

# Sliding Windows over Context-Free Languages

Moses Ganardi

Universität Siegen, Germany  
ganardi@eti.uni-siegen.de

Artur Jeż

University of Wrocław, Poland  
aje@cs.uni.wroc.pl

Markus Lohrey

Universität Siegen, Germany  
lohrey@eti.uni-siegen.de

---

## Abstract

We study the space complexity of sliding window streaming algorithms that check membership of the window content in a fixed context-free language. For regular languages, this complexity is either constant, logarithmic or linear [4]. We prove that every context-free language whose sliding window space complexity is  $\log_2(n) - \omega(1)$  must be regular and has constant space complexity. Moreover, for every  $c \in \mathbb{N}$ ,  $c \geq 1$  we construct a (nondeterministic) context-free language whose sliding window space complexity is  $\mathcal{O}(n^{1/c}) \setminus o(n^{1/c})$ . Finally, we give an example of a deterministic one-counter language whose sliding window space complexity is  $\Theta((\log n)^2)$ .

**2012 ACM Subject Classification** Theory of computation → Streaming models

**Keywords and phrases** sliding windows, streaming algorithms, context-free languages

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2018.15

## 1 Introduction

In many streaming applications, data items are outdated after a certain time and the sliding window model is a simple way to model this: *Sliding window algorithms* process an input sequence  $a_1a_2 \cdots a_m$  from left to right and have at time  $t$  only direct access to the current symbol  $a_t$ . Moreover, at each time instant  $t$  the algorithm is required to compute a value that depends on the last  $n$  symbols. The value  $n$  is called the *window size* and the last  $n$  symbols form the *active window* at time  $t$ . A general goal in the area of sliding window algorithms is to avoid the explicit storage of the window content (which requires  $\Omega(n)$  bits), and, instead, to work in considerably smaller space, e.g., polylogarithmic space in the window size  $n$ . An introduction into the sliding window model can be found in [1, Chapter 8].

In our recent papers [3, 4] we initiated the study of sliding window algorithms for regular languages. In general, a sliding window algorithm for a language  $L \subseteq \Sigma^*$  decides, at every time instant, whether the word in the active window belongs to  $L$ . In [4] we proved that for every regular language  $L$  the optimal space bound for a sliding window algorithm for  $L$  is either constant, logarithmic or linear in the window size. In [3] we also gave several characterizations for the three space classes: A regular language has a sliding window algorithm with space bound  $\mathcal{O}(\log n)$  (resp.,  $\mathcal{O}(1)$ ) if and only if it belongs to the Boolean closure of regular left ideals and regular length languages (resp., the Boolean closure of suffix-testable languages and regular length languages); see [3] for the formal definition of these language classes.



© Moses Ganardi, Artur Jeż, Markus Lohrey;  
licensed under Creative Commons License CC-BY

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).

Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 15; pp. 15:1–15:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we investigate to which extent the results from [3, 4] can be generalized to context-free languages. Our first main result (Theorem 2) states that if  $L$  is a context-free language that has a sliding window algorithm with space bound  $\log_2(n) - \omega(1)$  (recall that  $f(n) \in \omega(1)$  iff  $\forall c > 0 \exists m \forall n \geq m : f(n) \geq c$ ) then  $L$  must be regular. By the results from [3, 4] this implies that  $L$  has a constant space sliding window algorithm and is a Boolean combination of suffix-testable languages and regular length languages. Our proof uses a variant of the classical pumping lemma. The crucial observation is that taking a reversed Greibach normal form grammar for  $G$ , we can ensure that pumping in a word of length  $n$  does not affect a suffix of length  $o(n)$ .

Theorem 2 shows that, analogously to regular languages, there is a gap between  $\mathcal{O}(1)$  and  $\mathcal{O}(\log n)$  in the space complexity spectrum for context-free languages. This leads to the question whether there is also a gap between  $\mathcal{O}(\log n)$  and  $\mathcal{O}(n)$  (as it is the case for regular languages). We answer this question negatively. For this we construct from a linear bounded automaton (LBA) a context-free language, whose sliding window space complexity is related to the time complexity of the LBA in a certain way. The precise technical statement can be found in Theorem 9. From this result we obtain for every  $c \in \mathbb{N}$  a context-free language, whose optimal sliding window algorithm uses space  $\mathcal{O}(n^{1/c})$  (Theorem 10).

The context-free languages from the proof of Theorem 9 are non-deterministic. They are obtained by taking the complement of all accepting computations of an LBA on an input from  $a^*$  (as usual, a computation is encoded by a sequence of configuration words). These complements are context-free since one can guess errors, but they are not deterministic context-free. This leads to the question whether there exist deterministic context-free languages for which the optimal sliding window algorithm has space complexity  $o(n) \setminus \mathcal{O}(\log n)$ . We answer this question positively by constructing a deterministic one-counter language whose optimal sliding window algorithm uses space  $\mathcal{O}((\log n)^2)$  (Theorem 15).

The results from Theorem 10 and 15 are also shown for a more general sliding window model, which is known as the variable-size model in the literature. In the sliding window model discussed so far, the window size is fixed and for every window size there exists a streaming algorithm. In contrast, in the variable-size model, there is a single streaming algorithm and the window can grow and shrink. In other words, the arrival of new symbols and expiration of old symbols can happen independently. A formal definition can be found in Section 2. The space complexity of a variable-size streaming algorithm is measured with respect to the maximal window size seen in the past. In [4] it was shown that analogously to the fixed-size model, the space complexity of a regular language with respect to the variable-size model is either constant, logarithmic, or linear. Moreover, a regular language has space complexity  $\mathcal{O}(\log n)$  in the variable-size model if and only if it has space complexity  $\mathcal{O}(\log n)$  in the fixed-size model (on the other hand only trivial languages have constant space complexity in the variable-size model). Corollary 13 states that there exists a deterministic one-counter language whose optimal variable-size sliding window algorithm uses space  $\Theta((\log n)^2)$ .

Finally, we prove that our results for deterministic one-counter languages can be also shown for the reversals of the latter (i.e., for languages that can be accepted by a deterministic one-counter automaton that works from right to left). This is not obvious, since the reversal of a deterministic context-free language is in general not deterministic context-free. Moreover, the arguments for our space trichotomy result for regular languages [3, 4] mainly use a DFA for the reverse language, hence one might think that these arguments extend to reversals of deterministic context-free languages.

## 2 Preliminaries

For a function  $f : \mathbb{N} \rightarrow \mathbb{N}$ , we use the standard Landau notations  $\mathcal{O}(f)$ ,  $\Omega(f)$ ,  $o(f)$  and  $\Theta(f)$ .

We assume that the reader is familiar with the basic notions of formal languages, in particular regular languages, see e.g. [7] for more details. Let  $\Sigma$  be a finite alphabet of symbols. With  $\varepsilon$  we denote the empty word. For a word  $w = a_1 \cdots a_m \in \Sigma^*$  of length  $|w| = m$  we define  $w[i] = a_i$  and  $w[i : j] = a_i \cdots a_j$  if  $i \leq j$  and  $w[i : j] = \varepsilon$  if  $i > j$ . We define  $w[i : ] = w[i : m]$  and  $w[: j] = [1 : j]$ . Let  $\Sigma^n = \{w \in \Sigma^* : |w| = n\}$ ,  $\Sigma^{\leq n} = \{w \in \Sigma^* : |w| \leq n\}$ , and  $\Sigma^{\geq n} = \{w \in \Sigma^* : |w| \geq n\}$ . A word  $v \in \Sigma^*$  is a *prefix* (resp., *suffix*) of the word  $w$  if there exists a word  $u \in \Sigma^*$  such that  $w = vu$  (resp.,  $w = uv$ ). With  $\text{prefix}(w)$  we denote the set of all prefixes of  $w$ . For a word  $w = a_1 a_2 \cdots a_m$  let  $\text{rev}(w) = a_m \cdots a_2 a_1$  denotes the word  $w$  read from right to left.

