and **Arseny M. Shur**. Searching runs in streams. *Proceedings of the 26th International Symposium on String Processing and Information Retrieval, SPIRE 2019*, volume 11811 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2019. DOI (9)

[75] **Michael Mitzenmacher** and **Eli Upfal**. Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis. Cambridge University Press, New York, NY, USA, 2nd edition, 2017. (32)

[76] **Robert H. Morris Sr.** Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 1978. DOI (35)

[77] **Frank Neven**, **Thomas Schwentick**, and **Victor Vianu**. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.* 5(3):403–435, 2004. DOI (71)

[78] **Azaria Paz**. Introduction to Probabilistic Automata (Computer Science and Applied Mathematics). Academic Press, Inc., Orlando, FL, USA, 1971. (31)

[79] **Jean-Eric Pin**. Syntactic semigroups. **Grzegorz Rozenberg** and **Arto Salomaa**, editors, *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*, pages 679–746. Springer, 1997. DOI (4)

[80] **Benny Porat** and **Ely Porat**. Exact and approximate pattern matching in the streaming model. *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009*, pages 315–323. IEEE Computer Society, 2009. DOI (9)

[81] **Michael O. Rabin**. Probabilistic automata. *Inform. Control*, 6(3):230–245, 1963. DOI (31, 34)

[82] **Jakub Radoszewski** and **Tatiana Starikovskaya**. Streaming k-mismatch with error correcting and applications. *Information and Computation*, 271:104513, 2020. DOI (9)

[83] **J. Barkley Rosser** and **Lowell Schoenfeld**. Approximate formulas for some functions of prime numbers. *Illinois J. Math.* 6(1):64–94, 1962. (39)

[84] **Tim Roughgarden**. Communication complexity (for algorithm designers). *Found. Trends Theor. Comput. Sci.* 11(3-4):217–404, 2016. DOI (40, 44, 47)

[85] **Luc Segoufin** and **Victor Vianu**. Validating streaming XML documents. *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2002*, pages 53–64. ACM, 2002. DOI (9)

[86] **Tatiana Starikovskaya**. Communication and streaming complexity of approximate pattern matching. *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, 13:1–13:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. DOI (9)

[87] **Howard Straubing**. Finite semigroup varieties of the form $V * D$. *J. Pure Appl. Algebra*, 36:53–94, 1985. DOI (26)

[88] **Kanat Tangwongsan**, **Martin Hirzel**, and **Scott Schneider**. Low-latency sliding-window aggregation in worst-case constant time. *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017*, pages 66–77. ACM, 2017. DOI (9)

[89] **Haopeng Zhang**, **Yanlei Diao**, and **Neil Immerman**. On complexity and optimization of expensive queries in complex event processing. *Proceedings of the International Conference on Management of Data, SIGMOD 2014*, pages 217–228. ACM, 2014. DOI (2)