Language Recognition in the Sliding Window Model

DISSERTATION
zur Erlangung des Grades eines Doktors
der Naturwissenschaften

vorgelegt von

Moses Ganardi, M.Sc.

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen
Siegen 2019

Betreuer und erster Gutachter
Prof. Dr. Markus Lohrey
Universität Siegen

Zweiter Gutachter
Prof. Dr. Thomas Schwentick
Technische Universität Dortmund

Tag der mündlichen Prüfung
16. Oktober 2019

Abstract

In many streaming applications recent elements in the stream are considered more important than older elements. In the *sliding window model* we are given an unbounded stream of elements and the goal is to maintain a data structure which allows performing a certain query (e.g. computing a numerical quantity or verifying a property) on the set or sequence of the last n elements. The number n is called the window size, which can be either a fixed number or controlled online. The challenge is to devise streaming algorithms which avoid maintaining the window explicitly using $\Theta(n)$ space.

This thesis considers the language recognition problem in the sliding window problem: Given a formal language (in other words, a property) and a stream of symbols, maintain a small data structure which allows testing whether the current window, i.e. the suffix of length n, belongs to the language (satisfies the property). The main question that we aim to answer is: Which languages admit sliding window algorithms using sublinear space?

The first main result is a space trichotomy (constant, logarithmic, linear) for the space complexity of regular languages in the fixed- and the variable-size sliding window model, together with language-theoretic descriptions of the space classes. We also study the uniform setting where the regular language is considered as part of the input. On this basis we extend these results in various directions: (i) randomness, (ii) approximation, and (iii) subclasses of context-free languages. We prove a quatrochotomy for the randomized space complexity of regular languages. Concerning approximation, we present a constant-space sliding window property tester for every regular language, which distinguishes between words in the language and words that have large Hamming distance from the language. Finally, we give partial results on context-free languages over sliding windows and extend the space trichotomy for regular languages to the class of visibly pushdown languages.

Acknowledgements

First of all, I would like to thank my advisor Markus Lohrey who has been a great mentor to me and has been always open for ideas and discussions. Thanks for bearing a lot of my sloppy mistakes and for supporting me to become a better scientist. I am also very grateful to Thomas Schwentick for coexamining this thesis.

I am very glad for many opportunities to work with various coauthors. Thanks go to Stefan Göller, Artur Jeż, Konstantinos Mamouras, Tatiana Starikovskaya, and Georg Zetzsche. I would like to thank Andreas Krebs for contributing many ideas at the TüFTLeR seminar in 2016 and for hosting me for a week in Tübingen. In 2018, I had the wonderful opportunity to spend a month in Auckland and to work with Bakh Khoussainov on automatic structures.

I would like to thank various people from my years of study at RWTH Aachen, in particular Erich Grädel, Christof Löding, Peter Rossmanith, and Wolfgang Thomas, who sparked my interest in theoretical computer science. I am deeply grateful for Wolfgang Thomas who supported me to pursue a doctoral degree.

Furthermore, I would also like to thank all my current and former colleagues at the Lehrstuhl für Theoretische Informatik at University of Siegen: Michael Figelius, Danny Hucke, Seungbum Jo, Daniel König, Eric Nöth, Carl Philipp Reh, and Louisa Seelbach Benkner. You created a wonderful work atmosphere with many enjoyable conversations and interesting discussions.

Last but not least, I owe a lot of thanks to my friends, to my family, and to my wonderful Lionnie. Thanks for your love and for always cheering me on.

Contents

Chapter 1

Introduction

1.1 The streaming model

With increasing amounts of data, it has become a challenge how to process these data efficiently. In particular, the design and analysis of *streaming algorithms* has attained a lot of interest in recent years, both in practice and theory. Streaming algorithms receive their input as a sequence of elements or data items (in contrast to algorithms with random-access) and have to process each incoming element quickly, using only a small amount of memory. Such small space requirements arise for instance when searching in large databases (e.g. genome databases or web databases), analyzing internet traffic (e.g. click stream analysis), and monitoring networks. For surveys on data streaming, we refer to [2, 92].

From the perspective of this thesis we can summarize the literature on the streaming model roughly in three lines of research. Firstly, the vast majority of papers deals with computing aggregates and statistics over data streams, and related techniques. These include works on computing quantiles over data streams by Munro and Paterson [91], the AMS-algorithm (Alon, Matias, Szegedy) for computing frequency moments over data streams [5], the count-min sketch for computing heavy hitters [35], and Indyk's work on stable distributions and approximating the L_p-norm [68]. Concerning related techniques, let us also mention the probabilistic counters by Flajolet and Martin [50], and Vitter's reservoir sampling algorithm [111].

Secondly, there has been a growing body of research on streaming algorithms for formal languages. Hartmanis, Lewis and Stearns have studied the space complexity of context-free and context-sensitive languages on online Turing machines [84, 105]. More recently, Magniez, Mathieu and Nayak have presented a randomized streaming algorithm for the Dyck languages D_s using $O(\sqrt{n \log n})$ space and polylogarithmic time per letter [86]. Furthermore, they prove almost matching lower bounds. For the subclass **DLIN** of deterministic linear languages a randomized streaming algorithm using $O(\log n)$ space was given in [14]. Krebs, Limaye and Srinivasan considered streaming algorithms for the Dyck-2 language where the input may contain errors [80]. François et al. [52] presented a streaming property tester for visibly pushdown languages, which distinguishes words in the language from words that are ϵ-far from it, and works in space $(\log n/\epsilon)^{O(1)}$.

Thirdly, we list the fundamental problem of pattern matching. Here we are given a pattern of length n and a streaming text, and at each moment the algorithm must decide if the current suffix of length n matches the pattern. The classical algorithms by Knuth, Morris, Pratt [77], and Karp and Rabin [73] solve this problem in linear time and $O(n)$ space. Porat and Porat [98] showed that pattern matching can be solved in $O(\log n)$ space (in words) and $O(\log n)$ time per symbol of the text, and Breslauer and Galil [24] improved this result to $O(\log n)$ space and $O(1)$ time. Later the pattern matching problem was generalized to the case of multiple patterns [30, 63] and streams [62]. Golan, Kopelowitz and Porat [61] also considered the problem of pattern matching with d wildcards. More formally, the algorithm receives as an input a pattern, i.e. a string of length n that contains at most d wildcards (special symbols that match any symbol of the alphabet), and must find all substrings of the text matching the pattern. Their algorithm uses $O(d + \log n)$ time per symbol and $O(d \log n)$ space. Finally, the problem of pattern matching has been also studied in the variant where we must compute the distance (Hamming or edit) between the pattern and the current suffix [31, 32, 33, 62, 98, 104].

1.2 The sliding window model

In many applications data items in a stream are outdated after a certain time. 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, newer data items are more important than older ones. An easy and mathematically clean way to model this is the *sliding window model*. The user specifies a property φ and the length n of the window in which the property should be verified. For a given input stream the task of the algorithm is to decide at every time instant whether the suffix of length n satisfies the property φ or not. More generally φ may be an arbitrary function computed from the suffix of length n. Let us quote from the overview paper by Babcock et al. [11]: “In fact, for many such applications, sliding windows can be thought of not as an approximation technique reluctantly imposed due to the infeasibility of computing over all historical data, but rather as part of the desired query semantics explicitly expressed as part of the user’s query.”

The pattern matching problem and its variants can be considered as a particular sliding window problem; however, we usually consider properties φ over windows of arbitrary length.

The sliding window model was introduced in the seminal work by Datar et al. [36] where the authors considered the basic counting problem: Given a window length n and a stream of bits, maintain a count of the number of 1’s in the window. One can easily observe that an exact solution would require $\Theta(n)$ bits. Intuitively, the reason is that the algorithm cannot see the bit which is

$$\mathtt{H\ L\ \bar{S}\ \bar{S}\ L\ H}\ \boxed{\mathtt{\bar{S}\ L\ S\ L\ \bar{S}}}\ \mathtt{S\ H\ L\ L}\cdots$$
$$\longrightarrow$$

Figure 1.1: A stream of signals with a sliding window of length $n = 5$. Signals in gray will arrive in the future.

about to expire (the n-th most recent bit) without storing it explicitly. However, the authors show that using $O(\frac{1}{\epsilon}\log^2 n)$ memory bits one can maintain an approximate count up to a multiplicative factor of $1 \pm \epsilon$. Furthermore, they extend this result to arbitrary functions which satisfy certain additivity properties, e.g. L_p-norms for $p \in [1, 2]$. Braverman and Ostrovsky introduced the smooth histogram framework [22], to compute so-called smooth functions over sliding windows, which include all L_p-norms and frequency moments. Further work on computing aggregates, statistics and frequent elements in sliding window model can be found in [10, 13, 16, 17, 21, 37, 45, 56, 60]. The problem of sampling over sliding windows was first studied in [12] and later improved in [23]. As an alternative to sliding windows, Cohen and Strauss consider the problem of maintaining stream aggregates where the data items are weighted by a decay function [34].

A natural problem that has been surprisingly neglected is the language recognition problem over sliding windows. As a concrete motivating problem, consider a fire monitoring system that uses input data from a smoke detector and a heat detector. The data arrives in a streaming fashion and for simplicity suppose the smoke detector emits signals $\mathtt{S}$ (smoke) and $\bar{\mathtt{S}}$ (no smoke) and the heat detector emits signals $\mathtt{H}$ (high temperature) and $\mathtt{L}$ (low temperature). The signals from the two detectors are interleaved into one stream, and we define the following patterns over the stream that indicate the existence of fire: (i) the smoke detector emits two consecutive $\mathtt{S}$ signals, or (ii) the heat detector emits three consecutive $\mathtt{H}$ signals, or (iii) both signals $\mathtt{S}$ and $\mathtt{H}$ appear in either order (not necessarily consecutive). We want to trigger an alarm if any of the above patterns is identified in the last 10 minutes, and this 10-minute long window slides every time a new signal arrives. We assume that the signals arrive at regular intervals and hence the window always contains a fixed number of n signals.

In particular, this means that if the alarm is triggered because of an single occurrence of two consecutive $\mathtt{S}$, after 10 minutes this occurrence will fall off from the window and the alarm will stop unless it is triggered again. This problem can be viewed as a sliding window problem where the property is a language over the alphabet $\Sigma = \{\mathtt{S}, \bar{\mathtt{S}}, \mathtt{H}, \mathtt{L}\}$ of sensor values, namely the union of three languages defined by the following regular expressions:

$$\begin{aligned} e_i &= \Sigma^* \cdot \mathtt{S} \cdot (\mathtt{H} + \mathtt{L})^* \cdot \mathtt{S} \cdot \Sigma^* \\ e_{ii} &= \Sigma^* \cdot \mathtt{H} \cdot (\mathtt{S} + \bar{\mathtt{S}})^* \cdot \mathtt{H} \cdot (\mathtt{S} + \bar{\mathtt{S}})^* \cdot \mathtt{H} \cdot \Sigma^* \\ e_{iii} &= \Sigma^* \cdot (\mathtt{S} \cdot \Sigma^* \cdot \mathtt{H} + \mathtt{H} \cdot \Sigma^* \cdot \mathtt{S}) \cdot \Sigma^* \end{aligned}$$

A naïve approach to solve this problem would be to maintain the entire window of 10 minutes using $\Theta(n)$ bits, where n is the window size, and update it on every incoming signal. Then one could verify (offline) whether one of the three expressions match the window. It is not hard to see that there is a more efficient solution for this particular problem. For example, to verify the first property (smoke detector emits two consecutive S signals) one can maintain two numbers $p_1, p_2 \in \{1, \dots, n, \infty\}$ where p_i is the position of the i-th most recent S-signal in the window, if it exists, or $p_i = \infty$ otherwise. The window matches e_i if and only if $p_1, p_2 \leqslant n$, and one can clearly encode p_1 and p_2 using $O(\log n)$ bits. This thesis mainly deals with the question which languages admit space efficient streaming algorithms in the sliding window model.

1.3 Outline

In the following we briefly summarize the main results of this thesis and outline its structure. The thesis can roughly be divided into three main parts.

Deterministic algorithms for regular languages After introducing basic notions from formal languages and automata theory in Chapter 2, we formalize the *fixed-size* and the *variable-size* sliding window model in Chapter 3, and prove basic properties. In contrast to the fixed-size model (covered so far), variable-size windows are controlled by a sequence of operations (insert new symbol or remove the oldest symbol). Our first main result on regular languages is presented in Chapter 4. We prove that the sliding window space complexity of every regular language is either constant, logarithmic, or linear in the window size (both for fixed-size and variable-size windows). Furthermore we give descriptions of the $O(\log n)$ and the $O(1)$ classes. Next, we consider the uniform problem where the regular language is part of the input. We determine the computational complexity of deciding the space complexity, and prove almost tight bounds for the combined complexity, measured in the window size and the size of the automaton. In Chapter 5 we turn from regular languages to rational functions, which are functions computed by finite state transducers. We prove a dichotomy theorem on so-called suffix expansions of rational functions. Using this result we will extend the space trichotomy in the variable-size model to rational functions and later in Chapter 10 to visibly pushdown languages.

Randomized algorithms for regular languages The next part (Chapters 6 to 8) deals with the question if and how randomness can improve the space bounds of sliding window algorithms for languages. In Chapter 6 we present a $O(\log \log n)$ space randomized sliding window algorithm for regular suffix-free languages, and show that almost all other lower bounds from the deterministic setting transfer to the randomized setting. In light of this marginal improvement, we propose the framework of property testing over sliding windows in Chapter 7, which can be viewed as an approximate membership problem. The main result is a constant-space randomized sliding window property tester with two-sided error for all regular languages. Finally, in Chapter 8 we show that randomized sliding window algorithms satisfying a strict correctness definition can always be derandomized.

Context-free and visibly pushdown languages In Chapter 9 we investigate whether the results on regular languages can be extended to subclasses of context-free languages. We construct context-free languages with space complexity $\Theta(n^{1/c})$ for every number c, which indicates that context-free languages deviate strongly from regular languages in this context. The reason for this is that the sliding window space complexity is preserved under taking complements, and co-context free languages can encode computations of Turing machines. We prove that any sliding window algorithm for a context-free language which works in $o(\log n)$ space can in fact be reduced to $O(1)$ space. Finally, in Chapter 10 we prove a space trichotomy in the variable-size model for the class of visibly pushdown languages.

Chapter 2

Preliminaries

2.1 Basic notation

The set of natural numbers, including 0, is denoted by $\mathbb{N}$ and the set of integers is denoted by $\mathbb{Z}$. The logarithm of n to the base 2 is denoted by $\log n$, and we define $\log 0 = 0$. The *symmetric difference* of two sets A, B is defined as $A \triangle B = (A \cup B) \setminus (A \cap B)$. If $f \colon X \to Y$ is a function and $A \subseteq X$ is a subset, then $f(A) = \{f(x) \mid x \in A\}$ denotes the *image* of A under f. If $B \subseteq Y$ is a subset, then $f^{-1}(B) = \{x \in X \mid f(x) \in B\}$ denotes the *pre-image* of B under f. For a *partial function* $f \colon X \to Y$ we denote by $\mathrm{dom}(f) = \{x \in X \mid f(x) \text{ defined}\}$ the *domain* and by $\mathrm{im}(f) = \{f(x) \mid x \in \mathrm{dom}(f)\}$ the *image* of f. We identify a partial function $f \colon X \to Y$ with its *graph* $\{(x, f(x)) \mid x \in \mathrm{dom}(f)\} \subseteq X \times Y$.

Given functions $f, g \colon \mathbb{N}^k \to \mathbb{R}_{\geqslant 0}$, we use the following common asymptotic notation:

- $g(\overline{n}) = O(f(\overline{n})) \iff \exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall \overline{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$ with $n_1, \ldots, n_k \geqslant n_0 \colon g(\overline{n}) \leqslant c \cdot f(\overline{n})$
- $g(\overline{n}) = o(f(\overline{n})) \iff \forall c > 0\ \exists n_0 \in \mathbb{N}\ \forall \overline{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$ with $n_1, \ldots, n_k \geqslant n_0 \colon g(\overline{n}) \leqslant c \cdot f(\overline{n})$
- $g(\overline{n}) = \Omega(f(\overline{n})) \iff f(\overline{n}) = O(g(\overline{n}))$
- $g(\overline{n}) = \omega(f(\overline{n})) \iff f(\overline{n}) = o(g(\overline{n}))$
- $g(\overline{n}) = \Theta(f(\overline{n})) \iff g(\overline{n}) = O(f(\overline{n}))$ and $g(\overline{n}) = \Omega(f(\overline{n}))$

These notations should be viewed as *one-way equalities*. Often we will show lower bounds of the form $g(n) \geqslant c \cdot f(n)$ for some $c > 0$ and infinitely many $n \in \mathbb{N}$. In this case we write "$g(n) = \Omega(f(n))$ for infinitely many n" or "$g(n) = \Omega(f(n))$ infinitely often", see [76] for a discussion. A function $g(n)$ is *polynomial* or *polynomially bounded* if $g(n) = O(n^d)$ for some $d \geqslant 1$. We say that $g(n)$ is *exponential* if $g(n) \geqslant c^n$ for some $c > 1$ and infinitely many $n \in \mathbb{N}$.

2.2 Words, languages, monoids

Words and languages An *alphabet* Σ is a nonempty set of *letters* or *symbols*, which is usually finite. A *word* or *string* over an alphabet Σ is a finite sequence $w = a_1 a_2 \cdots a_n$ of letters $a_1, \dots, a_n \in \Sigma$. The *length* of w is the number $|w| = n$. The *empty word* is denoted by ε whereas small positive numbers are denoted by the lunate epsilon ϵ. The set of all words over Σ is denoted by Σ^*. A subset $L \subseteq \Sigma^*$ is called a *language* over Σ.

Let $w = a_1 \cdots a_n \in \Sigma^*$ be a word. Any word of the form $a_1 \cdots a_i$ is a *prefix* of w, a word of the form $a_i \cdots a_n$ is a *suffix* of w, and a word of the form $a_i \cdots a_j$ is a *factor* of w. The set of all prefixes of a word w is denoted by $\mathrm{Pref}(w)$. For a language L we define $\mathrm{Pref}(L) = \bigcup_{w \in L} \mathrm{Pref}(w)$ to be the set of prefixes of words in L. Similarly, $\mathrm{Suf}(w)$ and $\mathrm{Suf}(L)$ are the sets of suffixes of w and of words in L, respectively. The *reversal* of a word $w = a_1 a_2 \cdots a_n$ is defined as $w^R = a_n \cdots a_2 a_1$ and the *reversal* of a language L is $L^R = \{w^R \mid w \in L\}$. A *factorization* of $w \in \Sigma^*$ is formally a sequence $(w_1, \dots, w_k)$ of words $w_i \in \Sigma^*$ such that $w = w_1 \cdots w_k$. We specify the factorization informally by simply writing $w = w_1 \cdots w_k$.

Monoids A *monoid* $(M, \cdot, 1_M)$ consists of a set M, a binary operation $\cdot \colon M \times M \to M$ and a distinguished element $1 = 1_M \in M$ such that

- $\cdot$ is associative, i.e. $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ for all $x, y, z \in M$,
- 1 is an identity, i.e. $1 \cdot x = x \cdot 1 = x$ for all $x \in M$.

Usually we only specify the set M instead of $(M, \cdot, 1_M)$ and write st instead of $s \cdot t$. A *submonoid* of M is a subset $N \subseteq M$ which is closed under products and contains the identity, i.e. $s, t \in N$ implies $st \in N$ and $1_M \in N$. A *homomorphism* from a monoid M to a monoid N is a function $\varphi \colon M \to N$ such that $\varphi(1_M) = 1_N$ and $\varphi(xy) = \varphi(x)\varphi(y)$ for all $x, y \in M$.

Example 2.1. Let us give some examples for monoids.

- Clearly, every group is a monoid.
- If Q is a nonempty set then the set Q^Q of all functions $\tau \colon Q \to Q$ forms the *full transformation monoid* with function composition $(\sigma \cdot \tau)(q) = \tau(\sigma(q))$ as multiplication and the identity function on Q as the monoid identity.
- The *free monoid* over a set Σ is the set Σ^* with concatenation as the operation and the empty word as its identity. It has the universal property that for every monoid M and for every function $\varphi \colon \Sigma \to M$ there exists a unique extension to a homomorphism $\varphi \colon \Sigma^* \to M$.
- Given two monoids M and N the Cartesian product $M \times N = \{(x, y) \mid x \in M,\ y \in N\}$ forms a monoid, called the *direct product* of M and N, where $(x_1, y_1)(x_2, y_2) = (x_1 x_2, y_1 y_2)$ and $(1_M, 1_N)$ is the identity.

A *right action* of a monoid M on a set Q is a function $\cdot\colon Q \times M \to Q$ such that

- $(q \cdot x) \cdot y = q \cdot (xy)$ for all $q \in Q$ and $x, y \in M$,
- $q \cdot 1 = q$ for all $q \in Q$.

Dually, a *left action* of a monoid M on a set Q is a function $\cdot\colon M \times Q \to Q$ such that

- $x \cdot (y \cdot q) = (xy) \cdot q$ for all $x, y \in M$ and $q \in Q$,
- $1 \cdot q = q$ for all $q \in Q$.

Any function $\delta\colon Q \times \Sigma \to Q$ can be extended to a right action $\cdot\colon Q \times \Sigma^* \to Q$ of the free monoid Σ^* on Q: It is defined by $q \cdot \varepsilon = q$ and $q \cdot (wa) = \delta(q \cdot w, a)$ for all $q \in Q$, $w \in \Sigma^*$ and $a \in \Sigma$. Dually a function $\delta\colon \Sigma \times Q \to Q$ can be extended to a left action $\cdot\colon \Sigma^* \times Q \to Q$ by $\varepsilon \cdot q = q$ and $(aw) \cdot q = \delta(a, w \cdot q)$ for all $q \in Q$, $w \in \Sigma^*$ and $a \in \Sigma$.

The product of two sets $X, Y \subseteq M$ is $X \cdot Y = XY = \{xy \mid x \in X,\, y \in Y\}$. The power X^n is defined inductively by $X^0 = \{1_M\}$ and $X^{n+1} = X^n X$ for all $n \in \mathbb{N}$. The submonoid generated by a subset $X \subseteq M$ is the inclusion-wise smallest submonoid containing X, or equivalently, the set $X^* = \bigcup_{n \in \mathbb{N}} X^n$ of all products over X. It is also called the *Kleene-star* of X. The set of products of length at most n is denoted by $X^{\leqslant n} = \bigcup_{0 \leqslant k \leqslant n} X^k$. Similarly, we define the sets $X^{\geqslant n}$, $X^{<n}$ and $X^{>n}$.

2.3 Automata and regular languages

For a good introduction into the theory of formal languages and automata we refer to [18, 66, 78].

Automata An *automaton over a monoid* M is a tuple $\mathcal{A} = (Q, M, Q_-, \Delta, Q_+)$ where Q is the set of *states*, $Q_-, Q_+ \subseteq Q$ are designated sets of states, and $\Delta \subseteq Q \times M \times Q$ is the set of *transitions*. A *run* of $\mathcal{A}$ on an element $m \in M$ is a finite sequence $\pi = q_0 a_1 q_1 a_2 q_2 \cdots q_{n-1} a_n q_n \in Q(MQ)^*$ such that $m = a_1 \cdots a_n$ and $(q_{i-1}, a_i, q_i) \in \Delta$ for all $1 \leqslant i \leqslant n$. We call π *successful* if $q_0 \in Q_-$ and $q_n \in Q_+$. The subset accepted by $\mathcal{A}$ is defined as

$$L(\mathcal{A}) = \{m \in M \mid \text{there exists a successful run of } \mathcal{A} \text{ on } m\}.$$

If $\pi = q_0 a_1 q_1 \cdots a_n q_n$ and $\rho = p_0 b_1 p_1 \cdots b_\ell p_\ell$ are runs such that $q_n = p_0$ then their *composition* $\pi\rho$ is defined as $\pi\rho = q_0 a_1 \cdots a_n p_0 b_1 \cdots b_\ell p_\ell$. If Q and Δ are finite then $\mathcal{A}$ is *finite* and its *size* $|\mathcal{A}|$ is defined as the number of states. If $|Q_-| = 1$ or $|Q_+| = 1$ we only specify that particular state in the definition of $\mathcal{A}$ instead of Q_- or Q_+, respectively. We call $\mathcal{A}$ *trim* if every state in $\mathcal{A}$ occurs in some successful run. Clearly, one can make $\mathcal{A}$ trim by removing all states which do not occur in a successful run without changing $L(\mathcal{A})$.

We will view an automaton either as a *left automaton* or a *right automaton* to distinguish whether we view the automaton as reading the input from left to right, or from right to left. To prevent confusion we usually denote left automata with the letter $\mathcal{A}$ and right automata with the letter $\mathcal{B}$. We emphasize that left automata and right automata are formally the same objects.

Let $\mathcal{A} = (Q, M, I, \Delta, F)$ be a left automaton. We call I the set of *initial states* and F the set of *final states*. A transition $(p, a, q) \in \Delta$ is viewed as going from p to q and is depicted by $p \xrightarrow{a} q$. A run $\pi = q_0 a_1 q_1 \cdots q_{n-1} a_n q_n$ in $\mathcal{A}$ is said to go from q_0 to q_n and is depicted by

$$\pi\colon q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots q_{n-1} \xrightarrow{a_n} q_n,$$

or simply $\pi\colon q_0 \xrightarrow{a_1 \cdots a_n} q_n$. If $q_0 \in I$ then π is *initial*; if $q_n \in F$ then π is *final*. Now let $\mathcal{B} = (Q, M, F, \Delta, I)$ be a right automaton. We call I the set of *initial states* and F the set of *final states*. A transition $(p, a, q) \in \Delta$ is viewed as going from q to p and is depicted by a labeled arrow $p \xleftarrow{a} q$ pointing to the left. A run $\pi = q_0 a_1 q_1 \cdots q_{n-1} a_n q_n$ in $\mathcal{B}$ is said to go from q_n to q_0 and is depicted by

$$\pi\colon q_0 \xleftarrow{a_1} q_1 \xleftarrow{a_2} q_2 \cdots q_{n-1} \xleftarrow{a_n} q_n,$$

or simply $\pi\colon q_0 \xleftarrow{a_1 \cdots a_n} q_n$. If $q_n \in I$ then π is *initial*; if $q_0 \in F$ then π is *final*.

Notice that the distinction between left and right automata only affects the definitions of initial and final states, and the direction in which we read and write runs, but not the semantics, i.e. the accepted language.