### 2.1 Automata and streaming algorithms

We use standard definitions from automata theory. A *deterministic automaton* is a tuple  $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ , where  $Q$  is a possibly infinite set of states,  $\Sigma$  is an alphabet,  $q_0 \in Q$  is the initial states,  $\delta : Q \times \Sigma \rightarrow Q$  is the transition relation, and  $F$  is the set of final states. The transition function  $\delta$  is extended to a function  $\delta : Q \times \Sigma^* \rightarrow Q$  in the usual way and we set  $\mathcal{A}(x) = \delta(q_0, x)$  for all  $x \in \Sigma^*$ . The language accepted from a state  $q \in Q$  is denoted by  $L(\mathcal{A}, q) = \{x \in \Sigma^* \mid \delta(q, x) \in F\}$  and the language accepted by  $\mathcal{A}$  is defined by  $L(\mathcal{A}) = L(\mathcal{A}, q_0)$ . If  $Q$  is finite, then  $\mathcal{A}$  is a *deterministic finite automaton* (DFA).

A data stream is a finite sequence of data values. We make the assumption that these data values are from a finite set  $\Sigma$ . Thus, a data stream is a finite word  $w = a_1 a_2 \cdots a_m \in \Sigma^*$ . 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  $f : \Sigma^* \rightarrow A$  into some domain  $A$ , which means that at time instant  $t$  the streaming algorithm outputs the value  $f(a_1 a_2 \cdots a_t)$ . In this paper, we only consider the Boolean case  $A = \{0, 1\}$ ; in other words, the streaming algorithm tests membership in a fixed language. Thus, a *streaming algorithm* over  $\Sigma$  can be seen as a deterministic (possibly infinite) automaton  $\mathcal{A} = (S, \Sigma, s_0, \delta, F)$ . Furthermore, we abstract away from the actual computation and only analyze the space requirement, which in particular means that we encode the states of  $\mathcal{A}$  by bit strings. We describe this encoding by an injective function  $\text{enc} : S \rightarrow \{0, 1\}^*$ . The *space function*  $\text{space}(\mathcal{A}, \cdot) : \Sigma^* \rightarrow \mathbb{N}$  specifies the space used by  $\mathcal{A}$  on a certain input: For  $w \in \Sigma^*$  let  $\text{space}(\mathcal{A}, w) = \max\{|\text{enc}(\mathcal{A}(u))| : u \in \text{prefix}(w)\}$ . We also say that  $\mathcal{A}$  is a *streaming algorithm* for the accepted language  $L(\mathcal{A})$ .

### 2.2 Sliding window streaming models

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 “relevant past” 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*.

## 15:4 Sliding Windows over Context-Free Languages

In the following we formally define these two models.

**Fixed-size sliding windows.** Given a word  $w \in \Sigma^*$  of length  $m$  and a window size  $n \geq 0$ , we define  $\text{last}_n(w) \in \Sigma^n$  by

$$\text{last}_n(w) = \begin{cases} w[m-n+1:] & \text{if } n \leq m, \\ a^{n-m}w, & \text{if } n > m, \end{cases}$$

which is called the *active window*. Here  $a \in \Sigma$  is an arbitrary, but fixed, symbol, which fills the initial window. For a language  $L$  and  $n \geq 0$  let  $L_n = \{w \in \Sigma^* : \text{last}_n(w) \in L\}$ . A sequence  $\mathcal{A} = (\mathcal{A}_n)_{n \geq 0}$  is a *fixed-size sliding window algorithm* for a language  $L \subseteq \Sigma^*$  if for each  $n$  the  $\mathcal{A}_n$  is a streaming algorithm for  $L_n$ . Its *space complexity* is the function  $f_{\mathcal{A}}: \mathbb{N} \rightarrow \mathbb{N} \cup \{\infty\}$  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  $L_n$  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  $\Sigma^n$  and transitions  $au \xrightarrow{b} ub$  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. Let  $\mathcal{A}_n$  be the minimal DFA for  $L_n$  and encode each state of  $\mathcal{A}_n$  with at most  $\lfloor \log_2(a_n) \rfloor$  bits, where  $a_n$  is the number of states of  $\mathcal{A}_n$ . Then  $\mathcal{A} = (\mathcal{A}_n)_{n \geq 0}$  is an *optimal fixed-size sliding window algorithm*  $\mathcal{A}$  for  $L$ . Finally, we define  $F_L(n) = f_{\mathcal{A}}(n) = \lfloor \log_2(a_n) \rfloor$ . 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, for  $L = \{au : u \in \{a, b\}^*, |u| \text{ odd}\}$  we have  $F_L(2n) \in \Theta(n)$  and  $F_L(2n+1) \in \mathcal{O}(1)$ . 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 size.

**Variable-size sliding windows.** For an alphabet  $\Sigma$  we define the extended alphabet  $\bar{\Sigma} = \Sigma \cup \{\downarrow\}$ . In the variable-size model the *active window*  $\text{wnd}(u) \in \Sigma^*$  for a stream  $u \in \bar{\Sigma}^*$  is defined by

- $\text{wnd}(\varepsilon) = \varepsilon$
- $\text{wnd}(ua) = \text{wnd}(u)a$  for  $a \in \Sigma$
- $\text{wnd}(u\downarrow) = \varepsilon$  if  $\text{wnd}(u) = \varepsilon$
- $\text{wnd}(u\downarrow) = v$  if  $\text{wnd}(u) = av$  for  $a \in \Sigma$

A *variable-size sliding window algorithm* for a language  $L \subseteq \Sigma^*$  is a streaming algorithm  $\mathcal{A}$  for  $\{w \in \bar{\Sigma}^* : \text{wnd}(w) \in L\}$ . Following [3], we define its *space complexity* as the function  $v_{\mathcal{A}}: \mathbb{N} \rightarrow \mathbb{N} \cup \{\infty\}$  mapping each window size  $n$  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  $v_{\mathcal{A}}(n) = \max\{\text{space}(\mathcal{A}, u) : u \in \bar{\Sigma}^*, |\text{wnd}(v)| \leq n \text{ for all } v \in \text{prefix}(u)\}$ . By taking the minimal (possibly infinite) deterministic automaton for  $\{w \in \bar{\Sigma}^* : \text{wnd}(w) \in L\}$  and encoding states appropriately one can prove that there exists an optimal space bound:

► **Lemma 1** ([3]). *For every language  $L \subseteq \Sigma^*$  there exists a variable-size sliding window algorithm  $\mathcal{A}$  for  $L$  such that  $v_{\mathcal{A}}(n) \leq v_{\mathcal{B}}(n)$  for every variable-size sliding window algorithm  $\mathcal{B}$  for  $L$  and every  $n$ .*

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 1. Since any algorithm in the variable-size model yields an algorithm in the fixed-size model, we have  $F_L(n) \leq V_L(n)$ .

### 3 Sliding windows over context-free languages: below logspace

The goal of this section is to prove the following result:

► **Theorem 2.** *If  $L$  is a context-free language with  $F_L(n) \in \log_2(n) - \omega(1)$ , then  $L$  is regular and  $F_L(n) \in \mathcal{O}(1)$ .*

We start with some definitions. A language  $L \subseteq \Sigma^*$  is  *$k$ -suffix testable* if it is a finite Boolean combination of languages of the form  $\Sigma^*w$  where  $w \in \Sigma^{\leq k}$ . An equivalent condition is: for all  $x, y, z \in \Sigma^*$  with  $|z| = k$  we have  $xz \in L$  if and only if  $yz \in L$ . We call  $L$  *suffix testable* if it is  $k$ -suffix testable for some  $k$ . Note that every suffix testable language is regular. Let  $f: \mathbb{N} \rightarrow \mathbb{N}$  be a function. A language  $L \subseteq \Sigma^*$  is  *$f$ -suffix definable* if for all  $n \in \mathbb{N}$  and words  $u, v, w \in \Sigma^*$  such that  $|uw| = |vw| = n$  and  $|w| = f(n)$  we have  $uw \in L$  if and only if  $vw \in L$ . Similarly, one defines prefix testable and  $f$ -prefix definable languages. A *length language* is a language  $L \subseteq \Sigma^*$  such that for every  $n \geq 0$ , either  $\Sigma^n \subseteq L$  or  $L \cap \Sigma^n = \emptyset$ . We prove Theorem 2 in two steps:

► **Theorem 3.** *Every language  $L \subseteq \Sigma^*$  is  $(2^{F_L(n)+1} - 1)$ -suffix definable.*

► **Theorem 4.** *If a context-free language  $L$  is  $f$ -suffix definable for a function  $f(n) \in o(n)$ , then  $L$  is a finite Boolean combination of suffix testable languages and regular length languages.*

Combining Theorem 3 and 4 yields Theorem 2: If a context-free language  $L$  satisfies  $F_L(n) \in \log_2(n) - \omega(1)$  then  $L$  is  $f$ -suffix definable for a function  $f(n) \in o(n)$  by Theorem 3. Theorem 4 implies that  $L$  is a finite Boolean combination of suffix testable languages and regular length languages. Hence  $L$  is regular and  $F_L(n) \in \mathcal{O}(1)$ . The rest of this section is devoted to the proofs of Theorem 3 and 4.

