Querying Regular Languages over Sliding Windows

Moses Ganardi[1], Danny Hucke[2], and Markus Lohrey*[3]

1 **University of Siegen, Germany**
ganardi@eti.uni-siegen.de
2 **University of Siegen, Germany**
hucke@eti.uni-siegen.de
3 **University of Siegen, Germany**
lohrey@eti.uni-siegen.de

Abstract

We study the space complexity of querying regular languages over data streams in the sliding window model. The algorithm has to answer at any point of time whether the content of the sliding window belongs to a fixed regular language. A trichotomy is shown: For every regular language the optimal space requirement is either in $\Theta(n)$, $\Theta(\log n)$, or constant, where n is the size of the sliding window.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.4.3 Formal Languages

Keywords and phrases streaming algorithms, regular languages, space complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2016.18

1 Introduction

Streaming algorithms, i.e. algorithms that process a non-terminating stream $a_1a_2a_3\cdots$ of data values and which have at time t only direct access to the current symbol a_t, received a lot of attention in recent years, see [1] for a general reference. Two variants of streaming algorithms can be found in the literature:

- In the *standard model* the algorithm computes at time t a value $f(a_1\cdots a_t)$ that depends on the whole history.
- In the *sliding window model* the algorithm computes at time t a value $f(a_{t-n+1}\cdots a_t)$ that depends on the n last symbols (we should assume $t \geq n$ here). The value n is also called the *window size*.

For many applications, the sliding window model is more appropriate. Quite often data items in a stream are outdated after a certain time, and the sliding window model is a simple way to model this. The typical application is the analysis of a time series as it may arise in medical monitoring, web tracking, or financial monitoring. In all these applications, the most recent data items are more important than older ones.

A general goal in the area of sliding window algorithms is to avoid the explicit storage of the whole window, and, instead, to work in considerably smaller space, e.g. polylogarithmic space. In the seminal paper of Datar et al. [9], where the sliding window model was introduced, the authors prove that the number of 1's in a 0/1-sliding window of size n can be maintained in space $\frac{1}{\varepsilon}\cdot\log^2 n$ if one allows a multiplicative error of $1\pm\varepsilon$. A matching lower bound is provided as well in [9]. Other algorithmic problems that were addressed in

* M. Lohrey was supported by the DFG-project LO748/11-1.

36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016).
Editors: Akash Lal, S. Akshay, Saket Saurabh, and Sandeep Sen; Article No. 18; pp. 18:1–18:14

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

the extensive literature on sliding window streams include the computation of statistical data (e.g. computation of the variance and k-median [4], and quantiles [3]), optimal sampling from sliding windows [7], database querying (e.g. processing of join queries over sliding windows [12]) and graph problems (e.g. checking for connectivity and computation of matchings, spanners, and minimum spanning trees [8]) . The reader can find further references in the surveys [1, Chapter 8] and [6]. Another natural problem, whose investigation has so far been surprisingly neglected for the sliding window model, is the membership problem for a language or, equivalently, the computation of Boolean queries on the sliding window. In its general form, one fixes a language L over the alphabet of the data stream, and asks for an algorithm that can check at any time whether the content of the sliding window belongs to L. In this paper, we are mainly interested in the case, where L is a regular language.

Note that in the standard streaming model, it is trivial to solve the membership problem for a regular language L in constant space. For a data stream $a_1a_2a_3\cdots$ the algorithm simply runs a deterministic finite automaton for L and only stores the current state (which needs constant space since we assume the automaton to be fixed and not part of the input). This obvious fact might explain why the membership problem for regular languages in the streaming model has not received any attention so far. In contrast, there exist papers that deal with membership problems for (restricted classes of) context-free languages in the standard streaming model, see the paragraph on related work below.

Note that in the sliding window model the above algorithm (simulation of a DFA on the data stream) does not work. The problem is the removal of the left-most symbol from the sliding window in each step. A naïve approach is to store the whole window in $O(n)$ bits and simulate the DFA on this word. In fact, there exist very simple languages L for which this is the best possible solution in order to be able to decide at any point of time whether the current content of the sliding window belongs to L. An example is the language $a\{a,b\}^*$ of all words that start with a. The point is that by repeated 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 querying $a\{a,b\}^*$ has to use n bits of storage (where n is the window size). The main result of this paper is a trichotomy: The optimal space needed for querying a regular language L in the sliding window model falls into three classes with respect to its growth rate: constant space, $\Theta(\log n)$, and $\Theta(n)$, where n is the window size. We characterize the regular languages by its optimal growth rate algebraically in terms of the syntactic homomorphism and the left Cayley graph of the syntactic monoid of a regular language. The precise characterizations are a bit technical and will be presented in Section 4.

The sliding window model we have talked about so far is also known as the *fixed-size model*, since the sliding window has a fixed size n. In the literature there exists a second model as well which is known as the *variable-size model*, see e.g. [3]. In this model, the arrival of new data items and the expiration of old items can happen independently, which means that the sliding window can grow and shrink. We also determine the space complexity of querying a regular language for the variable-size model. Again, we prove the same trichotomy as above (constant space, $\Theta(\log n)$, and $\Theta(n)$), but the corresponding three classes of regular languages differ slightly from the situation in the fixed-size model.

Related work. In [5] the authors consider the problem of membership checking for various subclasses of context-free languages in the standard streaming model (where the whole history is checked for membership). For deterministic linear languages, a randomized streaming algorithm is presented which works in space $O(\log n)$ and has an inverse polynomial one-sided

error. On the other hand, a visibly pushdown language L exists, for which every randomized streaming algorithm with an error probability $< 1/2$ must use space $\Omega(n)$ [5].

One may consider our streaming algorithms as algorithms for testing membership of a dynamic word in a language L, where the update operations are restricted. In the variable-size model, these updates are the removal of the first symbol from the word, and appending a given symbol a to the word. Membership testing algorithms for regular languages that allow the replacement of the symbol at a specified position were studied in [11]. The focus of [11] is on the cell probe complexity of updates and membership queries.

2 Preliminaries

Let Σ be a finite alphabet. For a word $w = a_1 \cdots a_k \in \Sigma^*$ of length $|w| = k$ 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 $i > j$. A word $v \in \Sigma^*$ is a *suffix* of the word w if there exists a word $u \in \Sigma^*$ such that $w = uv$.