Rational subsets A subset $L \subseteq M$ of a monoid M is *rational* if it is accepted by a finite automaton $\mathcal{A}$ over M. Rational languages $L \subseteq \Sigma^*$ are called *regular*. The class of all regular languages is denoted by **Reg**. A *nondeterministic finite automaton (NFA)* is a finite automaton $\mathcal{A} = (Q, \Sigma^*, Q_-, \Delta, Q_+)$ over a free monoid Σ^* with $\Delta \subseteq Q \times \Sigma \times Q$, which is written in the form $\mathcal{A} = (Q, \Sigma, Q_-, \Delta, Q_+)$. A language is regular if and only if it is recognized by an NFA.

It is known that the set of rational subsets over a monoid M is the smallest set containing all finite subsets $L \subseteq M$ which is closed under union, products and Kleene-star. A *regular expression* α over M is built from single monoid elements, union $\cup$ (or also $+$), product $\cdot$ and Kleene-star *, which defines a subset $L(\alpha) \subseteq M$ in the natural way. Hence a subset in M is rational if and only if it is definable by a regular expression. This statement was proved by Kleene for the case of free monoids [75], and the general case is a straightforward adaption, cf. [57].

The rational subsets over a *commutative* monoid $(M, +)$ are precisely the *semilinear sets*, which are finite unions of *linear sets* $a + B^* = \{a + b \mid b \in B^*\}$ where $B \subseteq M$ is finite [42]. For example, linear sets over the free commutative monoid $(\mathbb{N}^d, +)$ have the form

$$\{\overline{u} + \sum_{i=1}^{m} k_i \overline{v}_i \mid k_1, \dots, k_m \in \mathbb{N}\}$$

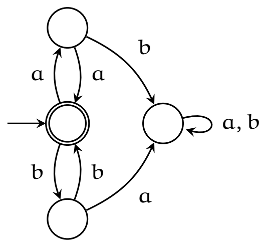

Figure 2.1: A deterministic finite automaton for the language $\{aa, bb\}^*$.

where $\bar{u}, \bar{v}_1, \dots, \bar{v}_m \in \mathbb{N}^d$. In dimension $d = 1$ it is easy to see that the semilinear sets over $(\mathbb{N}, +)$ are precisely finite unions of singleton sets and *arithmetic progressions* $c + d\mathbb{N} = \{c + dn \mid n \in \mathbb{N}\}$, for $c \in \mathbb{N}$, $d \geqslant 1$.

Deterministic automata A *(left-)deterministic automaton* is a left automaton of the form $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ where for all $p \in Q$ and $a \in \Sigma$ there exists exactly one transition $(p, a, q) \in \delta$. We view δ as a *transition function* $\delta \colon Q \times \Sigma \to Q$. The transition function δ can be extended to a right action $\cdot \colon Q \times \Sigma^* \to Q$ of the free monoid Σ^* on the state set Q. We write $\mathcal{A}(w)$ instead of $q_0 \cdot w$. *Deterministic finite automata (DFAs)* are the standard description for regular languages. An example for a DFA can be seen in Figure 2.1.

It is known that any NFA can be turned into an equivalent DFA by the powerset construction. If $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ is a NFA then the *powerset automaton* $2^{\mathcal{A}} = (2^Q, \Sigma, I, \delta, \mathcal{F})$ is defined by $\delta(P, a) = \{q \in Q \mid p \in P,\ (p, a, q) \in \Delta\}$ for all $P \in 2^Q$ and $a \in \Sigma$, and $\mathcal{F} = \{P \in 2^Q \mid P \cap F \neq \emptyset\}$. It is a DFA which equivalent to $\mathcal{A}$ and has $2^{|\mathcal{A}|}$ many states.

Equivalence relations For any equivalence relation $\sim$ on a set X we write $[x]_\sim$ for the $\sim$-class containing $x \in X$. If $Y \subseteq X$ is a subset then $Y/{\sim} = \{[y]_\sim \mid y \in Y\}$ is the set of all $\sim$-classes of elements in Y (the *quotient set* of Y). The *index* of $\sim$ is the cardinality of $X/{\sim}$. We denote by $\nu_\sim \colon X \to X/{\sim}$ the canonical projection with $\nu_\sim(x) = [x]_\sim$. If $\sim_1$ and $\sim_2$ are equivalence relations on a set X with $\sim_1 \subseteq \sim_2$ then $\sim_1$ is called *finer* than $\sim_2$ (or $\sim_1$ *refines* $\sim_2$), and $\sim_2$ is called *coarser* than $\sim_1$. The *intersection* $\sim_1 \cap \sim_2$ is again an equivalence relation on X. An equivalence relation $\sim$ on X *saturates* a subset $L \subseteq X$ if L is a union of $\sim$-classes, i.e. $L = \bigcup L/{\sim}$. The *kernel* of a function $f \colon X \to Y$ is the equivalence relation $\ker(f) = \{(x, x') \in X^2 \mid f(x) = f(x')\}$.

Congruences An equivalence relation $\sim$ on a monoid M is a *right (left) congruence* if $x \sim y$ implies $xz \sim yz$ ($zx \sim zy$), and it is a *congruence* if it is both a right congruence and a left congruence. If $\sim$ is a congruence on M then the set of $\sim$-classes $M/{\sim}$ forms again a monoid with the well-defined multiplication $[x]_\sim [y]_\sim = [xy]_\sim$.

There is a correspondence between right congruences and deterministic automata. Any deterministic automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ defines a right congruence $\sim_{\mathcal{A}}$ on Σ^* by $u \sim_{\mathcal{A}} v$ if and only if $\mathcal{A}(u) = \mathcal{A}(v)$, which saturates $L(\mathcal{A})$. Conversely, if $\sim$ is a right congruence on Σ^* which saturates a language $L \subseteq \Sigma^*$ then $\mathcal{A}_{L,\sim} = (\Sigma^*/\sim, \Sigma, [\varepsilon]_\sim, \delta, L/\sim)$ is a deterministic automaton for L where $\delta([w]_\sim, a) = [wa]_\sim$ for all $w \in \Sigma^*$ and $a \in \Sigma$.

For any language $L \subseteq \Sigma^*$ we define the *Myhill-Nerode right congruence* $\sim_L$ on Σ^* by

$$u \sim_L v \iff (uw \in L \iff vw \in L) \text{ for all } w \in \Sigma^*,$$

which is the coarsest right congruence on Σ^* saturating L. The Myhill-Nerode theorem states that L is regular if and only if $\sim_L$ has finite index [78]. The *minimal deterministic automaton* for L is $\mathcal{A}_L := \mathcal{A}_{L,\sim_L}$. It is the coarsest deterministic automaton for L (up to isomorphism) in the sense that $\sim_{\mathcal{A}_L}$ is coarser than $\sim_{\mathcal{A}}$ for any deterministic automaton $\mathcal{A}$ for L.

Syntactic monoid The *syntactic congruence* of a language $L \subseteq \Sigma^*$ is the equivalence relation defined by

$$u \equiv_L v \iff (xuy \in L \iff xvy \in L \text{ for all } x, y \in \Sigma^*).$$

It is finer than the Myhill-Nerode right congruence $\sim_L$. The monoid $\Sigma^*/{\equiv_L}$ is called the *syntactic monoid* of L and $\nu_{\equiv_L} \colon \Sigma^* \to \Sigma^*/{\equiv_L}$ is called the *syntactic homomorphism* of L. If $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is the minimal deterministic automaton for L then the syntactic monoid of L can be derived from $\mathcal{A}$ (and computed if $\mathcal{A}$ is finite). Every word $w \in \Sigma^*$ induces a state transformation $\tau(w) \colon Q \to Q$ by $(\tau(w))(q) = q \cdot w$. Since $\cdot$ is a right action on Q, the function $\tau \colon \Sigma^* \to Q^Q$ is a homomorphism from the free monoid Σ^* into the monoid Q^Q of all functions $\tau \colon Q \to Q$. Its image $\tau(\Sigma^*)$ is the *transition monoid* of $\mathcal{A}$ and is isomorphic to the syntactic monoid of L.

2.4 Context-free languages

A *context-free grammar* $\mathcal{G} = (N, \Sigma, S, P)$ over an alphabet Σ consists of a finite set of *nonterminals* or *variables* N (which is disjoint from Σ), a *start nonterminal* $S \in N$ and a set of *productions* $P \subseteq N \times (\Sigma \cup N)^*$. Productions $(A, \alpha) \in P$ are written as $A \to \alpha$. The one-step derivation $\Rightarrow_{\mathcal{G}} \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$ is defined by $x \Rightarrow_{\mathcal{G}} y$ if there exist $u, v \in (\Sigma \cup N)^*$ and a production $A \to \alpha \in P$ such that $x = uAv$ and $y = u\alpha v$. Its transitive reflexive closure is the derivation relation $\xRightarrow{*}_{\mathcal{G}}$. The language defined by $\mathcal{G}$ is

$$L(\mathcal{G}) = \{w \in \Sigma^* \mid S \xRightarrow{*}_{\mathcal{G}} w\}.$$

Languages $L \subseteq \Sigma^*$ defined by context-free grammars are called *context-free*. Given a context-free grammar $\mathcal{G} = (N, \Sigma, S, P)$, a *derivation tree* for $w \in \Sigma^*$ is a node-labeled rooted ordered tree with the following properties:

- Inner nodes are labeled by nonterminals $A \in N$ and the root is labeled by the start nonterminal S.

- Leaves are labeled by letters in Σ or ε. In the latter case, the leaf must be the only child of its parent.
- If a node s has children $s_1, \dots, s_k$ where s is labeled by A and $s_1, \dots, s_k$ are labeled by $\alpha_1, \dots, \alpha_k$ then there exists a production $A \to_G \alpha_1 \cdots \alpha_k$.
- If $a_1, \dots, a_\ell$ are the labels of the leaves read from left to right then $w = a_1 \cdots a_\ell$.

An important result in formal language theory is Parikh's theorem. Let $\Sigma = \{a_1, \dots, a_k\}$ be an ordered alphabet. Then the *Parikh mapping* $\Psi \colon \Sigma^* \to \mathbb{N}^k$ is defined by

$$\Psi(w) = (|w|_{a_1}, \dots, |w|_{a_k})$$

where $|w|_{a_i}$ is the number of occurrences of a_i in w.

Theorem 2.2 ([94]). *If L is context-free then its Parikh image $\Psi(L)$ is semilinear.*

Language growth The *growth (function)* of a language $L \subseteq \Sigma^*$ is the function $g(n) = |L \cap \Sigma^n|$, i.e. it counts the number of words in L of length n. The *cumulative growth (function)* of L is the function $g(n) = |L \cap \Sigma^{\leqslant n}|$. We remark that some authors also refer to the cumulative growth of a language as its growth. For most parts of this work the distinction between growth and cumulative growth is negligible because we only distinguish between polynomial and exponential growth. Notice that the following relations hold:

$$|L \cap \Sigma^n| \leqslant |L \cap \Sigma^{\leqslant n}| = \sum_{i=0}^{n} |L \cap \Sigma^i|$$

Lemma 2.3. *Let $g(n), G(n)$ be a functions such that $g(n) \leqslant G(n) \leqslant \sum_{k=0}^{n} g(k)$ for all $n \in \mathbb{N}$. Then $g(n)$ is polynomial (exponential) if and only if $G(n)$ is polynomial (exponential).*

Proof. If $g(n) = O(n^d)$ then $G(n) = O(n^{d+1})$. The only nontrivial statement left to show is that, if $G(n)$ is exponential then also $g(n)$ is exponential. Assume that $G(n) \geqslant c^n$ for some $c > 1$ and infinitely many $n \in \mathbb{N}$. By averaging we obtain that for infinitely many $n \in \mathbb{N}$ there exists $k \leqslant n$ such that

$$g(k) \geqslant \frac{c^n}{n}.$$

If n is large enough then $c^n/n \geqslant d^n$ for some $1 < d < c$. Let P be the set of all pairs (n, k) with $k \leqslant n$ which satisfy the inequality above. Since d^n tends towards infinity there must be infinite number of k-components in P. In other words, there exist infinitely many k and $n \geqslant k$ such that $g(k) \geqslant d^n \geqslant d^k$. This shows that $g(n)$ is exponential. $\square$

Corollary 2.4. *The growth of a language is polynomial (exponential) if and only if its cumulative growth is polynomial (exponential).*

A language L is a *bounded* language if there exist words $w_1, \dots, w_k$ such that $L \subseteq w_1^* \cdots w_k^*$. The classic growth theorem for context-free languages states that all context-free languages have either polynomial or exponential growth. We summarize related results in the following theorem.

Theorem 2.5 ([55, 59, 107, 110])**.** *Let L be a context-free language.*

- *The (cumulative) growth of L is either polynomial or exponential.*
- *L has polynomial growth if and only if L is a bounded language.*
- *If L has polynomial growth then there exists $d \in \mathbb{N}$ such that the growth of L is $O(n^d)$, and $\Omega(n^d)$ infinitely often (d is the order of growth).*
- *If L has exponential growth then its cumulative growth is $2^{\Omega(n)}$.*

2.5 Rational transductions

Let Σ be a finite *input alphabet* and Ω be a finite *output alphabet*. A *transduction* is any relation $T \subseteq \Sigma^* \times \Omega^*$. A transduction $T \subseteq \Sigma^* \times \Omega^*$ is *rational* if it is a rational subset of the product monoid $\Sigma^* \times \Omega^*$. Hence rational transductions are accepted by finite automata over $\Sigma^* \times \Omega^*$, which are called finite state transducers. A *rational function* is a partial function $t \colon \Sigma^* \to \Omega^*$ whose graph $\{(x, t(x)) \mid x \in \mathrm{dom}(t)\}$ is a rational transduction. Here we define transducers as finite automata over $\Sigma^* \times \Omega^*$ which can append a word to the output at the end of a run.

Left and right transducers A *left transducer* is a tuple $\mathcal{A} = (Q, \Sigma, \Omega, I, \Delta, F, o)$ such that $(Q, \Sigma^* \times \Omega^*, I, \Delta, F)$ is an automaton over $\Sigma^* \times \Omega^*$ and $o \colon F \to \Omega^*$ is a *terminal output function*. The notion of runs is inherited from the underlying automaton over $\Sigma^* \times \Omega^*$. To omit parentheses we write runs $p \xrightarrow{(x,y)} q$ in the form $p \xrightarrow{x|y} q$ and depict $o(q) = y$ by a transition $q \xrightarrow{|y}$ without input word and target state. If π is a run $p \xrightarrow{x|y} q$ we also say that π is a run on $x \in \Sigma^*$, and we define $\mathsf{out}(\pi) = y$ and $\mathsf{out}_F(\pi) = y\, o(q)$. The transduction defined by $\mathcal{A}$ is

$$T(\mathcal{A}) = \{(x, \mathsf{out}_F(\pi)) \mid \text{there exists a successful run } \pi \text{ in } \mathcal{A} \text{ on } x \in \Sigma^*\}.$$

Since the terminal output function can be eliminated by ε-transitions, left transducers precisely recognize all rational transductions.

Example 2.6. Let $\Sigma = \Omega = \{a, b\}$. The transduction

$$T = \{(aw, wa) \mid w \in \Sigma^*\} \cup \{(bw, wb) \mid w \in \Sigma^*\}$$

is rational as witnessed by the left transducer in Figure 2.2. It takes a nonempty word over Σ as input and shifts the leftmost symbol to the right end.

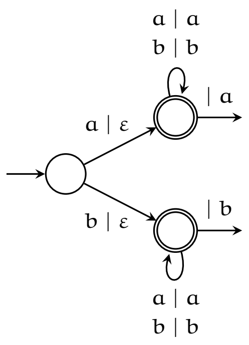

Figure 2.2: A transducer which shifts the leftmost symbol to the right end.

A *right transducer* is a tuple $\mathcal{A} = (Q, \Sigma, \Omega, F, \Delta, I, o)$ such that $(Q, \Sigma^* \times \Omega^*, F, \Delta, I)$ is a right automaton over $\Sigma^* \times \Omega^*$ and $o \colon F \to \Omega^*$ is a *terminal output function*. Runs are viewed as going from right and left, and we depict $o(q) = y$ by a transition $\xleftarrow{|y} q$. If π is a run $q \xleftarrow{x|y} p$ we define $\mathrm{out}(\pi) = y$ and $\mathrm{out}_F(\pi) = o(q)\, y$. The transduction defined by $\mathcal{A}$ is defined precisely as for left transducers, i.e. the set $T(\mathcal{A})$ of all pairs $(x, \mathrm{out}_F(\pi))$ such that π is a successful run in $\mathcal{A}$ on x.

Closure properties Similar to the regular languages, the class of rational transductions enjoy good closure properties [18]. The class of rational transductions is closed under inverse, reversal and composition where the *inverse* of T is $T^{-1} = \{(y, x) \mid (x, y) \in T\}$, the *reversal* of T is $T^R = \{(x^R, y^R) \mid (x, y) \in T\}$, and the composition of two transductions T_1, T_2 is

$$T_1 \circ T_2 = \{(x, z) \mid \text{there exists } y \text{ such that } (x, y) \in T_1 \text{ and } (y, z) \in T_2\}.$$

If $T \subseteq \Sigma^* \times \Omega^*$ is rational and $L \subseteq \Sigma^*$ is regular then the restriction $\{(x, y) \in T \mid x \in L\}$ is also rational. If $K \subseteq \Sigma^*$ is regular (context-free) and $T \subseteq \Sigma^* \times \Omega^*$ is rational then $TK = \{y \in \Omega^* \mid (x, y) \in T \text{ for some } x \in K\}$ is also regular (context-free). In particular the domain $\{x \mid (x, y) \in T\}$ and the image $\{y \mid (x, y) \in T\}$ of every rational transduction T is regular.

Left and right-subsequential functions Let $\mathcal{A} = (Q, \Sigma, \Omega, Q_-, \Delta, Q_+, o)$ be a (left or right) transducer. We call $\mathcal{A}$ *real-time* if $\Delta \subseteq Q \times \Sigma \times \Omega^* \times Q$. A *left-subsequential transducer* is a real-time left transducer $\mathcal{A} = (Q, \Sigma, \Omega, q_0, \Delta, F, o)$ which is *deterministic*, i.e. for every $p \in Q$ and $a \in \Sigma$ there exists at most one transition $p \xrightarrow{a|y} q$. A *right-subsequential transducer* is a real-time right transducer $\mathcal{A} = (Q, \Sigma, \Omega, F, \Delta, q_0, o)$ which is *deterministic*, i.e. for every $p \in Q$ and $a \in \Sigma$ there exists at most one transition $q \xleftarrow{a|y} p$. Clearly, left- and right-subsequential transducers define rational functions, the so-called *left- and right-subsequential functions*.

A result by Elgot and Mezei [43] states that every rational function t can be decomposed into a left- and a right-subsequential function, i.e. $t = \ell \circ r$ for some left-subsequential function ℓ and some right-subsequential function r. Reutenauer and Schützenberger [100] strengthened this result by making this decomposition canonical using Eilenberg's bimachines [40].

Synchronous rational relations Synchronous rational relations are a subclass of rational relations with better closure properties [74]. They are recognized by automata with k synchronous heads where k is the arity of the relation. Fix an alphabet Σ and let $\diamond \notin \Sigma$ be a fresh symbol. Given words $w_1 = a_{1,1} \cdots a_{1,n_1}, \dots, w_k = a_{k,1} \cdots a_{k,n_k}$ over Σ with lengths $n_1, \dots, n_k \in \mathbb{N}$ and maximum length $n = \max\{n_1, \dots, n_k\}$. We define the *convolution* of $w_1, \dots, w_k$ by

$$\otimes(w_1, \dots, w_k) = \begin{pmatrix} a'_{1,1} \\ a'_{2,1} \\ \vdots \\ a'_{k,1} \end{pmatrix} \begin{pmatrix} a'_{1,2} \\ a'_{2,2} \\ \vdots \\ a'_{k,2} \end{pmatrix} \cdots \begin{pmatrix} a'_{1,n_1} \\ a'_{2,n_2} \\ \vdots \\ a'_{k,n_k} \end{pmatrix}$$

where $a'_{i,j} = a_{i,j}$ for $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant n_i$ and $a'_{i,j} = \diamond$ otherwise. It is a word of length n over the alphabet $(\Sigma \cup \{\diamond\})^k$. The convolution of a k-ary relation $R \subseteq (\Sigma^*)^k$ is $\otimes R = \{\otimes(w_1, \dots, w_k) \mid (w_1, \dots, w_k) \in R\}$. A relation $R \subseteq (\Sigma^*)^k$ is *synchronous rational* (or *automatic*) if its convolution $\otimes R$ is a regular language over $(\Sigma \cup \{\diamond\})^k$. Synchronous rational relations form the basis for automatic structures [74]. Their good closure properties lead to decision procedures for automatic structures.

Proposition 2.7 ([74]). *Let $R, R_1, R_2 \subseteq (\Sigma^*)^k$ be synchronous rational k-ary relations. Then $(\Sigma^*)^k \setminus R$ and $R_1 \cup R_2$ are synchronous rational. For every $1 \leqslant i \leqslant k$ the $(k-1)$-ary relation $\{(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k) \mid (x_1, \dots, x_k) \in R\}$ is synchronous rational.*

Chapter 3

The sliding window model

3.1 Streaming algorithms

Before introducing sliding window algorithms we formalize the notion of streaming algorithms. A *stream* is a finite sequence of elements $a_1 \cdots a_m$, which arrive element per element from left to right. We make the assumption that these elements are from a finite set Σ, i.e. a stream is simply a finite word $x = a_1 a_2 \cdots a_m \in \Sigma^*$. Furthermore we assume that the symbols arrive in discrete time steps, i.e. at time instant $0 \leqslant t \leqslant m$ the prefix $a_1 \cdots a_t$ has been read. A *computational problem* is a function $\varphi \colon \Sigma^* \to Y$ where Σ is a finite alphabet and Y is a possibly infinite set of output values. In the simplest case the output is Boolean, i.e. $Y = \{0, 1\}$, and $\varphi \colon \Sigma^* \to \{0, 1\}$ is the *decision problem* or *membership problem* for the language $L = \{w \in \Sigma^* \mid \varphi(w) = 1\}$. We identify a language $L \subseteq \Sigma^*$ with its characteristic function $\chi_L \colon \Sigma^* \to \{0, 1\}$. Another example is the basic counting problem *count*: $\{0, 1\}^* \to \mathbb{N}$ where $\mathit{count}(a_1 \cdots a_n) = \sum_{i=1}^{n} a_i$ counts the number of 1's in the input bitstring $a_1 \cdots a_n$.

In contrast to classical (offline) algorithms, streaming algorithms may read the input only once from left to right. The algorithm maintains some data structure in memory. In the beginning the algorithm may perform some computation to build up the initial data structure. Then, at every time instant $1 \leqslant t \leqslant m$ the algorithm has only access to the symbol a_t and the current data structure and updates the data structure. We will mostly abstract away from the actual computation and only analyze the space requirement. The current data structure in memory will be abstractly referred to as the current *(memory) state* of the algorithm. Furthermore, the streaming algorithm produces an output value $y_t \in Y$ from the memory state at every time instant $0 \leqslant t \leqslant m$. Most streaming algorithms use randomness, i.e. the memory update operation can depend on coin tosses.

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

3.1.1 Streaming algorithms as automata

As a formal and abstract model of streaming algorithms we extend the classical notions of automata which either accept or reject inputs to automata with arbitrary output. Initially we will focus on deterministic streaming algorithms and discuss randomized streaming algorithms in Chapter 6. A *deterministic streaming algorithm* or a *deterministic automaton with output* $\mathcal{P} = (M, \Sigma, m_0, \delta, o)$ consists of

- a set M of *memory states*,
- a finite alphabet Σ,
- an initial state $m_0 \in M$,
- a transition function $\delta \colon M \times \Sigma \to M$, which extends to a right action $\cdot \colon M \times \Sigma^* \to M$,
- and an output function $o \colon M \to Y$ into some set Y of output values.

The letter $\mathcal{P}$ stands for *program*. We view (standard) deterministic automata as deterministic streaming algorithms with output function $o \colon M \to \{0,1\}$. As for deterministic automata we define $\mathcal{P}(x) = m_0 \cdot x$. The function $\varphi_{\mathcal{P}} \colon \Sigma^* \to Y$ computed by $\mathcal{P}$ is given by $\varphi_{\mathcal{P}}(x) = o(\mathcal{P}(x))$.

We are mainly interested in space complexity of streaming algorithms. The *space* of $\mathcal{P}$ (or *number of bits used by* $\mathcal{P}$) is given by $s(\mathcal{P}) = \log|M| \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$. If $s(\mathcal{P}) = \infty$ we will later measure the space restricted to input streams of bounded length or bounded window size. If a deterministic algorithm reads its input from left to right and uses $O(s)$ bits, say a Turing machine on its work tape or a random access machine in its registers, then it can be regarded as a deterministic streaming algorithm $\mathcal{P}$ as above with space complexity $s(\mathcal{P}) = O(s)$. Conversely, if $\mathcal{P}$ is a deterministic algorithm with finitely many memory states, then the memory states of $\mathcal{P}$ can be encoded using $O(s(\mathcal{P}))$ bits.

As in the case of languages there exists a canonical minimal deterministic automaton for every function $\varphi \colon \Sigma^* \to Y$. Any deterministic automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, o)$ for φ induces a right congruence $\sim_{\mathcal{A}}$ on Σ^* which refines $\ker(\varphi)$, namely

$$u \sim_{\mathcal{A}} v \iff \mathcal{A}(u) = \mathcal{A}(v).$$

Conversely, if $\sim$ is a right congruence on Σ^* which saturates $\ker(\varphi)$ then one can define a deterministic automaton with output $\mathcal{A}_{\varphi,\sim} = (\Sigma^*/{\sim}, \Sigma, [\varepsilon]_\sim, \delta, \bar{\varphi})$ which computes φ as follows. Set $\delta([w]_\sim, a) = [wa]_\sim$ for all $w \in \Sigma^*$ and $a \in \Sigma$, and $\bar{\varphi}([w]_\sim) = \varphi(w)$ for all $w \in \Sigma^*$. The Myhill-Nerode right congruence $\sim_\varphi$ on Σ^* is defined by

$$u \sim_\varphi v \iff \varphi(uw) = \varphi(vw) \text{ for all } w \in \Sigma^*,$$

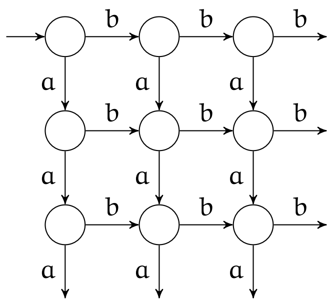

Figure 3.1: The minimal deterministic automaton for counting the maximum number of a's and b's.

which is the coarsest right congruence on Σ^* which refines $\ker(\varphi)$. The *minimal deterministic automaton with output* for φ is $\mathcal{A}_\varphi := \mathcal{A}_{\varphi,\sim_\varphi}$. It is the coarsest deterministic automaton for φ (up to isomorphism) in the sense that $\sim_{\mathcal{A}_\varphi}$ is coarser than $\sim_{\mathcal{A}}$ for any deterministic automaton $\mathcal{A}$ for φ.