#### 3.1 Proof of Theorem 3

For two languages  $L_1$  and  $L_2$  we define their distance  $d(L_1, L_2)$  as follows: If  $L_1 = L_2$ , then we set  $d(L_1, L_2) = 0$ , and otherwise  $d(L_1, L_2) = \sup\{|u| : u \in L_1 \Delta L_2\} + 1$  where  $L_1 \Delta L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$  denotes the symmetric difference of  $L_1$  and  $L_2$ . Notice that  $d(L_1, L_2) < \infty$  if and only if  $L_1 \Delta L_2$  is finite. If  $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$  is a DFA, we define the distance between two states  $p, q \in Q$  by  $d(p, q) = d(L(\mathcal{A}, p), L(\mathcal{A}, q))$ . We will use a result from [5, Lemma 1] stating that  $d(p, q) < \infty$  implies that  $d(p, q) \leq |Q|$ .

► **Lemma 5.** *Let  $L \subseteq \Sigma^*$  be regular and  $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$  be its minimal DFA. We have:*

- (i)  $d(p, q) \leq k$  if and only if  $\delta(p, z) = \delta(q, z)$  for all  $p, q \in Q$  and  $z \in \Sigma^k$ .
- (ii)  $L$  is  $k$ -suffix testable if and only if  $d(p, q) \leq k$  for all  $p, q \in Q$ .
- (iii) If there exists  $k \geq 0$  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  $d(p, q) = 0$  iff  $p = q$ , which is true because  $\mathcal{A}$  is minimal. For the induction step, we have  $d(p, q) \leq k + 1$  iff  $d(\delta(p, a), \delta(q, a)) \leq k$  for all  $a \in \Sigma$  iff  $\delta(p, z) = \delta(q, z)$  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  $z \in L(\mathcal{A}, p) \Delta L(\mathcal{A}, q)$ , then  $|z| < k$  because  $xz \in L$  iff  $yz \notin L$  and  $L$  is  $k$ -suffix testable. Now assume that  $d(p, q) \leq k$  for all  $p, q \in Q$  and consider  $x, y \in \Sigma^*$ ,  $z \in \Sigma^k$ . Since

$d(\mathcal{A}(x), \mathcal{A}(y)) \leq k$ , (i) implies  $\mathcal{A}(xz) = \mathcal{A}(yz)$ , and in particular  $xz \in L$  iff  $yz \in L$ . Therefore,  $L$  is  $k$ -suffix testable.

Point (iii) follows from (ii) and the above mentioned result from [5, Lemma 1].  $\blacktriangleleft$

**Proof of Theorem 3.** Let  $n \geq 0$  and  $L_n = \{w \in \Sigma^* : \text{last}_n(w) \in L\}$ . Let  $\mathcal{A}_n$  be the minimal DFA for  $L_n$ , which has at most  $f(n) := 2^{F_L(n)+1} - 1$  states. Since  $\text{last}_n(xy) = y$  for all  $x \in \Sigma^*$  and  $y \in \Sigma^n$ , the language  $L_n$  is  $n$ -suffix testable. Therefore  $L_n$  is  $f(n)$ -suffix testable by Lemma 5(iii). This implies that  $L$  is  $f$ -suffix definable because for all words  $u, v, w \in \Sigma^*$  such that  $|uw| = |vw| = n$  and  $|w| = f(n)$  we have  $uw \in L$  iff  $uw \in L_n$  iff  $vw \in L_n$  iff  $vw \in L$ .  $\blacktriangleleft$

### 3.2 Proof of Theorem 4

We prove the variant of Theorem 4 that talks about prefix-definability. This makes no difference, since the reversal of a context-free languages is again context-free. Also note that the requirement  $f(n) \in o(n)$  in Theorem 4 cannot be relaxed: For every  $k \geq 1$ , the language  $\{xay : x, y \in \{a, b\}^*, |x| = k|ay|\}$  is context-free and  $\lceil n/(k+1) \rceil$ -suffix definable but not even regular.

First, we show that in the proof of Theorem 4 we can restrict ourselves to functions  $f$  with the following property: A monotonic function  $f: \mathbb{N} \rightarrow \mathbb{N}$  has the *increasing plateau property* if for every  $k \geq 1$  there exists an  $n_0$  such that for all  $n \geq n_0$  we have:  $f(n+k) - f(n) \leq 1$ . Clearly, if  $f$  has the increasing plateau property then  $f \in o(n)$ .

**► Lemma 6.** *Let  $f: \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ . If  $f(n) \in o(n)$  then there exists a monotonic function  $g: \mathbb{N} \rightarrow \mathbb{N}$  with the increasing plateau property and such that  $f(n) \leq g(n)$  for all  $n \in \mathbb{N}$ .*

**Proof.** For a linear function  $g: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$  of the form  $g(x) = \alpha \cdot x + \beta$  we call  $\alpha$  the *slope* of  $g$ . We will first define a sequence of natural numbers  $n_1 < n_2 < n_3 \cdots$  such that  $f$  is bounded by a continuous piecewise linear function  $h: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$  that has slope  $1/i$  on the interval  $[n_i, n_{i+1}]$  and slope 0 on the interval  $[0, n_1]$ . Then we show that  $g: \mathbb{N} \rightarrow \mathbb{N}$  with  $g(n) = \lceil h(n) \rceil$  has the properties from the lemma.

First, for every  $i \geq 1$  we define  $n_i \in \mathbb{N}$  and a linear function  $h_i: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$  of slope  $1/i$  such that: (i)  $n_{i+1} > n_i$ , (ii) for all natural numbers  $n \geq n_i$  we have  $f(n) \leq h_i(n)$ , and (iii)  $h_i(n_{i+1}) = h_{i+1}(n_{i+1})$ .

Let  $n_1 \geq 0$  be the smallest natural number such that  $f(n) \leq n$  for  $n \geq n_1$  and  $f(n) \leq n_1$  for  $n < n_1$ . Clearly such an  $n_1$  exists, as  $f(n) \in o(n)$ . Define  $h_1$  by  $h_1(x) = x$  for all  $x \in \mathbb{R}_{\geq 0}$ . Hence, we have  $f(n) \leq h_1(n)$  for all  $n \geq n_1$ .

For the induction step, assume that  $n_i$  and the linear function  $h_i: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$  (of slope  $1/i$ ) are defined such that  $f(n) \leq h_i(n)$  for all  $n \geq n_i$ . Define the linear function  $u_{i+1}(x) = h_i(n_i) + (x - n_i)/(i + 1)$ , which has a slope  $1/(i + 1)$  and  $u_{i+1}(n_i) = h_i(n_i)$ . Then there is a smallest natural  $n_{i+1}$  such that  $n_{i+1} > n_i$  and  $u_{i+1}(n) \geq f(n)$  for each  $n \geq n_{i+1}$ . This holds because  $f(n) \in o(n)$ , and hence for any constants  $\alpha > 0, \beta \in \mathbb{R}$  we have  $f(n) \leq \alpha \cdot n + \beta$  for large enough  $n$ . Take this  $n_{i+1}$  and define the function  $h_{i+1}$  by  $h_{i+1}(x) = h_i(n_{i+1}) + (x - n_{i+1})/(i + 1)$ . It has slope  $1/(i + 1)$  and satisfies  $h_{i+1}(n_{i+1}) = h_i(n_{i+1})$ . Finally, for all  $n \geq n_{i+1}$  we have

$$\begin{aligned} h_{i+1}(n) &= h_i(n_{i+1}) + (n - n_{i+1})/(i + 1) \\ &= h_i(n_i) + (n_{i+1} - n_i)/i + (n - n_{i+1})/(i + 1) \\ &\geq h_i(n_i) + (n_{i+1} - n_i)/(i + 1) + (n - n_{i+1})/(i + 1) \\ &= h_i(n_i) + (n - n_i)/(i + 1) = u_{i+1}(n) \geq f(n). \end{aligned}$$

Hence,  $n_{i+1}$  and  $h_{i+1}$  have all the desired properties.

