Automata theory on sliding windows*

Moses Ganardi[1], Danny Hucke[1], Daniel König[1], Markus Lohrey[1], and Konstantinos Mamouras[2]

1 Universität Siegen, Germany
{ganardi,hucke,koenig,lohrey}@eti.uni-siegen.de
2 University of Pennsylvania, Philadelphia, USA
mamouras@seas.upenn.edu

Abstract

In a recent paper we analyzed the space complexity of streaming algorithms whose goal is to decide membership of a sliding window to a fixed language. For the class of regular languages we proved a space trichotomy theorem: for every regular language the optimal space bound is either constant, logarithmic or linear. In this paper we continue this line of research: We present natural characterizations for the constant and logarithmic space classes and establish tight relationships to the concept of language growth. We also analyze the space complexity with respect to automata size and prove almost matching lower and upper bounds. Finally, we consider the decision problem whether a language given by a DFA/NFA admits a sliding window algorithm using logarithmic/constant space.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages

Keywords and phrases regular languages, sliding window algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.32

1 Introduction

Streaming algorithms process an input sequence $a_1 a_2 \cdots a_m$ from left to right and have at time t only direct access to the current data value a_t. Such algorithms have received a lot of attention in recent years; see [1] for an introduction. The general goal of streaming algorithms is to avoid the explicit storage of the whole data stream. Ideally, a streaming algorithm works in constant space, in which case it reduces to a deterministic finite automaton (DFA), but polylogarithmic space with respect to the input length might be acceptable, too. These small space requirements are motivated by the current explosion in the size of the input data, which makes random access to the input often infeasible. Such a scenario arises for instance when searching in large databases (e.g., genome databases or web databases), analyzing internet traffic (e.g. click stream analysis), and monitoring networks.

The first papers on streaming algorithms are usually attributed to Munro and Paterson [28] and Flajolet and Martin [17], although the principle idea goes back to the work on online machines by Hartmanis, Lewis and Stearns from the 1960's [27, 31]. Extremely influential for the area of streaming algorithms was the paper of Alon, Matias, and Szegedy [2].

The sliding window model. Two streaming models can be found in the literature: In the *standard model* the algorithm reads a data stream $a_1 \cdots a_m$ from left to right. At time instant t it outputs a value $f(a_1 \cdots a_t)$ for a certain function f. In contrast, in the *sliding window*

* A full version of the paper with all proofs can be found in [19].

© Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey and Konstantinos Mamouras;
licensed under Creative Commons License CC-BY
35th Symposium on Theoretical Aspects of Computer Science (STACS 2018).
Editors: Rolf Niedermeier and Brigitte Vallée; Article No. 32; pp. 32:1–32:14

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

model the algorithm works on a sliding window. At time instant t, the active window is a certain suffix $a_{t-n+1} \cdots a_t$ of $a_1 \cdots a_t$ and the algorithm outputs $f(a_{t-n+1} \cdots a_t)$.

The sliding window model is the right approach for streaming applications, where data items are outdated after a certain time. A typical example is the analysis of a time series as it may arise in medical monitoring, web tracking, or financial monitoring. In all these applications, data items are usually no longer important after a certain time. Two variants of the sliding window model can be found in the literature; see e.g. [3]:

- *Fixed-size model:* The size of the active window is a fixed constant (the window size). In other words: at each time instant a new data value a_i arrives and the oldest data value from the sliding window expires.
- *Variable-size model:* The active window $a_{t-n+1} \cdots a_t$ is determined by an adversary. At every time instant the adversary can either remove the first data value from the window (expiration of a value) or add a new data value at the right end (arrival of a new value).

In the seminal paper of Datar et al. [15], where the fixed-size sliding window model was introduced, the authors show how to maintain the number of 1's in a sliding window of size n over the alphabet $\{0, 1\}$ in space $\frac{1}{\varepsilon} \cdot \log^2 n$ if one allows a multiplicative error of $1 \pm \varepsilon$. This is has been the starting point for a large number of further papers on the approximation of statistical data over sliding windows. Let us mention the work on computation of the variance and k-median [4], quantiles [3], and entropy [8] over sliding windows. Other computational problems that have been considered for the sliding window model include optimal sampling [9], various pattern matching problems [10, 11, 12, 13], database querying (e.g. processing of join queries [22]) and graph problems (e.g. checking for connectivity and computation of matchings, spanners, and spanning trees [14]). Further references on the sliding window model can be found in [1, Chapter 8] and [7].

Language recognition in the streaming model. A natural problem that has been surprisingly neglected for the streaming model is language recognition. The goal is to check whether an input string belongs to a given language L. Let us quote Magniez, Mathieu, and Nayak [26]: "Few applications [of streaming] have been made in the context of formal languages, which may have impact on massive data such as DNA sequences and large XML files. For instance, in the context of databases, properties decidable by streaming algorithm have been studied [30, 29], but only in the restricted case of deterministic and constant memory space algorithms." For Magniez et al. this was the starting point to study language recognition in the streaming model. Thereby they restricted their attention to the above mentioned standard streaming model. Note that in the standard model the membership problem for a regular language is trivial to solve: one simply simulates a DFA on the stream and thereby only store the current state. In [26] the authors present a randomized streaming algorithm for the (non-regular) Dyck language D_s with s pairs of parenthesis that works in space $\mathcal{O}(\sqrt{n} \log n)$ and time polylog(n) per symbol. Further investigations on streaming language recognition for various subclasses of context-free languages can be found in [5, 6, 18, 23, 24, 29, 30]. Let us emphasize that all these papers exclusively deal with the standard streaming model. Language recognition problems for the sliding window model have been completely neglected so far. This was the starting point for our previous paper [20].

Querying regular languages in the sliding window model. As mentioned above, in the standard streaming model the membership problem for a regular language can be solved in constant space by simulating a DFA. This solution does not work for the sliding window model. The problem is the removal of the left-most symbol from the sliding window. In

order to check whether the active window belongs to a certain language L one has to know this first symbol in general. In such a case one has to store the whole window content using $\mathcal{O}(n)$ bits (where n is the window size). A simple regular language where this phenomenon arises is the language $a\{a,b\}^*$ of all words that start with a. The point is that by repeatedly checking whether the sliding window content belongs to $a\{a,b\}^*$, one can recover the exact content of the sliding window, which implies that every sliding window algorithm for testing membership in $a\{a,b\}^*$ has to use n bits of storage (where n is the window size).

For a function $s(n)$ let $\mathsf{F}_{\mathsf{reg}}(s(n))$ be the class of all languages L with the following property: For every window size n there exists an algorithm that reads a data stream, uses only space $s(n)$ and correctly decides at every time instant whether the active window (the last n symbols from the stream) belongs to L. Note that this is a *non-uniform* model: for every window size n we use a separate algorithm. The class $\mathsf{V}_{\mathsf{reg}}(s(n))$ of languages that have variable-size sliding window algorithms with space complexity $s(n)$ is defined similarly, see page 6 for details. Our main result from [20] is a space trichotomy for regular languages:

1. $\mathsf{V}_{\mathsf{reg}}(o(n)) = \mathsf{F}_{\mathsf{reg}}(o(n)) = \mathsf{F}_{\mathsf{reg}}(\mathcal{O}(\log n)) = \mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$
2. $\mathsf{F}_{\mathsf{reg}}(o(\log n)) = \mathsf{F}_{\mathsf{reg}}(\mathcal{O}(1))$
3. $\mathsf{V}_{\mathsf{reg}}(o(\log n)) = \mathsf{V}_{\mathsf{reg}}(\mathcal{O}(1))$ = all trivial languages (empty and universal languages)

Each of the three cases is characterized in terms of the syntactic homomorphism and the left Cayley graph of the syntactic monoid of the regular language. The precise characterizations are a bit technical; see [20] for details.