Example 3.1. Consider the function $\varphi\colon \{a,b\}^* \to \mathbb{N}$ which counts the maximum number of a's or b's. Then $x \sim_\varphi y$ if and only if x and y contain the same number of a's and b's. Therefore the minimal deterministic automaton for φ needs to store the number of a's and b's read so far (the state space is $\mathbb{N} \times \mathbb{N}$) and outputs in state (n_a, n_b) the maximum of n_a and n_b. An excerpt of the automaton is displayed in Figure 3.1.

3.2 Fixed-size sliding window model

We fix an arbitrary padding symbol $\Box \in \Sigma$. Given a stream $x = a_1a_2\cdots a_m \in \Sigma^*$ and a *window length* or *window size* $n \in \mathbb{N}$, we define $\mathrm{last}_n(x) \in \Sigma^n$ by

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

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

Let $\varphi\colon \Sigma^* \to Y$ be a computational problem. The *sliding window problem* $\mathrm{SW}_n(\varphi)$ for φ and *window length* $n \in \mathbb{N}$ is the function $\varphi \circ \mathrm{last}_n\colon \Sigma^* \to Y$. A *sliding window algorithm (SW-algorithm)* for φ and window length $n \in \mathbb{N}$ is a streaming algorithm for $\mathrm{SW}_n(\varphi)$. The function $F_\varphi\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0} \cup \{\infty\}$ is defined by

$$F_\varphi(n) = \inf\{s(\mathcal{P}_n) \mid \mathcal{P}_n \text{ is an SW-algorithm for } \varphi \text{ and window length } n\}.$$

It is called the *space complexity* of φ *in the fixed-size sliding window model.*

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

Recall that we identify a language $L \subseteq \Sigma^*$ with its characteristic function $\chi_L \colon \Sigma^* \to \{0,1\}$. Therefore the sliding window problem for L is

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

A subtle point is that the space complexity $F_L(n)$ of a language L may depend on the underlying alphabet. This becomes apparent in Example 3.3 below, if the alphabet is restricted.

Lemma 3.2. *For any problem φ we have* $F_\varphi(n) = O(n)$.

Proof. A trivial SW-algorithm $\mathcal{P}_n$ for φ explicitly stores the active window of length n in a queue so that its value under φ can be computed. Formally, the state set of $\mathcal{P}_n$ is Σ^n and it has transitions of the form $au \xrightarrow{b} ub$ for $a, b \in \Sigma$, $u \in \Sigma^{n-1}$. Viewed as an edge-labeled graph this automaton is also known under the name *de Bruijn graph* [25]. Since every word $w \in \Sigma^n$ can be encoded with $O(\log|\Sigma| \cdot n)$ bits the algorithm uses $O(n)$ bits. ☐

Depending on the function φ there are more space efficient solutions. Usually sliding window algorithms are devised in the following way:

1. Specify some information or property $I(w)$ of the active window w and show that it can be *maintained* by a streaming algorithm. This means that given $I(bw)$ and $a \in \Sigma$ one can compute $I(wa)$.
2. Show that one can compute $\varphi(w)$ from the information $I(w)$.

Let us give simple examples of sliding window algorithms for regular languages.

Example 3.3. Let $\Sigma = \{a, b\}$ be the alphabet.

(i) Let $L = \Sigma^* a$ be the set of all words ending with a. Then $F_L(n) = O(1)$ because a streaming algorithm can maintain the last symbol of the stream, which is also the last symbol of the active window, in a single bit (either a or b).

(ii) Let $L = \Sigma^* a \Sigma^*$ be the set of all words containing a and let $n \in \mathbb{N}$ be the window size. A streaming algorithm can maintain the position $1 \leqslant i \leqslant n$ (from the right) of the most recent a-symbol in the window or set $i = \infty$ if the window contains no a-symbols. Depending on the chosen initial window we initialize i appropriately. On input a we set $i := 1$ and on input b we increment i and then set $i := \infty$ if $i > n$. The algorithm accepts if and only if $i \leqslant n$. Since the position i can be stored using $O(\log n)$ bits we have shown $F_L(n) = O(\log n)$.

(iii) Let $L = a\Sigma^*$ be the set of all words starting with a. We claim that $F_L(n) = \Theta(n)$. Since the trivial sliding window algorithm uses $O(n)$ bits it suffices to prove the lower bound. Let $\mathcal{P}_n$ be any SW-algorithm for L and window length $n \in \mathbb{N}$. We claim that $\mathcal{P}_n(x) \neq \mathcal{P}_n(y)$ for all $x, y \in \Sigma^n$ with $x \neq y$. Let $x = a_1 \cdots a_n \in \Sigma^n$ and $y = b_1 \cdots b_n \in \Sigma^n$ with $a_i \neq b_i$ for some $1 \leqslant i \leqslant n$. Since exactly one of the words $\mathrm{last}_n(xb^{i-1}) = a_i \cdots a_n b^{i-1}$ and $\mathrm{last}_n(yb^{i-1}) = b_i \cdots b_n b^{i-1}$ belongs to L the algorithm $\mathcal{P}_n$ must accept exactly one of the words xb^{i-1} and yb^{i-1}. Since $\mathcal{P}_n$ is deterministic, $\mathcal{P}_n(x) = \mathcal{P}_n(y)$ would imply $\mathcal{P}_n(xb^{i-1}) = \mathcal{P}_n(yb^{i-1})$, which proves the claim. Therefore $\mathcal{P}_n$ must contain at least $|\Sigma^n| = 2^n$ memory states and hence $F_L(n) = \Omega(n)$.

Notice that the complexity function $F_\varphi(n)$ is not necessarily monotonic. For instance, let L be the intersection of $a\Sigma^*$ and the set of words with even length. By Example 3.3(iii), we have $F_L(2n) = \Theta(n)$ but clearly we have $F_L(2n+1) = O(1)$ since for odd window lengths the algorithm can always reject. Therefore $F_L(n) = \Omega(n)$ only holds for infinitely many $n \in \mathbb{N}$.

Note that the fixed-size sliding window model is a *nonuniform* model: for every window size we have a separate streaming algorithm and these algorithms do not have to follow a common pattern. Working with a nonuniform model makes lower bounds stronger. In contrast, the variable-size sliding window model that we discuss next is a uniform model in the sense that there is a single streaming algorithm that works for every window length.

3.3 Variable-size sliding window model

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

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

The symbol $\downarrow$ represents the *pop operation*. We emphasize that a pop operation on an empty window leaves the window empty. Let $\varphi\colon \Sigma^* \to Y$ be a function. The *variable-size sliding window problem* $\mathrm{SW}(\varphi)$ is the function $\varphi \circ \mathrm{wnd}\colon \Sigma_\downarrow^* \to Y$. A *variable-size sliding window algorithm (variable-size SW-algorithm)* $\mathcal{P}$ for φ is a streaming algorithm for $\mathrm{SW}(\varphi)$.

Let $\mathrm{mwl}(a_1 \cdots a_m) = \max\{|\mathrm{wnd}(a_1 \cdots a_i)| : 0 \leqslant i \leqslant m\}$ be the *maximum window length* of a stream $u \in \Sigma^*$. If $\mathcal{P} = (M, \Sigma, m_0, \delta, o)$ is a streaming algorithm over the alphabet $\Sigma_\downarrow$ we define

$$M_{\leqslant n} = \{\mathcal{P}(w) \mid w \in \Sigma_\downarrow^*, \ \mathrm{mwl}(w) \leqslant n\}$$

and

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

The *space complexity* of $\mathcal{P}$ in the variable-size sliding window model is

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

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

Lemma 3.4. *If* $s(n) \geqslant \log n$ *is a monotonic function and* $\log |M_n| = O(s(n))$ *then* $v(\mathcal{P}, n) = O(s(n))$.

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

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

which proves the statement. □

Lemma 3.5. *Let* $\varphi \colon \Sigma^* \to Y$ *be a problem. There exists a space optimal variable-size SW-algorithm* $\mathcal{P}$ *for* φ*, i.e.* $v(\mathcal{P}, n) \leqslant v(\mathcal{Q}, n)$ *for every variable-size SW-algorithm* $\mathcal{Q}$ *for* φ *and every* $n \in \mathbb{N}$.

Proof. Let $\mathcal{P}$ be the minimal deterministic automaton for $\mathrm{SW}(\varphi) = \varphi \circ \mathrm{wnd}$ and $\mathcal{Q}$ be any deterministic automaton for $\mathrm{SW}(\varphi)$. Let $W_{\leqslant n} = \{w \in \Sigma_{\downarrow}^* \mid \mathrm{mwl}(w) \leqslant n\}$. Since $\sim_{\mathcal{P}}$ is coarser than $\sim_{\mathcal{Q}}$ we have

$$\begin{aligned} v(\mathcal{P}, n) &= \log |\{\mathcal{P}(w) \mid w \in W_{\leqslant n}\}| \\ &= \log |W_{\leqslant n} / {\sim_{\mathcal{P}}}| \leqslant \log |W_{\leqslant n} / {\sim_{\mathcal{Q}}}| \\ &= \log |\{\mathcal{Q}(w) \mid w \in W_{\leqslant n}\}| = v(\mathcal{Q}, n), \end{aligned}$$

which proves the statement. □

We define the space complexity of φ in the variable-side sliding window model by $V_\varphi(n) = v(\mathcal{P}, n)$, where $\mathcal{P}$ is a space optimal variable-size SW-algorithm for $\mathrm{SW}(\varphi)$ from Lemma 3.5.

Lemma 3.6. *For any problem* φ *and* $n \in \mathbb{N}$ *we have* $F_\varphi(n) \leqslant V_\varphi(n)$.

Proof. If $\mathcal{P}$ is a space optimal variable-size SW-algorithm for φ then one obtains an SW-algorithm $\mathcal{P}_n$ for window length $n \in \mathbb{N}$ as follows. Let us assume $n \geqslant 1$ (for $n = 0$ we use the trivial SW-algorithm). First one simulates $\mathcal{P}$ on the initial window $\Box^n$. For every incoming symbol $a \in \Sigma$ we perform a pop operation $\downarrow$ in $\mathcal{P}$, followed by inserting a. Since the maximum window length is bounded by n on any stream, the space complexity is bounded by $v(\mathcal{P}, n) = V_\varphi(n)$. □

A problem $\varphi \colon \Sigma^* \to Y$ is *trivial* if it is a constant function, i.e. $\varphi(u) = \varphi(v)$ for all $u, v \in \Sigma^*$. Correspondingly a language $L \subseteq \Sigma^*$ is trivial if it is either empty or universal, i.e. $L = \emptyset$ or $L = \Sigma^*$. The following lemma states that in the variable-size model one must at least maintain the current window size if the problem is nontrivial.

Lemma 3.7. *Let $\mathcal{P}$ be a variable-size SW-algorithm for a nontrivial problem $\varphi \colon \Sigma^* \to Y$. Then $\mathcal{P}(x)$ determines*[1]$|\mathrm{wnd}(x)|$ *for all $x \in \Sigma_\downarrow^*$ and therefore $V_\varphi(n) \geqslant \log(n+1)$.*

Proof. Let $y \in \Sigma^*$ be length-minimal such that $\varphi(\varepsilon) \neq \varphi(y)$. Let $x \in \Sigma_\downarrow^*$ with $|\mathrm{wnd}(x)| = m$. We read x into $\mathcal{P}$, followed by y and an infinite sequence of $\downarrow$. Then in state $\mathcal{P}(xy\downarrow^m)$ the algorithm outputs $\varphi(y)$, and in all states $\mathcal{P}(xy\downarrow^i)$ where $i > m$ it outputs $\varphi(\varepsilon)$, by minimality of $|y|$. Clearly this run determines m. For the second statement: If the algorithm reads any stream $a_1 \cdots a_n \in \Sigma^n$ it must visit $n+1$ pairwise distinct memory states and hence $v(\mathcal{P}, n) \geqslant \log(n+1)$. □

Clearly, the issue at hand is performing a pop operation on an empty window. Alternative definitions of the variable-size model are conceivable, e.g. one could neglect streams where the popping of an empty window occurs, or assume that the window size is always known to the algorithm.

3.4 Alternative characterizations

The problem of determining the space complexity $F_\varphi(n)$ and $V_\varphi(n)$ for a given problem φ boils down to understanding which windows have to be distinguished by the SW-algorithm, and which can be identified. In this section we study right congruences which capture the sliding window models. It turns out that the variable-size model has a clearer description in terms of right congruences than the fixed-size model. Since in many cases $V_L(n)$ and $F_L(n)$ are asymptotically equal we can view the variable-size model as an approximation of the fixed-size model which is easier to analyze.

3.4.1 Fixed-size model

For any problem $\varphi \colon \Sigma^* \to Y$ we define the equivalence relation ρ_φ on Σ^* by

$$x \;\rho_\varphi\; y \iff |x| = |y| = n \text{ and } \varphi(\mathrm{last}_n(xz)) = \varphi(\mathrm{last}_n(yz)) \text{ for all } z \in \Sigma^{\leqslant n}.$$

Let us show that this equivalence relation (in fact, right congruence) captures the space complexity $F_\varphi(n)$. Clearly we can extend the quantification from $z \in \Sigma^{\leqslant n}$ to $z \in \Sigma^*$ since for any $z \in \Sigma^{>n}$ we have $\mathrm{last}_n(xz) = \mathrm{last}_n(yz)$ and thus $\varphi(\mathrm{last}_n(xz)) = \varphi(\mathrm{last}_n(yz))$. Therefore we have

$$x \;\rho_\varphi\; y \iff |x| = |y| = n \text{ and } x \sim_{\mathrm{SW}_n(\varphi)} y. \tag{3.1}$$

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

Lemma 3.8. *We have* $x \sim_{\mathsf{SW}_n(\varphi)} \mathrm{last}_n(x)$ *for any* $x \in \Sigma^*$.

Proof. It suffices to show that $\mathrm{last}_n(xz) = \mathrm{last}_n(\mathrm{last}_n(x)\,z)$, which can be verified by a simple case distinction. Let $\Box \in \Sigma$ be the padding symbol. If $|xz| < n$ then $\mathrm{last}_n(xz) = \Box^{n-|xz|}xz$ and $\mathrm{last}_n(\mathrm{last}_n(x)\,z) = \mathrm{last}_n(\Box^{n-|x|}\,xz) = \Box^{n-|xz|}xz$. If $|xz| \geqslant n$ then $\mathrm{last}_n(xz)$ is the suffix of xz of length n. If $|z| \geqslant n$ then $\mathrm{last}_n(xz)$ is in fact a suffix of z; if $|z| < n$ then $\mathrm{last}_n(xz) = x'z$ where x' is the suffix of x of length $n - |z|$. In either case $\mathrm{last}_n(xz)$ is the suffix of $\mathrm{last}_n(x)\,z$ of length n. ☐

Proposition 3.9. *For every problem* $\varphi\colon \Sigma^* \to Y$ *we have* $F_\varphi(n) = \log|\Sigma^n/\rho_\varphi|$.

Proof. We have $F_\varphi(n) = \log|\Sigma^*/{\sim_{\mathsf{SW}_n(\varphi)}}|$. According to Lemma 3.8 we know that $|\Sigma^*/{\sim_{\mathsf{SW}_n(\varphi)}}| = |\Sigma^n/{\sim_{\mathsf{SW}_n(\varphi)}}|$ and by (3.1) the equivalence relations $\sim_{\mathsf{SW}_n(\varphi)}$ and ρ_φ coincide on Σ^n. Therefore $F_\varphi(n) = \log|\Sigma^n/\rho_\varphi|$. ☐

In particular this proves that the definition of the space complexity in the fixed-size sliding window model is independent of the choice of the padding symbol.

3.4.2 Variable-size model

Suffix expansions Consider an equivalence relation $\sim$ on Σ^*. The *suffix expansion* of $\sim$ is the equivalence relation $\approx$ on Σ^* defined by

$$a_1 \cdots a_n \approx b_1 \cdots b_m \iff n = m \text{ and } a_i \cdots a_n \sim b_i \cdots b_n, \text{ for all } 1 \leqslant i \leqslant n.$$

Notice that $\approx$ saturates each subset Σ^n. Furthermore, if $\sim$ is a right congruence then so is $\approx$ since $|u| = |v|$ implies $|ua| = |va|$ and $a_i \cdots a_n \sim b_i \cdots b_n$ implies $a_i \cdots a_n a \sim b_i \cdots b_n a$. The suffix expansion of the Myhill-Nerode right congruence $\sim_\varphi$ is denoted by $\approx_\varphi$. We also define suffix expansions for partial functions $t\colon \Sigma^* \to Y$ with *suffix-closed* domains $\mathrm{dom}(t)$. A language $L \subseteq \Sigma^*$ is suffix-closed if $xy \in L$ implies $y \in L$. The *suffix expansion* of t is the total function $\overleftarrow{t}\colon \mathrm{dom}(t) \to Y^*$ defined by

$$\overleftarrow{t}(a_1 \cdots a_n) = t(a_1 \cdots a_n)\, t(a_2 \cdots a_n) \,\cdots\, t(a_{n-1}a_n)\, t(a_n)$$

for all $a_1 \cdots a_n \in \mathrm{dom}(t)$. Here the range of $\overleftarrow{t}$ is the free monoid (alternatively, the set of all sequences) over Y. Notice that $\overleftarrow{t}$ is a length-preserving function. If $\sim$ is an equivalence relation on Σ^* then its suffix expansion $\approx$ is the kernel of $\overleftarrow{\nu}_\sim$.

Let $\varphi\colon \Sigma^* \to Y$ be a computational problem. For better readability we write $[x]_\varphi$ instead of $[x]_{\sim_\varphi}$ and ν_φ instead of $\nu_{\sim_\varphi}$. Using this notation the suffix expansion $\overleftarrow{\nu}_\varphi\colon \Sigma^* \to (\Sigma^*/{\sim_\varphi})^*$ is the length-preserving function

$$a_1 \cdots a_n \mapsto [a_1 \cdots a_n]_\varphi\, [a_2 \cdots a_n]_\varphi \cdots [a_n]_\varphi.$$

The following propositions state that the minimal information that any variable-size sliding window algorithm for φ has to maintain is the $\approx_\varphi$-class of the active window.

Proposition 3.10. *For every problem φ there exists a variable-size SW-algorithm $\mathcal{P}$ for φ with $\mathcal{P}(x) = \overleftarrow{v}_\varphi(\mathrm{wnd}(x))$ for all $x \in \Sigma_\downarrow^*$.*

Proof. We exhibit a streaming algorithm $\mathcal{P}$ which maintains the tuple $\overleftarrow{v}_\varphi(w)$ for the active window $w \in \Sigma^*$. In particular we know the window size n at every time instant. Initially the memory state is $\overleftarrow{v}_\varphi(\varepsilon)$, which is the empty sequence. Assume that the active window is $w = a_1 \cdots a_n \in \Sigma^*$.

- Suppose that $\mathcal{P}$ receives $\downarrow$. If $n \geqslant 1$ then $\overleftarrow{v}_\varphi(a_2 \cdots a_n)$ can be obtained from the sequence $\overleftarrow{v}_\varphi(a_1 \cdots a_n)$ by removing the first $\sim_\varphi$-class $[a_1 \cdots a_n]_\varphi$. If $n = 0$ then nothing is changed.
- Suppose that $\mathcal{P}$ receives a symbol $a \in \Sigma$. From the sequence $\overleftarrow{v}_\varphi(a_1 \cdots a_n)$ we can compute $\overleftarrow{v}_\varphi(a_1 \cdots a_n a) = [a_1 \cdots a_n a]_\varphi [a_2 \cdots a_n a]_\varphi \cdots [a]_\varphi$ since $\sim_\varphi$ is a right congruence.
- The first $\sim_\varphi$-class in $\overleftarrow{v}_\varphi(a_1 \cdots a_n)$ determines the value $\varphi(a_1 \cdots a_n)$.

These remarks define a variable-size SW-algorithm for φ. ☐

Proposition 3.11. *If φ is nontrivial then* $V_\varphi(n) = \log |\overleftarrow{v}_\varphi(\Sigma^{\leqslant n})| = \log |\Sigma^{\leqslant n}/{\approx_\varphi}|$.

Proof. In the algorithm from Proposition 3.10, the set of states reachable by streams with maximum window length n is precisely $\overleftarrow{v}_\varphi(\Sigma^{\leqslant n})$, which proves that $V_\varphi(n) \leqslant \log |\overleftarrow{v}_\varphi(\Sigma^{\leqslant n})|$.

Now consider a variable-size SW-algorithm $\mathcal{P}$ for φ with space complexity $v(\mathcal{P}, n)$. We have to show that $v(\mathcal{P}, n) \geqslant \log |\overleftarrow{v}_\varphi(\Sigma^{\leqslant n})|$, i.e. $|M_{\leqslant n}| \geqslant |\overleftarrow{v}_\varphi(\Sigma^{\leqslant n})|$ where $M_{\leqslant n} = \{\mathcal{P}(x) \mid \mathrm{mwl}(x) \leqslant n\}$. Let $x = a_1 a_2 \cdots a_m \in \Sigma^*$ be an input word of length $m \leqslant n$. By Lemma 3.7 the memory state $\mathcal{P}(x)$ determines the length $m = |x|$. It suffices to show that $\mathcal{P}(x)$ determines every congruence class $[a_k \cdots a_m]_\varphi$ for $1 \leqslant k \leqslant m$: Starting from memory state $\mathcal{P}(x)$ we read $k-1$ times $\downarrow$ into $\mathcal{P}$. Then, the active window is $a_k \cdots a_m$. We can determine the congruence class $[a_k \cdots a_m]_\varphi$ by reading every word $z \in \Sigma^*$ in parallel into $\mathcal{P}$ and computing $\varphi(a_k \cdots a_m z)$. To sum up, we have constructed an injection from $\overleftarrow{v}_\varphi(\Sigma^{\leqslant n})$ to $M_{\leqslant n}$, which proves the claim.

Finally, observe that a well-defined bijection $\Sigma^{\leqslant n}/{\approx_\varphi} \to \overleftarrow{v}_\varphi(\Sigma^{\leqslant n})$ is given by

$$[a_1 \cdots a_n]_{\approx_\varphi} \mapsto [a_1 \cdots a_n]_\varphi \cdots [a_n]_\varphi,$$

which proves the second equality. ☐

Note that Proposition 3.11 does not hold for trivial problems φ. In these cases, we have $V_\varphi(n) = 0$ and $\log |\overleftarrow{v}_\varphi(\Sigma^{\leqslant n})| = \log(n+1)$.

3.5 Related complexity measures

In the following we uncouple two concepts which underlie the variable-size model, namely that windows can grow to the right and shrink at the left end. These concepts yield lower bounds on $V_\varphi(n)$.

Standard streaming Firstly, if we only consider streams without $\downarrow$-operations then variable-size sliding window model becomes the *standard streaming model*. Given a streaming algorithm $\mathcal{P}$ for a problem φ we define the *standard streaming space complexity* as the logarithm of the number states reached on streams of length at most n, i.e. $\log |\{\mathcal{P}(w) \mid w \in \Sigma^{\leqslant n}\}|$. The optimal streaming algorithm is the minimal deterministic automaton with output for φ whose space complexity is given by

$$E_\varphi(n) = \log |\Sigma^{\leqslant n}/{\sim_\varphi}|.$$

Here we have adopted the notation $E_\varphi(n)$ from Hartmanis-Shank [65] who defined $E_L(n) = |\Sigma^{\leqslant n}/{\sim_L}|$ where $\sim_L$ is the canonical Myhill-Nerode *left*-congruence for L. We have $E_\varphi(n) \leqslant V_\varphi(n)$ since any variable-size sliding window algorithm for φ can be restricted to a (standard) streaming algorithm for φ.

A language L has constant complexity $E_L(n) = O(1)$ if and only if it is regular. Every deterministic one-counter language L satisfies $E_L(n) = O(\log n)$. In fact, the latter also holds for every language recognized by a deterministic pushdown automaton whose set of stack contents has polynomial growth.

Suffix complexity Secondly, we define the *suffix complexity* of φ by

$$S_\varphi(n) = \log |\overleftarrow{\varphi}(\Sigma^{\leqslant n})|.$$

Essentially[2], it specifies the minimal size of a data structure which supports *suffix queries* on a string $a_1 \cdots a_m \in \Sigma^{\leqslant n}$: Given a position $1 \leqslant i \leqslant m$, output $\varphi(a_i \cdots a_m)$. Since $[w]_\varphi$ determines $\varphi(w)$ we have $\log |\overleftarrow{\varphi}(\Sigma^{\leqslant n})| \leqslant \log |\Sigma^{\leqslant n}/{\approx_\varphi}|$ and thus $S_\varphi(n) \leqslant V_\varphi(n)$ for all nontrivial problems φ by Proposition 3.11.

There are simple languages L which show that, in general, $E_L(n)$ and $S_L(n)$ do not imply upper bounds on $V_L(n)$.

Proposition 3.12. *Let* $\Sigma = \{a, b, c\}$. *The language*

$$L = \Sigma^*\{vb^k \mid v \in \{a, c\}^*, |v|_a \geqslant k\}$$

satisfies $E_L(n) = O(\log n)$, $S_L(n) = O(\log n)$ *and* $V_L(n) = \Omega(n)$.

Proof. For a word $w \in \{a, b, c\}^*$ define $p(w) = (|v|_a, k) \in \mathbb{N}^2$ where vb^k is the maximal suffix of w satisfying $v \in \{a, c\}^*$. One can see that $p(w) = p(w')$ implies $w \sim_L w'$. Since $p(\{a, b, c\}^{\leqslant n}) \subseteq \{0, \ldots, n\}^2$ we obtain $E_L(n) = \log |\Sigma^{\leqslant n}/{\sim_L}| = O(\log n)$. For the second statement notice that L is a *left ideal*, i.e. $v \in L$ and $u \in \Sigma^*$ implies $uv \in L$. Therefore $\overleftarrow{\chi}_L(\Sigma^{\leqslant n}) \subseteq 1^*0^*$ and $S_L(n) = O(\log n)$.