We now define the function  $h : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ :

$$h(x) = \begin{cases} n_1 & \text{if } x \in [0, n_1] \\ h_i(x) & \text{if } x \in [n_i, n_{i+1}] \text{ for some } i \geq 1. \end{cases}$$

Since  $h_i(n_{i+1}) = h_{i+1}(n_{i+1})$  and  $h_1(n_1) = n_1$ ,  $h$  is uniquely defined. Finally, let  $g(n) = \lceil h(n) \rceil$  for all  $n \in \mathbb{N}$ .

Since  $f(n) \leq h_i(n)$  for all  $n \geq n_i$  and  $f(n) \leq f(n_1) \leq n_1$  for all  $n \leq n_1$ , we have  $f(n) \leq h(n) \leq g(n)$  for all  $n \in \mathbb{N}$ . Moreover,  $h$  is clearly monotonic, which implies that  $g$  is monotonic, too. It remains to show that  $g$  has the increasing plateau property.

Let  $k \geq 1$  and  $n \geq n_k$ . Since  $h$  is continuous and piecewise linear with slopes  $\leq 1/k$  on  $[n_k, \infty)$ , we have  $h(n+k) - h(n) \leq (n+k-n)/k = 1$ . This implies  $g(n+k) - g(n) \leq 1$ . ◀

► **Lemma 7.** *Let  $L$  be a context-free language and  $f : \mathbb{N} \rightarrow \mathbb{N} \setminus \{0\}$  be monotonic with  $f(n) \in o(n)$ . There are constants  $n_0$  and  $c > 0$  (only depending on  $L$  and  $f$ ) such that the following hold for every  $n \geq n_0$ :*

- $n \geq f(n) + c$  and
- for all words  $u, v$  with  $uv \in L$ ,  $|uv| = n$ ,  $|u| = f(n)$ , and  $|v| = n - f(n)$ , there exist words  $v', v''$  with  $|v'| = |v| - c$ ,  $|v''| = |v| + c$ , and  $uv', uv'' \in L$ .

**Proof.** Consider the following variant of the pumping lemma for context-free languages (see also [7, Chapter 6.1]), which simultaneously considers all languages defined by various nonterminals of the grammar; it can be shown in the same way as the standard variant:

Given a context-free grammar  $G$  with set of nonterminals  $N$  and productions  $P$ , let  $L_A$  denote the language generated by the grammar  $G_A$  with productions  $P$  and the start nonterminal  $A$ . Then there exists a natural number  $c_1$  depending only on  $G$  and not on  $A$ , such that if  $w \in L_A$  and  $|w| \geq c_1$ , then  $w$  can be written as  $w = w_1 w_2 w_3 w_4 w_5$  with:  $w_1 w_2^k w_3 w_4^k w_5 \in L_A$  for every  $k \geq 0$ ,  $|w_2 w_3 w_4| \leq c_1$  and  $|w_2 w_4| > 0$ . In particular, the word  $w_1 w_2^{1+c_1! / |w_2 w_4|} w_3 w_4^{1+c_1! / |w_2 w_4|} w_5$  of length  $|w| + c_1!$  belongs to  $L_A$ .

Let  $G$  be a grammar for  $L$  in Greibach normal form, i.e., all productions are of the form  $A \rightarrow aA_1 \cdots A_k$  for  $k \geq 0$ , nonterminals  $A, A_1, \dots, A_k$  and a terminal  $a$  (such a grammar exists for every context-free language); see also [7, Chapter 4.6]. Let  $r$  be the maximal length of the productions' right-hand sides in  $G$ , let  $N$  be the set of nonterminals, and let  $c_1$  be the constant from the above pumping lemma for  $G$ . We can assume that  $r \geq 2$ , otherwise  $L$  is finite and the lemma holds. Define  $c = c_1!$  and choose an  $n_0$  such that for all  $n \geq n_0$  the following three inequalities hold:

$$\frac{n}{f(n)} > 1 + (r-1)r^{2|N| \cdot (rc_1|N|+1)!} \quad (1)$$

$$\frac{n}{f(n)} > 1 + c_1(r-1) \quad (2)$$

$$n > f(n) + c$$

As the right-hand sides are constant and  $f(n) \in o(n)$ , such an  $n_0$  exists. Hence, for all  $n \geq n_0$  the following two inequalities hold ((1) is equivalent to (3) and (2) is equivalent to (4)):

$$\log_r \left( \frac{n - f(n)}{f(n)(r-1)} \right) > 2|N|(rc_1|N|+1)! \quad (3)$$

$$\frac{n - f(n)}{f(n)(r-1)} > c_1 \quad (4)$$



Consider a string  $uv$  of length  $n \geq n_0$  generated by  $G$ , where  $|u| = f(n)$ . Fix a leftmost derivation of  $uv$  and consider the first moment, at which the current sentential form has  $u$  as a prefix. This happens after  $|u| = f(n)$  derivation steps since  $G$  is in Greibach normal form. Apart from the prefix  $u$ , the rest of the sentential form has length at most  $1 + f(n)(r - 2) \leq f(n)(r - 1)$  and it derives the word  $v$  of length  $n - f(n)$ . So one of the nonterminals in the sentential form, say  $A$ , generates a word  $x$  with

$$|x| \geq \frac{n - f(n)}{f(n)(r - 1)} \stackrel{(4)}{>} c_1 . \quad (5)$$

The further analysis splits into several cases depending on the claim we want to prove.

We first show the second claim of the lemma, that there exists  $v''$  such that  $|v''| = |v| + c$  and  $uv'' \in L(G)$ . Since  $|x| \geq c_1$ , we can apply the pumping lemma and replace in the derivation of  $uv$  the word  $x$  by a word of length  $|x| + c_1! = |x| + c$ . The resulting derivation yields a word  $uv''$  with  $|v''| = |v| + c$ , as claimed.

So let us now prove that there is  $v'$  such that  $uv' \in L(G)$ , where  $|v'| = |v| - c$ . Again, consider the nonterminal  $A$  that generates a string  $x$  satisfying (5). Since the length of each right-hand side is at most  $r$ , there is a path  $\Pi$  in the derivation tree of length at least

$$\log_r \left( \frac{n - f(n)}{f(n)(r - 1)} \right) > 2|N| \cdot (rc_1|N| + 1) ! ,$$

where the estimation follows from (3). We are going to color some nodes on the path  $\Pi$  black or grey: if a node  $v$  on  $\Pi$  has a child that does not belong to  $\Pi$  and derives a string of length at least  $c_1$ , then we color  $v$  black. Then, as long as there are two uncolored nodes  $v, v'$  on  $\Pi$  ( $v$  above  $v'$ ) such that (i)  $v$  and  $v'$  are labelled with the same nonterminal, (ii) the path from  $v$  to  $v'$  has length at most  $|N|$ , and (iii) does not contain a black node, then we color  $v$  black and  $v'$  grey.

There can be at most  $|N|$  consecutive nodes on the path that are not colored and there are at least as many black nodes as grey nodes. Thus, the number of black nodes is at least

$$\left\lfloor \frac{1}{2} \left\lfloor \frac{2|N| \cdot (rc_1|N| + 1)!}{|N|} \right\rfloor \right\rfloor = (rc_1|N| + 1) !$$

For each black node we can shorten the derivation such that the derived word is shorter by at least 1 and at most  $rc_1|N|$  without affecting other colored nodes:

- For the first type of black nodes this follows directly from the pumping lemma. Note that we can apply the pumping lemma to a subtree that is disjoint from  $\Pi$ .
- For the second kind of black nodes, let  $v$  and  $v'$  be the corresponding nodes colored black and grey, respectively. We can delete the subtree rooted in  $v$  and replace it with the one rooted in  $v'$ . The length of the path is  $\leq |N|$ , the arity of the rules  $\leq r$  and each deleted nonterminal derived a string of length  $\leq c_1$  (otherwise it would be black).