We assume that the reader is familiar with the basic notions of formal languages, in particular regular languages. Our query algorithms for regular languages make use of the description of regular languages by finite monoids; see e.g. the textbook [14] for more details. A *monoid* is a set M together with an associative operation $\cdot : M \times M \to M$ and an element $1 \in M$ satisfying $1 \cdot x = x \cdot 1 = x$ for all $x \in M$. A function $h : M \to N$ between two monoids M, N is a *homomorphism* if $h(1) = 1$ and $h(x \cdot y) = h(x) \cdot h(y)$ for all $x, y \in M$. A language $L \subseteq \Sigma^*$ is *recognized* by a monoid M if there exists a homomorphism $h : \Sigma^* \to M$ from the free monoid Σ^* into a monoid M and a set $F \subseteq M$ such that $w \in L$ if and only if $h(w) \in F$ for all $w \in \Sigma^*$. It is well known that the class of regular languages is exactly the class of languages recognized by finite monoids. For every language $L \subseteq \Sigma^*$ the *syntactic congruence* $\equiv_L$ on Σ^* is defined by $u \equiv_L v$ if and only if for all $x, y \in \Sigma^*$: $xuy \in L$ iff $xvy \in L$. The set of congruence classes $\Sigma^*/\equiv_L$ forms a monoid, which is called the *syntactic monoid* of L and is denoted by $M(L)$. It is the smallest monoid which recognizes L. The function $h : \Sigma^* \to M(L)$ which maps a word u to its congruence class $[u]_{\equiv_L}$ is a homomorphism, called the *syntactic homomorphism* of L.

In this paper all graphs are finite, directed and vertex-colored. For a graph Γ we denote by $V(\Gamma)$ and $E(\Gamma)$ the set of vertices and edges of Γ, respectively. Graphs may have loops, i.e. $E(\Gamma)$ is an arbitrary subset of $V(\Gamma) \times V(\Gamma)$. For graphs Γ and Δ, a *homomorphism* from Γ to Δ is a function $\varphi : V(\Gamma) \to V(\Delta)$ such that for all $v \in V(\Gamma)$ the vertices v and $\varphi(v)$ have the same color and $(u, v) \in E(\Gamma)$ implies $(\varphi(u), \varphi(v)) \in E(\Delta)$. We call a graph Γ *homomorphic* to Δ if there exists a homomorphism from Γ to Δ. For a subset $S \subseteq V(\Gamma)$ we denote by $\mathrm{reach}_\Gamma(S)$ the subgraph of Γ which is induced by all nodes that are reachable from S. A graph Γ is *strongly connected* if for all $u \in V(\Gamma)$ we have $\mathrm{reach}_\Gamma(\{u\}) = \Gamma$. A *strongly connected component*, briefly SCC, of Γ is an inclusion maximal subset $S \subseteq V(\Gamma)$ such that the subgraph induced by S is strongly connected. The set of SCCs of a graph is partially ordered by $S_1 \preceq S_2$ iff a vertex in S_2 is reachable from a vertex in S_1. An SCC of Γ is *trivial* if it consists of a single node v and $(v, v) \notin E(\Gamma)$, otherwise the SCC is called *non-trivial*. A graph is a *directed cycle* if it is strongly connected and every vertex has outdegree (and indegree) 1. Our characterizations of regular languages will refer to homomorphisms from certain graphs (that we define below) to directed cycles. Note that every monochromatic graph is homomorphic to a directed cycle of size one.

For a monoid M and a subset A of M we denote by $\Gamma(M, A)$ the *(unlabelled) left Cayley graph* over the vertex set M with the edge set $\{(x, y) \mid y = a \cdot x \text{ for some } a \in A\}$. If the subset $A \subseteq M$ generates M, i.e. every element of M is a finite product over A, then $y \in M$

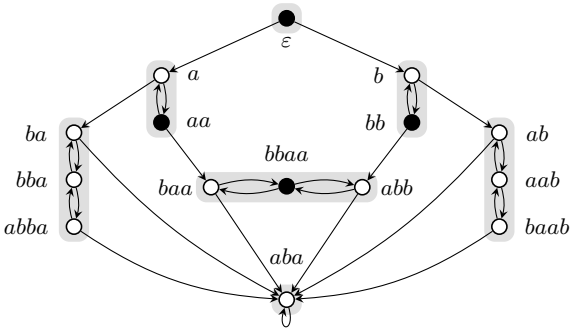

Figure 1 The left Cayley graph $\Gamma(M, A, F)$ from Example 1 for the language $(aa \mid bb)^*$. Vertices from F are black and SCCs are shaded.

is reachable from $x \in M$ in $\Gamma(M, A)$ if and only if $x \leq_{\mathcal{L}} y$ in M, which is defined by

$$x \leq_{\mathcal{L}} y \iff \exists \ell \in M : x = \ell \cdot y. \tag{1}$$

The $\mathcal{L}$-equivalence is defined by $x \equiv_{\mathcal{L}} y$ if and only if $x \leq_{\mathcal{L}} y \leq_{\mathcal{L}} x$. For a subset $F \subseteq M$ we denote with $\Gamma(M, A, F)$ the graph $\Gamma(M, A)$, where in addition all vertices from F (resp., $M \setminus F$) are colored with 1 (resp., 0).

▶ **Example 1.** Consider the regular language $L = (aa \mid bb)^*$. Let $h : \{a, b\}^* \to M$ be the syntactic homomorphism of L into its syntactic monoid M with 15 elements. Figure 1 shows the left Cayley graph $\Gamma(M, A, F)$, where $A = \{h(a), h(b)\}$ and $F = h(L)$. Note that every SCC is homomorphic to a directed cycle.

3 Sliding window models

In the literature, one distinguishes two sliding window models: The *fixed-size model* and the *variable-size model*, see also [3] for a discussion of these models.

3.1 The fixed-size model

A *data stream* over Σ is an infinite sequence $a_1 a_2 a_3 \cdots$ of symbols $a_i \in \Sigma$. The idea is that a data stream represents the sequence of data that is produced by some process. At time t, the observer of this process can only see symbol a_t.

Fix an $n \in \mathbb{N}$, which is called the *window size*. Moreover, fix a data stream $a_1 a_2 a_3 \cdots$. At time $t \geq 0$ the *sliding window* contains the word $a_{t-n+1} a_{t-n+2} \cdots a_t$ consisting of the n last symbols, where $a_i = a$ for a distinguished symbol $a \in \Sigma$ when $i \leq 0$. Thus, in the beginning the sliding window is filled with a's. Let us denote with $W_n(t)$ the content of the sliding window at time t.