In this paper we continue our investigation of sliding-window algorithms for regular languages. As a first contribution, we present very natural characterizations of the above language classes in 1. and 2.: The languages in 1. are exactly the languages that are reducible with a Mealy machine (working from right to left) to a regular language of polynomial growth. The regular languages of polynomial growth are exactly the bounded regular languages [33]. A language L is *bounded* if $L \subseteq w_1^* w_2^* \cdots w_n^*$ for words $w_1, w_2, \ldots, w_n$. In addition, we show that the class 1. is the Boolean closure of regular left ideals (regular languages L with $\Sigma^* L \subseteq L$) and regular length languages (regular languages where $|u| = |v|$ implies that $u \in L$ iff $v \in L$). The class 2. is characterized as the Boolean closure of suffix-testable languages (languages L where membership in L only depends on a suffix of constant length) and regular length languages. A natural example for the classes above is the problem of testing whether the sliding window contains a fixed pattern w as a factor (or as a suffix) since we can check membership of the left ideal $\Sigma^* w \Sigma^*$ (or of the suffix-testable language $\Sigma^* w$).

We also consider the sliding-window space complexity of regular languages in a uniform setting, where the size m (number of states) of an automaton for the regular language is also taken into account. In [20], we asked whether for DFAs of size m that accept languages in $\mathsf{F}_{\mathsf{reg}}(\mathcal{O}(\log n)) = \mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$, there exists a sliding-window streaming algorithm with space complexity $\mathrm{poly}(m) \cdot \log n$. Here, we give a negative answer by proving a lower bound of the form $\Omega(2^m \cdot \log n)$. Moreover, we also show almost matching upper bounds.

Finally, we prove that one can test in nondeterministic logspace (NL) and hence in deterministic polynomial time whether for a given DFA $\mathcal{A}$ the language $L(\mathcal{A})$ belongs to the above class 1. (resp., 2.). For NFAs these problems become PSPACE-complete.

2 Preliminaries

For an alphabet Σ and $n \geq 0$ let $\Sigma^{\leq n} = \{x \in \Sigma^* : |x| \leq n\}$. The set of all *prefixes* of $x \in \Sigma^*$ is $\mathrm{Pref}(x) = \{u \in \Sigma^* \colon \exists v \in \Sigma^* : x = uv\}$ and the *reversal* of $x = a_1 \cdots a_n$ is $x^{\mathsf{R}} = a_n \cdots a_1$. For a language $L \subseteq \Sigma^*$ let $\mathrm{Pref}(L) = \bigcup_{x \in L} \mathrm{Pref}(x)$ and $L^{\mathsf{R}} = \{x^{\mathsf{R}} : x \in L\}$. The *reversal* of

a function $\tau\colon \Sigma^* \to \Gamma^*$ is defined as $\tau^{\mathsf{R}}(x) = \tau(x^{\mathsf{R}})^{\mathsf{R}}$. Thus, $\tau(u) = v$ iff $\tau^{\mathsf{R}}(u^{\mathsf{R}}) = v^{\mathsf{R}}$.

We use $\log x$ as an abbreviation for $\lfloor \log_2 x \rfloor$. Note that if $(w_i)_{i\geq 0}$ is the length-lexicographic enumeration of $\{0,1\}^*$ then $|w_i| \leq \log i$. We use the following well-known bounds for binomial coefficients, where $1 \leq k \leq n$ and e is Euler's constant: $(n/k)^k \leq \binom{n}{k} \leq (e \cdot n/k)^k$.

We use standard definitions from automata theory. A *nondeterministic finite automaton* (NFA) is a tuple $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ where Q is a finite set of states, Σ is an alphabet, $I \subseteq Q$ is the set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $F \subseteq Q$ is the set of final states. A *deterministic finite automaton* (DFA) $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ has a single initial state $q_0 \in Q$ instead of I and a transition function $\delta\colon Q \times \Sigma \to Q$ instead of the transition relation Δ. A *deterministic automaton* has the same format as a DFA, except that the state set Q is not required to be finite. If $\mathcal{A}$ is deterministic, the transition function δ is extended to a function $\delta\colon Q \times \Sigma^* \to Q$ in the usual way and we define $\mathcal{A}(x) = \delta(q_0, x)$ for $x \in \Sigma^*$ and $L(\mathcal{A}) = \{x \in \Sigma^* \colon \mathcal{A}(x) \in F\}$ (the language accepted by $\mathcal{A}$).

Let $L \subseteq \Sigma^*$ be a language. The *left quotient* of $x \in \Sigma^*$ is $x^{-1}L = \{z \in \Sigma^* : xz \in L\}$. The *Myhill-Nerode congruence* $\sim_L$ is the equivalence relation on Σ^* defined by $x \sim_L y$ if and only if $x^{-1}L = y^{-1}L$. It is a *right congruence* on Σ^*, i.e. $x \sim_L y$ implies $xz \sim_L yz$ for all $x, y, z \in \Sigma^*$. If $\mathcal{A}$ is a (not necessarily finite) deterministic automaton for a language $L \subseteq \Sigma^*$, then $\mathcal{A}(x) = \mathcal{A}(y)$ implies $x \sim_L y$. The *minimal deterministic automaton* for L is $\mathcal{A}_L = (\Sigma^*/{\sim_L}, \Sigma, [\varepsilon]_{\sim_L}, \delta, \{[x]_{\sim_L} \colon x \in L\})$ with $\delta([x]_{\sim_L}, a) = [xa]_{\sim_L}$. Clearly, $L(\mathcal{A}_L) = L$.

For an NFA $\mathcal{A}$ we denote with $\mathcal{A}^{\mathsf{D}}$ the corresponding deterministic power set automaton (restricted to those states that are reachable from the initial state) and with $\mathcal{A}^{\mathsf{R}}$ the NFA obtained from $\mathcal{A}$ by reversing all transitions and swapping the set of initial states and the set of final states. Moreover, we define $\mathcal{A}^{\mathsf{RD}} = (\mathcal{A}^{\mathsf{R}})^{\mathsf{D}}$. Thus, $L(\mathcal{A}^{\mathsf{R}}) = L(\mathcal{A}^{\mathsf{RD}}) = L(\mathcal{A})^{\mathsf{R}}$. If an NFA $\mathcal{A}$ has m states, then both $\mathcal{A}^{\mathsf{D}}$ and $\mathcal{A}^{\mathsf{RD}}$ have at most 2^m states.

3 Streaming algorithms

A data stream is just a finite sequence of data values. We make the assumption that these data values are from a finite set Σ. Thus, a data stream is a finite word $w = a_1 \cdots a_m \in \Sigma^*$. A streaming algorithm reads the symbols of a data stream from left to right. At time instant t the algorithm has only 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 certain function $f\colon \Sigma^* \to A$ into some domain A, which means that at time instant t the streaming algorithm outputs the value $f(a_1 \cdots a_t)$. In this paper, we only consider the Boolean case $A = \{0,1\}$; in other words, the streaming algorithm tests membership in a fixed language. Furthermore, we abstract away from the actual computation and only analyze the space requirement. Formally, a *streaming algorithm* for $L \subseteq \Sigma^*$ is a deterministic (possibly infinite) automaton $\mathcal{A} = (S, \Sigma, s_0, \delta, F)$ with $L = L(\mathcal{A})$, where the states are encoded by bit strings. We describe this encoding by an injective function $\mathrm{enc}\colon S \to \{0,1\}^*$. The *space function* $\mathrm{space}(\mathcal{A}, \cdot)\colon \Sigma^* \to \mathbb{N}$ specifies the space used by $\mathcal{A}$ on a certain input: For $w \in \Sigma^*$ let $\mathrm{space}(\mathcal{A}, w) = \max\{|\mathrm{enc}(\mathcal{A}(u))| : u \in \mathrm{Pref}(w)\}$.