So for some  $m \in \{1, 2, \dots, rc_1|N|\}$  there are  $\frac{(rc_1|N|+1)!}{rc_1|N|} > (rc_1|N|)!$  different possibilities to shorten the derived word by  $m$  letters. We choose  $(rc_1|N|)!/m$  of them so that the word is shortened by  $(rc_1|N|)!$  letters. Thus we showed that there exists  $v'$  such that  $uv' \in L(G)$  and  $|uv'| = n - (rc_1|N|)!$ . As  $c = c_1!$  divides  $(rc_1|N|)!$ , by applying  $((rc_1|N|)!/c - 1)$  times the already proved second claim of the lemma we can first obtain a word  $uz \in L(G)$  such that  $|uz| = n + (rc_1|N|)! - c$  and then use the argument above to obtain a word  $uv' \in L(G)$  such that  $|uv'| = |uz| - (rc_1|N|)! = |uv| - c$ . Here, we use monotonicity of  $f$ , which ensures that the prefix  $u$  is not touched when using the above argument for the longer word  $uz$ . ◀



► **Lemma 8.** *Let  $L$  be a context-free language that is  $f$ -prefix definable for a function  $f(n) \in o(n)$ . Then there exists a constant  $\alpha$  such that for every word  $u$  of length  $\alpha$  and all words  $v, w$  with  $|v| = |w|$  we have  $uv \in L$  if and only if  $uw \in L$ .*

**Proof.** By Lemma 6 there exists a monotonic function  $g(n) \in o(n)$  having the increasing plateau property and such that  $f(n) \leq g(n)$  for all  $n \geq 0$ . Hence,  $L$  is still  $g$ -prefix definable. Moreover, let  $f' \in o(n)$  be defined by  $f'(n) = g(n) + 1$  for all  $n$ . Take the constants  $n_0$  and  $c$  from Lemma 7 for  $L$  and  $f'$  (instead of  $f$ ). Choose  $m$  such that (i)  $m \geq n_0 + c$  and (ii)  $g(n) - g(n - c) \leq 1$  for all  $n \geq m$ , which is possible by the increasing plateau property. We take  $\alpha = g(m)$ . Heading for a contradiction, let us take words  $u, v, w$  such that  $|u| = \alpha$ ,  $|v| = |w|$ ,  $uv \in L$  and  $uw \notin L$ . We can assume that  $|v| = |w|$  is minimal with these properties. Let  $n = |uv| = |uw|$  in the following. We now distinguish two cases.

**Case 1.**  $n \leq m$ , which implies  $g(n) \leq g(m) = |u|$ . Hence,  $uv$  and  $uw$  have the same prefix of length  $g(n)$ . Since  $L$  is  $g$ -prefix definable, we have  $uv \in L$  iff  $uw \in L$ , which is a contradiction.

**Case 2.**  $n > m$ , and thus  $n > n_0 + c$  and  $g(n) \geq g(m) = |u|$ . Since  $n - g(m) \geq n - g(n) = n - f'(n) + 1 > c > 0$ , we can write  $v = v_1av_2$  and  $w = w_1bw_2$  such that  $a, b \in \Sigma$  and  $|uv_1| = |uw_1| = g(n)$ . Thus,  $|uv_1a| = |uw_1b| = f'(n)$ . By Lemma 7 there exists a word  $v'_2$  with  $|v'_2| = |v_2| - c$  and  $uv_1av'_2 \in L$ . Take any word  $w'_2$  of length  $|w'_2| = |w_2| - c$ . By the length-minimality of  $v$  and  $w$  we must have  $uw_1bw'_2 \in L$  (note that  $c > 0$ ). Note that  $|uw_1bw'_2| = |uw| - c = n - c > n_0$ . Therefore, we can apply Lemma 7 to the word  $uw_1bw'_2$ . Note that  $g(n) - g(n - c) \leq 1$  since  $n \geq m$ . Thus,  $f'(n - c) = g(n - c) + 1 \geq g(n)$  and the prefix of  $uw_1bw'_2$  of length  $f'(n - c)$  starts with  $uw_1$ . We can conclude with Lemma 7 that there is a word  $w''_2$  such that  $uw_1w''_2 \in L$  and  $|uw_1w''_2| = n$ . But since  $|uw_1| = g(n)$  and  $|uw_1w''_2| = n$ , the  $g$ -prefix definability of  $L$  implies that  $uw_1y \in L$  for all words  $y$  of length  $n - g(n)$ . In particular, we get  $uw_1bw_2 = uw \in L$ , which is a contradiction. ◀

We can now prove (the prefix version of) Theorem 4:

**Proof of Theorem 4.** Let  $L$  be a  $f$ -prefix definable context-free language with  $f(n) \in o(n)$ . Let  $\alpha$  be the constant from Lemma 8. For every word  $u$  of length  $\alpha$  let  $L_u = \{w : uw \in L\}$ . Each of these finitely many languages is context-free and by Lemma 8 it is a length language. It is a direct consequence of Parikh's theorem [8] (or the fact that every unary context-free language is regular) that a context-free length language is regular. Hence, every  $L_u$  (for  $|u| = \alpha$ ) is a regular length language. We can now decompose  $L$  as follows:

$$L = (L \cap \Sigma^{<\alpha}) \cup \bigcup_{u \in \Sigma^\alpha} uL_u = (L \cap \Sigma^{<\alpha}) \cup \bigcup_{u \in \Sigma^\alpha} (u\Sigma^* \cap \Sigma^\alpha L_u).$$

Since  $L_u$  is a regular length language, also  $\Sigma^\alpha L_u$  is a regular length language. Moreover,  $u\Sigma^*$  is prefix testable. Finally, every finite language (and hence  $L \cap \Sigma^{<\alpha}$ ) is a finite Boolean combination of prefix testable languages. This shows the theorem. ◀

## 4 Sliding windows over context-free languages: above logspace

In this section, we show that the trichotomy result for regular languages [4] does not carry over the context-free languages. More precisely, we show that for every natural number  $c \geq 1$  there exists a one-counter language  $L_c$  such that  $F_{L_c}(n) \in \mathcal{O}(n^{1/c}) \setminus o(n^{1/c})$  and  $V_{L_c}(n) \in \Theta(n^{1/c})$ . Recall that a one-counter language is a language that can be accepted by a nondeterministic pushdown automaton with a singleton pushdown alphabet (a so called

one-counter automaton). Also recall that a linear bounded automaton (LBA for short) is a Turing machine that only uses the space that is occupied by the input word; see also [7, Chapter 9.3]. We first show the following technical result:

► **Theorem 9.** *Let  $t(k)$  be a monotonically increasing function and  $M$  be an LBA which halts on input  $a^k$  after exactly  $t(k)$  steps. Let  $f(n)$  be the function with<sup>1</sup>*

$$f(n) = \begin{cases} k & \text{if } n = k(t(k) + 3) \text{ for some } k \\ 0 & \text{else} \end{cases}$$

and let  $g(n) = \max\{f(m) : m \leq n\}$ . *There is a one-counter language  $L$  such that  $F_L(n) \in \Theta(f(n))$  and  $V_L(n) \in \Theta(g(n))$ .*

**Proof.** Let  $\Gamma$  be the tape alphabet of  $M$  and  $Q$  the set of states of  $M$ . A configuration of  $M$  is encoded by a word from  $\Gamma^*(Q \times \Gamma)\Gamma^*$  over the alphabet  $\Delta := \Gamma \cup (Q \times \Gamma)$ . A computation of  $M$  on an input  $a^k$  ( $k \geq 1$ ) is a sequence of configurations  $c_0 \vdash_M \cdots \vdash_M c_{t(k)}$  where  $|c_i| = k$  for all  $1 \leq i \leq t(k)$ ,  $c_0 = (q_0, a)a^{k-1}$  is the start configuration on input  $a^k$ , every  $c_{i+1}$  is obtained from  $c_i$  by applying a transition of  $M$  for  $0 \leq i \leq t(k) - 1$ , and  $c_{t(k)}$  is an accepting computation. Let  $\Delta' = \{x' \mid x \in \Delta\}$  be a disjoint copy of  $\Delta$  and define  $w'$  for a word  $w \in \Delta^*$  by applying the homomorphism  $x \mapsto x'$  ( $x \in \Delta$ ) to  $w$ . Finally, let  $K$  be the set of all words

$$c_0 \text{ rev}(c_1)' c_2 \text{ rev}(c_3)' \cdots c_{t(k)} s \text{ rev}(s) \text{ or} \quad (6)$$

$$c_0 \text{ rev}(c_1)' c_2 \text{ rev}(c_3)' \cdots \text{rev}(c_{t(k)})' s \text{ rev}(s) \quad (7)$$

such that  $k \geq 1$ ,  $c_0 \vdash_M \cdots \vdash_M c_{t(k)}$  is a computation on input  $a^k$ ,  $s \in \{0, 1\}^*$  is an arbitrary word of length  $k$  (0 and 1 are arbitrary symbols not in  $\Delta \cup \Delta'$ ), and  $t(k)$  even (resp., odd) in case (6) (resp., (7)). Notice that the words in (6) and (7) have length  $k(t(k) + 3)$ . We can assume that  $M$  never goes back to the initial state  $q_0$ . This ensures that every word has at most one non-empty suffix that is a prefix of a word from  $K$ .