In the *fixed-size sliding-window model* we want to answer queries about the window content $W_n(t)$, where the window size n is fixed. For this, the algorithm has at time t access to the n-th symbol a_t and a previously computed data structure, that w.l.o.g. can be assumed to be a bit string $S_n(t) \in \{0, 1\}^*$. The goal is to compute the query, based on a_t and $S_n(t)$. The simplest solution is to store (a binary coding of) $W_n(t)$ in $S_n(t)$, but in many cases we can find a better solution, where $S_n(t)$ is considerably smaller than $W_n(t)$. Moreover, we would like to have such a query algorithm for every window size n. Note that this is a *non-uniform model*: For every n we may have a different query algorithm. This will

not be crucial for our upper bounds, since our algorithms will work for all window sizes n (which is a parameter in the algorithms). But working with a non-uniform model makes our lower bounds stronger.

In this paper, we are only interested in Boolean queries, i.e. queries that output a single bit. Let us fix a language $L \subseteq \Sigma^*$. We say that L is *streamable in space $s(n)$ in the fixed-size model* if for every window size n there exists an algorithm such that for every input stream $a_1a_2a_3\cdots$ the following holds:

- The algorithm maintains a bit string $S_n(t)$ of length at most $s(n)$, where $S_n(0)$ is an arbitrary bit string of length at most $s(n)$ (it corresponds to an initial state).
- At every time $t \geq 0$, the algorithm has only access to $S_n(t)$ and a_{t+1}. Based on these data, the algorithm computes $S_n(t+1)$ and decides correctly whether $W_n(t) \in L$.

3.2 The variable-size model

In the fixed-size model, at every time a new data item arrives and the oldest data item is removed from the window. In contrast, in the *variable-size sliding-window model* the arrival of new data items and the expiration of old items is decoupled and can happen independently. This means that the window can grow and shrink. One can think of an adversary that executes an infinite sequence of operations $\mathsf{op}_1, \mathsf{op}_2, \mathsf{op}_3 \cdots$, where every op_i is either a pop-operation or a $\mathsf{push}(a)$-operation for a symbol $a \in \Sigma$. A pop-operation deletes the first symbol from the window; this corresponds to the situation where the first item in the window expires and falls out of the window (if the window is already empty it stays empty after a pop). A $\mathsf{push}(a)$-operation appends the symbol a at the right end of the sliding window; this corresponds to the arrival of an a in the data stream. In this way we can define for an infinite sequence $\mathsf{op}_1, \mathsf{op}_2, \mathsf{op}_3 \cdots$ of operations $\mathsf{op}_i \in \{\mathsf{pop}\} \cup \{\mathsf{push}(a) \mid a \in \Sigma\}$ the window content $W(t)$ at time $t \in \mathbb{N}$, where $W(0) = \varepsilon$. We say that the language L is *streamable in space $s(n)$ in the variable-size model* if there exists an algorithm such that for every infinite sequence $\mathsf{op}_1, \mathsf{op}_2, \mathsf{op}_3 \cdots$ of operations the following holds:

- At every time $t \geq 0$, the algorithm stores a bit string $S(t)$ of length at most $s(|W(t)|)$, where $S(0) = \varepsilon$.
- At time $t \geq 0$, the algorithm has only access to $S(t)$ and the operation op_{t+1}. Based on these data, the algorithm computes $S(t+1)$ and decides correctly whether $W(t) \in L$.

Note the uniformity of this definition. There is a single algorithm that has to work for every window size. Also note that if L is streamable in space $s(n)$ in the variable-size model, then L is also streamable in space $s(n)$ in the fixed-size model.

The variable-size model captures various other streaming models that appeared in the literature. For instance, the standard model that was mentioned in the introduction corresponds to the case where no pop-operations are allowed. Another realistic model is the *time-stamp based model*, where the data items arrive at arbitrary time points (which are real numbers) and the sliding window contains all data values with an arrival time from the interval $[t-\tau, t]$, where t is the current time and τ is a fixed duration. Also the time-stamp based model can be simulated by the variable-size model, see [3] for details.

4 Streaming algorithms for regular languages

In this section, we will prove our main results. Let $L \subseteq \Sigma^*$ be a regular language. We will query the content of the sliding window for membership in L. Let $M = M(L)$ be the syntactic monoid of L and $h : \Sigma^* \to M$ be the syntactic homomorphism. Let $F = h(L)$, hence $L = h^{-1}(F)$. We simply write Γ for the two-colored left Cayley graph $\Gamma(M, A, F)$

Table 1 The trichotomy results for querying regular languages in the sliding window model.

	constant space	logarithmic space	linear space
fixed-size model	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
variable-size model	$\{\emptyset, \Sigma^*\}$	$(\mathcal{C}_1 \cup \mathcal{C}_2) \setminus \{\emptyset, \Sigma^*\}$	$\mathcal{C}_3$

where $A = h(\Sigma)$. Recall that Γ is a finite directed graph, possibly with loops. For the rest of this section we fix Σ, L, M, h, A, and Γ. It is important for our results that L is fixed, and not part of the input. This implies that the monoid M and the graph Γ can be hard-wired into our algorithms.

We partition the set of all regular languages over Σ into three classes $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, where

- $L \in \mathcal{C}_1$ if and only if for every non-trivial SCC S of Γ the subgraph $\mathrm{reach}_\Gamma(S)$ is homomorphic to a directed cycle,
- $L \in \mathcal{C}_2$ if and only if $L \notin \mathcal{C}_1$ and every SCC of Γ is homomorphic to a directed cycle,
- $L \in \mathcal{C}_3$ if and only if $L \notin (\mathcal{C}_1 \cup \mathcal{C}_2)$.

For instance, the language $(aa \mid bb)^*$ from Example 1 belongs to $\mathcal{C}_2$. Other examples for languages in $\mathcal{C}_1 \cup \mathcal{C}_2$ are *open* languages, i.e. languages of the form $\Sigma^* L$ where L is a regular language over Σ. Examples for languages in $\mathcal{C}_1$ are languages of the form $\Sigma^* w$ for $w \in \Sigma^*$.