In the above streaming model, the output value of the streaming algorithm at time t depends on the whole past $a_1 a_2 \cdots a_t$ of the data stream. However, in many practical applications one is only interested in the "relevant part of the past". Two formalizations of this can be found in the literature:

- Only the suffix of $a_1 a_2 \cdots a_t$ of length n is relevant. Here, n is a fixed constant. This streaming model is called the *fixed-size sliding window model.*
- The relevant suffix of $a_1 a_2 \cdots a_t$ is determined by an adversary. In this model, at every

time instant the adversary can either remove the first symbol from the active window (expiration of a data value), or add a new symbol at the right end (arrival of a new data value). This streaming model is also called the *variable-size sliding window model*.

Fixed-size sliding windows. Given a word $w = a_1 a_2 \cdots a_m \in \Sigma^*$ and a window length $n \geq 0$, we define the *active window* $\mathrm{last}_n(w) = a_{m-n+1} a_{m-n+2} \cdots a_m \in \Sigma^n$, where we set $a_i = a$ for $i \leq 0$. Here $a \in \Sigma$ is an arbitrary symbol, which fills the window initially. A sequence $\mathcal{A} = (\mathcal{A}_n)_{n \geq 0}$ is a *fixed-size sliding window algorithm* for $L \subseteq \Sigma^*$ if each $\mathcal{A}_n$ is a streaming algorithm for $\{w \in \Sigma^* : \mathrm{last}_n(w) \in L\}$. Its *space complexity* is the function $f_{\mathcal{A}} \colon \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ where $f_{\mathcal{A}}(n)$ is the maximum encoding length of a state in $\mathcal{A}_n$.

Note that for every language L and every n the language $\{w \in \Sigma^* : \mathrm{last}_n(w) \in L\}$ is regular, which ensures that $\mathcal{A}_n$ can be chosen to be a DFA and hence $f_{\mathcal{A}}(n) < \infty$ for all $n \geq 0$. The trivial fixed-size sliding window algorithm for L is the sequence $\mathcal{B} = (\mathcal{B}_n)_{n \geq 0}$, where $\mathcal{B}_n$ is the DFA with state set Σ^n and the transition mapping $\delta(au, b) = ub$ for $a, b \in \Sigma$, $u \in \Sigma^{n-1}$. States of $\mathcal{B}_n$ can be encoded with $\mathcal{O}(\log|\Sigma| \cdot n)$ bits. By minimizing each $\mathcal{B}_n$, we obtain an *optimal fixed-size sliding window algorithm* $\mathcal{A}$ for L. Finally, we define $F_L(n) = f_{\mathcal{A}}(n)$. Thus, F_L is the space complexity of an optimal fixed-size sliding window algorithm for L. Notice that F_L is not necessarily monotonic. For instance, take $L = \{au \colon u \in \{a,b\}^*, |u| \text{ odd}\}$. Then, we have $F_L(2n) \in \Theta(n)$ and $F_L(2n+1) \in O(1)$. The above trivial algorithm $\mathcal{B}$ yields $F_L(n) \in \mathcal{O}(n)$ for every language L.

Note that the fixed-size sliding window is a *non-uniform* 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 non-uniform 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 length.

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

$$\mathrm{wnd}(\varepsilon) = \varepsilon \qquad \mathrm{wnd}(u\downarrow) = \varepsilon \text{ if } \mathrm{wnd}(u) = \varepsilon$$
$$\mathrm{wnd}(ua) = \mathrm{wnd}(u)a \qquad \mathrm{wnd}(u\downarrow) = v \text{ if } \mathrm{wnd}(u) = av$$

A *variable-size sliding window algorithm* for a language $L \subseteq \Sigma^*$ is a streaming algorithm $\mathcal{A}$ for $\{w \in \overline{\Sigma}^* : \mathrm{wnd}(w) \in L\}$. Its *space complexity* is the function $v_{\mathcal{A}}$ mapping a window length n to the maximum number of bits used by $\mathcal{A}$ on inputs producing an active window of size at most n. Formally, it is the monotonic function $v_{\mathcal{A}} \colon \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ with $v_{\mathcal{A}}(n) = \max\{\mathrm{space}(\mathcal{A}, u) : u \in \overline{\Sigma}^*, |\mathrm{wnd}(v)| \leq n \text{ for all } v \in \mathrm{Pref}(u)\}$. This definition of $v_{\mathcal{A}}(n)$ slightly deviates from the one given in [20], where the space complexity is defined as $v'_{\mathcal{A}}(n) = \max\{|\mathrm{enc}(\mathcal{A}(u))| : u \in \overline{\Sigma}^*, |\mathrm{wnd}(u)| = n\}$. One easily sees that $v_{\mathcal{A}}(n) = \max_{k \leq n} v'_{\mathcal{A}}(k)$ and hence $v_{\mathcal{A}}(n) = v'_{\mathcal{A}}(n)$ if $v'_{\mathcal{A}}(n)$ is monotonic. An advantage of our definition of $v_{\mathcal{A}}(n)$ is that for every language an optimal variable-size sliding window algorithm exists. We obtain this algorithm from the minimal deterministic automaton for $\{w \in \overline{\Sigma}^* : \mathrm{wnd}(w) \in L\}$.

▶ **Lemma 3.1.** *For every $L \subseteq \Sigma^*$ there exists a variable-size sliding window algorithm $\mathcal{A}$ such that $v_{\mathcal{A}}(n) \leq v_{\mathcal{B}}(n)$ for every variable-size sliding window algorithm $\mathcal{B}$ for L and all n.*

We define $V_L(n) = v_{\mathcal{A}}(n)$, where $\mathcal{A}$ is a space *optimal variable-size sliding window algorithm* for L from Lemma 3.1. Since any algorithm in the variable-size model yields an algorithm in the fixed-size model, we have $F_L(n) \leq V_L(n)$.

Space complexity classes and closure properties. For a function $s\colon \mathbb{N} \to \mathbb{N}$ we define the classes $\mathsf{F}(s)$ and $\mathsf{V}(s)$ of all languages $L \subseteq \Sigma^*$ which have a fixed-size (variable-size, respectively) sliding window algorithm with space complexity bounded by $s(n)$. For a class $\mathcal{C}$ of functions we define $\mathsf{X}(\mathcal{C}) = \bigcup_{s\in\mathcal{C}} \mathsf{X}(s)$ for $\mathsf{X} \in \{\mathsf{F},\mathsf{V}\}$.

Several times we will exploit closure properties of the classes $\mathsf{F}(\mathcal{O}(s))$ and $\mathsf{V}(\mathcal{O}(s))$ (for a function $s(n)$). We need the following definitions: A *Mealy machine* $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta)$ consists of a finite set of states Q, an input alphabet Σ, an output alphabet Γ, an initial state $q_0 \in Q$ and the transition function $\delta\colon Q \times \Sigma \to Q \times \Gamma$. For every $q \in Q$ the machine computes a length-preserving transduction $\tau_q\colon \Sigma^* \to \Gamma^*$ in the usual way: $\tau_q(\varepsilon) = \varepsilon$ and if $\delta(p, a) = (q, b)$ then $\tau_p(au) = b\,\tau_q(u)$. We call $\tau^{\mathsf{R}}_{q_0}$ the *$\leftarrow$-transduction* computed by $\mathcal{M}$. Thus, a $\leftarrow$-transduction is computed by a Mealy machine that works on an input word from right to left. If L is regular and τ is a $\leftarrow$-transduction, then $\tau(L)$ and $\tau^{-1}(L)$ are regular as well. A $\leftarrow$-transduction τ is called a *$\leftarrow$-reduction* from $K \subseteq \Sigma^*$ to $L \subseteq \Gamma^*$ if $K = \tau^{-1}(L)$.