For the language  $L$  from the theorem, we take the complement of  $K$ . It is not hard to see that  $L$  can be recognized by a nondeterministic one-counter automaton by guessing an error in the input word  $w$ . Possible errors are the following, where we call a block of  $w$  a maximal factor from  $\Delta^+ \cup (\Delta')^+ \cup \{0, 1\}^+$  in  $w$ ,  $m$  is the number of blocks of  $w$  and  $u_i$  denotes the  $i$ -th block of  $w$  for  $1 \leq i \leq m$ :

1.  $m < 2$ ,
2.  $u_1$  is not an initial configuration, i.e., of the form  $(q_0, a)a^{k-1}$  for some  $k \geq 1$ ,
3. for some odd  $i < m$ ,  $u_i$  is not a configuration,
4. for some even  $i < m$ ,  $u_i$  is not of the form  $c'$  for a configuration  $c$ ,
5.  $u_{m-1}$  is not an accepting configuration,
6. there exists  $1 \leq i < m - 1$  with  $|u_i| \neq |u_{i+1}|$ ,
7.  $|u_m| \neq 2|u_{m-1}|$ ,
8. there exists  $1 \leq i < m - 1$  odd such that  $u_i \vdash_M \text{rev}(u)$  does not hold for  $u' = u_{i+1}$ ,
9. there exists  $1 \leq i < m - 1$  even such that  $\text{rev}(u) \vdash_M u_{i+1}$  does not hold for  $u' = u_i$ ,
10.  $u_m$  is not a palindrome over the alphabet  $\{0, 1\}^*$ .

The conditions in points 1–5 are regular. Points 6–10 can be checked with a single counter.

The upper bound in the theorem has to be shown for the variable-size model. Since  $F_K(n) = F_L(n)$  and  $V_K(n) = V_L(n)$ , it is enough to show the bounds for the language  $K$ . Let us first present a variable-size streaming algorithm with space complexity  $\mathcal{O}(g(n))$ . Assume that  $w$  is the active window. The algorithm stores the following data  $n, i, t, k, \ell, s$ :

<sup>1</sup> Since  $t(k)$  is monotonically increasing, the number  $k$  in the first case is unique.

- $n = |w|$  is the length of the active window.
- $i$  is the smallest position  $x$  such that  $w[x:]$  is a prefix of a word from  $K$ . If this prefix is empty, then  $i = n + 1$ .
- $t$  is the number of blocks in  $w[i:]$  minus 1 (where  $i$  is from the previous point); this tells us the number of computation steps that  $M$  has executed.
- $k$  is the largest number  $y$  such that  $w[i:]$  starts with  $(q_0, a)a^{y-1}$ ; hence,  $k$  tells us the input length.
- In case  $1 \leq t \leq t(k)$ ,  $\ell$  is the length of the last block of  $w[i:]$  (if  $t = 0$  or  $t = t(k) + 1$  we store some dummy value in  $\ell$ ).
- In case  $t = t(k) + 1$ ,  $s$  is the maximal suffix of  $w[i:]$  from  $\{0, 1\}^*$ . If the length of this suffix exceeds  $k$  then  $s$  stores only its prefix of length  $k$ .

It is easy to see that these variables can be updated. The main observation is that in case  $1 \leq t \leq t(k)$  and  $\ell < k$  then the algorithm internally simulates  $M$  for  $t$  steps on input  $a^k$ . In this way, the algorithm can check whether the arriving symbol is the right one, namely the (possibly primed)  $(\ell + 1)$ -th symbol of the configuration reached after  $t$  steps on input  $a^k$ . In this case, the algorithm sets  $\ell := \ell + 1$ , otherwise the algorithm sets  $i := n + 1$ . If  $t$  is set to  $t(k) + 1$  then the algorithm starts to accumulate the window suffix  $s \in \{0, 1\}^*$  up to length  $k$ . If  $s$  has length  $k$  then the next  $k$  arriving symbols are compared in reversed order with  $s$ . If a match is obtained, the algorithm accepts if  $i = 1$  at the same time.

Let us now compute the space complexity of the algorithm. The numbers  $n$ ,  $i$ ,  $t$ ,  $k$  and  $\ell$  need  $\mathcal{O}(\log n)$  bits. Recall that  $s$  has maximal length  $k$ . But we only store symbols in  $s$  if  $n \geq k(t(k) + 1) \geq \lfloor k/3 \rfloor (t(\lfloor k/3 \rfloor) + 3)$ , since the window must already contain a complete computation on input  $a^k$  before  $s$  becomes non-empty. We get  $\lfloor k/3 \rfloor = f(\lfloor k/3 \rfloor (t(\lfloor k/3 \rfloor) + 3)) \leq g(n)$ , i.e.,  $k \leq 3g(n) + 3$ . Finally, since  $g(n)$  is the maximal value  $k$  such that  $k(t(k) + 3) \leq n$  and  $t(k) \in 2^{\mathcal{O}(k)}$ , we get  $g(n) \in \Omega(\log n)$ . This shows that the algorithm works in space  $\mathcal{O}(g(n))$ .

To show that  $F_K(n) \in \mathcal{O}(f(n))$  we can argue similarly. Of course, in the fixed-size model, we do not have to store the window size  $n$ . If the window size  $n$  is not of the form  $k(t(k) + 3)$  for some  $k$  then the algorithm always rejects and no space at all is needed. Otherwise, since  $t(k)$  is monotonically increasing, there is a unique  $k$  with  $n = k(t(k) + 3)$ .

Finally, we show that  $F_K(n) \geq f(n)$  for all  $n$ , which implies that  $V_K(n) \geq g(n)$  for all  $n$  since  $V_K(n) \geq F_K(n)$  and  $V_K(n)$  is monotonic. It suffices to consider a window size  $n = k(t(k) + 3)$  for some  $k$ , as otherwise  $f(n) = 0$ . Hence,  $f(n) = k$ . Moreover, consider an accepting computation  $c_0 \vdash_M c_1 \vdash_M \cdots \vdash_M c_{t(k)}$  where  $|c_0| = \cdots = |c_{t(k)}| = k$ . Let us assume that  $k$  is even; the case that  $k$  is odd is analogous. Now consider the  $2^k$  many distinct words  $w(s) := 0^k c_0 \text{rev}(c_1)' c_2 \text{rev}(c_3)' \cdots c_{t(k)}$  for  $s \in \{0, 1\}^k$ . The length of these words is  $n = k(t(k) + 3)$ , which is the window size.