For the fixed-size model we will show that (i) languages in $\mathcal{C}_1$ are streamable in constant space, (ii) languages in $\mathcal{C}_2$ are streamable in space $O(\log n)$ but not streamable in space $o(\log n)$, and (iii) languages in $\mathcal{C}_3$ are not streamable in space $o(n)$. For the variable-size model, languages in $\mathcal{C}_3$ are still not streamable in space $o(n)$, but here only the languages $\emptyset$ and Σ^* are streamable in constant space. The remaining languages $(\mathcal{C}_1 \cup \mathcal{C}_2) \setminus \{\emptyset, \Sigma^*\}$ are streamable in space $O(\log n)$ but not streamable in space $o(\log n)$ in the variable-size model. Table 1 summarizes both trichotomies.

▶ **Example 2.** Let $L_1 = \{a, b\}^* a$ be the set of all words that end with an a. Obviously, L_1 is streamable in constant space in the fixed-size model: The algorithm has to store nothing. At each time t one can determine from the current symbol a_t whether the window content belongs to L_1, which is the case for $a_t = a$. Similarly, for every finite word $w \in \{a, b\}^*$ the language $\{a, b\}^* w$ is streamable in constant space in the fixed-size model: The algorithm has to store the last $|w| - 1$ symbols from the stream. Note that this argument fails for the variable-size model: In fact, L_1 is not streamable in constant space in the variable-size model; this follows from Theorem 7 in Section 4.2.

▶ **Example 3.** Let $L_2 = \{a, b\}^* a \{a, b\}^*$ be the set of all words that contain an a. This language is streamable in space $O(\log n)$ in the variable-size model. The algorithm stores (i) the current window size n (using $O(\log n)$ bits), and (ii) the position p of the right-most a in the window (using $O(\log n)$ bits). We set p to 0 if the window contains no a. This information can be easily updated: For a pop-operation, the algorithm sets $n := \max\{0, n-1\}$ and $p := \max\{0, p-1\}$. For a push(a)-operation, the algorithm sets $n := n + 1$ and $p := n$. Finally, for a push(b)-operation only n is incremented.

On the other hand, L_2 is not streamable in space $o(\log n)$ in the fixed-size model: If L_2 would be streamable in space $o(\log n)$ then one could represent every number $1 \leq i \leq n$ by a bit string of length $o(\log n)$, namely by the $o(\log n)$-size data structure $d(i)$ obtained after moving the word $b^{i-1} a b^{n-i}$ into the sliding window, where n is the window size. To recover i from $d(i)$ one continues the stream with b's and thereby simulates the query algorithm for L_2

starting with the data structure $d(i)$. The smallest number of b's after which a membership query for L_2 is answered negatively is i.

▶ **Example 4.** Let $L_3 = a\{a,b\}^*$ be the set of all words that start with an a. An argument similar to Example 3 shows that L_3 is not streamable in space $o(n)$ in the fixed-size model. More precisely, if L_3 would be streamable in space $o(n)$ in the fixed-size model, then one could represent every word $w \in \{a,b\}^n$ by a bit string of length $o(n)$, namely by the $o(n)$-size data structure $d(w)$ obtained after moving the word w into the sliding window, where n is the window size. To recover w from $d(w)$ one simulates the query algorithm starting with the data structure $d(w)$. The query result after seeing $i-1$ further symbols from the stream yields the i-th symbol of w: A positive (resp., negative) query answer yields an a (resp., b).

4.1 Upper bounds

In this section we will prove two upper bounds on the space for querying regular languages in the sliding window model. First, we show that every language in $\mathcal{C}_1 \cup \mathcal{C}_2$ is streamable in logarithmic space in both streaming models.

▶ **Theorem 5.** *If every SCC of Γ is homomorphic to a directed cycle, then L is streamable in space $O(\log n)$ in the variable-size model and hence also in the fixed-size model.*

Proof. Since the fixed-size model can be simulated by the variable-size model, it suffices to present an algorithm for the variable-size model.

Let $w \in \Sigma^*$ be a word of length n and $\mathrm{Suf}(w)$ be the set of suffixes of w, which includes the empty word and w itself. Define the preorder $\preceq$ on $\mathrm{Suf}(w)$ by $u \preceq v$ iff $h(u) \leq_{\mathcal{L}} h(v)$, where $\leq_{\mathcal{L}}$ is defined in (1). This is in fact a total preorder: If $v \in \mathrm{Suf}(w)$ is a suffix of $u \in \mathrm{Suf}(w)$ then $u \preceq v$. But note that we may have $u \preceq v \preceq u$ for two different suffixes of w. The word w is a smallest element w.r.t. $\preceq$. The induced equivalence relation $\equiv$ is defined by $u \equiv v$ iff $u \preceq v \preceq u$. Clearly, $u \equiv v$ iff $h(u) \equiv_{\mathcal{L}} h(v)$. As usual, denote with $\mathrm{Suf}(w)/_{\equiv}$ the set of equivalence classes of $\equiv$. Note that $|\mathrm{Suf}(w)/_{\equiv}|$ is bounded by a constant which only depends on the monoid M and not on the window size n. One can identify the elements of $\mathrm{Suf}(w)/_{\equiv}$ with intervals on the set of positions of w. Hence we can represent $(\mathrm{Suf}(w), \preceq)$ by storing a constant number of interval endpoints using $O(\log n)$ bits. Our streaming algorithm (for window size n) stores the following data:

- the total preorder $(\mathrm{Suf}(w), \preceq)$, using $O(\log n)$ bits,
- the function $f : \mathrm{Suf}(w)/_{\equiv} \to M$ defined by $f(C) = h(v)$ where v is the shortest suffix in the equivalence class C, using $O(1)$ bits.

We describe these data conveniently by a sequence

$$p_0, m_1, p_1, m_2, p_2, \ldots, m_{k-1}, p_{k-1}, m_k, p_k \tag{2}$$

such that the following holds:

- $1 \leq k \leq |M|$,
- $0 = p_0 < p_1 < \cdots < p_{k-1} < p_k = n+1$,
- $m_1, \ldots, m_k \in M$ and m_k is the unit element of M.

The meaning of this sequence is the following: The equivalence classes of $\equiv$ are the sets $C_i = \{w[p:n] \mid p_{i-1} < p \leq p_i\}$ for $1 \leq i \leq k$ (the class C_k contains the empty suffix for $p = p_k = n+1$). The monoid element m_i is $h(w[p_i:n])$ for $1 \leq i \leq k$ (hence, $m_k = 1$ is the unit element). Thus, $m_i = h(v)$ where v is the shortest suffix in its equivalence class C_i.

On the sequence (2) we can now perform the desired queries: In order to test whether $w \in L$, one has to check whether $h(w) \in F$. For this we consider the monoid element m_1.