▶ **Lemma 3.2.** *For any function $s(n)$ the classes $\mathsf{F}(\mathcal{O}(s))$ and $\mathsf{V}(\mathcal{O}(s))$ are closed under (i) Boolean operations and (ii) $\leftarrow$-reductions.*

Space trichotomy for regular languages. In [20] we proved a trichotomy theorem on sliding window algorithms for regular languages. We identified a partition of the class of regular languages into three classes which completely characterize the sliding window space complexity in both the fixed-size and the variable-size model. The definition of the three classes is given in terms of the syntactic homomorphism and the left Cayley graph of the syntactic monoid of the regular language, see [20].

For $\mathsf{X} \in \{\mathsf{F},\mathsf{V}\}$ and a class $\mathcal{C}$ of functions we abbreviate $\mathsf{X}(\mathcal{C}) \cap \mathsf{REG}$ by $\mathsf{X}_{\mathsf{reg}}(\mathcal{C})$, where REG is the class of all regular languages.

▶ **Theorem 3.3** ([20]). *The following holds:*

- $\mathsf{V}_{\mathsf{reg}}(o(n)) = \mathsf{F}_{\mathsf{reg}}(o(n)) = \mathsf{F}_{\mathsf{reg}}(\mathcal{O}(\log n)) = \mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$
- $\mathsf{F}_{\mathsf{reg}}(o(\log n)) = \mathsf{F}_{\mathsf{reg}}(\mathcal{O}(1))$
- $\mathsf{V}_{\mathsf{reg}}(o(\log n)) = \mathsf{V}_{\mathsf{reg}}(\mathcal{O}(1)) =$ *all trivial languages (empty and universal languages)*

Strictly speaking, [20, Theorem 7] only claims $\mathsf{V}_L(n) \notin O(1)$ for all non-trivial languages L. However, the proof of [20, Theorem 7] does imply the stronger bound $\mathsf{V}_L(n) \notin o(\log n)$. This statement will also be reproved in the following section.

Let us comment on a subtle point. When making statements about the space complexity functions $V_L(n)$ and $F_L(n)$ it is in general important to fix the underlying alphabet. For instance according to the third point from Theorem 3.3 we have $V_L(n) \in \mathcal{O}(1)$ for the language $L = \{a\}^*$ if the underlying alphabet is $\{a\}$. On the other hand, if the underlying alphabet is $\{a, b\}$ then $V_L(n) \notin \mathcal{O}(1)$ (in fact, L then belongs to $\mathsf{V}_{\mathsf{reg}}(\Theta(\log n))$).

4 Space complexity and language growth

In this section we reprove the space trichotomy (Theorem 3.3) for the variable-size model. For this we relate the function $V_L(n)$ to the growth of a certain derived language and then use the well known results about the growth of regular languages. We need the following definition. For a language $L \subseteq \Sigma^*$ define the mapping $\psi_L\colon \Sigma^* \to (\Sigma^*/{\sim_L})^*$ by $\psi_L(a_1 \cdots a_n) = [a_1 \cdots a_n]_{\sim_L} [a_2 \cdots a_n]_{\sim_L} \cdots [a_n]_{\sim_L}$. Notice that ψ_L is a length-preserving mapping from Σ^* to the set of words over the alphabet $\Sigma^*/{\sim_L}$. Although $\Sigma^*/{\sim_L}$ may be infinite (namely for non-regular L), the image $\psi_L(\Sigma^{\le n})$ has at most $|\Sigma|^{n+1} - 1$ elements for each $n \ge 0$.

▶ **Theorem 4.1.** *For every language* $\emptyset \subsetneq L \subsetneq \Sigma^*$ *we have* $V_L(n) = \log|\psi_L(\Sigma^{\leq n})|$.

Proof sketch. We first exhibit a variable-size sliding window algorithm $\mathcal{A}$ with $v_{\mathcal{A}}(n) = \log|\psi_L(\Sigma^{\leq n})|$. The idea is that on input $w \in \overline{\Sigma}^*$ the algorithm $\mathcal{A}$ is in state $\mathcal{A}(w) = \psi_L(\mathrm{wnd}(w))$. Consider an active window $a_1 \cdots a_n \in \Sigma^*$. Three observations are crucial:

- $\psi_L(a_2 \cdots a_n)$ is obtained from $\psi_L(a_1 \cdots a_n)$ by removing the first $\sim_L$-class $[a_1 \cdots a_n]_{\sim_L}$.
- $\psi_L(a_1 \cdots a_n)$ and $a \in \Sigma$ determine $\psi_L(a_1 \cdots a_n a) = [a_1 \cdots a_n a]_{\sim_L} \cdots [a_n a]_{\sim_L}[a]_{\sim_L}$, since $\sim_L$ is a right-congruence
- The first $\sim_L$-class in $\psi_L(a_1 \cdots a_n)$ determines whether $a_1 \cdots a_n \in L$.

These remarks define a variable-size sliding window algorithm for L with state set $\psi_L(\Sigma^*)$. It is easy to define a binary encoding of the states such that this variable-size sliding window algorithm has space complexity $\log|\psi_L(\Sigma^{\leq n})|$.

Conversely, consider a variable-size sliding window algorithm $\mathcal{A}$ for L with space complexity $v(n) = v_{\mathcal{A}}(n)$. To prove that $v(n) \geq \log|\psi_L(\Sigma^{\leq n})|$, one shows that for every input $x = a_1 a_2 \cdots a_m \in \Sigma^*$ of length $m \leq n$, the state $\mathcal{A}(x)$ determines (i) $m = |x|$ and (ii) $\psi_L(a_1 \cdots a_m)$. Hence, every value $\psi_L(x)$ for $x \in \Sigma^{\leq n}$ can be encoded by a bit string of length at most $v(n)$, namely $\mathrm{enc}(\mathcal{A}(x))$. Since there are $|\psi_L(\Sigma^{\leq n})|$ such values, it follows that $2^{v(n)+1} - 1 \geq |\psi_L(\Sigma^{\leq n})|$, which implies $v(n) \geq \log|\psi_L(\Sigma^{\leq n})|$. ◀

Lemma 4.1 fails for $L = \emptyset$ or $L = \Sigma^*$, where $V_L(n) = 0$ and $\log|\psi_L(\Sigma^{\leq n})| = \log(n+1)$.

We can use Lemma 4.1 to reprove the space trichotomy for regular languages in the variable-size sliding window model. For this, we need the following simple lemma:

▶ **Lemma 4.2.** *If* $L \subseteq \Sigma^*$ *is regular, then* ψ_L *is a* $\leftarrow$*-transduction. In particular,* $\psi_L(\Sigma^*)$ *and* $\psi_L(L)$ *are regular. Furthermore* ψ_L *is a* $\leftarrow$*-reduction from* L *to* $\psi_L(L)$.

The *growth* of a language $L \subseteq \Sigma^*$ is the function $g(n) = |\{x \in L : |x| \leq n\}|$. Since the growth of every regular language is either $\Theta(n^d)$ for some integer $d \geq 0$ or $\Omega(r^n)$ for some $r > 1$ [21, Section 2.3], Lemma 4.1 and 4.2 reprove the trichotomy theorem for variable-size windows: For a regular language L, $V_L(n)$ is either in $\mathcal{O}(1)$, $\Theta(\log n)$ or $\Theta(n)$. Furthermore, since $|\psi_L(\Sigma^{\leq n})| \geq n+1$ we have $V_L(n) \in \Omega(\log n)$ for every non-trivial language L.