It remains to show that $V_L(n) = \Omega(n)$. Let $\mathcal{P}$ be a variable-size SW-algorithm for L and read in parallel two distinct words $w \neq w' \in \{a, c\}^n$ into $\mathcal{P}$. After a suitable number of $\downarrow$-operations the windows have the form aw'' and cw'' for some $w'' \in \{a, c\}^*$. By inserting $k = |aw''|_a$ many b-symbols we obtain a window

[2]under the assumption that the length m is stored and hence $S_\varphi(n) \geqslant \log(n + 1)$.

$aw''b^k \in L$ and a window $cw''b^k \notin L$. This proves that $\mathcal{P}$ must distinguish all 2^n words in $\{a,c\}^n$ and hence its space complexity is $\Omega(n)$. □

3.6 Connection to language growth

If a language is sparse then its sliding window problem becomes easy.

Proposition 3.13. *If* $L \subseteq \Sigma^*$ *has growth* $g(n)$*, then* $F_L(n) = O(\log g(n) + \log n)$.

Proof. Let $n \in \mathbb{N}$ be a window size and let $w_1, \dots, w_m$ be an arbitrary enumeration of $L \cap \Sigma^n$ where $m = g(n)$. We can solve this problem using the Aho-Corasick algorithm [3] for the *dictionary matching problem*: Given a text t and a set of dictionary words (or keywords) $\{d_1, \dots, d_m\}$, report all occurrences of the dictionary words in the text. It builds a deterministic automaton of size at most $\sum_{i=1}^m |d_i|$ and runs it over the text. In our case the dictionary words are $w_1, \dots, w_m$, which have total size $g(n) \cdot n$. Hence the states of the Aho-Corasick automaton can be encoded using $O(\log g(n) + \log n)$ bits.

For completeness sake let us describe the Aho-Corasick algorithm in our abstract computation model to solve the problem in the specified space bounds. If $L \cap \Sigma^n = \emptyset$ then the SW-algorithm for window length n always rejects, and so we can assume the contrary. We show that a streaming algorithm can maintain the longest suffix $v = a_i \cdots a_n$ of the active window $w = a_1 \cdots a_n$ such that v is a prefix of a word $w_j \in L \cap \Sigma^n$. Since the empty suffix $v = \varepsilon$ is always the prefix of some word in $L \cap \Sigma^n$, which was assumed to be nonempty, the suffix v is well-defined. Notice that v can be encoded by the binary encoded number j using $O(\log g(n))$ bits and the binary encoded number i using $O(\log n)$ bits. Of course, there may exist several words w_j having v as a prefix; in this case the concrete choice of w_j does not matter. This information clearly suffices to check whether the active window w belongs to L since $w \in L$ if and only if $|v| = n$. Moreover, we can update the information: If $a_{n+1} \in \Sigma$ is the next symbol from the stream, then the longest suffix v' of $a_2 \cdots a_n a_{n+1}$ with $v' \in \text{Pref}(L \cap \Sigma^n)$ is also a suffix of $va_{n+1} = a_i \cdots a_n a_{n+1}$. Therefore we can compute v' from v and a_{n+1}. □

In particular $F_L(n) = O(\log n)$ for all languages with polynomial growth. We can extend this statement to the variable-size model under the stronger condition that L is bounded. We need a known fact on bounded languages.

Lemma 3.14 ([59, Lemma 1.1(c)]). *If* L *is a bounded language and* K *is a set of factors of words in* L *then* K *is bounded.*

Proposition 3.15. *Let* L *be a language.*

(i) If $\text{Pref}(L)$ *has cumulative growth* $g(n)$ *then* $V_L(n) = O(\log g(n))$.

(ii) If L *is bounded then* $V_L(n) = O(\log n)$.

Proof. Let us describe a variable-size SW-algorithm for L with space complexity $O(\log n)$. Again we maintain the longest suffix $v = a_i \cdots a_n$ of the active window $w = a_1 \cdots a_n$ which is a prefix of some word in L, i.e. v is the longest suffix of w with $v \in \mathrm{Pref}(L)$. This suffix $v \in \mathrm{Pref}(L)$ of w can be encoded using $O(\log g(|v|))$ bits, which is bounded by $O(\log g(|w|))$ since cumulative growth functions are monotonic. Therefore this SW-algorithm has space complexity $O(\log g(n))$.

The second statement follows from the first since $\mathrm{Pref}(L)$ is bounded whenever L is bounded by Lemma 3.14. $\square$

We remark that Proposition 3.15(ii) fails under the weaker conditions that L is polynomially growing.

Proposition 3.16. *There exists a language L with polynomial growth such that $V_L(n) = \Theta(n)$.*

Proof. Define L to be the set of all words of the form axb^m where $x \in \{a,b\}^*$ with $2^{|x|} \leqslant m$. Then $|L \cap \{a,b\}^n| \leqslant |\{x \in \{a,b\}^* : |x| \leqslant \log n\}| = O(n)$ and hence the growth of L is polynomial.

However, we claim that $V_L(n) = \Theta(n)$. Consider any variable-size SW-algorithm $\mathcal{P}$ for L and two distinct words $x \neq y$ from $\{a,b\}^n$ for some $n \in \mathbb{N}$. To show the lower bound it suffices to show that $\mathcal{P}(x) \neq \mathcal{P}(y)$. Let $1 \leqslant i \leqslant n$ be any position where x and y differ. Then the active window of $x \downarrow^{i-1}$ and $y \downarrow^{i-1}$ are of the form ax' and by' (or vice versa) for some $x', y' \in \{a,b\}^{n-i}$. Since $ax'b^m \in L$ but $bx'b^m \notin L$ for sufficiently large $m \in \mathbb{N}$ the algorithm $\mathcal{P}$ must distinguish x and y. $\square$

3.7 Closure properties

Throughout this work we need simple closure properties in the streaming and the sliding window model.

Lemma 3.17. *Let $\mathcal{P}_i$ be a streaming algorithm for $\varphi_i \colon \Sigma^* \to Y$ for $1 \leqslant i \leqslant k$, and let $\tau \colon Y^k \to Z$ be any function. Define $\varphi \colon \Sigma^* \to Z$ where $\varphi(x) = \tau(\varphi_1(x), \dots, \varphi_k(x))$.*

(i) There exists a streaming algorithm $\mathcal{P}$ for φ such that $s(\mathcal{P}) \leqslant \sum_{i=1}^k s(\mathcal{P}_i)$.

(ii) $F_\varphi(n) \leqslant \sum_{i=1}^k F_{\varphi_i}(n)$

(iii) $V_\varphi(n) \leqslant \sum_{i=1}^k V_{\varphi_i}(n)$

Proof. For (i) let $\mathcal{P}$ be the product automaton of the k automata $\mathcal{P}_i$. If $\mathcal{P}_i$ outputs some value y_i for $1 \leqslant i \leqslant k$, then $\mathcal{P}$ outputs $\tau(y_1, \dots, y_k)$. It has $\prod_{i=1}^k 2^{s(\mathcal{P}_i)} = 2^{\sum_{i=1}^k s(\mathcal{P}_i)}$ many states and hence $s(\mathcal{P}) = \sum_{i=1}^k s(\mathcal{P}_i)$.

Point (ii) follows immediately from point (i) and the observation that

$$\varphi(\mathrm{last}_n(x)) = \tau(\varphi_1(\mathrm{last}_n(x)), \dots, \varphi_k(\mathrm{last}_n(x)))$$

for all $x \in \Sigma^*$.

For (iii) let $\mathcal{P}_i$ be the minimal deterministic automaton for φ_i and let $\mathcal{P}$ be the product automaton as in (i). If $x \in \Sigma_\downarrow^*$ is a stream with maximum window length $\leqslant n$ then $\mathcal{P}(x) = (\mathcal{P}_1(x), \dots, \mathcal{P}_k(x))$. We can bound the number of such memory states by $\prod_{i=1}^k 2^{\nu(\mathcal{P}_i, n)}$. By the calculation above we get $\nu(\mathcal{P}, n) = \sum_{i=1}^k \nu(\mathcal{P}_i, n)$. □

In the following let $X \in \{F, V\}$ and Σ be an alphabet. Let us look at language closure properties in the sliding window models. A simple corollary of Lemma 3.17 is that space complexity classes form a Boolean algebra, i.e. they are closed under union, intersection and complementation, for both the fixed-size and the variable-size model.

Corollary 3.18. *For any function* $s(n)$*, the class* $\{L \subseteq \Sigma^* \mid X_L(n) = O(s(n))\}$ *forms a Boolean algebra.*

For a word $u \in \Sigma^*$ and a language $L \subseteq \Sigma^*$ we define the *left quotient* $u^{-1}L = \{w \in \Sigma^* \mid uw \in L\}$ and the *right quotient* $Lu^{-1} = \{w \in \Sigma^* \mid wu \in L\}$.

Lemma 3.19. *Consider a function* $s(n)$ *satisfying* $s(n+k) = O(s(n))$ *for all* $k \in \mathbb{N}$*, e.g.* $s(n) = n^c$ *or* $s(n) = \log^c n$ *for some* $c \geqslant 0$*. Then* $\{L \subseteq \Sigma^* \mid X_L(n) = O(s(n))\}$ *is closed under right quotients but in general not under left quotients.*

Proof. Consider $u \in \Sigma^*$ and $L \subseteq \Sigma^*$. First we consider the fixed-size model. Let $\mathcal{P}$ be an SW-algorithm for L and window length $n + |u|$ for some $n \in \mathbb{N}$. The algorithm $\mathcal{P}'$ simulates $\mathcal{P}$ where a memory state s in $\mathcal{P}'$ is accepting if and only if $s \cdot u$ is accepting in $\mathcal{P}$. Then we have

$$\begin{aligned} \mathcal{P}' \text{ accepts } x &\iff \mathcal{P}' \text{ accepts } xu \\ &\iff \text{last}_{n+|u|}(xu) = \text{last}_n(x)\, u \in L \\ &\iff \text{last}_n(x) \in Lu^{-1} \end{aligned}$$

for all $x \in \Sigma^*$, i.e. $\mathcal{P}'$ is an SW-algorithm for L and window length n which has the same space complexity as $\mathcal{P}$. A similar argument holds for the variable-size model.

A counterexample that the statement does not for left quotients is the following. Let $L = ca\{a, b\}^*$ over the alphabet $\{a, b, c\}$. One can see that $V_L(n) = O(\log n)$ since an SW-algorithm can track the position of the most recent ca-factor using $O(\log n)$ bits. On the other hand $c^{-1}L = a\{a, b\}^*$ has complexity $F_{c^{-1}L}(n) = \Omega(n)$ by Example 3.3. □

In particular, space complexity classes do not form a variety of languages [109]: a class of languages $\mathcal{L} \subseteq 2^{\Sigma^*}$ is a *variety* if it is closed under (i) Boolean operations, (ii) left and right quotients, and (iii) inverse homomorphisms, i.e. if $L \in \mathcal{L}$ and $\psi\colon \Sigma^* \to \Sigma^*$ is a homomorphism then $\psi^{-1}(L) \in \mathcal{L}$. There is a one-to-one correspondence between varieties of languages and varieties of monoids [41], which allows one to use algebraic tools in formal language theory. For example, the class of *star-free* languages (the closure of all finite languages under products and Boolean operations) contains exactly those languages whose syntactic monoid is *aperiodic*. Because of the restricted closure properties in the sliding window model we have to analyse automata and cannot reason purely algebraically about monoids.

For completeness we also prove nonclosure under inverse homomorphisms.

Lemma 3.20. *The class* $\{L \subseteq \Sigma^* \mid X_L(n) = O(\log n)\}$ *is not closed under inverse homomorphisms.*

Proof. Let $E \subseteq \{a, b\}^*$ be the set of words with even length. We have $V_E(n) = O(\log n)$ because it suffices to maintain the window length. Consider the homomorphism defined by $\psi(a) = a$ and $\psi(b) = \varepsilon$. Then $\psi(x) \in E$ if and only if x contains an even number of a's. We can prove $F_{\psi^{-1}(E)} = \Omega(n)$ by a fooling argument as in Example 3.3. Let $\mathcal{P}_n$ be a sliding window algorithm for $\psi^{-1}(E)$ and window length n. Let $w \neq w' \in \{a, b\}^n$ be two distinct windows, which are read into $\mathcal{P}_n$ in parallel. After inserting a suitable number of b-symbols the two windows are identical up to the first symbol. Hence, one window contains an even number of a's whereas the other contains an odd number. Therefore $\mathcal{P}_n$ has space complexity $\Omega(n)$. □

Finally, we will look at reductions between languages. A function $\tau \colon \Sigma^* \to \Gamma^*$ is a *reduction* from $K \subseteq \Sigma^*$ to $L \subseteq \Gamma^*$ if $x \in K$ if and only if $\tau(x) \in L$. The reductions that we use are computed by a right-sequential transducers which output a single output letter per input letter.

A *right Mealy machine* $\mathcal{M} = (Q, \Sigma, \Omega, \delta, q_0)$ consists of a finite state set Q, an input alphabet Σ, an output alphabets Ω, a transition function $\delta \colon \Sigma \times Q \to Q \times \Omega$ and an initial state $q_0 \in Q$. We view $\mathcal{M}$ as a right-subsequential transducer $\mathcal{M} = (Q, \Sigma, \Omega, Q, \Delta, q_0, o)$ where

$$\Delta = \{(q, a, b, p) \mid \delta(p, a) = (q, b), p \in Q, a \in \Sigma\}$$

and $o(q) = \varepsilon$ for all $q \in Q$. Functions computed by right Mealy machines are called *$\leftarrow$-transduction*. Every $\leftarrow$-transduction $\tau \colon \Sigma^* \to \Gamma^*$ is *length-preserving*, i.e. $|\tau(x)| = |x|$ for all $x \in \Sigma^*$. A reduction from K to L which is a $\leftarrow$-transduction, is a *$\leftarrow$-reduction* from K to L, and we say that K is *$\leftarrow$-reducible* to L.

Lemma 3.21. *If* K *is $\leftarrow$-reducible to* L *via a Mealy machine with* m *states, then* $X_K(n) \leqslant m \cdot X_L(n)$. *In particular, for any function* $s(n)$ *the class* $\{L \subseteq \Sigma^* \mid X_L(n) = O(s(n))\}$ *is closed under $\leftarrow$-reductions.*

Proof. Let $\tau \colon \Sigma^* \to \Gamma^*$ be a $\leftarrow$-reduction from $K \subseteq \Sigma^*$ to $L \subseteq \Gamma^*$ where τ is computed by the right Mealy machine $\mathcal{M} = (Q, \Sigma, \Omega, \delta, q_0)$. For a state $p \in Q$ let $\tau_p = T(\mathcal{M}_p)$ where $\mathcal{M}_p = (Q, \Sigma, \Omega, \delta, p)$. We assume that $Q = \{1, \dots, m\}$.

Let us start with the fixed-size model. Consider an SW-algorithm $\mathcal{P}_n$ for L and window length n. We construct an SW-algorithm $\mathcal{Q}_n$ for K which simulates m copies of $\mathcal{P}_n$ in parallel. It maintains the tuple

$$(\mathcal{P}_n(\tau_1(x)), \dots, \mathcal{P}_n(\tau_d(x))),$$

which is possible since $\mathcal{P}_n(\tau_p(xa))$ can be computed by determining the unique transition $q \xleftarrow{a|b} p$ in $\mathcal{M}$ and computing $\mathcal{P}_n(\tau_p(xa)) = \mathcal{P}_n(\tau_q(x)\, b)$. Clearly, the number of bits used by $\mathcal{Q}_n$ is m times the number of bits used by $\mathcal{P}_n$.

For the variable-size model define the Mealy machine $\overline{\mathcal{M}} = (Q, \Sigma_\downarrow, \Omega_\downarrow, \delta', q_0)$ obtained from $\mathcal{M}$ by adding the transitions $q \xleftarrow{\downarrow|\downarrow} q$ for every $q \in Q$. Let $\overline{\tau}\colon \Sigma_\downarrow^* \to \Gamma_\downarrow^*$ be the transduction computed by $\overline{\mathcal{M}}$ and $\overline{\tau}_p$ be the transduction computed by $\overline{\mathcal{M}}$ in state p.

Let us prove $\tau_p(\mathsf{wnd}(x)) = \mathsf{wnd}(\overline{\tau}_p(x))$ for all $x \in \Sigma_\downarrow^*$ and $p \in Q$ by induction on $|x|$. If $x = \varepsilon$ this is clear. Assume $x = ya$ for some $y \in \Sigma_\downarrow^*$ and $a \in \Sigma$. If $q \xleftarrow{a|b} p$ then

$$\begin{aligned}\tau_p(\mathsf{wnd}(x)) &= \tau_p(\mathsf{wnd}(y)\, a) = \tau_q(\mathsf{wnd}(y))\, b = \mathsf{wnd}(\overline{\tau}_q(y))\, b \\ &= \mathsf{wnd}(\overline{\tau}_q(y)\, b) = \mathsf{wnd}(\overline{\tau}_p(ya)) = \mathsf{wnd}(\overline{\tau}_p(x)).\end{aligned}$$

Now assume $x = y{\downarrow}$ for some $y \in \Sigma_\downarrow^*$. If $\mathsf{wnd}(y) = \varepsilon$ then $\mathsf{wnd}(x) = \varepsilon$ and hence $\tau_p(\mathsf{wnd}(x)) = \varepsilon = \mathsf{wnd}(\overline{\tau}_p(x))$. Otherwise $\mathsf{wnd}(y) = cw$ for some $c \in \Sigma$ and $w \in \Sigma^*$. By induction hypothesis we know that $\tau_p(\mathsf{wnd}(y)) = \mathsf{wnd}(\overline{\tau}_p(y))$. Since τ_p is length-preserving we have $|cw| = |\tau_p(\mathsf{wnd}(y))| = |\mathsf{wnd}(\overline{\tau}_p(y))|$. Therefore we can decompose $\mathsf{wnd}(\overline{\tau}_p(y)) = dv$ into $d \in \Sigma$ and $v \in \Sigma^*$ with $|v| = |w|$. By the properties of a Mealy machine we have $\tau_p(w) = v$. This implies

$$\begin{aligned}\tau_p(\mathsf{wnd}(x)) &= \tau_p(\mathsf{wnd}(y{\downarrow})) = \tau_p(w) = v = \mathsf{wnd}(dv{\downarrow}) \\ &= \mathsf{wnd}(\overline{\tau}_p(y){\downarrow}) = \mathsf{wnd}(\overline{\tau}_p(y{\downarrow})) = \mathsf{wnd}(\overline{\tau}_p(x)),\end{aligned}$$

which concludes the proof of the claim.

Now consider a variable-size SW-algorithm $\mathcal{P}$ for L. We construct a variable-size SW-algorithm $\mathcal{Q}$ for K which simulates m copies of $\mathcal{P}$ in parallel. It maintains the tuple

$$(\mathcal{P}(\overline{\tau}_1(x)), \dots, \mathcal{P}(\overline{\tau}_d(x))) \tag{3.2}$$

for an input stream $x \in \Sigma_\downarrow^*$. Clearly the memory state $\mathcal{P}(\overline{\tau}_{q_0}(x)) = \mathcal{P}(\overline{\tau}(x))$ determines whether $\mathsf{wnd}(\overline{\tau}(x)) = \tau(\mathsf{wnd}(x))$ belongs to L. Since τ is a reduction from K to L we have $\mathsf{wnd}(x) \in K$ if and only if $\tau(\mathsf{wnd}(x)) \in L$. On input $a \in \Sigma_\downarrow$ we can update the tuple in (3.2) since $\mathcal{P}(\overline{\tau}_p(xa))$ is the memory state $\mathcal{P}(\overline{\tau}_q(x)\, b)$ where $q \xleftarrow{a|b} p$ is a transition in $\overline{\mathcal{M}}$.

The bound on the space complexity is easy to verify: If $x \in \Sigma_\downarrow^*$ is an input stream for $\mathcal{Q}$ with $|\mathsf{wnd}(x)| = n$ then $\mathcal{Q}(x)$ is a tuple of length m containing memory states of the form $\mathcal{P}(\overline{\tau}_p(x))$. Since $|\mathsf{wnd}(\overline{\tau}_p(x))| = |\tau_p(\mathsf{wnd}(x))| = |\mathsf{wnd}(x)| = n$ each memory state $\mathcal{P}(\overline{\tau}_p(x))$ can be stored using $v(\mathcal{P}, n)$ bits. □

3.8 Remarks

Some authors distinguish between *sequence-based* and *time-based windows*, see [23, 60]. Sequence-based windows correspond to our fixed-size sliding window model whereas a time-based window at time t contains those elements whose timestamp lies in some interval $[t-T, t]$ for some interval length T. The difference to the variable-size model is that deletions from the window are implicit.

A time-efficient algorithm for many sliding window problems is the De-amortized Banker's Aggregator (DABA) by Tangwongsan, Hirzel and Schneider [108]. It receives a stream of monoid elements and maintains a $O(n)$ size data structure in worst case constant-time per element which allows to compute the monoid product over a sliding window.

The space complexity $E_L(n)$ in the standard streaming model is related to the notion of *automaticity* [102]. The automaticity of a language L measures for all $n \in \mathbb{N}$ the minimal size of a DFA whose language coincides with L on all words of length at most n. Clearly every regular language has constant automaticity. Karp [72] proved that every nonregular language has automaticity at least $(n+3)/2$ for infinitely many n. Since the automaticity of L is a lower bound on $2^{E_L(n)}$ this implies that $E_L(n)$, and hence $V_L(n)$, must be at least $\Omega(\log n)$ infinitely often for every nonregular language L.

Chapter 4

Regular languages

In this chapter we will show that the space complexity of every regular language in both sliding window models is either constant, logarithmic or linear. In Example 3.3 we have already seen prototypical languages for these three space complexities, namely $\Sigma^* a$ (constant), $\Sigma^* a \Sigma^*$ (logarithmic) and $a\Sigma^*$ (linear). Intuitively, for languages of logarithmic space complexity it suffices to maintain a constant number of positions in the window. For languages of constant space complexity it suffices to maintain a constant-length suffix of the window.

Next, we characterize the regular languages with logarithmic space complexity as the Boolean combinations of regular left ideals and regular length languages. This holds for both the fixed-size and the variable-size model. The regular languages with constant space complexity in the fixed-size model are the Boolean combinations of suffix testable languages and regular length languages. In the variable-size model the only regular languages with constant complexity are the empty and the universal language.

If a regular language is given as a DFA $\mathcal{A}$ and it admits a sliding window algorithm with $O(\log n)$ complexity then we devise a $O(2^{|\mathcal{A}|} \cdot |\mathcal{A}| \cdot \log n)$ space algorithm. We will prove that this exponential dependence on the DFA size is unavoidable. Finally we study the decision problems whether a given regular language has logarithmic or constant space complexity in the two sliding window models. Both problems are NL-complete if the regular language is given as a DFA. They become PSPACE-complete if the regular language is given as an NFA.

The results of this chapter appeared in [G1, G2].

4.1 Space trichotomy

It turns out that the appropriate representation of a regular language for the analysis in the sliding window model are right-deterministic finite automata. A *right-deterministic finite automaton (rDFA)* is a finite right automaton $\mathcal{B} = (Q, \Sigma, F, \Delta, q_0)$ where for all $p \in Q$ and $a \in \Sigma$ there exists exactly one transition $(q, a, p) \in \Delta$. We write $\mathcal{B}$ in the form $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ where $\delta \colon \Sigma \times Q \to Q$ is the transition function, which extends to a left action $\cdot \colon \Sigma^* \times Q \to Q$.

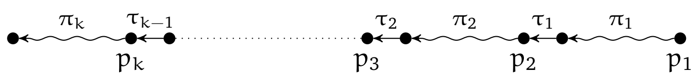

Figure 4.1: The SCC-factorization of a run.

The reason why we use rDFAs instead of DFAs can be explained intuitively for the variable-size sliding window model as follows. The variable-size model contains operations in both “directions”: On the one hand a variable-size window can be extended on the right. On the other hand the window can be shortened to an arbitrary suffix. For regular languages the extension to longer windows is “tame” because the Myhill-Nerode right congruences have finite index. Hence it remains to control the structure of all suffixes with respect to the regular language, which is best captured by an rDFA for the language.

4.1.1 Logarithmic space

In the following let $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ be a right-deterministic finite automaton. A state $q \in Q$ is *reachable* from $p \in Q$ if there exists a run from p to q, and we write $q \preceq_{\mathcal{B}} p$. We say that q is *reachable* if it is reachable from the initial state q_0. A set of states $P \subseteq Q$ is reachable if all $p \in P$ are reachable. The reachability relation $\preceq_{\mathcal{B}}$ is a *preorder* on Q, i.e. it is reflexive and transitive. Two states $p, q \in Q$ are *strongly connected* if p is reachable from q and q is reachable from p. The equivalence classes of $\preceq_{\mathcal{B}}$ are the *strongly connected components (SCCs)* of $\mathcal{B}$. A subset $P \subseteq Q$ is *strongly connected* if it is contained in a single SCC, i.e. all pairs $p, q \in P$ are strongly connected.

Path summaries Consider a run π in $\mathcal{B}$. We call π a P-*run* for a subset $P \subseteq Q$ if all states occurring in π are contained in P. We call π *internal* if π is a P-run for some SCC P. The *SCC-factorization* of π is the unique factorization $\pi = \pi_k \tau_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$ into maximal internal (possibly empty) subruns π_i and transitions τ_i in $\mathcal{B}$. By maximality of the π_i each transition τ_i connects two distinct SCCs. An SCC-factorization is abstractly visualized in Figure 4.1. Let $p_k, \dots, p_1 \in Q$ be the starting states of the runs $\pi_k, \dots, \pi_1$. Then the *path summary* of π is defined as

$$\mathrm{ps}(\pi) = (|\pi_k|, p_k), \dots, (|\tau_2\pi_2|, p_2), (|\tau_1\pi_1|, p_1) \in (\mathbb{N} \times Q)^*.$$

In other words, it specifies the first state that is visited in an SCC, and the length of the run until reaching the next SCC or the end of the word, respectively. The leftmost length $|\pi_k|$ can be zero but all other lengths $|\tau_i\pi_i|$ are positive. We define $\pi_{w,q}$ to be the run of $\mathcal{B}$ on w starting from q, and $\mathrm{PS}_{\mathcal{B}}(w) = \{\mathrm{ps}(\pi_{w,q}) \mid q \in Q\}$.