Consider now the minimal DFA  $\mathcal{A}_n$  for the language  $K_n$ , and let  $r$  be the number of states of  $\mathcal{A}_n$  (hence,  $F_K(n) = \lfloor \log_2 r \rfloor$ ). We claim that  $\mathcal{A}_n(w(s)) \neq \mathcal{A}_n(w(u))$  for all  $s, u \in \{0, 1\}^k$  with  $s \neq u$ . To see this, assume that  $\mathcal{A}_n(w(s)) = \mathcal{A}_n(w(u))$  for  $s, u \in \{0, 1\}^k$  with  $s \neq u$ . Hence,  $\mathcal{A}_n(w(s) \text{rev}(s)) = \mathcal{A}_n(w(u) \text{rev}(s))$ . On the other hand, the above definition of  $w(s)$  and  $w(u)$  implies  $w(s) \text{rev}(s) \in K$  and  $w(u) \text{rev}(s) \notin K$ , which yields a contradiction. We get  $r \geq 2^k$ , and thus  $F_K(n) \geq k = f(n)$ . ◀

Theorem 9 yields a quite dense spectrum of space complexity functions for context-free languages. We only prove the existence of context-free languages with sliding-window space complexity  $n^{1/c}$  for  $c \in \mathbb{N}$ ,  $c \geq 1$ :

► **Theorem 10.** *For every  $c \in \mathbb{N}$ ,  $c \geq 1$ , there exists a one-counter language  $L_c$  such that  $F_{L_c}(n) \in \mathcal{O}(n^{1/c}) \setminus o(n^{1/c})$  and  $V_{L_c}(n) \in \Theta(n^{1/c})$ .*

**Proof.** One can easily construct a deterministic LBA  $M$  that on input  $a^k$  terminates after exactly  $k^{c-1}$  steps. For instance, an LBA that terminates after exactly  $k^2$  steps makes  $k$  phases, where in each phase the read-write head moves from the left input end to the right end or vice versa and thereby replaces the first  $a$  that is seen on the tape by a  $b$ -symbol. This construction can be iterated to obtain the above LBA  $M$  for an arbitrary  $k$ . The mapping  $f(n)$  from Theorem 9 then satisfies  $f(k(t(k) + 3)) = f(k(k^{c-1} + 3)) = f(k^c + 3k) = k$  and  $f(n) = 0$  if  $n$  is not of the form  $k^c + 3k$  for some  $k$ . This implies  $f(n) \in \mathcal{O}(n^{1/c}) \setminus o(n^{1/c})$  and  $g(n) \in \Theta(n^{1/c})$  for the mapping  $g(n)$  from Theorem 9. Hence, by Theorem 9 there is a one-counter language  $L_c$  with the properties stated in the theorem. ◀

To fully exploit Theorem 9 one would have to analyze the spectrum of time complexity functions of linear bounded automata. We are not aware of specific results in this direction.

## 5 Sliding windows over deterministic one-counter languages

The context-free language  $L_c$  from Theorem 10 is not deterministic context-free and it is open whether the same result can be obtained for deterministic context-free languages. In this section we exhibit a deterministic one-turn one-counter language with space complexity  $\Theta((\log n)^2)$  in the variable-size (resp., fixed-size) model. A  $t$ -turn pushdown automaton has the property that in any accepting run there are at most  $t$  alternations between push and pop operations [6].

We start with the variable-size model. A maximal factor  $\beta$  in a word  $w \in \{a, b\}^*$  of the form  $\beta = ab^i$  is called a *block* of length  $i + 1$  in  $w$  (this notion is not related to the blocks used in the proof of Theorem 9). Define the language

$$L = \{ab^kav : k \geq 0, v \in \{a, b\}^{\leq k}\} \cup ab^*, \quad (8)$$

which is recognized by a deterministic one-turn one-counter automaton. Put differently,  $L$  contains those words  $w \in \{a, b\}^*$  which begin with a block of length  $\geq |w|/2$ .

► **Lemma 11.** *We have  $V_L(n) \in \mathcal{O}((\log n)^2)$ .*

**Proof.** Any word  $w \in \{a, b\}^*$  can be uniquely factorized as  $w = b^s \beta_m \beta_{m-1} \cdots \beta_2 \beta_1$  where  $s, m \geq 0$  and each  $\beta_i$  is a block. A block  $\beta_i$  is *relevant* if it is at least as long as the remaining suffix, i.e.  $|\beta_i| \geq \sum_{j=1}^{i-1} |\beta_j|$ . For an active window  $w \in \{a, b\}^*$  our variable-size algorithm maintains the window size and for each relevant block  $\beta_i$  its starting position and its length. If the first symbol in the window expires, every relevant block stays relevant (and the starting position is decremented) with the possible exception of a relevant block with starting position 1, which is removed. If an  $a$  arrives, we create a new relevant block of length 1. If a  $b$  arrives, we prolong the rightmost relevant block (which is also rightmost among all blocks) by 1. Furthermore, using this information we can determine whether  $w \in L$ : This is the case if and only if the leftmost relevant block starts at the first position and its length is at least  $n/2$  where  $n$  is the current window size.

To show that the space complexity of the algorithm is  $\mathcal{O}((\log n)^2)$ , it suffices to show that each word  $w \in \{a, b\}^n$  has  $\mathcal{O}(\log n)$  relevant blocks. Let  $\gamma_k, \gamma_{k-1}, \dots, \gamma_2, \gamma_1$  be the sequence of relevant blocks in  $w$ . Then we know that  $|\gamma_i| \geq \sum_{j=1}^{i-1} |\gamma_j|$  for all  $1 \leq i \leq k$ . Inductively, we show that  $|\gamma_i| \geq 2^{i-2} |\gamma_1|$  for all  $2 \leq i \leq k$ . This is immediate for  $i = 2$  and for the induction step, observe that  $|\gamma_i| \geq |\gamma_1| + \sum_{j=2}^{i-1} |\gamma_j| \geq |\gamma_1| + |\gamma_1| \sum_{j=2}^{i-1} 2^{j-2} = 2^{i-2} |\gamma_1|$  for all  $i \geq 3$ . This proves  $k = \mathcal{O}(\log n)$ , which concludes the proof. ◀

► **Lemma 12.** *We have  $V_L(n) \in \Omega((\log n)^2)$ .*

**Proof.** For each  $k \geq 0$  we define *arrangements* of length  $3^k$ : The word  $a$  is the only arrangement of length  $3^0 = 1$ . An arrangement of length  $3^{k+1}$  is any word of the form  $ub^{3^k}v$  where  $u, v \in \{a, b\}^{3^k}$ ,  $u$  begins with  $a$  and has at most one other  $a$ -symbol and  $v$  is any arrangement of length  $3^k$ . Notice that an arrangement  $ub^{3^k}v$  contains one or two blocks in the factor  $ub^{3^k}$ , one of which is relevant. If  $\alpha_1 \neq \alpha_2$  are distinct arrangements of length  $3^k$ , consider the maximal common suffix  $\alpha_3$  of  $\alpha_1$  and  $\alpha_2$  which is again an arrangement. Consider the suffixes of  $\alpha_1, \alpha_2$  of length  $3|\alpha_3|$ . By the construction, these suffixes are also arrangements. Hence, their “middle parts” consist solely of  $b$ 's, so they have the common suffix  $b^{|\alpha_3|}\alpha_3$ . Since  $\alpha_1$  and  $\alpha_2$  differ, there exists a number  $\ell \geq |\alpha_3|$  such that  $\alpha_1$  has the suffix  $ab^\ell\alpha_3$  and  $\alpha_2$  has the suffix  $b^{\ell+1}\alpha_3$ , or vice versa.

Now consider a variable-size sliding window algorithm for  $L$ . We claim that the algorithm can distinguish any two distinct arrangements  $\alpha_1 \neq \alpha_2$  of length  $3^k$ . Consider two instances of the algorithm, where the active windows are  $\alpha_1$  and  $\alpha_2$ , respectively. By performing a suitable number of  $\downarrow$ -operations the two windows contain the words  $ab^\ell\alpha_3$  and  $b^{\ell+1}\alpha_3$ , respectively. Since  $|\alpha_3| \leq \ell$ , we have  $ab^\ell\alpha_3 \in L$  and  $b^{\ell+1}\alpha_3 \notin L$ .