Note that $m_1 \equiv_{\mathcal{L}} h(w)$. Hence, the vertices $h(w)$ and $m_1 = h(w[p_1 : n])$ belong to the same SCC S of Γ. Note that $h(w) = h(w[1 : p_1 - 1])m_1$. We cannot store this word $w[1 : p_1 - 1]$, in fact we do not even store its image under h. But, by assumption, the SCC S of Γ that contains $h(w)$ and m_1 has a homomorphism φ onto a directed cycle Θ. Thus, we can compute $\varphi(h(w))$ by traversing the cycle from $\varphi(m_1)$ for $p_1 - 1$ steps (the homomorphic image of Γ under φ is hard-wired into the algorithm). The color of $\varphi(h(w))$ in Θ then indicates whether $h(w) \in F$ (i.e. $w \in L$) or $h(w) \notin F$ (i.e. $w \notin L$).

For a pop-operation on w and $p_1 > 1$, the algorithm updates the sequence (2) to

$$p_0, m_1, p_1 - 1, m_2, p_2 - 1, \ldots, m_{k-1}, p_{k-1} - 1, m_k, p_k - 1.$$

Otherwise, if $p_1 = 1$ then the algorithm updates the sequence (2) to

$$p_1 - 1, m_2, p_2 - 1, \ldots, m_{k-1}, p_{k-1} - 1, m_k, p_k - 1.$$

Finally, let us consider a push(a)-operation on w. Note that $\equiv_{\mathcal{L}}$ is a right congruence, i.e. $x \equiv_{\mathcal{L}} y$ implies $xz \equiv_{\mathcal{L}} yz$ for all $x, y, z \in M$. This means that our interval-representation of $(\mathrm{Suf}(wa), \preceq)$ can be obtained from the interval-representation of $(\mathrm{Suf}(w), \preceq)$ by possibly merging successive intervals. In order to detect, which intervals have to be merged, note that for all $u, v \in \mathrm{Suf}(w)$ we have

$$ua \equiv va \iff h(u)h(a) \equiv_{\mathcal{L}} h(v)h(a) \iff f([u]_{\equiv})h(a) \equiv_{\mathcal{L}} f([v]_{\equiv})h(a),$$

because $h(u) \equiv_{\mathcal{L}} f([u]_{\equiv})$ and $h(v) \equiv_{\mathcal{L}} f([v]_{\equiv})$, and the fact that $\equiv_{\mathcal{L}}$ is a right congruence. Using this, we can detect whether two successive intervals that represent the classes $[u]_{\equiv}$ and $[v]_{\equiv}$ have to be merged into a single interval. Formally, we process the sequence (2) as follows: We walk over the sequence from left to right. For every $1 \leq i \leq k - 1$ we check whether $m_i h(a) \equiv_{\mathcal{L}} m_{i+1} h(a)$. If this is true, then we remove m_i, p_i from the sequence, otherwise we replace m_i, p_i by $m_i h(a), p_i$. Then we continue with $i + 1$ (if $i < k - 1$). Finally, we check whether $h(a) \equiv_{\mathcal{L}} 1$. If this holds, then we replace $m_k, p_k = 1, n + 1$ by $1, n + 2$, otherwise we replace $1, n + 1$ by $h(a), n + 1, 1, n + 2$. ◀

Next we show that languages in $\mathcal{C}_1$ are streamable in constant space in the fixed-size model.

▶ **Theorem 6.** *Let* $\mathrm{reach}_\Gamma(S)$ *be homomorphic to a directed cycle for every non-trivial SCC* S *of* Γ*. Then* L *is streamable in space* $O(1)$ *in the fixed-size model.*

Proof. Observe that every path in Γ of length at least $c := |V(\Gamma)|$ (a constant) contains a vertex in a non-trivial SCC S and therefore ends in $\mathrm{reach}_\Gamma(S)$. Fix a window size n. If $n < c$, we store the window content explicitly and can test whether $w \in L$, e.g. using an automaton for L. Now assume $n \geq c$. For a window content $w \in \Sigma^*$ we explicitly store the suffix v of length c. Clearly this suffix can be updated when a new symbol arrives in the window. Also v suffices to test whether $w \in L$. We compute $h(v)$ and a non-trivial SCC S such that $h(v)$ is contained in $\mathrm{reach}_\Gamma(S)$. Let $\varphi : \mathrm{reach}_\Gamma(S) \to \Theta$ be the homomorphism into a directed cycle Θ. Then we compute $\varphi(h(w))$ by traversing Θ starting from the vertex $\varphi(h(v))$ for $n - c$ steps. The color of $\varphi(h(w))$ determines whether $w \in L$. ◀

As in most previous work on the sliding window model, our focus is on the space requirements of query algorithms. But it is also interesting to note that in Theorem 5 and 6 we can achieve constant time for all update and query operations on the RAM model with register length $O(\log n)$. Let us show this for Theorem 5 first. Recall that the sequence

(2) that we manipulate in the proof of Theorem 5 has constant length. For a pop- or push(a)-operation, the manipulation of (2) only needs constant time. To see this, note that the numbers p_i in (2) are only incremented or decremented and that all operations in the monoid M need constant time since M is fixed. Finally, for a membership query we traverse the cycle Θ starting from $\varphi(m_1)$ for $p_1 - 1$ steps. To do this in constant time, we store also the numbers $(p_i - 1) \bmod \ell(\Theta)$, where $\ell(\Theta)$ is the length of the cycle. We have to maintain these remainders for all (constantly many) cycle lengths $\ell(\Theta_1), \ldots, \ell(\Theta_c)$, where $\Theta_1, \ldots, \Theta_c$ are the cycles to which the SCCs of Γ are homomorphic. For Theorem 6 it suffices to traverse Θ for $(n - c) \bmod \ell(\Theta)$ steps.

4.2 Lower bounds

In this section, we prove matching lower bounds for the upper bounds from the previous section. Let us first show that constant space in the variable-size model makes it impossible to query any non-trivial language. Roughly speaking, the reason is that in order to query a non-trivial language in the variable-size model one has to know when the sliding window is empty. But for this, one has to maintain the size of the window, for which $\log n$ bits are needed.