Let us conclude this section with a result that bounds for all languages the fixed-size space function $F_L(n)$ in terms of the growth of L.

▶ **Theorem 4.3.** *If* $L \subseteq \Sigma^*$ *has growth* $g(n)$*, then* $F_L(n) \in \mathcal{O}(\log g(n) + \log n)$.

5 Logspace sliding-window algorithms

In this section we will study the class $\mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$. We will (i) give several new and very natural characterizations of $\mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$, (ii) will exhibit a new logspace sliding-window algorithm that is more space efficient in terms of the automata size compared to our previous algorithm from [20], and (iii) will match our new space bound by an almost tight lower bound. Before we state the results, we have to introduce a couple of definitions.

A *strongly connected component* (SCC for short) of a DFA $\mathcal{B} = (Q, \Sigma, q_0, \delta, F)$ is an inclusion-maximal subset $C \subseteq Q$ such that for all $p, q \in C$ there exist words $u, v \in \Sigma^*$ such that $\delta(p, u) = q$ and $\delta(q, v) = p$. An SCC $C \subseteq Q$ is *well-behaved* if for all $q \in C$ and $u, v \in \Sigma^*$ with $|u| = |v|$ and $\delta(q, u), \delta(q, v) \in C$ we have: $\delta(q, u) \in F$ if and only if $\delta(q, v) \in F$. If every SCC in $\mathcal{B}$ which is reachable from q_0 is well-behaved, then $\mathcal{B}$ is called *well-behaved*.

A language $L \subseteq \Sigma^*$ is called a *left ideal (right ideal)* if $\Sigma^* L \subseteq L$ ($L\Sigma^* \subseteq L$). A language $L \subseteq \Sigma^*$ is called a *length language* if for all $n \in \mathbb{N}$, either $\Sigma^n \subseteq L$ or $L \cap \Sigma^n = \emptyset$. Clearly, L

is a length language iff L^{R} is a length language, and L is left ideal iff L^{R} is a right ideal. In this section we prove the main characterization theorem for the class $\mathsf{V}_{\mathsf{reg}}(\mathcal{O}(\log n))$:

▶ **Theorem 5.1.** *Let $L \subseteq \Sigma^*$ be regular. The following statements are equivalent:*
1. $L \in \mathsf{F}(\mathcal{O}(\log n))$
2. $L \in \mathsf{V}(\mathcal{O}(\log n))$
3. *L^{R} is recognized by a well-behaved DFA.*
4. *L is $\leftarrow$-reducible to a regular language of polynomial growth.*
5. *L is a Boolean combination of regular left ideals and regular length languages.*

Our proof of the direction from 3. to 2. will also yield a better space bound in terms of automata size. In [20] we presented a variable-size sliding window algorithm using space $\mathcal{O}(m^m \cdot (m \cdot \log(m) + \log(n)))$ for a regular language that is given by a DFA with m states.

▶ **Theorem 5.2.** *Let $\mathcal{A}$ be a DFA or NFA with m states such that $\mathcal{A}^{\mathsf{RD}}$ is well-behaved. There are constants c_m, d_m that only depend on m such that the following holds for $L = L(\mathcal{A})$:*
- *If $\mathcal{A}$ is a DFA then $V_L(n) \le (2^m \cdot m + 1) \cdot \log n \ + \ c_m$ for n large enough.*
- *If $\mathcal{A}$ is an NFA then $V_L(n) \le (4^m + 1) \cdot \log n \ + \ d_m$ for n large enough.*

Finally we prove a lower bound for the fixed-size model (and hence also for the variable-size model) that almost matches the space bound in Theorem 5.2:

▶ **Theorem 5.3.** *For all $k \ge 1$ there exists a language $L_k \subseteq \{0, \ldots, k\}^*$ recognized by a DFA with $k+3$ states such that $L_k \in \mathsf{F}(\mathcal{O}(\log n))$ and $F_{L_k}(n) \ge (2^k - 1) \cdot (\log n - k)$.*

We start with the proof of Theorem 5.2.

5.1 Proof of Theorem 5.2

We need one more definition for the proof of Theorem 5.2. Let $\mathcal{B}$ be a well-behaved DFA with m states and let ρ be a run in $\mathcal{B}$, which does not necessarily start in the initial state. Let $C_1, \ldots, C_k$ be the sequence of pairwise different SCCs that are visited by ρ in that order. The *path summary* of ρ is the sequence $S(\rho) = (p_1, \ell_1, p_2, \ell_2, \ldots, p_k, \ell_k)$ where p_i is the first state in C_i visited by ρ, and $\ell_i \ge 0$ is the number of symbols read in ρ from the first occurrence of p_i until the first state from C_{i+1} (or until the end for p_k). The number of different path summaries $S(\rho)$, where ρ ranges over all runs in $\mathcal{B}$ of length n can be bounded by (e is Euler's constant)

$$m^m \cdot \binom{n+m-1}{m-1} \le m^m \cdot \binom{n+m}{m} \le m^m \cdot \left(\frac{e \cdot (n+m)}{m}\right)^m \le e^m \cdot (n+m)^m. \qquad (1)$$

Here, (i) m^m is the number of sequences of m states (we can repeat the last state in a path summary so that we have exactly m states) and (ii) $\binom{n+m-1}{m-1}$ is the number of ordered partitions of n into m summands.

We can now prove Theorem 5.2. Let $L \subseteq \Sigma^*$ be regular and given by a finite DFA or NFA $\mathcal{A}$. Let $\mathcal{B} = \mathcal{A}^{\mathsf{RD}}$, which is well-behaved. A set $D \subseteq \Sigma^*$ *distinguishes* L if for all $x, y \in \Sigma^*$ with $x \not\sim_L y$ there exists $z \in D$ such that exactly one of the words xz and yz belongs to L. If $\mathcal{A}$ is a DFA with m states, then there are at most m distinct left quotients $x^{-1}L$. Since every family of m sets has a distinguishing set of size at most $m-1$ [16], we get a set D of size at most $m-1$ that distinguishes L. If $\mathcal{A}$ is an NFA with m states, we can clearly choose $|D| \le 2^m - 1$ by determinizing $\mathcal{A}$.

For a window content $w = a_1 \cdots a_n$ we define a 0-1-matrix $A_w \colon D \times \{1, \ldots, n\} \to \{0,1\}$ by $A_w(z, i) = 1$ iff $a_i \cdots a_n z \in L$. Note that the i-th column $A_w(\cdot, i)$ determines $[a_i \cdots a_n]_{\sim_L}$,

and vice versa. Hence, the matrix A_w determines $\psi_L(w)$ and vice versa, i.e., $|\psi_L(\Sigma^{\le n})| = |\{A_w : w \in \Sigma^{\le n}\}|$. By Lemma 4.1, it therefore suffices to bound $|\{A_w : w \in \Sigma^{\le n}\}|$.

We can encode each row $A_w(z, \cdot)$ of A_w succinctly as follows. Consider one row indexed by $z \in D$. Let ρ_z be the run of $\mathcal{B}$ on the word $(wz)^{\mathsf{R}}$ and $\tilde{\rho}_z$ be the subrun of ρ_z which only reads the suffix w^{R} of $(wz)^{\mathsf{R}}$. One can reconstruct $A_w(z, \cdot)$ from the path summary $S(\tilde{\rho}_z)$. Thus A_w can be encoded by $|D|$ many path summaries. With (1) and the fact that $\mathcal{B}$ has at most 2^m states, we get the bound

$$|\{A_w : w \in \Sigma^{\le n}\}| \le \sum_{i=0}^{n} e^{2^m|D|} \cdot (i + 2^m)^{2^m|D|} \le (n+1) \cdot e^{2^m|D|} \cdot (n + 2^m)^{2^m|D|}.$$

Hence, for the DFA case (where $|D| \le m - 1$) we have

$$V_L(n) = \log|\psi_L(\Sigma^{\le n})| \le \log(n+1) + 2^m \cdot m \cdot (\log e + \log(n+2^m)) \le (2^m \cdot m + 1) \cdot \log n \, + \, c_m$$

for n large enough, where c_m can be chosen as $1 + 2^m \cdot m \cdot \log e + m^2 \cdot 2^m$. The calculation for the NFA case (where $|D| \le 2^m - 1$) is analogous.