Example 4.1. Consider the rDFA $\mathcal{B}$ in Figure 4.2. It consists of three SCCs, namely the blue SCC $\{p\}$, the red SCC $\{q, r\}$, and the purple SCC $\{s, t\}$. All its runs on the

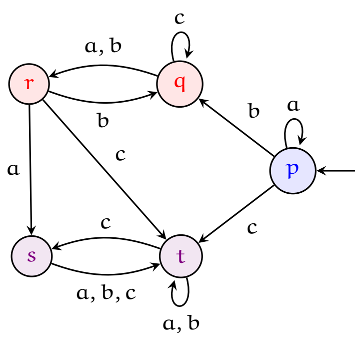

Figure 4.2: An rDFA partitioned into its SCCs.

word $w = babccbaa$ are listed here:

$$
\begin{array}{l}
t \overset{b}{\leftarrow} s \overset{a}{\leftarrow} r \overset{b}{\leftarrow} q \overset{c}{\leftarrow} q \overset{c}{\leftarrow} q \overset{b}{\leftarrow} p \overset{a}{\leftarrow} p \overset{a}{\leftarrow} p \\
t \overset{b}{\leftarrow} t \overset{a}{\leftarrow} t \overset{b}{\leftarrow} t \overset{c}{\leftarrow} s \overset{c}{\leftarrow} t \overset{b}{\leftarrow} s \overset{a}{\leftarrow} r \overset{a}{\leftarrow} q \\
t \overset{b}{\leftarrow} t \overset{a}{\leftarrow} t \overset{b}{\leftarrow} t \overset{c}{\leftarrow} s \overset{c}{\leftarrow} t \overset{b}{\leftarrow} t \overset{a}{\leftarrow} s \overset{a}{\leftarrow} r \\
t \overset{b}{\leftarrow} t \overset{a}{\leftarrow} t \overset{b}{\leftarrow} t \overset{c}{\leftarrow} s \overset{c}{\leftarrow} t \overset{b}{\leftarrow} t \overset{a}{\leftarrow} t \overset{a}{\leftarrow} s \\
t \overset{b}{\leftarrow} t \overset{a}{\leftarrow} t \overset{b}{\leftarrow} t \overset{c}{\leftarrow} s \overset{c}{\leftarrow} t \overset{b}{\leftarrow} t \overset{a}{\leftarrow} t \overset{a}{\leftarrow} t
\end{array}
$$

Then $\mathrm{PS}_{\mathcal{B}}(w)$ contains the following path summaries:

$$(1, s)(4, q)(3, p),\ (6, s)(2, q),\ (7, s)(1, r),\ (8, s),\ (8, t).$$

The *path summary algorithm for* $\mathcal{B}$ is a streaming algorithm over $\Sigma_{\downarrow}$ described in Algorithm 4.1 where the acceptance condition is defined later.

Lemma 4.2. *Algorithm 4.1 correctly maintains* $\mathrm{PS}_{\mathcal{B}}(w)$ *for the active window* $w \in \Sigma^*$.

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

Assume $S = \mathrm{PS}_{\mathcal{B}}(w)$ for some window $w \in \Sigma^*$ and $a \in \Sigma$. The claim is that the algorithm computes $S' = \mathrm{PS}_{\mathcal{B}}(wa)$ from S. Suppose that π' is a run in $\mathcal{B}$ on wa. It has the form $\pi' = \pi\,(p_1, a, p_0)$ and $\mathrm{ps}(\pi) \in S$. Let $\pi = \pi_k \tau_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$ be the SCC-factorization of π. If p_0 and p_1 are strongly connected then the SCC-factorization of π' is $\pi' = \pi_k \tau_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1'$ where $\pi_1' = \pi_1\,(p_1, a, p_0)$, and otherwise $\pi' = \pi_k \tau_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1\,(p_1, a, p_0)$. In this way the algorithm computes $\mathrm{ps}(\pi')$ from $\mathrm{ps}(\pi)$.

Now consider the case $a = \downarrow$. We have $w = \varepsilon$ if and only if $\mathrm{PS}_{\mathcal{B}}(w) = \{0\} \times Q$, and in this case the set of path summaries S is unchanged. Otherwise assume $w = bv$ for some $b \in \Sigma$. We claim that the algorithm computes $S' = \mathrm{PS}_{\mathcal{B}}(v)$ from S. Suppose that π' is a run in $\mathcal{B}$ on v which ends in state $p \in Q$. If $q \overset{b}{\leftarrow} p$ in $\mathcal{B}$ then let $\pi = (q, b, p)\,\pi'$, and we have $\mathrm{ps}(\pi) \in S$. Let $\pi = \pi_k \tau_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$

Algorithm 4.1: The path summary algorithm

```
initialize $S = \{0\} \times Q$;
foreach input $a \in \Sigma_\downarrow$ do
    $S' = \emptyset$;
    if $a \in \Sigma$ then
        for $p_0 \in Q$ do
            let $p_1 = a \cdot p_0$ and $(\ell_k, p_k) \cdots (\ell_1, p_1) \in S$;
            if $p_0$ and $p_1$ are strongly connected then
                add $(\ell_k, p_k) \cdots (\ell_1 + 1, p_0)$ to $S'$;
            else
                add $(\ell_k, p_k) \cdots (\ell_1, p_1)(1, p_0)$ to $S'$;
    if $a = \downarrow$ then
        if $S = \{0\} \times Q$ then
            $S' = S$;
        else
            for $(\ell_k, p_k) \cdots (\ell_1, p_1) \in S$ do
                if $\ell_k \geqslant 1$ then
                    add $(\ell_k - 1, p_k) \cdots (\ell_1, p_1)$ to $S'$;
                else
                    add $(\ell_{k-1} - 1, p_{k-1}) \cdots (\ell_1, p_1)$ to $S'$;
    replace $S$ by $S'$;
```

be the SCC-factorization of π. If $|\pi_k| \geqslant 1$ then $\pi' = \pi_k' \tau_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$ is the SCC-factorization of π' where $\pi_k = (q, b, p)\, \pi_k'$. Otherwise π_k is empty and $\tau_{k-1} = (q, b, p)$. Therefore $\pi' = \tau_{k-1} \cdots \tau_2 \pi_2 \tau_1 \pi_1$ is SCC-factorization of π'. In this way the algorithm computes $\mathrm{ps}(\pi')$ from $\mathrm{ps}(\pi)$. □

Proposition 4.3. *The path summary algorithm for $\mathcal{B}$ has space complexity* $O(|\mathcal{B}|^2 \cdot (\log n + \log |\mathcal{B}|))$.

Proof. A single path summary $\mathrm{ps}(\pi)$ consists of a sequence of at most $|\mathcal{B}|$ states, which can be encoded in $O(|\mathcal{B}| \cdot \log |\mathcal{B}|)$ bits, and a sequence $(\ell_k, \dots, \ell_1)$ of $k \leqslant |\mathcal{B}|$ numbers up to $|\pi|$, which can be encoded in $O(|\mathcal{B}| \cdot \log |\pi|)$. Since $\mathrm{PS}_{\mathcal{B}}(w)$ contains $|\mathcal{B}|$ such path summaries in total $\mathrm{PS}_{\mathcal{B}}(w)$ can be encoded in $O(|\mathcal{B}|^2 \cdot (\log |w| + \log |\mathcal{B}|))$ bits. □

Well-behaved rDFAs Now we describe the class of automata $\mathcal{B}$ for which the set of path summaries determines whether the window is accepted by $\mathcal{B}$. A subset $P \subseteq Q$ is *convex* (with respect to the preorder $\preceq_{\mathcal{B}}$) if $p \preceq_{\mathcal{B}} q \preceq_{\mathcal{B}} r$ and $p, r \in P$ implies $q \in P$. In particular, every SCC is convex, and every convex set is a union of SCCs. A convex subset $P \subseteq Q$ is *well-behaved* if for any two P-runs π_1, π_2 which start in the same state and have equal length, either both π_1 and π_2 are accepting or both are rejecting. If every reachable SCC in $\mathcal{B}$ is well-behaved then $\mathcal{B}$ is called *well-behaved*. A path summary is called *accepting* if it is the path summary of some accepting run. Notice that a path summary $\mathrm{ps}(\pi) = (\ell_k, p_k) \cdots (\ell_1, p_1)$ is accepting if and only if there exists an accepting run of length ℓ_k starting in p_k.

Lemma 4.4. *Let $\mathcal{B}$ be well-behaved and let π be a run in $\mathcal{B}$ starting in a reachable state. Then π is accepting if and only if $\mathrm{ps}(\pi)$ is accepting.*

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

Hence, we let path summary algorithm for $\mathcal{B}$ accept if the path summary starting in q_0 is accepting. Combining Proposition 4.3 and Lemma 4.4 yields:

Proposition 4.5. *If $\mathcal{B}$ is well-behaved then $L = L(\mathcal{B})$ has space complexity $V_L(n) = O(|\mathcal{B}|^2 \cdot \log n)$, which is $O(\log n)$ for fixed $\mathcal{B}$.*

Proof. Let $n \in \mathbb{N}$ be a window size. If $n \leqslant |\mathcal{B}|$ then the trivial streaming algorithm for $\mathrm{SW}_n(L)$ uses $O(n) \leqslant O(|\mathcal{B}|)$ bits. If $n > |\mathcal{B}|$ then we use the path summary algorithm for $\mathrm{SW}_n(L)$ which uses $O(|\mathcal{B}|^2 \cdot (\log n + \log |\mathcal{B}|)) \leqslant O(|\mathcal{B}|^2 \cdot \log n)$ bits. □

Implementation details To implement the above algorithm on a realistic computation model, we have to be able to efficiently determine whether a path summary is accepting. Given a number $d \geqslant 1$, a set of natural numbers $X \subseteq \mathbb{N}$ is d-*periodic* if we have $x \in X$ if and only if $x + d \in X$. A state $q \in Q$ is *transient* if $x \cdot q \neq q$ for all $x \in \Sigma^+$. Every transient state in $\mathcal{B}$ forms an SCC of size one (a *transient SCC*); however, not every SCC of size one is transient.

Lemma 4.6. *Let $P \subseteq Q$ be a well-behaved subset in $\mathcal{B}$ and $p_0 \in P$ be nontransient. Then $\mathrm{Acc}(P, p_0) = \{|\pi| : \pi$ is an accepting P-run starting in $p_0\}$ is d-periodic for some $d \leqslant |Q|$.*

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

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

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

In the following we describe how to implement Proposition 4.5. We do the following preprocessing on the well-behaved rDFA $\mathcal{B}$. Using depth-first search we compute all SCCs in $\mathcal{B}$. For every SCC P we pick a state $p \in P$ and compute the distance $\mathrm{dist}(p, q)$ from p to all states $q \in P$ using any shortest paths algorithm. Furthermore let d be the minimal length of a nonempty run from p to p itself, which is the period d from Lemma 4.6. If no such run exists then we store the information that p is transient. Otherwise we assign to each state $q \in P$ the distance from p modulo d. By traversing an arbitrary P-run of length d from p we can compute a bit vector of length d which represents $\mathrm{Acc}(P, p_0)$. Using this information we can easily answer whether a path summary $(\ell_k, p_k) \cdots (\ell_1, p_1)$ is accepting: it is accepting if and only if either p_k is transient, $\ell_k = 0$ and $p_k \in F$, or $\mathrm{dist}(p_0, p_k) + \ell_k \bmod d$ belongs to $\mathrm{Acc}(P, p_0)$ where P is the SCC of p_k and p_0 is the picked state in P.

Non-well-behaved rDFAs Now let us prove a linear lower bound for automata which are not well-behaved. Let $L \subseteq \Sigma^*$ be a language. We say that L *separates* two words $x, y \in \Sigma^*$ if $|\{x, y\} \cap L| = 1$. We say that L *separates* two languages $K_1, K_2 \subseteq \Sigma^*$ if $K_1 \subseteq L$ and $K_2 \cap L = \emptyset$, or $K_2 \subseteq L$ and $K_1 \cap L = \emptyset$.

Lemma 4.7. *If $\mathcal{B}$ is not well-behaved then there exist words $u_1, u_2, v_1, v_2, z \in \Sigma^*$ where $|u_i| = |v_i|$ for $i = 1, 2$ such that $L = L(\mathcal{B})$ separates $u_2\{u_1u_2, v_1v_2\}^*z$ and $v_2\{u_1u_2, v_1v_2\}^*z$.*

Proof. Since $\mathcal{B}$ is not well-behaved there are states $p \in Q$, $q \in F$, $r \in Q \setminus F$ and words $u = u_1u_2, v = v_1v_2, z \in \Sigma^*$ such that $|u_2| = |v_2|$ and

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

We can ensure that $|u_1| = |v_1|$ and hence also $|u| = |v|$: If $k = |u|$ and $\ell = |v|$ we replace u_1 by $u^{\ell-1}u_1$ and v_1 by $v^{k-1}v_1$, which preserves all properties above. Then $u_2\{u, v\}^*z$ and $v_2\{u, v\}^*z$ are separated by L. ☐

Proposition 4.8. *Under the conditions of Lemma 4.7 the language $L = L(\mathcal{B})$ satisfies $F_L(n) = \Omega(n)$ for infinitely many n, and $V_L(n) = \Omega(n)$.*

Proof. Let $u_1, u_2, v_1, v_2, z \in \Sigma^*$ be the words from Lemma 4.7 and let $u = u_1u_2$ and $v = v_1v_2$. Now consider an SW-algorithm $\mathcal{P}_n$ for L and window length $n = |u_2| + |u| \cdot (m - 1) + |z|$ for some $m \geqslant 1$. We prove that $\mathcal{P}_n$ has at least 2^m many states by showing that $\mathcal{P}_n(x) \neq \mathcal{P}_n(y)$ for any $x \neq y \in \{u, v\}^m$. Notice that $|\{u, v\}^m| = 2^m$ since $u \neq v$ and $|u| = |v|$.

Read two distinct words $x, y \in \{u, v\}^m$ into two instances of $\mathcal{P}_n$. Consider the last $\{u, v\}$-block where x and y differ. Without loss of generality assume $x = x'us$ and $y = y'vs$ for some $x', y', s \in \{u, v\}^*$. By reading $x'z$ into both instances the window of the x-instance becomes $\mathrm{last}_n(xx'z) = u_2sx'z$ and the window of the y-instance becomes $\mathrm{last}_n(yx'z) = v_2sx'z$. By Lemma 4.7 the two windows are separated by L, and therefore the algorithm $\mathcal{P}_n$ must accept one of the streams $xx'z$ and $yx'z$, and reject the other. In conclusion $\mathcal{P}_n(x) \neq \mathcal{P}_n(y)$ and hence $\mathcal{P}_n$ must use $\Omega(m) = \Omega(n)$ bits.

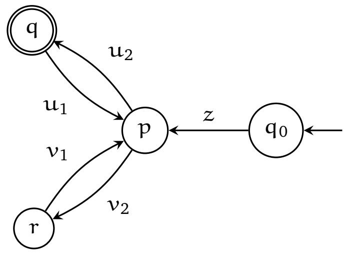

Figure 4.3: Forbidden pattern for well-behaved rDFAs where $|u_1| = |v_1|$ and $|u_2| = |v_2|$.

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

From Proposition 4.5 and Proposition 4.8 we obtain:

Corollary 4.9. *Let* $X \in \{F, V\}$. *A regular language* $L \subseteq \Sigma^*$ *satisfies* $X_L(n) = O(\log n)$ *if and only if* L *is recognized by a well-behaved rDFA.*

Finally, let us observe that the lower bound from Proposition 4.8 already holds for the suffix complexity $S_L(n)$:

Proposition 4.10. *Under the conditions of Lemma 4.7 the language* $L = L(\mathcal{B})$ *has suffix complexity* $S_L(n) = \Omega(n)$.

Proof. We claim that $|\{u_1u_2, v_1v_2\}^m z/{\approx_L}| \geqslant 2^m$. This holds because any two distinct words from $\{u_1u_2, v_1v_2\}^m z$ have equal-length suffixes of the form u_2xz and v_2xz for some $x \in \{u_1u_2, v_1v_2\}^*$, which are separated by L. Hence $S_L(m) = \log|\Sigma^{\leqslant m}/{\approx_L}| = \Omega(m)$ holds over an arithmetic progression and therefore $S_L(n) = \Omega(n)$ by monotonicity of $S_L(n)$. □

Together with $S_L(n) \leqslant V_L(n)$ this implies that for every regular language L all three functions $F_L(n)$, $V_L(n)$ and $S_L(n)$ are (simultaneously) at most logarithmic or at least linear.

4.1.2 Constant space

Next we study which regular languages have sublogarithmic complexity. Recall that in the variable-size model any such language must be trivial because the algorithm must at least maintain the current window size by Lemma 3.7.

Corollary 4.11. *The empty language* $L = \emptyset$ *and the universal language* $L = \Sigma^*$ *satisfy* $V_L(n) = O(1)$. *All other languages satisfy* $V_L(n) = \Omega(\log n)$.

Theorem 4.12 (Trichotomy in the variable-size model). *Every regular language has space complexity* $\Theta(1)$, $\Theta(\log n)$ *or* $\Theta(n)$ *in the variable-size sliding window model.*

Proof. Let L be regular. If L empty or universal then the space complexity is $O(1)$. Otherwise, it is $\Omega(\log n)$ by Corollary 4.11. Furthermore, if it is accepted by a well-behaved rDFA then its space complexity is $\Theta(\log n)$ by Proposition 4.5, and otherwise $\Theta(n)$ by Proposition 4.8. □

Now we can focus on the fixed-size model. Let $U(\mathcal{B}) \subseteq Q$ be the set of states $q \in Q$ such that exists a nontransient state $p \in Q$ such that q is reachable from p and p is reachable from the initial state q_0. Notice that $q \in U(\mathcal{B})$ if and only if there exist runs of unbounded length from q_0 to q (hence the symbol U for unbounded).

Proposition 4.13. *If* $U(\mathcal{B})$ *is well-behaved then* L *has space complexity* $F_L(n) = O(|\mathcal{B}|)$*, which is* $O(1)$ *for fixed* $\mathcal{B}$*.*

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

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

of $\mathcal{B}$ on $\mathrm{last}_n(x)$. By the choice of k some state $p \in Q$ must occur twice in the run $q \xleftarrow{\mathrm{last}_k(x)} q_0$, Therefore, p is nontransient and all states in the run $r \xleftarrow{s} q$ belong to $U(\mathcal{B})$. Since $U(\mathcal{B})$ is well-behaved r is final if and only if some run of length $|s|$ starting in q is accepting. This can be determined since $|s| = n - k$ is known. □

Lemma 4.14. *If* $U(\mathcal{B})$ *is not well-behaved then there exist words* $x, y, z \in \Sigma^*$ *where* $|x| = |y|$ *such that* $L = L(\mathcal{B})$ *separates* xy^*z *and* y^*z*.*

Proof. Since $U(\mathcal{B})$ is not well-behaved there are $U(\mathcal{B})$-runs π and ρ from the same starting state q such that $|\pi| = |\rho|$ and exactly one of the runs π and ρ is accepting. By definition of $U(\mathcal{B})$ the state q is reachable from a nontransient state p via some run σ, which itself is reachable from the initial state q_0, say $p \xleftarrow{z} q_0$. We can replace π by $\pi\sigma$ and ρ by $\rho\sigma$ preserving the properties of being $U(\mathcal{B})$-runs and $|\pi| = |\rho|$. Assume that π and ρ are runs on words $v \in \Sigma^*$ and $w \in \Sigma^*$. Since p is nontransient we can construct internal runs from p to p of unbounded lengths. Consider such a run $p \xleftarrow{u} p$ of length $|u| \geqslant |v| = |w|$. Then L separates vu^*z and wu^*z. Factorize $u = u_1u_2$ such that $|u_2| = |v| = |w|$. Notice that all words in u_2u^*z reach the same state in $\mathcal{B}$ and hence either u_2u^*z is either contained in L or disjoint from L. Then L separates u_2u^*z and vu^*z, or u_2u^*z and wu^*z. Hence L also separates $(u_2u_1)^*u_2u_1u_2z$ from either $vu_1(u_2u_1)^*u_2u_1u_2z$ or from $wu_1(u_2u_1)^*u_2u_1u_2z$. □

Proposition 4.15. *Under the conditions of Lemma 4.14 the language* $L = L(\mathcal{B})$ *satisfies* $F_L(n) = \Omega(\log n)$ *for infinitely many* n*.*

Proof. Let $x, y, z \in \Sigma^*$ be the words from Lemma 4.14. Consider an SW-algorithm $\mathcal{P}_n$ for L and window length $n = |x| + |y| \cdot m + |z|$ for some $m \geqslant 1$. We prove that $\mathcal{P}_n$ has at least m many states by showing that $\mathcal{P}_n(xy^i) \neq \mathcal{P}_n(xy^j)$ for any $1 \leqslant i < j \leqslant m$. Let $1 \leqslant i < j \leqslant m$. Then we have

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

and

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

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

This implies the space trichotomy in the fixed-size model:

Theorem 4.16 (Trichotomy in the fixed-size model)**.** *In the fixed-size sliding window model every regular language has space complexity*

- $O(1)$,
- $O(\log n)$ *and* $\Omega(\log n)$ *infinitely often, or*
- $O(n)$ *and* $\Omega(n)$ *infinitely often.*

4.1.3 Alternative proofs

In the rest of this section we give alternative proofs for the space trichotomies, using the equivalence relations defined in Section 3.4. Recall that by Proposition 3.11 for any nontrivial language L we have

$$V_L(n) = \log |\overleftarrow{\nu}_L(\Sigma^{\leqslant n})|$$

for all $n \in \mathbb{N}$. If L is regular then $\sim_L$ has finite index and ν_L has a finite range, which yields a regular representation of $\overleftarrow{\nu}_L(\Sigma^*)$.

Lemma 4.17. *If* $L \subseteq \Sigma^*$ *is regular, then* $\overleftarrow{\nu}_L$ *is a* $\longleftarrow$*-transduction. In particular,* $\overleftarrow{\nu}_L(\Sigma^*)$ *and* $\overleftarrow{\nu}_L(L)$ *are regular. Furthermore* $\overleftarrow{\nu}_L$ *is a* $\longleftarrow$*-reduction from* L *to* $\overleftarrow{\nu}_L(L)$.

Proof. Since L is regular the quotient set $\Sigma^*/{\sim_L}$ is finite. Let $h\colon \Sigma^* \to M$ be the syntactic homomorphism of L into the syntactic monoid M of L. Since the syntactic congruence refines the Myhill-Nerode congruence, there exists a function $\nu\colon M \to \Sigma^*/{\sim_L}$ such that $[x]_L = \nu(h(x))$ for all $x \in \Sigma^*$. Define the right Mealy machine $\mathcal{M} = (M, \Sigma, \Sigma^*/{\sim_L}, \delta, 1)$ and transitions

$$h(a) \cdot m \xleftarrow{a|\nu(h(a)\cdot m)} m$$

for all $m \in M$, $a \in \Sigma$. This Mealy machine computes the $\longleftarrow$-transduction $\overleftarrow{\nu}_L$.

To see that $\tilde{\nu}_L$ is indeed a reduction from L to $\tilde{\nu}_L(L)$ notice $x \in L$ clearly implies $\tilde{\nu}_L(x) \in \tilde{\nu}_L(L)$. Conversely, if $\tilde{\nu}_L(x) \in \tilde{\nu}_L(L)$ then there exists $y \in L$ with $\tilde{\nu}_L(x) = \tilde{\nu}_L(y)$. Therefore we have $x \approx_L y$, which implies $x \sim_L y$ and thus $x \in L$. □

The trichotomy theorem for variable-size windows can now be reproven.

Alternative proof of Theorem 4.12. If L is empty or universal then $V_L(n) = O(1)$, and otherwise $V_L(n) = \Omega(\log n)$ by Corollary 4.11. In the latter case its space complexity is $V_L(n) = \log |\tilde{\nu}_L(\Sigma^{\leqslant n})|$ by Proposition 3.11, which is the logarithm of the cumulative growth function of the language $\tilde{\nu}_L(\Sigma^*)$. Since $\tilde{\nu}_L(\Sigma^*)$ is regular by Lemma 4.17 its cumulative growth is either polynomial or $2^{\Omega(n)}$. Therefore $V_L(n)$ is either $O(\log n)$ or $\Omega(n)$. □

For the fixed-size model recall the equivalence relation ρ_L on Σ^* defined as:

$$x \;\rho_L\; y \iff |x| = |y| = n \text{ and}$$
$$(\mathrm{last}_n(xz) \in L \iff \mathrm{last}_n(yz) \in L) \text{ for all } z \in \Sigma^{\leqslant n}.$$

Lemma 4.18. *If $L \subseteq \Sigma^*$ is regular then ρ_L is synchronous rational.*

Proof. First consider the relation $T \subseteq (\Sigma^*)^3$ of all tuples (x, y, z) such that $n = |x| = |y| \geqslant |z|$, $\mathrm{last}_n(xz) \in L$ and $\mathrm{last}_n(yz) \notin L$. We claim that T is synchronous rational by constructing an automaton for $\otimes T$. Notice that $x \;\rho_L\; y$ if and only if $|x| = |y| = n$ and there exists no $z \in \Sigma^{\leqslant n}$ such that $(x, y, z) \in T$ or $(y, x, z) \in T$. If T is synchronous rational then so is ρ_L by the closure properties of synchronous rational relations.