▶ **Theorem 7.** *If $L \subseteq \Sigma^*$ and $\emptyset \subsetneq L \subsetneq \Sigma^*$, then L is not streamable in space $O(1)$ in the variable-size model.*

Proof. Towards a contradiction assume that L is streamable in the variable-size model in space m, where m is a constant, which means that the algorithm has at most 2^m pairwise distinct data structures. We can assume that $\varepsilon \in L$, otherwise consider the complement $\Sigma^* \setminus L$ which is also streamable in space m. Let further $w \in \Sigma^*$ be a word such that $w \notin L$. Consider the $2^m + 1$ words $w^0, w^1, w^2, \ldots, w^{2^m}$. There are two numbers $0 \leq i < j \leq 2^m$ such that the stream prefixes w^i and w^j lead to the same data structure. After $(j - 1) \cdot |w|$ further pop-operations the window contains $\varepsilon \in L$ and the word $w \notin L$, respectively, which is a contradiction. ◀

For the remaining lower bounds, we need the following simple graph theoretic lemma:

▶ **Lemma 8.** *Let Γ be a finite directed vertex-colored graph (possibly with loops) and let s be a vertex from which all vertices of Γ are reachable. Assume that all vertices have outdegree ≥ 1 and s has indegree ≥ 1. If Γ is not homomorphic to a directed cycle, then there exist paths π_0, π_1 of the same length from s to vertices s_0, s_1 which have distinct colors.*

Proof. Let V_n be the set of vertices which are reachable from s via a path of length n for $n \geq 0$. The union $\bigcup_{n \geq 0} V_n$ is the set of vertices reachable from s, which by assumption is $V(\Gamma)$. Towards a contradiction assume that every set V_n is monochromatic. Let $\approx$ be the transitive-reflexive closure of the binary relation R on $V(\Gamma)$ defined by $R = \bigcup_{n \geq 0} V_n \times V_n$. Then, every equivalence class of $\approx$ is monochromatic. Hence, we can construct the quotient graph $\Gamma/_\approx = (\{[v]_\approx \mid v \in V(\Gamma)\}, \{([u]_\approx, [v]_\approx) \mid (u, v) \in E(\Gamma)\})$. Moreover, the equivalence class $[u]_\approx$ has the same color as all its elements. Clearly, Γ is homomorphic to $\Gamma/_\approx$.

We claim that every vertex in $\Gamma/_\approx$ has out-degree 1: Since every vertex in Γ has outdegree ≥ 1, the same holds for $\Gamma/_\approx$. Moreover, if a vertex v is contained in some set V_n, then all successors of v are contained in V_{n+1}. This implies that R respects the successor relation, i.e. whenever $(u, v) \in R$ and $(u, u'), (v, v') \in E(\Gamma)$, then also $(u', v') \in R$. Hence, also $\approx$ respects the successor relation. This proves that every vertex in $\Gamma/_\approx$ has out-degree 1. Finally, each node has an incoming edge since s has an incoming edge and all other nodes are reachable from s. It follows that $\Gamma/_\approx$ must in fact be a directed cycle. ◀

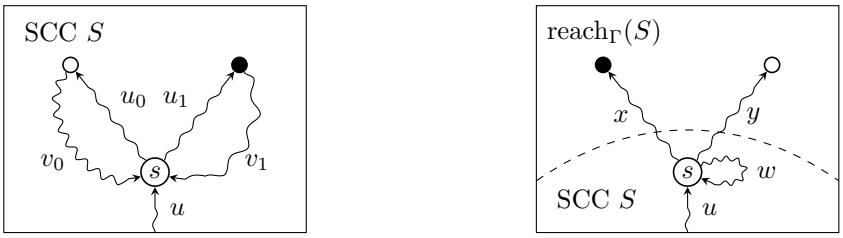

Figure 2 The origin of the words used in the proofs of Theorem 9 (left) and Theorem 10 (right).

Now we show that languages in $\mathcal{C}_3$ are not streamable in space $o(n)$ in the fixed-size model.

▶ **Theorem 9.** *If some SCC S of Γ is not homomorphic to a directed cycle, then L is not streamable in space $o(n)$ in the fixed-size model and hence not streamable in space $o(n)$ in the variable-size model.*

Proof. We apply Lemma 8 with an arbitrary node $s \in S$ to the subgraph of Γ induced by S. Therefore, there exist paths π_0 and π_1 of the same length k from s to nodes $s_0, s_1 \in S$, which are colored differently, say $s_0 \notin F$ and $s_1 \in F$. Let $u_0, u_1 \in \Sigma^k$ be words representing the paths π_0, π_1 and let $u \in \Sigma^*$ such that $h(u) = s$, which exists since h is surjective. Since S is strongly connected, there also exist paths π_0' and π_1' from s_0 and s_1, respectively, back to s. This results in words $v_0, v_1 \in \Sigma^+$ such that $h(v_0u_0u) = h(u) = h(v_1u_1u)$, $h(u_0u) \notin F$, $h(u_1u) \in F$. The situation is shown in Figure 2 on the left. Choose numbers $p, q > 0$ such that $z_0 = (v_0u_0)^p$ and $z_1 = (v_1u_1)^q$ have the same length. We get $h(z_0u) = h(z_1u) = h(u)$. Let x_0, x_1 such that $z_0 = x_0u_0$ and $z_1 = x_1u_1$ and hence $|x_0| = |x_1|$.

Let $z = z_0$ (we could also set $z = z_1$). We have $h(u) = h(z^mu)$ for every m. Note that $z \neq \varepsilon$. By replacing x_0 and x_1 by z^mx_0 and z^mx_1, respectively, for m large enough we can therefore assume that $|x_0| = |x_1| \geq |u|$. Let $z = z'z''$ with $|z''| + |u| = |x_0|$.

Assume now that L is streamable in space $o(n)$ in the fixed-size model. We will deduce a contradiction. Consider an arbitrary bit string $\alpha = a_1 \cdots a_n \in \{0,1\}^n$ of length n. We encode this bit string by the word $w(\alpha) = z_{a_1}z_{a_2} \cdots z_{a_n}z'$ of length $n' = \Theta(n)$. Let n' be the window size. For n large enough, there must exist bit strings $\alpha = a_1 \cdots a_n$ and $\beta = b_1 \cdots b_n$ of length n such that $\alpha \neq \beta$ but after moving $w(\alpha)$ and $w(\beta)$ into the sliding window, the same internal data structure arises. Let $1 \leq i \leq n$ be a position such that w.l.o.g. $a_i = 0$ and $b_i = 1$. We now move the word $z''z^{i-1}u$ of length $(i-1)|z| + |z''| + |u| = (i-1)|z| + |x_0| = (i-1)|z| + |x_1|$ into the sliding window. The window contents are then:

$$\begin{aligned} u_0 z_{a_{i+1}} \cdots z_{a_n} z' z'' z^{i-1} u &= u_0 z_{a_{i+1}} \cdots z_{a_n} z^i u \\ u_1 z_{b_{i+1}} \cdots z_{b_n} z' z'' z^{i-1} u &= u_1 z_{b_{i+1}} \cdots z_{b_n} z^i u \end{aligned}$$