5.2 Proof of Theorem 5.1

The equivalence of 1. and 2. is shown in [20] (its the only direction that we do not reprove), and the direction from 3. to 2. is stated in Theorem 5.2. The implication from 2. to 4. follows from Lemma 4.1 and 4.2. The direction from 2. to 3. is shown in the full version [19], where we actually show that $V_L(n) \in \Omega(n)$ if L^{R} is recognized by a DFA that is not well-behaved. To prove that 5. implies 3. we show in [19] that (i) the minimal DFA for a regular right ideal or a regular length language is well-behaved, and (ii) that the class of languages accepted by well-behaved DFAs is closed under Boolean operations.

It remains to show the implication from 4. to 5. First, a straightforward argument shows that the class of Boolean combinations of regular left ideals and regular length languages is closed under pre-images of $\longleftarrow$-transductions (see [19]). Therefore, it suffices to prove that every regular language of polynomial growth is a Boolean combination of regular left ideals and regular length languages. Since a language L and its reversal L^{R} have the same growth, we can instead show that every regular language of polynomial growth is a Boolean combination of regular right ideals and regular length languages. The idea is to decompose every regular language of polynomial growth as a finite union of languages recognized by so called linear cycle automata.

In the following we will allow *partial* DFAs $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ where $\delta : Q \times \Sigma \to Q$ is a partial function. An SCC C of a partial DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is called a *cycle* if for every $p \in C$ there exists at most one $a \in \Sigma$ such that $\delta(p, a) \in C$. Note that a singleton SCC $C = \{p\}$ such that $\delta(p, a) \neq p$ whenever $\delta(p, a)$ is defined is a cycle, too. Such a cycle is called *trivial*. A partial DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is a *linear cycle automaton* if

- for all $p, q \in Q$ there exists at most one symbol $a \in \Sigma$ such that $\delta(p, a) = q$,
- every SCC C of $\mathcal{A}$ is a (possibly trivial) cycle,
- there is an enumeration $C_1, \dots, C_k$ of the SCCs of $\mathcal{A}$ such that there is a unique transition from C_i to C_{i+1} for $1 \le i \le k - 1$, and there is no transition from C_i to C_j for $j > i + 1$,
- q_0 belongs to C_1,
- $|F| = 1$ and the unique final state belongs to C_k.

▶ **Lemma 5.4.** *If L is a regular language with polynomial growth, then L is a finite union of languages recognized by linear cycle automata.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be the minimal DFA for a regular language $L \subseteq \Sigma^*$ of polynomial growth. We first remove from $\mathcal{A}$ all states from which no state in F is reachable; then $\mathcal{A}$ becomes a partial DFA. By [21, Lemma 2] for every $q \in Q$ there exists a word $u_q \in \Sigma^*$ such that the language $\{w \in \Sigma^* : \delta(q, w) = q\}$ is a subset of u_q^*. Thus, for every SCC C of $\mathcal{A}$ and every state $q \in C$ there is at most one symbol $a \in \Sigma$ with $\delta(q, a) \in C$.

A *path description* is a sequence $P = (q_0, C_0, p_0, a_0, q_1, C_1, p_1, a_2, \ldots, q_k, C_k, p_k)$ where $C_0, \ldots, C_k$ is a chain in the partial ordering on the SCCs of $\mathcal{A}$, $q_i, p_i \in C_i$ for all $0 \le i \le k$, $\delta(p_i, a_i) = q_{i+1}$ for all $0 \le i < k$ and $p_k \in F$. There are only finitely many path descriptions. To every accepting run of $\mathcal{A}$ we assign a path description, which indicates the SCCs traversed in the run and the transitions that lead from one SCC to the next SCC. We can write $L(\mathcal{A})$ as a finite union of languages over all path descriptions. For every path description P, we take the set of all words accepted by a run of $\mathcal{A}$ whose path description is P.

Consider a single path description $P = (q_0, C_0, p_0, a_0, q_1, C_1, p_1, a_2, \ldots, q_k, C_k, p_k)$ and let $\mathcal{B}$ be the restriction of $\mathcal{A}$ to the SCCs C_i. Furthermore all transitions between two distinct SCCs are removed except for the transitions (p_i, a_i, q_{i+1}). Finally, p_k becomes the only final state of $\mathcal{B}$. Then $\mathcal{B}$ is indeed a linear cycle automaton. ◀

By the previous lemma, it suffices to decompose the language accepted by a linear cycle automaton as a Boolean combination of regular length languages and regular right ideals. By a simple pumping argument (see [19]) we can reduce to linear cycle automata, in which each cycle has the same length. Let us consider such an automaton $\mathcal{A}$ and let $L = L(\mathcal{A})$: There are numbers $p, q \ge 0$ such that each word in L has length $p + qn$ for some $n \ge 0$. Here q is the uniform length of the non-trivial cycles in $\mathcal{A}$. We claim that L is the intersection of the three languages

- $L\Sigma^*$, which is a regular right ideal,
- $\{x \in \Sigma^* : \mathrm{Pref}(x) \subseteq \mathrm{Pref}(L)\}$, which is the complement of a regular right ideal,
- $\Sigma^p(\Sigma^q)^*$, which is a length language.

Clearly L is contained in the described intersection. Conversely, consider a word x in the intersection. We have $x = yz$ where $y \in L$. Hence, $|y| = p+qn$ for some n. Since $|x| = p+qn'$ for some n', the length $|z|$ is divided by q. Since $y \in L$, $\mathcal{A}(y)$ is the unique final state of $\mathcal{A}$, which belongs to the unique maximal SCC C of $\mathcal{A}$. If C is non-trivial, then it is a cycle of length q and also $\mathcal{A}(yz)$ is the final state, i.e., $x \in L$. If C is trivial, then $y, yz \in L$ implies $z = \varepsilon$ and x is also accepted by $\mathcal{A}$. This concludes the proof for the direction from 4. to 5.

5.3 Proof of Theorem 5.3

The languages L_k ($k \ge 0$) from Theorem 5.3 are defined as

$$L_0 = 0^+ \quad \text{and} \quad L_k = L_{k-1} \cup L_{k-1}\, k\, \{0, \ldots, k-1\}^* \text{ for } k \ge 1.$$

Observe that a word $a_1 \cdots a_n \in \{0, \ldots, k\}^*$ belongs to L_k if and only if $n \ge 1$, $a_1 = 0$ and for each $1 \le i \le n$ it holds that $a_i = 0$ or $a_i \ne \max_{1 \le j \le i-1} a_j$. We can construct a DFA $\mathcal{A}_k$ for L_k with $k + 3$ states, which stores the maximum value seen so far in its state, see Figure 1.

To prove that each L_k belongs to $\mathsf{V}(\mathcal{O}(\log n))$, we show that L_k is a Boolean combination of regular left ideals. Given a word $x = a_1 \cdots a_n \in \Sigma^*$ and a language $L \subseteq \Sigma^*$, a position $1 \le i \le n$ is an *L-alternation point*, if exactly one of the words $a_i \cdots a_n$ and $a_{i+1} \cdots a_n$ belongs to L. Denote by $\mathrm{alt}_L(x)$ the number of L-alternation points in x. We need the following two lemmas, which are proven in the full version [19].