It remains to show that T is synchronous rational. Observe that $\otimes T$ contains all convolutions

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \cdots \begin{pmatrix} x_k \\ y_k \\ z_k \end{pmatrix} \begin{pmatrix} x_{k+1} \\ y_{k+1} \\ \diamond \end{pmatrix} \cdots \begin{pmatrix} x_n \\ y_n \\ \diamond \end{pmatrix} \tag{4.1}$$

such that $x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_k \in \Sigma$, $0 \leqslant k \leqslant n$, $x_{k+1} \cdots x_n z_1 \cdots z_k \in L$ and $y_{k+1} \cdots y_n z_1 \cdots z_k \notin L$. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be a DFA for L. For all tuples $(p, q, p', q') \in Q^4$ one can construct an automaton $\mathcal{A}_{p,q,p',q'}$ which accepts exactly all convolutions of the form (4.1) such that $q_0 \cdot x_{k+1} \cdots x_n = p$, $q_0 \cdot y_{k+1} \cdots y_n = q$, $p \cdot z_1 \cdots z_k = p'$ and $q \cdot z_1 \cdots z_k = q'$. It first simulates two copies of $\mathcal{A}$ on $z_1 \cdots z_k$ starting from p and q, respectively, and verifies whether p' and q' are reached. Then it simulates two copies of $\mathcal{A}$ on the suffixes $x_{k+1} \cdots x_n$ and $y_{k+1} \cdots y_n$, starting from the initial state q_0 and verifying whether p and q are reached. Then we have $\otimes T = \bigcup_{p,q,p',q' \in Q} L(\mathcal{A}_{p,q,p',q'})$, and therefore T is synchronous rational. □

Alternative proof of Theorem 4.16. Proposition 3.9 states $F_L(n) = \log |\Sigma^n/\rho_L|$. By definition every ρ_L-class is contained in some Σ^n. Let $R \subseteq \Sigma^*$ be the set of all lexicographic minimal representatives from each ρ_L-class. It satisfies $F_L(n) = \log |R \cap \Sigma^n|$. Since the lexicographic linear ordering on Σ^* is synchronous rational, R is a regular set. By Theorem 2.5 the growth of R is either (i) $O(1)$, or (ii) polynomial in n and $\Omega(n)$ for infinitely many n, or (iii) exponential in n. By taking logarithms we get the trichotomy for $F_L(n)$. □

4.2 Characterization of the space classes

Next we will provide natural characterizations of the languages with space complexity $O(\log n)$ and $O(1)$. For the log-space class this will be done by the characterization via well-behaved rDFAs. For the constant space class we will utilize a distance notion between states in a DFA.

4.2.1 Constant space complexity

We now start with the description for constant space. Recall that this class only contains the trivial languages in the variable-size model by Corollary 4.11.

A language $L \subseteq \Sigma^*$ is called a *length language* if for all $n \in \mathbb{N}$, either $\Sigma^n \subseteq L$ or $L \cap \Sigma^n = \emptyset$. A language $L \subseteq \Sigma^*$ is called k-*suffix testable* if for all $x, y \in \Sigma^*$ and $z \in \Sigma^k$ we have

$$xz \in L \iff yz \in L.$$

Equivalently, L is a Boolean combination of languages of the form $\Sigma^* w$ where $w \in \Sigma^{\leqslant k}$. We call L *suffix testable* if it is k-suffix testable for some $k \geqslant 0$. We emphasize that the notions of length languages and suffix testable languages only make sense with respect to an underlying alphabet. Clearly, every finite language is suffix testable: If k is the maximum length of a word in $L \subseteq \Sigma^*$ then L is $(k+1)$-suffix testable since $L = \bigcup_{w \in L} \{w\}$ and $\{w\} = \Sigma^* w \setminus \bigcup_{a \in \Sigma} \Sigma^* a w$. The class of suffix testable languages corresponds to the variety **D** of definite monoids [106]. The main theorem of this section is:

Theorem 4.19. *Let $L \subseteq \Sigma^*$ be regular. The following statements are equivalent:*

(i) $F_L(n) = O(1)$

(ii) L *is a finite Boolean combination of suffix testable languages and regular length languages.*

The following definitions are useful, which are also studied in [54]. We define the distance $d(K, L)$ by

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

Notice that $d(K, L) < \infty$ if and only if $K \triangle L$ is finite. For a DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ and a state $p \in Q$, we define $\mathcal{A}_p = (Q, \Sigma, p, \delta, F)$. For two states $p, q \in Q$, we define the distance $d(p, q) = d(L(\mathcal{A}_p), L(\mathcal{A}_q))$. If we have two runs $p \xrightarrow{u} p'$ and $q \xrightarrow{u} q'$ where $p' \in F$, $q' \notin F$ and $|u| \geqslant |Q|^2$ then some state pair occurs twice in the runs and we can pump the runs to unbounded lengths. Therefore

$d(p, q) < \infty$ implies $d(p, q) \leqslant |Q|^2$. In fact $d(p, q) < \infty$ implies $d(p, q) \leqslant |Q|$ by [54, Lemma 1].

Lemma 4.20. *Let* $L \subseteq \Sigma^*$ *be regular and* $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ *be its minimal DFA. We have:*

(i) $d(p, q) \leqslant k$ *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* $d(p, q) \leqslant k$ *for all* $p, q \in Q$.

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

Proof. The proof of (i) is an easy induction: If $k = 0$, the statement is $d(p, q) = 0$ iff $p = q$, which is true because $\mathcal{A}$ is minimal. For the induction step, we have $d(p, q) \leqslant k + 1$ iff $d(\delta(p, a), \delta(q, a)) \leqslant k$ 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 $z \in L(\mathcal{A}_p) \triangle L(\mathcal{A}_q)$, then $|z| < k$ because $xz \in L$ iff $yz \notin L$ and L is k-suffix testable.

Now assume that $d(p, q) \leqslant k$ for all $p, q \in Q$ and consider $x, y \in \Sigma^*$, $z \in \Sigma^k$. Since $d(\mathcal{A}(x), \mathcal{A}(y)) \leqslant k$, (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 from [54, Lemma 1]. □

Lemma 4.21. *For any* $L \subseteq \Sigma^*$ *and* $n \geqslant 0$*, the language* $\mathsf{SW}_n(L)$ *is* $2^{F_L(n)}$*-suffix testable.*

Proof. Let $\mathcal{P}_n$ be an SW-algorithm for L and window length n with space complexity $F_L(n)$. Therefore $\mathcal{P}_n$ has at most $2^{F_L(n)}$ states. The language $\mathsf{SW}_n(L)$ is n-suffix testable because

$$\mathsf{SW}_n(L) = \{x \in \Sigma^{\leqslant n-1} \mid \mathrm{last}_n(x) \in L\} \cup \Sigma^*(L \cap \Sigma^n).$$

By Lemma 4.20(iii) $\mathsf{SW}_n(L)$ is $2^{F_L(n)}$-suffix testable. □

Corollary 4.22. *Let* $L \subseteq \Sigma^*$ *be a language. Then* $F_L(n) = O(1)$ *if and only if there exists a* $k \geqslant 0$ *such that* $\mathsf{SW}_n(L)$ *is* k*-suffix testable for all* $n \geqslant 0$.

Proof. The left-to-right direction follows from Lemma 4.21. If each $\mathsf{SW}_n(L)$ is k-suffix testable, then an SW-algorithm for window length n only needs to maintain the last k symbols in the stream to test membership of the active window of length n in $\mathsf{SW}_n(L)$, or equivalently in L. □

Proof of Theorem 4.19. First, let $L \subseteq \Sigma^*$ be a regular language with $F_L(n) = O(1)$. By Lemma 4.21 there exists $k \geqslant 0$ such that $\mathsf{SW}_n(L)$ is k-suffix testable for all $n \geqslant 0$. We can express L as the Boolean combination

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

where the right quotient $Lz^{-1} = \{x \in \Sigma^* \mid xz \in L\}$ is regular [18, Chapter 3, Example 5.7]. The set $L \cap \Sigma^{\leqslant k-1}$ is finite and hence suffix testable. It remains to show that each Lz^{-1} is a length language. Consider two words $x, y \in \Sigma^*$ of the same length $|x| = |y| = n$. Since $|xz| = |yz| = n + k$ and $\mathsf{SW}_{n+k}(L)$ is k-suffix testable we have $xz \in L$ iff $yz \in L$, and hence $x \in Lz^{-1}$ iff $y \in Lz^{-1}$.

For the other direction note that: (i) If L is a length language or a suffix testable language then clearly $F_L(n) = O(1)$, and (ii) $\{L \subseteq \Sigma^* \mid F_L(n) = O(1)\}$ is closed under Boolean operations by Corollary 3.18. This proves the theorem. □

We give another characterization of the regular languages with constant space complexity based on the minimal DFA, which will be used later for a decision procedure.

Proposition 4.23. *Let $L \subseteq \Sigma^*$ be regular and $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be its minimal DFA. Then $F_L(n) = O(1)$ if and only if for all $x, y \in \Sigma^*$ with $|x| = |y|$ and $z \in \Sigma^{|Q|}$ we have $\mathcal{A}(xz) = \mathcal{A}(yz)$.*

Proof. Assume that $F_L(n) = O(1)$. By Corollary 4.22 there exists $k \geqslant 0$ such that each $\mathsf{SW}_n(L)$ is k-suffix testable. Let $x, y \in \Sigma^*$ with $|x| = |y| = n$. For all $z \in \Sigma^{\geqslant k}$ we have $xz \in \mathsf{SW}_{n+|z|}(L)$ iff $yz \in \mathsf{SW}_{n+|z|}(L)$. Thus, $xz \in L$ iff $yz \in L$ for all $z \in \Sigma^{\geqslant k}$, and hence $d(\mathcal{A}(x), \mathcal{A}(y)) \leqslant k$. Again by [54, Lemma 1] we get $d(\mathcal{A}(x), \mathcal{A}(y)) \leqslant |Q|$. By Lemma 4.20(i) we have $\mathcal{A}(xz) = \mathcal{A}(yz)$ for all $z \in \Sigma^{|Q|}$.

Conversely, assume that $\mathcal{A}(xz) = \mathcal{A}(yz)$ for all $x, y \in \Sigma^*$ with $|x| = |y|$ and $z \in \Sigma^{|Q|}$. This means that one can simulate the automaton on the window of length n by only storing the last $|Q|$ many symbols and hence in space $O(1)$. □

4.2.2 Characterization of the log-space class

In the following we will give two natural characterizations of the regular languages with space complexity $O(\log n)$. By Corollary 4.9 we can equivalently consider the fixed- or the variable-size model. A language $L \subseteq \Sigma^*$ is called a *left ideal* if $\Sigma^* L \subseteq L$, or equivalently $\Sigma^* L = L$. Every suffix testable language is a Boolean combination of languages $\Sigma^* w$ and therefore of regular left ideals. In this section we will prove the following theorem.

Theorem 4.24. *Let $L \subseteq \Sigma^*$ be regular. The following statements are equivalent:*

(i) $F_L(n) = O(\log n)$

(ii) $V_L(n) = O(\log n)$

(iii) L *is recognized by a well-behaved rDFA.*

(iv) L *is $\leftarrow$-reducible to a regular language of polynomial growth.*

(v) L *is a finite Boolean combination of regular left ideals and regular length languages.*

The equivalence of the points (i), (ii), (iii) is stated in Corollary 4.9. The other equivalences are shown by the chain of implications $(ii) \to (iv) \to (v) \to (iii)$.

The implication from (ii) to (iv) follows from previous statements: If $V_L(n) = O(\log n)$ and L is nontrivial then by Proposition 3.11 has $\overleftarrow{v}_L(\Sigma^*)$ polynomial (cumulative) growth and hence also its subset $\overleftarrow{v}_L(L)$ has polynomial growth. By Lemma 4.17 there exists a $\leftarrow$-reduction from L to $\overleftarrow{v}_L(L)$ (namely $\overleftarrow{v}_L$), and $\overleftarrow{v}_L(L)$ is regular. If $L = \emptyset$ then it has polynomial growth and is $\leftarrow$-reducible to itself. If $L = \Sigma^*$ then it is $\leftarrow$-reducible to a unary universal language a^*. In the rest of the section, we prove the implications $(v) \to (iii)$ and $(iv) \to (v)$.

Implication $(v) \to (iii)$

The implication $(v) \to (iii)$ follows from the following two lemmas.

Lemma 4.25. *If a regular language $L \subseteq \Sigma^*$ is a left ideal or a length language, then any rDFA $\mathcal{B}$ for L is well-behaved.*

Proof. Let $\mathcal{B}$ be an rDFA for L. If L is a length language then for all reachable states q and all runs π, π' starting from q with $|\pi| = |\pi'|$ we have π is accepting if and only if π' is accepting.

If L is a left ideal, then whenever a final state p is reachable, and q is reachable from p, then q is also final. Hence, for every reachable SCC P in $\mathcal{B}$ either all states of P are final or all states of P are nonfinal. □

Lemma 4.26. *The class of languages $L \subseteq \Sigma^*$ recognized by well-behaved rDFAs is closed under Boolean operations.*

Proof. If $\mathcal{B}$ is well-behaved then the complement automaton $\overline{\mathcal{B}}$ is also well-behaved. Given two well-behaved rDFAs $\mathcal{B}_1, \mathcal{B}_2$, we claim that the product automaton $\mathcal{B}_1 \times \mathcal{B}_2$ recognizing the intersection language is also well-behaved. Suppose that $\mathcal{B}_i = (Q_i, \Sigma, F_i, \delta_i, q_i)$ for $i = 1, 2$. The product automaton for the intersection language is defined by $\mathcal{B}_1 \times \mathcal{B}_2 = (Q_1 \times Q_2, \Sigma, F_1 \times F_2, \delta, (q_1, q_2))$ where $\delta(a, (q_1, q_2)) = (\delta_1(a, q_1), \delta_2(a, q_2))$ for all $q_1 \in Q_1, q_2 \in Q_2$ and $a \in \Sigma$. Consider an SCC S of $\mathcal{B}_1 \times \mathcal{B}_2$ which is reachable from the initial state and let $(p_1, p_2), (q_1, q_2), (r_1, r_2) \in S$ such that

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

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

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

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

Implication $(iv) \to (v)$

It remains to show the implication from (iv) to (v).

Lemma 4.27. *The class of finite Boolean combinations of regular left ideals and regular length languages is closed under pre-images of* $\leftarrow$*-transductions.*

Proof. For any function $\tau : \Sigma^* \to \Gamma^*$ and $K, L \subseteq \Gamma^*$ we have $\tau^{-1}(K \cup L) = \tau^{-1}(K) \cup \tau^{-1}(L)$ and $\tau^{-1}(\Gamma^* \setminus L) = \Sigma^* \setminus \tau^{-1}(L)$. Now assume that τ is a $\leftarrow$-transduction. If L is regular then $\tau^{-1}(L)$ is also regular by the closure properties of rational transductions. Since τ is length-preserving, the pre-image of a length language L under τ is again a length language (namely L itself). Finally, we claim that $\tau^{-1}(\Gamma^* L) = \Sigma^* \tau^{-1}(L)$, i.e. pre-images of left ideals under τ are left ideals again, and hence, regular left ideals are closed under preimages of $\leftarrow$-transductions.

Since τ is recognized by a Mealy machine we have $\mathrm{Suf}(\tau(x)) = \tau(\mathrm{Suf}(x))$. The statement $x \in \tau^{-1}(\Gamma^* L)$ is equivalent to the existence of a suffix x' of x with $\tau(x') \in L$. Since τ is recognized by a Mealy machine, this is again equivalent to saying that some suffix of $\tau(x)$ belongs to L. This can also be written as $x \in \Sigma^* \tau^{-1}(L)$, which concludes the proof. $\square$

Towards the proof of the direction (iv) to (v) in Theorem 4.24 it remains to prove that every regular language of polynomial growth is a finite Boolean combination of regular left ideals and regular length languages. Since L and its reversal L^R have the same growth we can equivalently write every regular language of polynomial growth as a finite Boolean combination of regular right ideals and regular length languages. A *right ideal* is a language $L \subseteq \Sigma^*$ with $L\Sigma^* \subseteq L$. The idea is to decompose every regular language of polynomial growth as a finite union of languages recognized by so-called linear cycle automata, which can be viewed as regular expressions $v_0 u_1^* v_1 \cdots u_k^* v_k$ that can be parsed uniquely from left to right.

A *partial DFA* $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is defined as a DFA except that its transition function $\delta \colon Q \times \Sigma \to Q$ is a partial function. If the language accepted by a partial DFA $\mathcal{A}$ is nonempty it can be *reduced* by removing all states which are either not reachable from q_0 or from which no final state can be reached.

Theorem 4.28 ([107, Theorem 1 and Lemma 6])**.** *Let* $\mathcal{A}$ *be a reduced partial DFA for* L*. Then* L *has polynomial growth if and only if for every state* p *in* $\mathcal{A}$ *there exists at most one transition* $p \xrightarrow{a} q$ *such that* p *and* q *are strongly connected.*

A partial DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ is a *linear cycle automaton* if

- the set of SCCs is linearly ordered by the reachability order,
- for all $p, q \in Q$ there exists at most one symbol $a \in \Sigma$ such that $p \xrightarrow{a} q$,
- for all $1 \leqslant i \leqslant m$ and $p \in C_i$ there exists at most one transition $p \xrightarrow{a} q$ with $q \in C_i$,

- for all $1 \leqslant i \leqslant m-1$ there exists exactly one transition $p \xrightarrow{a} q$ with $p \in C_i$ and $q \notin C_i$, and this state q belongs to C_{i+1} (we call p the *exit state* of C_i and q the *entry state* of C_{i+1}),
- $|F| = 1$ and $F \subseteq C_m$.

Notice that every SCC in a linear cycle automaton is either transient or a *cycle*, i.e. restricted to the SCC, every state has exactly one incoming and one outgoing transition.

Lemma 4.29. *If L is a regular language with polynomial growth, then L is a finite union of languages recognized by linear cycle automata.*

Proof. We can assume that $L \neq \emptyset$ (since $L = \emptyset$ is the empty union). Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be a reduced partial DFA for L. If π is a run in $\mathcal{A}$ let $d(\pi) = (s_1, t_1, \dots, s_m, t_m) \in Q^{2m}$ be the sequence of states in $\mathcal{A}$ such that π traverses exactly m SCCs where s_i and t_i are the first and the last state visited in the i-th SCC. We call $d(\pi)$ the *path description* of π. Notice that there are only finitely many path descriptions.

Now consider a single path description $d = (s_1, t_1, \dots, s_m, t_m)$ which starts in the initial state $s_1 = q_0$ and ends in a final state $t_m \in F$. We obtain a partial DFA $\mathcal{A}_d$ from $\mathcal{A}$ as follows: restrict $\mathcal{A}$ to all SCCs containing $s_1, \dots, s_m$, and remove all transitions between two distinct SCCs except for the transitions $\{(t_i, s_{i+1}) \mid 1 \leqslant i \leqslant m-1\}$. Finally, t_m is declared as the only final state. By Theorem 4.28, $\mathcal{A}_d$ is indeed a linear cycle automaton and we have $L(\mathcal{A}) = \bigcup_d L(\mathcal{A}_d)$ where the union is taken over all path descriptions d starting in q_0 and ending in a final state. $\square$

Lemma 4.30. *Let $\mathcal{A}$ be a linear cycle automaton. There are linear cycle automata $\mathcal{A}_1, \dots, \mathcal{A}_s$ such that $L(\mathcal{A}) = \bigcup_{i=1}^{s} L(\mathcal{A}_i)$ and in each $\mathcal{A}_i$ all cycles have uniform length.*

Proof. Let $m_1, \dots, m_k$ be the lengths of each cycle in $\mathcal{A}$ and m be the least common multiple of $m_1, \dots, m_k$. The language $L(\mathcal{A})$ is the finite union of all languages accepted by linear cycle automata that are obtained from $\mathcal{A}$ by doing the following replacement for every nontrivial cycle

$$C : q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \ \cdots \ q_{m_i-1} \xrightarrow{a_{m_i-1}} q_{m_i} \xrightarrow{a_{m_i}} q_1$$

of $\mathcal{A}$. W.l.o.g. assume that q_1 is either the initial state of $\mathcal{A}$ or the entry state of C. Choose an arbitrary number $0 \leqslant d_i < m/m_i$ (we then take the finite union over all such choices). We replace C by a path P of length $d_i m_i$ followed by cycle C' of length m, having the form

$$P : q_1' \xrightarrow{(a_1 \cdots a_{m_i})^{d_i}} q_1, \quad C' : q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \cdots q_{m-1} \xrightarrow{a_{m-1}} q_m \xrightarrow{a_m} q_1,$$

where $a_1 a_2 \cdots a_m = (a_1 a_2 \cdots a_{m_i})^{m/m_i}$. All states on the path P except for q_1 are new and also all states $q_{m_i+1}, \dots, q_m$ are new. If q_1 is the initial state of

$\mathcal{A}$ then q_1' is the new initial state. Otherwise, the unique transition entering C is redirected to the new state q_1'. The union of the languages recognized by all automata of this form is $L(\mathcal{A})$. $\square$

Lemma 4.31. *Let $\mathcal{A}$ be a linear cycle automaton in which all cycles have uniform length. Then $L(\mathcal{A})$ is a Boolean combination of regular right ideals and regular length-languages.*

Proof. Let $L \subseteq \Sigma^*$ be the language recognized by $\mathcal{A}$. There are numbers $p, q \geqslant 0$ such that each word in L has length $p + qn$ for some $n \geqslant 0$. Here q is the uniform cycle length in $\mathcal{A}$. We claim that L is the intersection of the three languages

(i) $L \cdot \Sigma^*$, which is a regular right ideal,

(ii) $\{x \in \Sigma^* \mid \mathrm{Pref}(x) \subseteq \mathrm{Pref}(L)\}$, which is the complement of the regular right ideal $(\Sigma^* \setminus \mathrm{Pref}(L)) \cdot \Sigma^*$,

(iii) $\Sigma^p(\Sigma^q)^*$, which is a regular length language.

Clearly L is contained in the described intersection. Conversely, consider a word x in the intersection. We can factorize $x = yz$ for some $y \in L$ by (i). Hence, $|y| = p + qn$ for some n by the assumption above. Since $|x| = p + qn'$ for some n' by (iii), the length $|z| = |x| - |y|$ is divided by q. By (ii) x is a prefix of a word in L and hence $\mathcal{A}$ has some initial run π on x, say $\pi\colon q_0 \xrightarrow{y} s \xrightarrow{z} t$. Since $y \in L$, the state s must be the unique final state of $\mathcal{A}$, which belongs to the unique minimal SCC C of $\mathcal{A}$. If C is a cycle, then it has length q and hence $t = s$, i.e. $x \in L$. If C is transient, then $z = \varepsilon$ and therefore $x = y \in L$. $\square$

This concludes the proof for the direction from (iv) to (v) in Theorem 4.24.

4.2.3 Boolean combinations of regular left ideals

Let us look at the subclass of finite Boolean combinations of regular left ideals. Such languages describe Boolean combinations of the form "there exists a suffix with some regular property" and "every suffix has some regular property". They can furthermore be described by *weak rDFAs*, i.e. rDFAs $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ where every reachable SCC is either contained or disjoint from F. Every weak rDFA is well-behaved.

Theorem 4.32 ([69])**.** *A language is a finite Boolean combination of regular left ideals if and only if it is recognized by a weak rDFA.*

Alternatively, this language class can be characterized using the notion of alternation points. Given a word $x = a_1 \cdots a_n \in \Sigma^*$ and a language $L \subseteq \Sigma^*$, a position $1 \leqslant i \leqslant n$ is an L-*alternation point* if $a_i \cdots a_n$ and $a_{i+1} \cdots a_n$ are separated by L. Denote by $\mathrm{alt}_L(x)$ the number of L-alternation points in x.

Lemma 4.33. *If $L \subseteq \Sigma^*$ is a Boolean combination of at most k left ideals then $\mathrm{alt}_L(x) \leqslant k$ for all $x \in \Sigma^*$. Conversely, if $L \subseteq \Sigma^*$ is regular and $\mathrm{alt}_L(x) \leqslant k$ for all $x \in \Sigma^*$ then L is a Boolean combination of at most k regular left ideals.*

Proof. If L is a Boolean combination of left ideals $L_1, \dots, L_k$, then each L-alternation point in a word is an L_i-alternation point for some $1 \leqslant i \leqslant k$. Since L_i is a left ideal, each word has at most one L_i-alternating point and we obtain $\mathrm{alt}_L(x) \leqslant k$ for all $x \in \Sigma^*$.

Conversely, assume that $\mathrm{alt}_L(x) \leqslant k$ for all $x \in \Sigma^*$. Without loss of generality assume $\varepsilon \in L$, which ensures that $x \in L$ if and only if $\mathrm{alt}_L(x)$ is even. If $\varepsilon \notin L$, then $x \in L$ if and only if $\mathrm{alt}_L(x)$ is odd, and we can argue similarly as below. We define $P_i = \{x \in \Sigma^* \mid \mathrm{alt}_L(x) \geqslant i\}$ for $i \geqslant 0$ and write L as

$$L = \bigcup_{0 \leqslant i \leqslant k \text{ even}} (P_i \setminus P_{i+1}).$$

Each P_i is a left ideal because prolonging a word on the left only increases the number of L-alternation points. Furthermore, each P_i is regular: by enriching a DFA for L with a counter up to i, a DFA can verify that the input x satisfies $\mathrm{alt}_L(x) \geqslant i$. Using the fact that $P_0 = \Sigma^*$ and $P_i = \emptyset$ for all $i > k$, we can write L as

$$L = \begin{cases} (\Sigma^* \setminus P_1) \cup (P_2 \setminus P_3) \cup \cdots \cup (P_{k-2} \setminus P_{k-1}) \cup P_k, & \text{if } k \text{ is even} \\ (\Sigma^* \setminus P_1) \cup (P_2 \setminus P_3) \cup \cdots \cup (P_{k-1} \setminus P_k), & \text{if } k \text{ is odd.} \end{cases}$$

This proves that L is a Boolean combination of the regular left ideals $P_1, \dots, P_k$, which concludes the proof. □

We can add yet another equivalent statement to Theorem 4.24.

Proposition 4.34. *Let $L \subseteq \Sigma^*$ be regular and assume that L is a finite Boolean combination of left ideals and length languages. Then any rDFA for L is well-behaved.*

Proof. Let $\mathcal{B}$ be an rDFA for L and assume that $\mathcal{B}$ is not well-behaved. By Lemma 4.7 there exist words u_1, u_2, v_1, v_2, z such that $|u_i| = |v_i|$ for $i = 1, 2$ and L separates $u_2\{u_1u_2, v_1v_2\}^*z$ and $v_2\{u_1u_2, v_1v_2\}^*z$.

The given representation yields left ideals $L_1, \dots, L_k$ and sets of lengths $N_1, \dots, N_m \subseteq \mathbb{N}$ such that membership of a word w to L is determined by whether $w \in L_i$ for $1 \leqslant i \leqslant k$ and whether $|w| \in N_j$ for $1 \leqslant j \leqslant m$. Let $c \colon \mathbb{N} \to C$ be the finite coloring where $C = \{0,1\}^m$ and the bitvector $c(n) = (c_1, \dots, c_m)$ indicates whether $n \in N_j$ for $1 \leqslant j \leqslant m$. Then there exists an infinite monochromatic subset $N \subseteq \{|u_2| + d|u_1u_2| + |z| : d \in \mathbb{N}\}$, say $N = \{|u_2| + d|u_1u_2| + |z| : d \in D\}$ where

$$D = \{d_1, d_1 + d_2, d_1 + d_2 + d_3, \dots\}$$

for some $d_1, d_2, \dots \geqslant 1$. Hence membership of a word w in $\{u_2, v_2\}\{u_1u_2, v_1v_2\}^*z$ of length $n \in N$ to L is only determined by whether $w \in L_i$ for $1 \leqslant i \leqslant k$. Equivalently, there exists a Boolean combination L' of $L_1, \dots, L_k$ such that L and L' agree on all words of length $n \in N$.