Of course, the stream prefixes $w(\alpha)z''z^{i-1}u$ and $w(\beta)z''z^{i-1}u$ must still lead to the same data structure. But we have $h(u_0 z_{a_{i+1}} \cdots z_{a_n} z^i u) = h(u_0u) \notin F$ and $h(u_1 z_{b_{i+1}} \cdots z_{b_n} z^i u) = h(u_1u) \in F$, which is a contradiction. ◀

For a word $w \in \Sigma^*$ of length k we define the signature $\delta(w) = b_1 \cdots b_k \in \{0,1\}^*$ such that $b_i = 1$ if $h(w[i:k]) \in F$ and $b_i = 0$ otherwise. To complete our trichotomy, we finally show that languages in $\mathcal{C}_2$ are not streamable in space $o(\log n)$ in the fixed-size model.

▶ **Theorem 10.** *If some SCC S of Γ is non-trivial and* reach$_\Gamma(S)$ *is not homomorphic to a directed cycle, then L is not streamable in space $o(\log n)$ in the fixed-size model and hence not streamable in space $o(\log n)$ in the variable-size model.*

Proof. Let S be the strongly connected component of Γ which is not trivial and where $\mathrm{reach}_\Gamma(S)$ is not homomorphic to a directed cycle. Pick an arbitrary node $s \in S$. It must have indegree ≥ 1 since S is non-trivial. Since h is surjective there exists $u \in \Sigma^*$ with $h(u) = s$. We apply Lemma 8 to s and the subgraph $\mathrm{reach}_\Gamma(S)$. This yields words $x, y \in \Sigma^+$ of equal length, say $k \geq 1$, which correspond to the paths π_0, π_1 in the lemma, such that $h(xu) \in F$ and $h(yu) \notin F$. Further, since S is strongly connected and non-trivial, there exists a non-trivial path π from s back to s, which yields a word $w \in \Sigma^+$ with $h(wu) = h(u)$. The situation is shown in Figure 2 on the right. Let $\ell = |w|$ and write k uniquely as $k = c \cdot \ell + (\ell - p + 1) = (c+1) \cdot \ell - p + 1$ for $c \geq 0$ and $1 \leq p \leq \ell$. Consider the word $w[p : \ell]w^c$. In Γ this word yields the path consisting of c repetitions of the circle π followed by $\ell - p + 1$ more steps of π such that the whole path has length k. If $h(w[p : \ell]w^c u) \in F$ then we can replace x by $w[p : \ell]w^c$, otherwise we can replace y by $w[p : \ell]w^c$. Without loss of generality we can assume that $x = w[p : \ell]w^c$. From $h(xu) \in F$ and $h(yu) \notin F$ it follows that the signatures $\delta(xu)$ and $\delta(yu)$ differ in the first position. We can assume that for each position $i > 1$ we have $\delta(xu)[i] = \delta(yu)[i]$, otherwise we update the words x and y to the suffixes $x[i : k]$ and $y[i : k]$, respectively, where i is the maximal position such that $\delta(xu)[i] \neq \delta(yu)[i]$.

Now assume that L is streamable in space $s(n) \in o(\log n)$ in the fixed-size model. We will deduce a contradiction. For n large enough we consider the n words $z_i = uw^{n-i}yw^i$ $(1 \leq i \leq n)$ of equal length $n' = \ell \cdot n + |u| + |y| = \ell \cdot n + |u| + k \in \Theta(n)$. Large enough here means that $2^{s(n')} < n$; such an n exists since $s(n) \in o(\log n)$ and $n' \in \Theta(n)$. We now fix the window size to n' and move the words z_i $(1 \leq i \leq n)$ into the window. Since $2^{s(n')} < n$, there exist $i < j$ such that after moving z_i and z_j in the window, the same internal data structure arises. Hence the two stream prefixes $z_i w^{n-i}u$ and $z_j w^{n-i}u$ also lead to the same internal data structure. Moreover, after the stream prefix $z_i w^{n-i}u = uw^{n-i}yw^n u$ the content of the sliding window is $yw^n u$ (the suffix of $uw^{n-i}yw^n u$ of length $n' = \ell \cdot n + |u| + k$), which does not belong to L since $h(yw^n u) = h(yu) \notin F$. So, it remains to show that the suffix of length n' of the stream prefix $z_j w^{n-i}u = uw^{n-j}yw^{n+j-i}u$ belongs to L. We distinguish two cases (recall that $k = c \cdot \ell + (\ell - p + 1)$): If $j - i \geq c + 1$, then the suffix of $uw^{n-j}yw^{n+j-i}u$ of length n' is $w[p : \ell]w^{n+c}u = xw^n u$ which belongs to L since $h(xw^n u) = h(xu) \in F$. If $j - i \leq c$, then the suffix of $uw^{n-j}yw^{n+j-i}u$ of length $n' = \ell \cdot n + |u| + k$ is $y[1 + (j-i)\ell : k]w^{n+j-i}u$. We have $h(y[1 + (j-i)\ell : k]w^{n+j-i}u) = h(y[1 + (j-i)\ell : k]u)$. Now recall that the signatures $\delta(xu)$ and $\delta(yu)$ only differ in the first position. Since $1 + (j-i)\ell \geq 2$ it follows that $h(y[1 + (j-i)\ell : k]u) \in F$ if and only if $h(x[1 + (j-i)\ell : k]u) \in F$. Since $x = w[p : \ell]w^c$ and $j - i \leq c$ we have $x[1 + (j-i)\ell : k] = w[p : \ell]w^{c-j+i}$. Thus, we have $h(x[1 + (j-i)\ell : k]u) = h(w[p : \ell]w^{c-j+i}u) = h(w[p : \ell]w^c u) = h(xu) \in F$, which finally show that $y[1 + (j-i)\ell : k]w^{n+j-i}u$ belongs to L.