▶ **Lemma 5.5.** *Let $L \subseteq \Sigma^*$ be regular. Then L is a Boolean combination of at most k regular left ideals if and only if $\mathrm{alt}_L(x) \le k$ for all $x \in \Sigma^*$.*

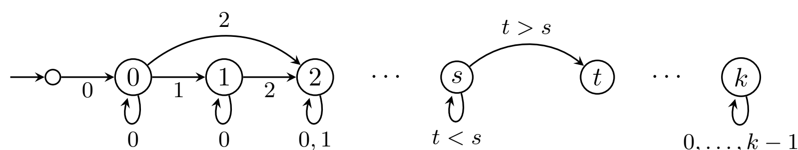

Figure 1 A DFA for L_k. Omitted transitions lead to a sink state. All non-sink states are final.

▶ **Lemma 5.6.** *For all $k \geq 0$ and $x \in \mathbb{N}^*$ we have $\mathrm{alt}_{L_k}(x) \leq 2^{k+2} - 2$. Moreover, $V_{L_k}(n) \leq (2^{k+3} \cdot (k+3) + 1) \cdot \log n \;+\; c_k$ for n large enough, where c_k only depends on k.*

For the proof of the if-direction in Lemma 5.5, one writes L as a Boolean combination of the sets $\{x \in \Sigma^* : \mathrm{alt}_L(x) \geq i\}$ $(1 \leq i \leq k)$. Lemma 5.6 is shown by induction on k.

We can now prove Theorem 5.3. Define the languages $Z_0 = 0^*$ and $Z_k = Z_{k-1}\, k\, Z_{k-1}$ for $k \geq 1$. An example word from Z_3 is 0010002100300010020010. Note that every suffix of $x \in Z_k$ that starts with 0 belongs to L_k and every suffix of $x \in Z_k$ that starts with $a > 0$ does not belong to L_k. The former follows by induction on k; the latter holds since $L_k \subseteq 0\,\mathbb{N}^*$.

Fix some $k \geq 1$ and let $\mathcal{B} = (\mathcal{B}_n)_{n \geq 0}$ be a fixed-size sliding window algorithm for L_k. Consider a window size n. We claim that $\mathcal{B}_n(x) \neq \mathcal{B}_n(y)$ for all $x, y \in Z_k$ with $|x| = |y| = n$ and $x \neq y$. To see this, write $x = zau$ and $y = zbv$ with $a, b \in \{0, \ldots, k\}$, $a \neq b$. We must have $a = 0$ and $b > 0$ or vice versa. Assume that $a = 0$ and $b > 0$. Thus, $au \in L_k$ and $bv \notin L_k$. Hence, we have $\mathrm{wnd}(x0^{|z|}) = au0^{|z|} \in L_k$ and $\mathrm{wnd}(y0^{|z|}) = bv0^{|z|} \notin L_k$. But if $\mathcal{B}_n(x) = \mathcal{B}_n(y)$, then also $\mathcal{B}_n(x0^{|z|}) = \mathcal{B}_n(y0^{|z|})$, which yields a contradiction.

It follows that $\mathcal{B}_n$ has at least $\binom{n}{2^k-1} \geq (n/(2^k-1))^{2^k-1} \geq (n/2^k)^{2^k-1}$ many states, which implies $v_{\mathcal{B}}(n) \geq (2^k - 1) \cdot (\log n - k)$.

6 Constant space sliding-window algorithms

Lemma 4.1 implies that $V_L(n) \geq \log n$ if $\emptyset \neq L \neq \Sigma^*$. Thus, only trivial languages have a constant-space variable-size streaming algorithm. This changes in the fixed-size window model. In [20] we characterized those regular languages L in $\mathsf{F}(\mathcal{O}(1))$ in terms of the left Cayley graph of the syntactic monoid of L. Here we give a more natural characterization.

A language $L \subseteq \Sigma^*$ is called *k-suffix testable* if for all $x, y \in \Sigma^*$ and $z \in \Sigma^k$ we have: $xz \in L \iff yz \in L$. Equivalently, L is a Boolean combination of languages of the form $\Sigma^* w$ where $w \in \Sigma^{\leq k}$. We call L *suffix testable* if it is k-suffix testable for some $k \geq 0$. Clearly, every finite language is suffix testable: if $L \subseteq \Sigma^{\leq k}$ then L is $(k+1)$-suffix testable. The class of suffix testable languages corresponds to the variety $\mathbf{D}$ of definite monoids [32]. Our main result about the class $\mathsf{F}(\mathcal{O}(1))$ is the following; its proof can be found in the full version [19]:

▶ **Theorem 6.1.** *A regular language $L \subseteq \Sigma^*$ belongs to $\mathsf{F}(\mathcal{O}(1))$ if and only if L is a finite Boolean combination of suffix testable languages and regular length languages.*

7 Deciding space complexity in the sliding window model

In this section, we consider the complexity of the following decision problems:

- DFA(1): Given a DFA $\mathcal{A}$, does $L(\mathcal{A})$ belong to $\mathsf{F}(\mathcal{O}(1))$?
- NFA(1): Given an NFA $\mathcal{A}$, does $L(\mathcal{A})$ belong to $\mathsf{F}(\mathcal{O}(1))$?
- DFA($\log n$): Given a DFA $\mathcal{A}$, does $L(\mathcal{A})$ belong to $\mathsf{F}(\mathcal{O}(\log n)) = \mathsf{V}(\mathcal{O}(\log n))$?
- NFA($\log n$): Given an NFA $\mathcal{A}$, does $L(\mathcal{A})$ belong to $\mathsf{F}(\mathcal{O}(\log n)) = \mathsf{V}(\mathcal{O}(\log n))$?

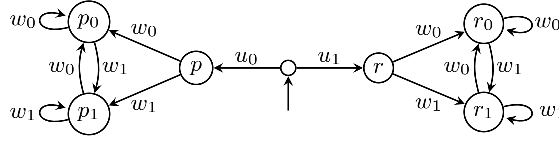

Figure 2 A critical tuple (u_0, u_1, w_0, w_1).

Recall that by Theorem 3.3, $L(\mathcal{A})$ belongs to $\mathsf{V}(\mathcal{O}(1))$ iff $L(\mathcal{A})$ is trivial. The latter problem can be shown to be NL-complete (resp., PSPACE-complete) if $\mathcal{A}$ is a DFA (resp., an NFA) using standard constructions. The same complexity bounds hold for the above problems:

▶ **Theorem 7.1.** *The following hold:*

- DFA(1) *and* DFA($\log n$) *are* NL-*complete.*
- NFA(1) *and* NFA($\log n$) *are* PSPACE-*complete.*

We only sketch the NL upper bound for DFA($\log n$); the other parts of Theorem 7.1 are shown in the full version [19]. We can assume that the input DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is minimal; see the argument for the NL upper bound for DFA(1) in [19]. Let $L = L(\mathcal{A})$. For $u, x_0, x_1 \in \Sigma^*$ we define $Q(u, x_0, x_1) = \{\mathcal{A}(ux) : x \in \{x_0, x_1\}^*\}$, which is the set of states of $\mathcal{A}$ reachable from the initial state by first reading u and then an arbitrary product of copies of x_0 and x_1. We call a tuple (u_0, u_1, w_0, w_1) of words *critical*, if (i) $|u_0| = |u_1| \geq 1$, (ii) u_i is a suffix of w_i for all $i \in \{0, 1\}$ and (iii) $Q(u_0, w_0, w_1) \cap Q(u_1, w_0, w_1) = \emptyset$. Using critical tuples, we can state another characterization of the class $\mathsf{V}_{\mathsf{reg}}(\log n)$:

▶ **Lemma 7.2.** *We have $L \notin \mathsf{V}_{\mathsf{reg}}(\log n)$ if and only if there exists a critical tuple in $\mathcal{A}$.*

We show that if there exists a critical tuple, then there exists a critical tuple (u_0, u_1, w_0, w_1) such that $|Q(u_0, w_0, w_1)| \leq 3 \geq |Q(u_1, w_0, w_1)|$. Assume that (u_0, u_1, w_0, w_1) is a critical tuple. Let $h \colon \Sigma^* \to M$ be the canonical homomorphism into the transition monoid M of $\mathcal{A}$, which right acts on Q via $Q \times M \to Q$, $(q, m) \mapsto q \cdot m = m(q)$. Notice that $Q(u_i, w_0, w_1) = \{\mathcal{A}(u_i) \cdot m : m \in \{h(w_0), h(w_1)\}^*\}$, where X^* denotes the submonoid of M generated by $X \subseteq M$. It suffices to define a new critical tuple (u_0, u_1, x_0, x_1) with the property that $h(x_i) \cdot h(x_j) = h(x_j)$ for all $i, j \in \{0, 1\}$. This implies $\{h(x_0), h(x_1)\}^* = \{1, h(x_0), h(x_1)\}$, and hence, $|Q(u_0, x_0, x_1)| \leq 3 \geq |Q(u_1, x_0, x_1)|$.

Notice that if (u_0, u_1, w_0, w_1) is critical, then also $(u_0, u_1, y_0 w_0, y_1 w_1)$ is critical for all $y_0, y_1 \in \{w_0, w_1\}^*$. Let $\omega \geq 1$ be a number such that m^ω is idempotent for all $m \in M$. By choosing $e_0 = (h(w_0)^\omega h(w_1)^\omega)^\omega h(w_0)^\omega$ and $e_1 = (h(w_0)^\omega h(w_1)^\omega)^\omega$ we indeed obtain $e_i e_j = e_j$ for all $i, j \in \{0, 1\}$. Hence we define $x_0 = (w_0^\omega w_1^\omega)^\omega w_0^\omega$ and $x_1 = (w_0^\omega w_1^\omega)^\omega$.

To decide whether $L \notin \mathsf{V}_{\mathsf{reg}}(\log n)$ it therefore suffices to check whether there is a critical tuple (u_0, u_1, w_0, w_1) such that $|Q(u_0, w_0, w_1)| \leq 3 \geq |Q(u_1, w_0, w_1)|$. Figure 2 illustrates the substructure we need to detect in $\mathcal{A}$. We show that the existence of such a structure can be verified in NL. To do so, we reduce to testing emptiness of one-counter automata, which is known to be in NL [25]. For two states $p, r \in Q$ let $\mathcal{A}_{p,r} = (Q, \Sigma, p, \delta, \{r\})$ be the automaton $\mathcal{A}$ with initial state p and final state r, and let $L(p, r) = L(\mathcal{A}_{p,r})$. The algorithm iterates over all disjoint sets $\{p, p_0, p_1\}, \{r, r_0, r_1\} \subseteq Q$. For $i \in \{0, 1\}$ let $\mathcal{A}_i$ be a DFA for the language $L(p, p_i) \cap L(p_0, p_i) \cap L(p_1, p_i) \cap L(r, r_i) \cap L(r_0, r_i) \cap L(r_1, r_i)$. Now consider the language $\{v_0 \# u_0 \# v_1 \# u_1 : v_i u_i \in L(\mathcal{A}_i) \text{ for } i \in \{0, 1\}, |u_0| = |u_1| \geq 1, u_0 \in L(q_0, p), u_1 \in L(q_0, r)\}$ for which one can construct in logspace a one-counter automaton. The counter is used to verify the constraint $|u_0| = |u_1|$. The language above is empty if and only if $\mathcal{A}$ has a critical tuple. This concludes the proof that DFA($\log n$) belongs to NL.

References

1 Charu C. Aggarwal. *Data Streams - Models and Algorithms*. Springer, 2007.
2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.
3 Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of PODS 2004*, pages 286–296. ACM, 2004.
4 Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of PODS 2003*, pages 234–243. ACM, 2003.
5 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013.
6 Ajesh Babu, Nutan Limaye, and Girish Varma. Streaming algorithms for some problems in log-space. In *Proceedings of TAMC 2010*, volume 6108 of *Lecture Notes in Computer Science*, pages 94–104. Springer, 2010.
7 Vladimir Braverman. Sliding window algorithms. In *Encyclopedia of Algorithms*, pages 2006–2011. Springer, 2016.
8 Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *Proceedings of FOCS 2007*, pages 283–293. IEEE Computer Society, 2007.
9 Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *J. Comput. Syst. Sci.*, 78(1):260–272, 2012.
10 Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Trans. Algorithms*, 10(4):22:1–22:12, 2014.
11 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary matching in a stream. In *Proceedings of ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2015.
12 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *Proceedings of SODA 2016*, pages 2039–2052. SIAM, 2016.
13 Raphaël Clifford and Tatiana A. Starikovskaya. Approximate hamming distance in a stream. In *Proceedings of ICALP 2016*, volume 55 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
14 Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic graphs in the sliding-window model. In *Proceedings of ESA 2013*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013.
15 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.
16 A. Policriti F. Parlamento and K. Rao. Witnessing differences without redundancies. *Proceedings of the American Mathematical Society*, 125(2):587–594, 1997.
17 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.
18 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proceedings of ESA 2016*, volume 57 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
19 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. Technical report, arXiv.org, 2018. `https://arxiv.org/abs/1702.04376`.
20 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *Proceedings of FSTTCS 2016*, volume 65 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

21 Pawel Gawrychowski, Dalia Krieger, Narad Rampersad, and Jeffrey Shallit. Finding the growth rate of a regular or context-free language in polynomial time. *International Journal on Foundations of Computer Science*, 21(4):597–618, 2010.

22 Lukasz Golab and M. Tamer Özsu. Processing sliding window multi-joins in continuous queries over data streams. In *Proceedings of VLDB 2003*, pages 500–511. Morgan Kaufmann, 2003.

23 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.

24 Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming algorithms for recognizing nearly well-parenthesized expressions. In *Proceedings of MFCS 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 412–423. Springer, 2011.

25 Michel Latteux. Langages á un compteur. *Journal of Computer and System Sciences*, 26(1):14–33, 1983.

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

27 Philip M. Lewis II, Richard Edwin Stearns, and Juris Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Proceedings of SWCT (FOCS) 1965*, pages 191–202. IEEE Computer Society, 1965.

28 J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

29 Luc Segoufin and Cristina Sirangelo. Constant-memory validation of streaming XML documents against dtds. In *Proceedings of ICDT 2007*, volume 4353 of *Lecture Notes in Computer Science*, pages 299–313. Springer, 2007.

30 Luc Segoufin and Victor Vianu. Validating streaming XML documents. In *Proceedings of PODS 2002*, pages 53–64. ACM, 2002.

31 Richard Edwin Stearns, Juris Hartmanis, and Philip M. Lewis II. Hierarchies of memory limited computations. In *Proceedings of SWCT (FOCS) 1965*, pages 179–190. IEEE Computer Society, 1965.

32 Howard Straubing. Finite semigroup varieties of the form $V * D$. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.

33 Andrew Szilard, Sheng Yu, Kaizhong Zhang, and Jeffrey Shallit. Characterizing regular languages with polynomial densities. In *Proceedings of MFCS 1992*, volume 629 of *Lecture Notes in Computer Science*, pages 494–503. Springer, 1992.