Define $w_1 = u_2(u_1u_2)^{d_1}z$. For even $i \geqslant 2$ let $w_i = v_2(u_1u_2)^{d_i-1}u_1w_{i-1}$; for odd $i \geqslant 2$ let $w_i = u_2(u_1u_2)^{d_i-1}v_1w_{i-1}$. Observe that for all $i \geqslant 1$ we have $|w_i| \in N$, w_i is a suffix of w_{i+1}, and L separates w_i and w_{i+1}. Hence the words w_i have an unbounded number of L'-alternation points, which contradicts Lemma 4.33. $\square$

4.2.4 Summary of language classes

Let us summarize the language classes (over a fixed alphabet Σ) discussed in this chapter:

- **Reg**: the class of all regular languages over Σ
- **Len**: the class of regular length languages
- **LI**: the class of regular left ideals
- **ST**: the class of suffix testable languages

A class of languages $\mathbf{A}$ over Σ is *Boolean closed* if $K, L \in \mathbf{A}$ implies $\Sigma^* \setminus L \in \mathbf{A}$ and $K \cup L \in \mathbf{A}$. If $\mathbf{A}_1, \dots, \mathbf{A}_n$ are classes of languages over some alphabet Σ, then $\langle \mathbf{A}_1, \dots, \mathbf{A}_n \rangle$ denotes the Boolean closure of $\bigcup_{i=1}^n \mathbf{A}_i$, i.e. the smallest Boolean closed class which contains $\bigcup_{i=1}^n \mathbf{A}_i$. By the results from this chapter for every regular language $L \subseteq \Sigma^*$ we have:

Theorem 4.35 (Fixed-size model). *For every regular language L we have:*

- *If* $L \in \langle \mathbf{ST}, \mathbf{Len} \rangle$ *then* $F_L(n) = O(1)$.
- *If* $L \notin \langle \mathbf{ST}, \mathbf{Len} \rangle$ *then* $F_L(n) = \Omega(\log n)$ *infinitely often.*
- *If* $L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$ *then* $F_L(n) = O(\log n)$.
- *If* $L \notin \langle \mathbf{LI}, \mathbf{Len} \rangle$ *then* $F_L(n) = \Omega(n)$ *infinitely often.*

Theorem 4.36 (Variable-size model). *For every regular language L we have:*

- *If L is trivial then* $V_L(n) = O(1)$.
- *If L is nontrivial then* $V_L(n) = \Omega(\log n)$.
- *If* $L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$ *then* $V_L(n) = O(\log n)$.
- *If* $L \notin \langle \mathbf{LI}, \mathbf{Len} \rangle$ *then* $V_L(n) = \Omega(n)$.

4.3 The uniform problem

In this section we consider the setting where the automaton is part of the input. Let us first summarize the results from Section 4.1 which are based on properties of the right-deterministic finite automaton. Suppose that $\mathcal{B}$ is an rDFA for $L \subseteq \Sigma^*$. Then:

(i) If $\mathcal{B}$ is well-behaved then $V_L(n) = O(|\mathcal{B}|^2 \cdot \log n)$ (Proposition 4.5).

(ii) If $U(\mathcal{B})$ is well-behaved then $F_L(n) = O(|\mathcal{B}|)$ (Proposition 4.13).

Let us first provide matching lower bounds for the above upper bounds. Afterwards we consider the problem where the regular language is given by a DFA/NFA.

4.3.1 Bounds for rDFAs

Proposition 4.37. *For every $m \in \mathbb{N}$ there exists a language $L_m \subseteq \{a, b, 0\}^*$ recognized by a well-behaved rDFA $\mathcal{B}_m$ with $2m + 1$ states such that $F_{L_m}(n) \geqslant \frac{1}{18} \cdot |\mathcal{B}_m|^2 \cdot \log n$ for sufficiently large $n \in \mathbb{N}$*[1].

Proof. For $m \in \mathbb{N}$ we define an rDFA $\mathcal{B}_m = (Q_m, \Sigma, F_m, \delta, q_0)$ as follows. The state set is $Q_m = \{q_0, \dots, q_{2m}\}$, the alphabet is $\Sigma = \{a, b, 0\}$ and the initial state is q_0. The transitions in $\mathcal{B}_m$ are

- $q_{i+1} \xleftarrow{b} q_i$ for all $0 \leqslant i \leqslant m - 1$,
- $q_{i+1} \xleftarrow{a} q_i$ for all $m \leqslant i \leqslant 2m - 1$,
- and the loops $q_i \xleftarrow{a} q_i$ for all $0 \leqslant i \leqslant m - 1$, $q_i \xleftarrow{b} q_i$ for all $m \leqslant i \leqslant 2m$ and $r \xleftarrow{0} r$ for all $r \in Q$.

Finally we set $F = \{q_i \in Q_m \mid i \text{ even}\}$. Notice that the SCCs in $\mathcal{B}_m$ are singletons and Q_m is linearly ordered by $\preceq_{\mathcal{B}_m}$. Let $\pi\colon \{a, b, 0\}^* \to \{a, b\}^*$ be the projection to the subalphabet $\{a, b\}$. Since 0 is a neutral letter in $\mathcal{B}_m$ we have $x \cdot q = \pi(x) \cdot q$ for all $q \in Q$ and $x \in \{a, b, 0\}^*$.

Now consider the set $X = \pi^{-1}((a^{m-1}b)^m)$. If $n \geqslant m^2$ then X contains $\binom{n}{m^2}$ words of length n. Let $n \geqslant m^2$ and $\mathcal{P}_n$ be an SW-algorithm for $L(\mathcal{B}_m)$ and window length n. Read two distinct words $x, y \in X$ of length n into two instances of $\mathcal{P}_n$, which will be distinguished in the following. We can factorize $x = x_1cz$ and $y = y_1c'z$ such that $x_1, y_1, z \in \{a, b, 0\}^*$, and $\{c, c'\}$ is either $\{a, 0\}$ or $\{b, 0\}$. Without loss of generality let $c' = 0$.

First assume that $c = b$. By inserting $k = n - |bz|$ of 0-symbols into both instances we obtain the windows $bz0^k$ and $0z0^k$. Since the number of b's in the windows differs by one and the windows contain at most m many b's the windows are separated by $L(\mathcal{B}_m)$.

Now assume that $c = a$. There exist numbers $0 \leqslant i, j \leqslant m - 1$ such that $\pi(z) = a^i b(a^{m-1}b)^j$. Since x_1 contains $m - j - 1$ many b-symbols we can insert into both instances of $\mathcal{P}_n$ the word $b^{m-j-1}0^k$ where $k = n - (m - j - 1) - |az|$ and obtain the windows $azb^{m-j-1}0^k$ and $0zb^{m-j-1}0^k$. Since

$$\pi(azb^{m-j-1}0^k) = a^{i+1}b(a^{m-1}b)^j b^{m-j-1}$$

[1]Here and in Theorem 4.42 the bounds hold for all $n \geqslant n_m$ for some threshold n_m depending on m.

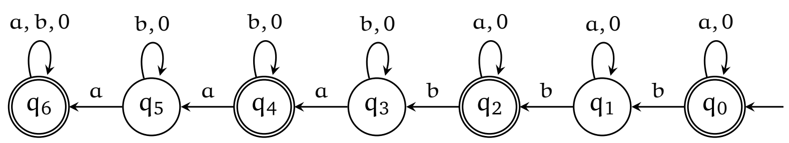

Figure 4.4: Lower bound automaton $\mathcal{A}_3$ from Proposition 4.37.

and

$$\pi(0zb^{m-j-1}0^k) = a^i b(a^{m-1}b)^j b^{m-j-1},$$

the window $\mathrm{last}_n(xb^{m-j-1}0^k)$ reaches state q_{m+i+1} in $\mathcal{B}_m$ whereas the window $\mathrm{last}_n(xb^{m-j-1}0^k)$ reaches state q_{m+i}. Hence the windows are separated by $L(\mathcal{B}_m)$.

Since X contains $\binom{n}{m^2}$ words of length n we have $\binom{n}{m^2}$ many streams which are pairwise distinguished by $\mathcal{P}_n$. Therefore $\mathcal{P}_n$ uses at least

$$\log\binom{n}{m^2} \geqslant m^2 \cdot \log\frac{n}{m^2}$$

bits. If n is sufficiently large (satisfying $\sqrt{n} \geqslant m^2$) then this is at least

$$m^2 \cdot \log\sqrt{n} = \frac{1}{2}m^2 \log n \geqslant \frac{1}{2}\left(\frac{|\mathcal{B}_m|}{3}\right)^2 \log n = \frac{1}{18}|\mathcal{B}_m|^2 \log n$$

bits. □

Proposition 4.38. *For every $m \in \mathbb{N}$ there exists a language $L_m \subseteq \{a,b\}^*$ recognized by an rDFA $\mathcal{B}_m$ such that $U(\mathcal{B}_m)$ is well-behaved, $|\mathcal{B}_m| = m+2$ and $F_{L_m}(n) \geqslant m$ for all $n \geqslant m$.*

Proof. Let L_m be the set of words whose m-th last letter is a. It is recognized by an rDFA $\mathcal{B}_m$ consisting of m transient states, a single final state, and a single nonfinal sink state. Furthermore $U(\mathcal{B}_m)$ is well-behaved. Let $\mathcal{P}_n$ be a sliding window algorithm for L_m and window size $n \geqslant m$. We claim that $\mathcal{P}_n(x) \neq \mathcal{P}_n(y)$ for all $x \neq y \in \{a,b\}^m$: If $x \neq y \in \{a,b\}^m$ differ at some position $1 \leqslant i \leqslant m$ then $\mathrm{last}_n(xa^{i-1})$ and $\mathrm{last}_n(ya^{i-1})$ differ at the m-th last position. Therefore $\mathcal{P}_n$ accepts exactly one of the streams xa^{i-1} and ya^{i-1}, and thus $\mathcal{P}_n$ must distinguish x and y. In conclusion $\mathcal{P}_n$ must have at least 2^m states. □

Hence, the upper bounds in Proposition 4.5 and Proposition 4.13 are optimal.

4.3.2 Upper bounds for DFAs/NFAs

If the regular language is given by a NFA $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$, one can convert it into an equivalent rDFA by reversing the transitions (formally, we just view it as a right automaton) and applying the powerset construction, which entails an exponential blowup (it has $2^{|\mathcal{A}|}$ states). We obtain the rDFA $\mathcal{A}^{\mathsf{RD}} = (2^Q, \Sigma, \mathcal{I}, \delta, F)$. with transitions

$$\delta(a, P) = \{q \in Q \mid p \in P,\ (q, a, p) \in \Delta\}, \text{ for all } P \subseteq Q, a \in \Sigma,$$

and final states

$$\mathcal{I} = \{P \subseteq Q \mid P \cap I \neq \emptyset\}.$$

By applying the bounds for rDFAs we get the following space bounds:

- If $\mathcal{A}^{\mathsf{RD}}$ is well-behaved then $V_{L(\mathcal{A})} = O(4^{|\mathcal{A}|} \cdot \log n)$ by Proposition 4.5.
- If $U(\mathcal{A}^{\mathsf{RD}})$ is well-behaved then $F_{L(\mathcal{A})} = O(2^{|\mathcal{A}|})$ by Proposition 4.13.

We will improve both bounds in the case where $\mathcal{A}$ is a DFA.

Theorem 4.39. *Let $\mathcal{A}$ be a finite automaton for $L \subseteq \Sigma^*$ and assume that $\mathcal{B} = \mathcal{A}^{\mathsf{RD}}$ is well-behaved.*

- *If $\mathcal{A}$ is a DFA then $V_L(n) = O(2^{|\mathcal{A}|} \cdot |\mathcal{A}| \cdot \log n)$.*
- *If $\mathcal{A}$ is an NFA then $V_L(n) = O(4^{|\mathcal{A}|} \cdot \log n)$.*

Proof. The NFA case was already discussed above. Now assume that $\mathcal{A}$ is a DFA for L. We need the following combinatorial result. Consider a family of m pairwise distinct sets $\{X_1, \dots, X_m\}$. Then it is known that there exists a *differentiating set* D of size at most $m - 1$, i.e. for any $1 \leqslant i < j \leqslant m$ we have $D \cap (X_i \triangle X_j) \neq \emptyset$ [95].

A set $D \subseteq \Sigma^*$ *distinguishes* L if for all $x, y \in \Sigma^*$ with $x \not\sim_L y$ there exists $z \in D$ such that exactly one of the words xz and yz belongs to L. We apply the result above to the set of left quotients. Since there are at most $|\mathcal{A}|$ distinct left quotients $x^{-1}L$ we get a set D of size at most $|\mathcal{A}| - 1$ that distinguishes L.

For a window $w = a_1 \cdots a_n$ we define a 0-1-matrix $M_w \colon D \times \{1, \dots, n\} \to \{0, 1\}$ by $M_w(z, i) = 1$ iff $a_i \cdots a_n z \in L$. Notice that the i-th column $M_w(\cdot, i)$ determines $[a_i \cdots a_n]_{\sim_L}$, and vice versa, for all $1 \leqslant i \leqslant n$. Thus M_w determines $[a_1 \cdots a_n]_{\approx_L}$, and vice versa. By Proposition 3.11 it suffices to provide an encoding of M_w with the desired bounds.

We can encode each row $M_w(z, \cdot)$ of M_w succinctly as follows. Consider one row indexed by $z \in D$. Let π_z be the initial run of $\mathcal{B}$ on the word wz and $\tilde{\pi}_z$ be the subrun of π_z which only reads the prefix w of wz. One can reconstruct $M_w(z, \cdot)$ from the path summary of $\tilde{\pi}_z$ by Lemma 4.4 because $\mathcal{B}$ is well-behaved. Thus M_w can be represented by a list of $|D|$ many path summaries. A single path summary $\mathsf{ps}(\pi_z)$ for $z \in D$ can be encoded using $O(|\mathcal{B}| \cdot (\log n + \log |\mathcal{B}|))$. Hence we can encode M_w using $O(2^{|\mathcal{A}|} \cdot |\mathcal{A}| \cdot (\log n + |\mathcal{A}|))$ bits.

To get rid of the addend $|\mathcal{A}|$ we do a case distinction on the window size as in the proof of Proposition 4.5. Parallel to the above algorithm which preserves the matrix M_w succinctly, we maintain the suffix of length $|\mathcal{A}|$ in the stream and the window length $n \in \mathbb{N}$. Whenever the window size is strictly smaller than $|\mathcal{A}|$ we remove the representation of M_w. Then the algorithm only uses

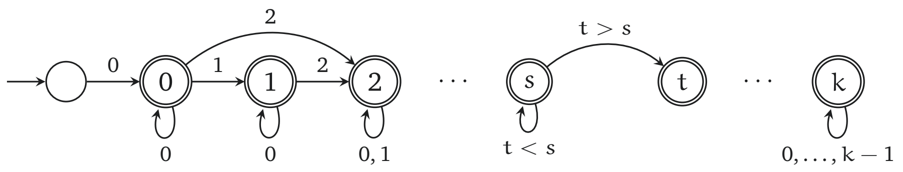

Figure 4.5: An automaton for L_k. Omitted transitions lead to a sink state. All states are final, except from the initial and the sink state.

$O(|\mathcal{A}| + \log n)$ bits, which satisfies the claimed bound. Whenever the window size is increased from $|\mathcal{A}| - 1$ to $|\mathcal{A}|$ we reconstruct the representation of M_w from the explicitly stored window of length $n = |\mathcal{A}|$. The representation of M_w (using path summaries) takes $O(2^{|\mathcal{A}|} \cdot |\mathcal{A}| \cdot (\log n + |\mathcal{A}|))$ bits, which is bounded by $O(2^{|\mathcal{A}|} \cdot |\mathcal{A}| \cdot \log n)$. □

Theorem 4.40. *Let $\mathcal{A}$ be a finite automaton for $L \subseteq \Sigma^*$, let $\mathcal{B} = \mathcal{A}^{\mathsf{RD}}$, and assume that $U(\mathcal{B})$ is well-behaved.*

- *If $\mathcal{A}$ is a DFA then $F_L(n) = O(|\mathcal{A}|)$.*
- *If $\mathcal{A}$ is an NFA then $F_L(n) = O(2^{|\mathcal{A}|})$.*

Proof. The NFA case was already discussed above. If $\mathcal{A}$ is a DFA with k states then the sliding window algorithm $\mathcal{P}_n$ for window size n simply maintains $\mathrm{last}_k(w)$ where w is the stream prefix read so far. This requires $O(|\mathcal{A}|)$ bits. If $n < k$ then we know the window explicitly and we can test membership to L. If $n \geqslant k$ then the active window $\mathrm{last}_n(w)$ and the word $\mathrm{last}_n(\mathrm{last}_k(w))$ have the common suffix $\mathrm{last}_k(w)$. By Proposition 4.23 we can decide whether the window belongs to L, using the fact that $\mathrm{last}_n(w) \in L$ if and only if $\mathrm{last}_n(\mathrm{last}_k(w)) \in L$. □

4.3.3 Lower bounds for DFAs

In the following we show that the exponential space bound in Theorem 4.39 is unavoidable, already for the fixed-size sliding window model. For $k \geqslant 0$ we define the language $L_k \subseteq \{0, \dots, k\}^*$ by

- $L_0 = 0^+$, and
- $L_k = L_{k-1} \cup L_{k-1}\, k \{0, \dots, k-1\}^*$ for $k \geqslant 1$.

Observe that a word $a_1 \cdots a_n \in \{0, \dots, k\}^*$ belongs to L_k if and only if $n \geqslant 1$, $a_1 = 0$ and for each $1 \leqslant i \leqslant n$ it holds that $a_i = 0$ or $a_i \neq \max_{1 \leqslant j \leqslant i-1} a_j$. We can construct a DFA $\mathcal{A}_k$ for L_k with $k + 3$ states, which stores the maximum value seen so far in its state, see Figure 4.5.

To prove that each L_k has space complexity $O(\log n)$ in the variable-size model, we show that L_k is a Boolean combination of regular left ideals by showing that the number of L_k-alternation points is bounded.

Lemma 4.41. *For all* $k \geqslant 0$ *and* $x \in \mathbb{N}^*$ *we have* $\mathrm{alt}_{L_k}(x) \leqslant 2^{k+2} - 2$ *and hence* $V_{L_k}(n) = O(\log n)$.

Proof. We prove the lemma by induction on $k \geqslant 0$. Clearly each word has at most 2 alternation points with respect to $L_0 = 0^+$. Now let $k \geqslant 1$ and $x \in \mathbb{N}^*$. If all occurring numbers in x are at most $k-1$, then $\mathrm{alt}_{L_k}(x) = \mathrm{alt}_{L_{k-1}}(x)$ and the claim follows by induction. Otherwise consider the last occurrence of a number $\geqslant k$ and factorize $x = y\ell z$ where $y \in \mathbb{N}^*$, $\ell \geqslant k$ and $z \in \{0, \dots, k-1\}^*$. If $\ell > k$, then the first $|y|$ positions of x cannot contain L_k-alternation points and we get

$$\mathrm{alt}_{L_k}(x) \leqslant 1 + \mathrm{alt}_{L_k}(z) = 1 + \mathrm{alt}_{L_{k-1}}(z) \leqslant 2^{k+1} - 1 \leqslant 2^{k+2} - 2.$$

Now assume $x = ykz$. By the definition of L_k each L_k-alternation point in x is either (i) an L_{k-1}-alternation point in y, (ii) an L_{k-1}-alternation point in z, or (iii) position $|y| + 1$ (i.e. the last position, where k occurs). Hence we have

$$\mathrm{alt}_{L_k}(x) \leqslant 1 + \mathrm{alt}_{L_{k-1}}(y) + \mathrm{alt}_{L_{k-1}}(z) \leqslant 1 + (2^{k+1} - 2) + (2^{k+1} - 2) \leqslant 2^{k+2} - 2.$$

By Lemma 4.33 and Theorem 4.24 we also conclude $V_{L_k}(n) = O(\log n)$. □

Theorem 4.42. *For each* $k \geqslant 1$ *there exists a language* $L_k \subseteq \{0, \dots, k\}^*$ *recognized by a DFA* $\mathcal{A}_k$ *with* $k + 3$ *states such that* $V_{L_k}(n) = O(\log n)$ *and* $F_{L_k}(n) \geqslant \frac{1}{32} \cdot 2^{|\mathcal{A}_k|} \cdot \log n$ *for sufficiently large* $n \in \mathbb{N}$.

Proof. Of course, we take the languages L_k considered in this section. We define the languages $Z_0 = 0^*$ and $Z_k = Z_{k-1}\ k\ Z_{k-1}$ for $k \geqslant 1$. An example word from Z_3 is 0010002100300010020010. The crucial fact about words $x \in Z_k$ that we are using is the following: Every suffix of x that starts with 0 belongs to L_k and every suffix of x that starts with $a > 0$ does not belong to L_k. The former follows by induction on k; the latter holds since words in L_k start with 0.

Fix some $k \geqslant 1$ and let $\mathcal{P}_n$ be an SW-algorithm for L_k and window length $n \geqslant 2^k - 1$. We claim that $\mathcal{P}_n$ distinguishes all $\binom{n}{2^k-1}$ words in Z_k of length n. Let $x, y \in Z_k$ such that $|x| = |y| = n$ and $x \neq y$. Read x and y into two instances of $\mathcal{P}_n$. We can write $x = zau$ and $y = zbv$ with $a, b \in \{0, \dots, k\}$, $a \neq b$. We must have $a = 0$ and $b > 0$ or vice versa. Assume that $a = 0$ and $b > 0$. Thus, $au \in L_k$ and $bv \notin L_k$. Hence, we have $\mathrm{last}_n(x0^{|z|}) = au0^{|z|} \in L_k$ and $\mathrm{last}_n(y0^{|z|}) = bv0^{|z|} \notin L_k$. Therefore we can distinguish x and y in $\mathcal{P}_n$ by the word $0^{|z|}$, which proves the claim.

The claim above implies that $\mathcal{B}_n$ has at least $\binom{n}{2^k-1}$ many states. Hence, the space complexity of $\mathcal{B}$ is at least

$$\log \binom{n}{2^k - 1} \geqslant (2^k - 1) \log \left(\frac{n}{2^k - 1} \right)$$

many bits. If n is sufficiently large (satisfying $\sqrt{n} \geqslant 2^k - 1$) then this is at least

$$2^{k-1} \cdot \log \sqrt{n} = 2^{k-2} \cdot \log n = 2^{|\mathcal{A}_k| - 5} \cdot \log n$$

many bits. $\square$

Notice that the lower bound matches the upper bound of $O(2^{|\mathcal{A}|} \cdot |\mathcal{A}| \cdot \log n)$ from Theorem 4.39 up to a linear factor in $|\mathcal{A}|$. In particular the space complexity must be the exponential in the DFA size $|\mathcal{A}|$. A matching lower bound of $4^{|\mathcal{A}|} \cdot \log n$ for the NFA case remains open.

In the following we reduce the alphabet size of the DFA $\mathcal{A}_k$ in Theorem 4.42 to a constant with a logarithmic size blowup. Let $\Sigma = \{0, \dots, k\}$ be a linearly ordered alphabet and let $\mathsf{bin}\colon \Sigma \to \{0,1\}^{\lceil \log k \rceil}$ be the binary encoding function where the most significant bit is to the left. We define the bijection $\mathsf{code}\colon \Sigma^* \to \{0,1,\#\}^*$ by $\mathsf{code}(a_1 \cdots a_n) = \# \,\mathsf{bin}(a_1) \# \,\mathsf{bin}(a_2) \cdots \# \,\mathsf{bin}(a_n)$. Notice that $|\mathsf{code}(x)| = (\lceil \log k \rceil + 1) \cdot |x|$ for all $x \in \Sigma^*$. We need to prove that L_k and $\mathsf{code}(L_k)$ have the same asymptotic space complexity and that $\mathsf{code}(L_k)$ has a small DFA.

Lemma 4.43. *Let Σ be an alphabet, $c = \lceil \log |\Sigma| \rceil + 1$ and $L \subseteq \Sigma^*$. We have $F_L(n) \leqslant F_{\mathsf{code}(L)}(cn) \leqslant F_L(n) + O(c + \log(cn))$ for all $n \in \mathbb{N}$.*

Proof. Let $n \in \mathbb{N}$, and $m = c \cdot n$. If $\mathcal{Q}_m$ is an SW-algorithm for $\mathsf{code}(L)$ and window size m then we can construct an SW-algorithm $\mathcal{P}$ for L and window size n. If $\square \in \Sigma$ is the padding symbol then initially $\mathcal{P}_m$ is simulated on $\mathsf{code}(\square^n)$ to reach its initial state. Then, on every input symbol $a \in \Sigma$ it reads $\# \,\mathsf{bin}(a)$ into $\mathcal{Q}_m$ and copies its output. This shows that $F_L(n) \leqslant F_{\mathsf{code}(L)}(cn)$ for all $n \in \mathbb{N}$.

Now let $\mathcal{P}_n$ be an SW-algorithm for L and window size n. We describe an SW-algorithm $\mathcal{Q}_m$ for $\mathsf{code}(L)$ and window size m. Notice that if the window length is not divided by c then the sliding window problem for $\mathsf{code}(L)$ becomes trivial. Now assume that m is divided by c. For an input stream let s be its longest suffix which is a prefix of some word in $\mathsf{code}(\Sigma^*)$. The algorithm maintains its length $|s|$ up to threshold $m+1$. This can be done by memorizing the last c symbols explicitly. If s is not the complete window, i.e. $|s| = m$, the algorithm always rejects. Otherwise it adopts the output of the following simulation of $\mathcal{P}_n$. If the suffix of length c in the window contains two or more #-symbols, we reset $\mathcal{P}_n$, i.e. it is set back to its initial state. If the suffix of length c in the window is of the form $\# \,\mathsf{bin}(a)$ for some $a \in \Sigma$, we simulate $\mathcal{P}_n$ on a, otherwise we also reset $\mathcal{P}_n$. $\square$

It is easy to construct a DFA $\mathcal{A}'_k$ for $\mathsf{code}(L_k)$ of size $O(|\mathcal{A}_k| \cdot k) \leqslant O(k^2)$. Using the properties of $\mathcal{A}_k$ we can improve this bound to $O(k \log k)$.