To sum up, we found two stream prefixes $z_i w^{n-i}u$ and $z_j w^{n-i}u$, which lead to the same internal data structure, but after seeing $z_i w^{n-i}u$ the window content does not belongs L, whereas after seeing $z_j w^{n-i}u$ the window content belongs to L. This is a contradiction. ◀

5 Streaming algorithms for non-regular languages

It would be interesting to know whether our classification can be extended to larger language classes. As a first step, one might consider deterministic context-free languages or the subclass of visibly pushdown languages [2]. All visibly pushdown languages that we have considered so far fall into our trichotomy.

▶ **Example 11.** Let $L_4 = \{a^k b^k \mid k \geq 0\}$. This language is streamable in space $O(\log n)$ in the variable-size model. The algorithm stores (i) the current window length n and the unique numbers k, m such that $a^k b^m$ is the longest suffix of the window that belongs to a^*b^*. Note that $1 \leq k + m \leq n$. This information can be maintained: For a pop-operation, k and m are not changed unless $n = k + m$. In this case, k is decremented if $k > 0$. If $k = 0$ then m is decremented. For a push(b)-operation, m is incremented. Finally, for a push(a)-operation, the algorithm sets $k := 1$ and $m := 0$ if $m > 0$. If $m = 0$, then k is incremented.

Assume that L_4 is streamable in space $o(\log n)$ in the fixed-size model. Similar to Example 3 we would be able to represent every number $1 \leq i \leq n$ by a bit string of length $o(\log n)$, namely by the data structure obtained by inserting the word $a^{n+i}b^{n-i}$ into a sliding window of size $2n$.

▶ **Example 12.** Let L_5 be the Dyck-language over a single pair `(,)` of brackets. We claim that L_5 is not streamable in space $o(n)$ in the fixed-size model. In order to get a contradiction, assume that L_5 is streamable in space $o(n)$ in the fixed-size model. As in Example 4 we deduce that every bit string of length n can be represented with $o(n)$ many bits. For this, we encode a bit string $\alpha = a_1 a_2 \cdots a_n$ ($a_i \in \{0, 1\}$) by the word $u(\alpha) = u_1 u_2 \cdots u_n$, where $u_i =$ `()()` if $a_i = 0$ and $u_i =$ `(())` if $a_i = 1$. Note that $|u(\alpha)| = 4n$. We then represent α by the $o(n)$-size data structure $d(\alpha)$ obtained by moving $u(\alpha)$ in the sliding window, where the window size is $4n$. To recover a_i from $d(\alpha)$ one continues the data stream with $2i - 1$ many repetitions of `()`. Then, the window content belongs to L_5 if and only if $a_i = 0$.

The following example shows that there exists a non-context-free language whose optimal space requirement is $\Theta(\sqrt{n})$ in the fixed-size model.

▶ **Example 13.** Let $L_6 = \{w^k \mid n \geq 0, w \in \{a, b\}^*, |w| = k\}$. We claim that in the fixed-size model, L_6 is streamable in space $O(\sqrt{n})$ but not in space $o(\sqrt{n})$. If the window size n is not a square, then the query algorithm can always answer with no. So, assume that the window size is $n = m^2$. The algorithm then stores for the window content w (i) the length-m suffix s of w and (ii) the largest position p such that $m + 1 \leq p \leq n$ and $w[p] \neq w[p - m]$, where we set $p = m$ if such a position does not exist. Note that $w \in L_6$ if and only if $p = m$. This information s, p can be maintained. For s this is clear. To maintain p, the algorithm checks whether the next symbol in the stream is the first symbol of s. If this is the case, the algorithm sets $p := \max\{p - 1, m\}$, otherwise it sets $p := n$.

The argument that L_6 is not streamable in space $o(\sqrt{n})$ in the fixed-size model is similar to the argument in Example 4. One shows that from the data structure that is obtained by moving w^m (with $w \in \{a, b\}^m$) into the sliding window, one can recover the word w.

In the variable-size model, L_6 is not even streamable in space $o(n)$: It is a basic result in communication complexity that equality checking of two words x and y of length n needs $\Omega(n)$ bits of communication. Assume that L_6 is streamable in space $o(n)$ in the variable-size model. Then Alice, who initially has access to x, and Bob, who has access to y, could check $x = y$ by exchanging $o(n)$ bits, where $n = |x| = |y|$: Alice pushes the word x into the window and then sends the $o(n)$-size data structure to Bob. Bob then pushes the word y^{n-1} into the window and afterwards check whether the window content belongs to L_6, which is the case if and only if $x = y$.

In the long version of this paper, we will present for every $k \geq 2$ an example for a non-deterministic context-free language that in the fixed-size model is streamable in space $O(n^{1/k})$ but not streamable in space $o(n^{1/k})$.

6 Future work

Our results on querying regular languages in the sliding window model open several avenues for further research. First of all, one might also consider randomized query algorithms for the sliding window model. For the standard streaming model randomized query algorithms were studied in [5] for subclasses of context-free languages.

We also plan to investigate whether our trichotomy can be extended to larger language classes. As remarked above, it fails for non-deterministic context-free languages. But it is open whether there exists a deterministic context-free language or even a visibly pushdown language L such that in the fixed-size (resp., variable-size) model L is streamable in space $o(n)$ but not streamable in space $O(\log n)$.

It would be interesting to know the space complexity of querying regular languages in the sliding window model, when the regular language is part of the input, and, for instance, given by a deterministic finite automaton (DFA). The syntactic monoid of $L(A)$, where A is an m-state DFA, can have size m^m [13]. This yields the space bound $O(\log(n) \cdot m \cdot \log(m) \cdot m^m)$ in the proof of Theorem 5, where n is the window size. But maybe a better algorithm exists.

Finally, one might also study weighted automata in the sliding window model. A weighted automaton computes for an input word a value from a semiring, which is the Boolean semiring for classical finite automata; see [10] for details. The goal would be to maintain the semiring value to which the sliding window content maps.

Acknowledgements. We thank Philipp Reh for spotting a mistake in an earlier version of the paper.

References

1 Charu C. Aggarwal. *Data Streams – Models and Algorithms*. Springer, 2007.

2 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of STOC 2004*, pages 202–211. ACM Press, 2004.

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. *Theor. Comput. Sci.*, 494:13–23, 2013.

6 Vladimir Braverman. Sliding window algorithms. In *Encyclopedia of Algorithms*, pages 2006–2011. Springer, 2016.

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

8 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.

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

10 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 2009.

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

12 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.

13 Markus Holzer and Barbara König. On deterministic finite automata and syntactic monoid size. *Theor. Comput. Sci.*, 327(3):319–347, 2004.

14 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, Boston, Basel, Berlin, 1994.