It is easy to see that the number  $n_k$  of arrangements of length  $3^k$  is exactly  $\prod_{i=0}^{k-1} 3^i$ : to construct an arrangement of length  $3^k$ , note that among its first  $3^{k-1}$  letters the first one is  $a$  and there is at most one further  $a$ . So, there are  $3^{k-1}$  choices for the prefix of length  $3^{k-1}$ . The next  $3^{k-1}$  letters are fixed, and then one of  $n_{k-1}$  many arrangements of length  $3^{k-1}$  follows. Thus  $n_k = 3^{k-1}n_{k-1}$  and  $n_0 = 1$ , which yields the claim. Note that  $\log_3(n_k) = \sum_{i=0}^{k-1} i = \Theta(k^2)$ . Therefore, the algorithm needs  $\Omega((\log n)^2)$  bits of space. ◀

► **Corollary 13.** *There exists a deterministic one-turn one-counter language  $L$  such that  $V_L(n) = \Theta((\log n)^2)$ .*

The language  $L$  from (8) is an example of a language where the space complexity in the fixed-size model is strictly below the space complexity in the variable-size model:

► **Lemma 14.** *We have  $F_L(n) \in \mathcal{O}(\log n)$ .*

**Proof.** Let  $n \geq 0$  be the window size. For the active window we store (i) the starting position of the leftmost block of length at least  $n/2$  (if such a block does not exist we set a special flag) and (ii) the length of the unique suffix from  $ab^*$  (again, a flag is set if the window content is in  $b^*$ ). This information can be stored with  $\mathcal{O}(\log n)$  bits and it can be updated at each step. Moreover, the active window belongs to  $L$  if the leftmost block of length at least  $n/2$  starts at position 1. ◀

We now prove the variant of Corollary 13 for the fixed-size model: For the language  $L$  from (8) let  $K = Lc^*$ , which is a deterministic one-turn one-counter language.

► **Theorem 15.** *We have  $F_K(n) = \Theta((\log n)^2)$ .*

**Proof.** Let  $n$  be the window size. Consider the maximal suffix of the active window which has the form  $vc^i$  where  $v \in \{a, b\}^*$ . Using  $\mathcal{O}(\log n)$  bits we maintain the starting position of that suffix and the length  $|v|$ . Furthermore, we maintain the same data structure as in the proof of Lemma 11 for the word  $v \in \{a, b\}^*$ . The algorithm accepts iff  $v$  begins at the first position, the leftmost relevant block also starts at the first position and has length at least  $|v|/2$ . In total, the space complexity is bounded by  $\mathcal{O}((\log n)^2)$ .

The proof for the lower bound is similar to the proof of Lemma 12. Let  $k$  be maximal such that  $3^k \leq n$ . Then the number of bits required to encode an arrangement of length  $3^k$  is  $\Omega((\log n)^2)$ . The rest of the proof follows the proof of Lemma 12; we only have to replace every  $\downarrow$ -operation by the insertion of a  $c$ -symbol. ◀

For the language  $L$  from (8) let  $L' = \{\#^j \text{rev}(u)v\$^i : u \in L, i \geq 0, v \in \{a, b\}^i, j \geq 1\}$ . Its reversal  $\text{rev}(L') = \{\$^i v \mid i \geq 0, v \in \{a, b\}^i\} L \#^+$  is accepted by a deterministic one-counter three-turn automaton.

► **Theorem 16.** *We have  $V_{L'}(n) = \mathcal{O}((\log n)^2)$  and  $F_{L'}(n) \notin o((\log n)^2)$ .*

**Proof.** We first exhibit a variable-size sliding window algorithm for  $L'$ . Of course, we maintain the window size  $n$ . For the active window consider its longest suffix of the form  $\#^j w \$^i$  where  $w \in \{a, b\}^*$  and  $i, j \geq 0$ . Using  $\mathcal{O}(\log n)$  bits we can maintain the numbers  $i, j$ , the length  $|w|$ , and the maximal number  $k$  such that  $b^k$  is a suffix of  $w$ .

Furthermore, if  $j \geq 1$  we maintain for each relevant block in  $\text{rev}(w)$  its starting position and its length, which requires  $\mathcal{O}((\log n)^2)$  bits (see the proof of Lemma 11). Let us argue why this information can be maintained. Let  $n, i, j, k$  and  $w$  have the meaning from the previous paragraph. If  $j$  is set from 0 to 1, then  $w$  is empty and  $\text{rev}(w)$  contains no blocks. If  $j \geq 1$  we can prolong  $w$  as long as the active window does not end with  $\$$ -symbols ( $i = 0$ ). In this case, every time an  $a$ -symbol arrives, a new block in  $\text{rev}(w)$  is formed, which has length  $k + 1$ . If it is not relevant, then it is immediately discarded. Also notice that when  $w$  is prolonged by  $a$  or  $b$ , all relevant blocks in  $\text{rev}(w)$  stay relevant. A  $\downarrow$ -operation only affects  $w$  if  $j \in \{0, 1\}$  and  $n = j + |w| + i$ . In this case  $j$  is set to zero, and we no longer have to store the relevant blocks of  $w$ .

It remains to show the lower bound. Let the window size  $n$  be of the form  $2 \cdot 3^k$ . Again we show that any fixed-size sliding window algorithm for  $L'$  must distinguish any two distinct arrangements. Let  $\alpha_1 \neq \alpha_2$  be two arrangements of length  $3^k$ . As shown in the proof of Lemma 12, there exists a number  $0 \leq \ell < 3^k$  and an arrangement  $\alpha_3$  of length at most  $\ell$  such that  $\alpha_1$  and  $\alpha_2$  have the suffixes  $ab^\ell \alpha_3 \in L$  and  $b^{\ell+1} \alpha_3 \notin L$ , respectively (or vice versa). Without loss of generality,  $\alpha_1 = v_1 ab^\ell \alpha_3$  and  $\alpha_2 = v_2 b^{\ell+1} \alpha_3$  for some  $v_1, v_2 \in \{a, b\}^*$ . Both words  $v_1$  and  $v_2$  have length  $r := 3^k - (\ell + 1 + |\alpha_3|)$ . We have

$$\begin{aligned} \text{last}_n(\#^{3^k} \text{rev}(\alpha_1)\$^r) &= \#^{3^k-r} \text{rev}(ab^\ell \alpha_3) \text{rev}(v_1)\$^r \in L' \quad \text{and} \\ \text{last}_n(\#^{3^k} \text{rev}(\alpha_2)\$^r) &= \#^{3^k-r} \text{rev}(b^{\ell+1} \alpha_3) \text{rev}(v_2)\$^r \notin L'. \end{aligned}$$

This shows that the algorithm must distinguish the words  $\#^{3^k} \text{rev}(\alpha_1)$  and  $\#^{3^k} \text{rev}(\alpha_2)$ . Note that adding a  $\$$  at the right end of the word removes the right-most symbol ( $a$  or  $b$ ) in the factor which has to belong to  $\text{rev}(L)$  in order to have a word from  $L'$ . The rest of the proof follows the arguments from the proof of Lemma 12. ◀

## 6 Open problems

Our results lead to several open problems; in particular for deterministic context-free languages: Are there deterministic context-free languages where the optimal space bound (for the variable-size or the fixed-size model) is in  $o(n) \cap \omega((\log n)^2)$ ?

An interesting subclass of the deterministic context-free languages are the visibly pushdown languages [2, 9], which are also known as input-driven languages. Visibly pushdown languages have better algorithmic properties than general deterministic context-free languages [2, 9]. Our deterministic context-free languages from Section 5 are not visibly pushdown languages. This leads to the question, whether our space trichotomy result for regular languages [4] extends to visibly pushdown languages (or at least visibly one-counter languages).



---

**References**

---

- 1 Charu C. Aggarwal. *Data Streams - Models and Algorithms*. Springer, 2007.
- 2 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, STOC 2004*, pages 202–211. ACM Press, 2004.
- 3 Moses Ganardi, Danny HucKe, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, volume 96 of *LIPICs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- 4 Moses Ganardi, Danny HucKe, and Markus Lohrey. Querying regular languages over sliding windows. In *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, volume 65 of *LIPICs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- 5 Paweł Gawrychowski and Artur Jež. Hyper-minimisation made efficient. In *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science, MFCS 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 356–368. Springer, 2009.
- 6 Seymour Ginsburg and Edwin H Spanier. Finite-turn pushdown automata. *SIAM Journal on Control*, 4(3):429–453, 1966.
- 7 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison–Wesley, Reading, MA, 1979.
- 8 Rohit Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.
- 9 Burchard von Braunmühl and Rutger Verbeek. Input-driven languages are recognized in  $\log n$  space. In *Proceedings of the 1983 International FCT-Conference, FCT 1983*, volume 158 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 1983.