Lemma 4.44. *For all $k \geqslant 1$ there exists a DFA $\mathcal{A}'_k$ for $\mathsf{code}(L_k)$ with $O(k \log k)$ states.*

Proof. Recall that in each state s of $\mathcal{A}_k$ (except from the initial and the sink state) the input symbol t is compared to s. If $s = t$ then the automaton goes to a sink state. If $s < t$ then it moves to state t, and if $s > t$ then it stays in state s.

Figure 4.6 shows the DFA $\mathcal{A}'_3$ for $\mathsf{code}(L_3)$. In general a DFA $\mathcal{A}'_k$ for $\mathsf{code}(L_k)$ contains a tree of states which are labeled with bit strings from $\{0,1\}^{\leqslant \lceil \log k \rceil}$. Every state s from $\mathcal{A}_k$ is represented by state $\mathsf{code}(s)$ in $\mathcal{A}'_k$, which is a leaf in the tree. Attached to it is a gadget of size $O(\log k)$ which processes inputs of the

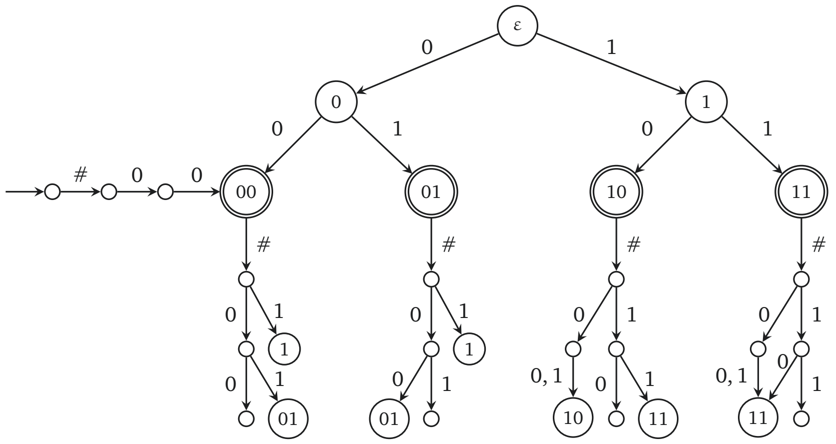

Figure 4.6: The automaton $\mathcal{A}'_3$ for $\mathsf{code}(L_3)$. Omitted transitions lead to a sink state. States with the same name are identified.

form $\#\,\mathsf{code}(t)$ for $t \in \Sigma$. If $s = t$ then the automaton leads to a sink state. If the automaton detects that $s < t$ (in Figure 4.6 it branches off to the right) then it moves to the corresponding state in the tree. If the automaton detects that $t < s$ (it branches off to the left) then it reads the remaining bits in the code word and returns to state $\mathsf{code}(s)$. □

Theorem 4.45. *There exists a number* $d > 0$ *such that for each* $k \geqslant 1$ *there exists a language* $L'_k \subseteq \{0, 1, \#\}^*$ *recognized by a DFA* $\mathcal{A}'_k$ *with* $\Theta(k \log k)$ *states such that* $V_{L'_k}(n) = O(\log n)$ *and* $F_{L'_k}(n) \geqslant d \cdot 2^{|\mathcal{A}'_k|/\log|\mathcal{A}'_k|} \cdot \log n$ *for infinitely many* $n \in \mathbb{N}$.

Proof. Let $\mathcal{A}'_k$ be the automaton for $L'_k = \mathsf{code}(L_k)$ of size $O(k \log k)$ from Lemma 4.44 By Lemma 4.43 we know that

- $F_{L'_k}(cn) \geqslant F_{L_k}(n)$, where $c = \lceil \log k \rceil + 2$, and
- $V_{L'_k}(n) = O(\log n)$ because $V_{L_k}(n) = O(\log n)$.

For $n \in \mathbb{N}$ large enough Theorem 4.42 implies

$$F_{L'_k}(cn) \geqslant F_{L_k}(n) \geqslant 2^{k-2} \cdot \log n$$

and hence

$$F_{L'_k}(n) \geqslant 2^{k-2} \cdot \log \sqrt{n} \geqslant 2^{k-3} \cdot \log n$$

for infinitely many n with $\sqrt{n} \geqslant c$. Since $|\mathcal{A}'_k| = O(k \log k)$ we have $k = \Omega(|\mathcal{A}'_k|/\log|\mathcal{A}'_k|)$, and therefore $2^{k-3} \geqslant d \cdot 2^{|\mathcal{A}'_k|/\log|\mathcal{A}'_k|}$ for some $d > 0$ and k large enough. This implies the desired bound. □

4.4 Deciding the space complexity

In this section we consider the computational complexity of the following decision problems.

- DFA(1): Given a DFA $\mathcal{A}$ for L, does $F_L(n) = O(1)$ hold?
- NFA(1): Given an NFA $\mathcal{A}$ for L, does $F_L(n) = O(1)$ hold?
- DFA$(\log n)$: Given a DFA $\mathcal{A}$ for L, does $F_L(n) = O(\log n)$ hold?
- NFA$(\log n)$: Given an NFA $\mathcal{A}$ for L, does $F_L(n) = O(\log n)$ hold?

In the following we will show that both DFA-problems are NL-complete whereas the NFA-problems are PSPACE-complete (under logspace reductions). The *DFA (NFA) universality problem* asks whether a given DFA (NFA) over Σ accepts Σ^*. It is NL-complete for DFAs and is PSPACE-complete for NFAs [88].

Of course we can also replace $F_L(n)$ by $V_L(n)$ in the two latter problems DFA$(\log n)$ and NFA$(\log n)$ by Corollary 4.9.

Proposition 4.46. *The following question is* NL*-complete (*PSPACE*-complete): Given a DFA (NFA) $\mathcal{A}$ for* L*, does $V_L(n) = O(1)$ hold?*

Proof. By Corollary 4.11 one has to check whether $L(\mathcal{A}) = \emptyset$ or $L(\mathcal{A}) = \Sigma^*$. The upper bounds hold since emptiness of DFAs and NFAs can be tested in NL (by guessing an accepted word) and testing universality for DFAs (NFAs) is in NL (PSPACE). For the lower bounds, we can reduce the universality problem to the question $L(\mathcal{A}) \in \{\emptyset, \Sigma^*\}$. Given a DFA (NFA) $\mathcal{A}$ we can test in logspace whether the empty word is accepted by $\mathcal{A}$ by testing whether there exists an initial state which is final. If $\varepsilon \notin L(\mathcal{A})$ then $L(\mathcal{A}) \neq \Sigma^*$ and we can return a negative instance (say, an automaton for the nontrivial language $\{a\}$). Otherwise, we know that $L(\mathcal{A}) \neq \emptyset$ and we can return $\mathcal{A}$ itself. □

4.4.1 The DFA case

We start with the NL-hardness for the DFA case:

Theorem 4.47. DFA(1) *and* DFA$(\log n)$ *are* NL*-hard.*

Proof. We reduce from the NL-complete reachability problem in finite directed graphs. Given a finite directed graph $G = (V, E)$ and two vertices $s, t \in V$, the question is whether there exists a path from s to t. We can assume that $s \neq t$ and that each vertex $v \in V$ has exactly two successors $v_a, v_b \in V$. Let $\mathcal{A} = (V \cup \{\bot\}, \{a, b, c\}, s, \delta, \{t\})$ be a DFA where

$$\delta(v, x) = \begin{cases} v_x & \text{if } v \in V \setminus \{t\},\ x \in \{a, b\}, \\ t & \text{if } v = t,\ x \in \{a, b, c\}, \\ \bot & \text{otherwise,} \end{cases}$$

and let $L = L(\mathcal{A})$. Since $s \neq t$, we can write L as $K\{a, b, c\}^*$ for some $K \subseteq \{a, b\}^+$. Furthermore, there exists a path from s to t in G if and only if $K \neq \emptyset$. If $K = \emptyset$, then $L = \emptyset$ and $F_L(n) = O(1)$. If $K \neq \emptyset$, then we claim that $F_L(n)$ is not in $O(\log n)$. Take any word $x \in K$ of length $|x| = k$. Then $x\{x, c^k\}^* \subseteq L$ and $c^k\{x, c^k\}^* \cap L = \emptyset$. By Proposition 4.8 we know $F_L(n) = \Omega(n)$ infinitely often. □

Theorem 4.48. DFA(1) *is* NL*-complete.*

Proof. Let us first assume that the input DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ for $L = L(\mathcal{A})$ is minimal. Later, we will argue how to handle the general case. Since nondeterministic logspace is closed under complement, it suffices to decide whether $F_L(n)$ is not in $O(1)$. By Proposition 4.23 this is the case if and only if there exist words $x, y, z \in \Sigma^*$ such that $|x| = |y|$, $|z| = |Q|$ and $\mathcal{A}(xz) \neq \mathcal{A}(yz)$. The existence of such words can be easily verified in nondeterministic logspace: One simulates $\mathcal{A}$ on two words of the same length (the words x, y), and thereby only stores the current state pair. At every time instant, the algorithm can nondeterministically decide to continue the simulation from the current state pair (p, q) with a single word (the word z) for $|Q|$ steps. The algorithm accepts if at the end the two states are distinct.

The general case, where $\mathcal{A}$ is not minimal is handled as follows: Assume that $\mathcal{A} = (\{1, \dots, k\}, \Sigma, 1, \delta, F)$ is the input DFA. It is known that DFA equivalence is in NL [28]. Hence, one can test in nondeterministic logspace, whether two states $p, q \in Q$ are equivalent (in the sense that $\delta(p, w) \in F$ iff $\delta(q, w) \in F$ for all $w \in \Sigma^*$). We will use this problem as an NL-oracle in the above NL-algorithm for minimal DFAs. More precisely, let $\mathcal{A}' = (Q, \Sigma, 1, \delta', F')$ be the minimal DFA for $\mathcal{A}$, where we assume that Q is the set of all states $q \in \{1, \dots, k\}$ such that there is no state $p < q$ that is equivalent to q. We run the NL-algorithm above for minimal DFAs on $\mathcal{A}'$ without explicitly constructing $\mathcal{A}'$. If we have to compute a successor state $\delta'(q, a)$ (where $q \in Q$) we compute, using the above NL-oracle the smallest state that is equivalent to $\delta(q, a)$.

The above argument shows that DFA(1) belongs to $\mathrm{NL}^{\mathrm{NL}}$. Finally, we use the well-known identity $\mathrm{NL} = \mathrm{NL}^{\mathrm{NL}}$ [67]. □

In the rest of the section, we show that one can also decide in nondeterministic logspace whether $V_L(n) = O(\log n)$ (or equivalently $F_L(n) = O(\log n)$). As in the proof of Theorem 4.48 we can assume that L is given by its minimal DFA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$. An equivalence relation $\sim$ on a set X *separates* two subsets $X_1, X_2 \subseteq X$ if $x_1 \not\sim x_2$ for all $x_1 \in X_1$ and $x_2 \in X_2$. Recall that the Myhill-Nerode right congruence $\sim_L$ satisfies $\mathcal{A}(x) = \mathcal{A}(y)$ if and only if $x \sim_L y$. Therefore, two sets of words $X_1, X_2 \subseteq \Sigma^*$ are separated by $\sim_L$ if and only if the sets of reached states $\{\mathcal{A}(x_i) \mid x_i \in X_i\}$ are disjoint for $i = 1, 2$.

Given a right congruence $\sim$ over Σ^*. A tuple $(u_2, v_2, u, v) \in (\Sigma^*)^4$ of words is *critical* in $\sim$ if $|u_2| = |v_2|$, u_2 is a suffix of u, v_2 is a suffix of v, and $\sim$ separates $u_2\{u, v\}^*$ and $v_2\{u, v\}^*$.

Lemma 4.49. *We have* $V_L(n) = \Theta(n)$ *if and only* $\sim_L$ *has a critical tuple.*

Proof. If $V_L(n) = \Theta(n)$, then any rDFA $\mathcal{B}$ for L is not well-behaved by Theorem 4.12 and Proposition 4.5. By Lemma 4.7 there are words $u_1, u_2, v_1, v_2, z \in \Sigma^*$ such that $|u_2| = |v_2|$, $u = u_1u_2$, $v = v_1v_2$, and L separates $u_2\{u,v\}^*z$ and $v_2\{u,v\}^*z$. For all $x, y \in \{u,v\}^*$ the words u_2xz and v_2yz are separated by L, and hence $u_2x \not\sim_L v_2y$.

Conversely, assume that there are words $u_1, u_2, v_1, v_2 \in \Sigma^*$ where $|u_2| = |v_2|$, $u = u_1u_2$ and $v = v_1v_2$, such that $\sim_L$ separates $u_2\{u,v\}^*$ and $v_2\{u,v\}^*$. We clearly have $u_2 \neq v_2$ and hence $u \neq v$. Further, we can choose numbers $k, \ell \geqslant 1$ such that u^k and v^ℓ have the same length. We redefine u_1 to be $u^{k-1}u_1$ and v_1 to be $v^{\ell-1}v_1$. This ensures $|u| = |v|$ and $|u_1| = |v_1|$. Moreover, the new resulting sets $u_2\{u,v\}^*$ and $v_2\{u,v\}^*$ are contained in the original sets and therefore also separated by $\sim_L$.

Now consider a variable-size SW-algorithm $\mathcal{P}$ for L and let $m \in \mathbb{N}$. We claim that $\mathcal{P}$ must distinguish any distinct streams $x, y \in \{u,v\}^m$, i.e. $\mathcal{P}(x) \neq \mathcal{P}(y)$. This is because after $|u_1| = |v_2|$ many pop-operations, the windows are of the form u_2w and v_2w for some $w \in \{u,v\}^*$. By assumption $u_2w \not\sim_L v_2w$, and hence there exists a word $z \in \Sigma^*$ such that L separates u_2wz and v_2wz. Hence we can separate the states in $\mathcal{P}$ by reading z, which implies $\mathcal{P}(x) \neq \mathcal{P}(y)$. This implies that $V_L(n) = \Omega(n)$ infinitely often and in fact $V_L(n) = \Theta(n)$ by Theorem 4.12. □

In the following we show how to decide whether a finite index right congruence (presented by a DFA) has a critical tuple. This requires basic arguments in the language of finite semigroups. A semigroup S is *right zero* if $xy = y$ for all $x, y \in S$. An element $e \in S$ is *idempotent* if $e^2 = e$. It is known that for every finite semigroup S there exists a number $\omega \in \mathbb{N} \setminus \{0\}$ such that s^ω is idempotent for all $s \in S$.

Lemma 4.50. *Let* $h \colon \Sigma^* \to M$ *be a homomorphism into a finite monoid* M. *For any* $u, v \in \Sigma^*$ *there exist* $u' \in \{u,v\}^*u$ *and* $v' \in \{u,v\}^*v$ *such that* $\{h(u'), h(v')\}$ *forms a right zero subsemigroup.*

Proof. Let $u' = (u^\omega v^\omega)^\omega u^\omega$ and $v' = (u^\omega v^\omega)^\omega$. Setting $e' = h(u')$ and $f' = h(v')$ we have the equations $e'e' = f'e' = e'$ and $f'f' = e'f' = f'$. □

Lemma 4.51. *Assume that* $\sim$ *is a finite index right congruence with a critical tuple.*

(i) For any homomorphism $h \colon \Sigma^* \to M$ *into a finite monoid* M *there exists a critical tuple* (u_2, v_2, u, v) *in* $\sim$ *such that* $|h(u_2\{u,v\}^*)| = |h(v_2\{u,v\}^*)| = 1$.

(ii) There exists a critical tuple (u_2, v_2, u, v) *in* $\sim$ *such that* $u_2\{u,v\}^*$ *and* $v_2\{u,v\}^*$ *are contained in single* $\sim$*-classes.*

Proof. Assume that (u_2, v_2, u, v) is a critical tuple. By Lemma 4.50 there exist $u' \in \{u,v\}^*u$ and $v' \in \{u,v\}^*v$ such that $\{h(u), h(v)\}$ forms a right zero subsemigroup. Notice that u_2 is a suffix of u' and v_2 is a suffix of v'. Furthermore, since $\{u',v'\}^* \subseteq \{u,v\}^*$ the subsets $u_2\{u',v'\}^*$ and $v_2\{u',v'\}^*$ are still separated by $\sim$, therefore (u_2, v_2, u', v') is a critical tuple.

Now replace u by u' and v by v'. Observe that (u_2u, v_2u, uu, vu) is again a critical tuple because u_2u is a suffix of uu, v_2u is a suffix of vu, $|u_2u| = |v_2u|$, and

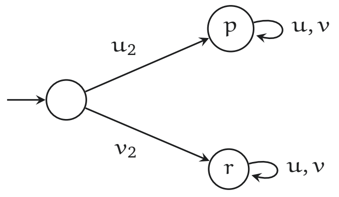

Figure 4.7: The minimal DFA contains this pattern if (u_2, v_2, u, v) is a critical tuple where $u_2\{u, v\}^*$ and $v_2\{u, v\}^*$ are contained in single $\sim_L$-classes.

because $u_2u\{uu, vu\}^* \subseteq u_2\{u, v\}^*$ and $v_2u\{uu, vu\}^* \subseteq v_2\{u, v\}^*$. Furthermore we have $h(u_2u\{uu, vu\}^*) = \{h(u_2u)\}$ and $h(v_2u\{uu, vu\}^*) = \{h(v_2u)\}$. This proves statement (i).

Statement (ii) is a simple corollary of (i). Define $x \equiv y$ if and only if $\ell x \sim \ell y$ for all $\ell \in \Sigma^*$, which is a congruence on Σ^* and refines $\sim$. Every class $[x]_\equiv$ can be associated with the function $\Sigma^*/\sim \to \Sigma^*/\sim$ given by $[\ell]_\sim \mapsto [\ell x]_\sim$. Since $\sim$ has finite index the number of such functions is also finite and hence $\equiv$ has finite index. Hence we can take the canonical homomorphism $h\colon \Sigma^* \to \Sigma^*/\equiv$ and apply (i). □

We are ready to present the decision procedure.

Lemma 4.52. *Given a minimal DFA $\mathcal{A}$, one can test in nondeterministic logspace whether $\mathcal{A}$ has a critical tuple.*

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be a minimal DFA. Figure 4.7 illustrates the structure we need to detect in $\mathcal{A}$. To do so, we represent all critical tuples by a regular language. Consider the language of all words $u_1\#v_1\#(u_2 \otimes v_2)$ such that there are states $p \neq r \in Q$ and

- $q_0 \xrightarrow{u_2} p$,
- $q_0 \xrightarrow{v_2} r$,
- $p \xrightarrow{u_1u_2} p$ and $r \xrightarrow{u_1u_2} r$,
- $p \xrightarrow{v_1v_2} p$ and $r \xrightarrow{v_1v_2} r$,
- and $|u_2| = |v_2| \geqslant 1$.

One can construct in logspace an NFA for the language above, which can be tested for emptiness in NL. The NFA initially guesses the two distinct states p and r, and verifies the six runs. The length constraint can be verified due to the encoding as a convolution $u_2 \otimes v_2$. The language above is empty if and only if $\mathcal{A}$ has a critical tuple. □

Lemma 4.49, Lemma 4.52 and Theorem 4.16 together imply:

Corollary 4.53. $\text{DFA}(\log n)$ *is* NL*-complete.*

4.4.2 The NFA case

In this section, we show that the problems $\textsc{Nfa}(1)$ and $\textsc{Nfa}(\log n)$ are both $\textsc{Pspace}$-complete. The upper bounds follow easily from Theorem 4.48 and Corollary 4.53 and the following fact (see [85, Lemma 1]): If a mapping f can be computed by a Turing-machine with a polynomially bounded work tape (the output can be of exponential size) and L is a language that can be decided in polylogarithmic space, then $f^{-1}(L)$ belongs to $\textsc{Pspace}$. Note that from a given NFA $\mathcal{A}$ one can compute an equivalent DFA using polynomially bounded work space: One iterates over all subsets of the state set of $\mathcal{A}$; the current subset is stored on the work tape. For every subset and input symbol one then writes the corresponding transition of the DFA on the output tape.

Theorem 4.54. $\textsc{Nfa}(1)$ *is* $\textsc{Pspace}$*-complete.*

Proof. By the above remark it suffices to establish $\textsc{Pspace}$-hardness of $\textsc{Nfa}(1)$. For this we will reduce the NFA universality problem to $\textsc{Nfa}(1)$. W.l.o.g. consider the alphabet $\Sigma = \{a, b\}$. For an NFA $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ we define $\rho(\mathcal{A})$ to be the automaton that results from $\mathcal{A}$ by adding a new initial state $\bar{q}$ with an a-labeled self-loop and a b-labeled transition from every state of F to $\bar{q}$. The only final state of $\rho(\mathcal{A})$ is $\bar{q}$. More formally, we define $\rho(\mathcal{A})$ as follows:

$$\rho(\mathcal{A}) = (Q \cup \{\bar{q}\}, \Sigma, I \cup \{\bar{q}\}, \Delta \cup \{(q, b, \bar{q}) \mid q \in F\} \cup \{(\bar{q}, a, \bar{q})\}, \{\bar{q}\}).$$

Notice that the ρ-construction implies $L(\rho(\mathcal{A})) = a^* \cup L(\mathcal{A})\, b\, a^*$. It is then easy to verify that $L(\mathcal{A}) = \Sigma^*$ iff $L(\rho(\mathcal{A})) = \Sigma^*$. If $L(\mathcal{A}) = \Sigma^*$ then clearly $F_{L(\rho(\mathcal{A}))}(n) = O(1)$. Conversely, assume that $F_{L(\rho(\mathcal{A}))}(n) = O(1)$. By Theorem 4.19 there exists a number $k \in \mathbb{N}$ such that $a^* \cup L(\mathcal{A})\, b\, a^*$ is a Boolean combination of k-suffix testable languages and regular length languages. Let $x \in \{a, b\}^n$ be any word of length n. Since $a^{n+1+k} \in L(\rho(\mathcal{A}))$ and xba^k share the same k-suffix and are of the same length, we also know that $xba^k \in L(\rho(\mathcal{A}))$ and hence $x \in L(\mathcal{A})$. This proves that $\mathcal{A}$ is universal.

We have thus established that the log-space construction $\mathcal{A} \mapsto \rho(\mathcal{A})$ reduces the universality problem for NFAs to $\textsc{Nfa}(1)$. $\square$

Theorem 4.55. $\textsc{Nfa}(\log n)$ *is* $\textsc{Pspace}$*-complete.*

Proof. It remains to show that $\textsc{Nfa}(\log n)$ is $\textsc{Pspace}$-hard, which can be shown by reducing the NFA universality problem to $\textsc{Nfa}(\log n)$. W.l.o.g. the alphabet of the input automaton is $\Sigma = \{a, b\}$, and we also consider the extended alphabet $\Gamma = \{a, b, c\}$. We will view the automaton as a finite right automaton $\mathcal{B} = (Q, \Sigma, F, \Delta, I)$ with $\Delta \subseteq Q \times \Sigma \times Q$. We define $\mathcal{B}'$ to be the automaton that results from $\mathcal{B}$ by adding a new initial and final state $\bar{q}$ with a- and b-labeled self-loops, a c-labeled transition from every state of I to $\bar{q}$, and a c-labeled transition from $\bar{q}$ to every state of $\mathcal{B}$. The only initial state of $\mathcal{B}'$ is $\bar{q}$. More formally, we define

$$\begin{aligned}
\mathcal{B}' &= (Q \cup \{\bar{q}\}, \Gamma, F \cup \{\bar{q}\}, \Delta', \{\bar{q}\}), \text{ where} \\
\Delta' &= \Delta \cup \{(q, c, \bar{q}) \mid q \in I\} \cup \{(\bar{q}, c, q) \mid q \in Q\} \cup \{(\bar{q}, x, \bar{q}) \mid x \in \{a, b\}\}.
\end{aligned}$$

We claim that $L(\mathcal{B}) = \Sigma^*$ if and only if $F_{L(\mathcal{B}')}(n) = O(\log n)$. To prove this we consider the powerset automaton $2^{\mathcal{B}'} = (2^{Q \cup \{\bar{q}\}}, \Gamma, \mathcal{F}, \delta, \{\bar{q}\})$ with transitions

$$\{q \mid p \in P, q \overset{x}{\leftarrow}_{\mathcal{B}} p\} \overset{x}{\leftarrow} P \text{ for all } P \subseteq Q \cup \{\bar{q}\}, x \in \Gamma$$

and

$$\mathcal{F} = \{P \subseteq Q \cup \{\bar{q}\} \mid P \cap (F \cup \{\bar{q}\}) \neq \emptyset\}.$$

All transitions which are reachable from the initial state $\{\bar{q}\}$ are of the form

- $\{\bar{q}\} \overset{x}{\leftarrow} \{\bar{q}\}$ for $x \in \{a, b\}$,
- $I \overset{c}{\leftarrow} \{\bar{q}\}$,
- $P' \overset{x}{\leftarrow} P$ for $x \in \{a, b\}$ and $P, P' \subseteq Q$,
- $\{\bar{q}\} \overset{c}{\leftarrow} P$ for $P \subseteq Q$.

In particular, the reachable part is strongly connected. Because of the self-loops on the initial final state $\{\bar{q}\}$ the rDFA $2^{\mathcal{B}'}$ is well-behaved if and only if all reachable states are final, i.e. $L(2^{\mathcal{B}'}) = L(\mathcal{B}') = \Gamma^*$. It suffices to prove that $L(\mathcal{B}) = \Sigma^*$ if and only if $L(\mathcal{B}') = \Gamma^*$.

First assume that $L(\mathcal{B}) = \Sigma^*$. Take any word $w \in \Gamma^*$, which can be written as $w = w_k c w_{k-1} c \cdots c w_1 c w_0$ where $w_0, \dots, w_k \in \Sigma^*$. Then there exist initial accepting runs $q_i \overset{w_i}{\longleftarrow} p_i$ in $\mathcal{B}$ for $0 \leqslant i \leqslant k$. This allows to construct a run on w in $\mathcal{B}'$, namely

$$\cdots q_2 \overset{w_2}{\longleftarrow} p_2 \overset{c}{\leftarrow} \bar{q} \overset{w_1}{\longleftarrow} \bar{q} \overset{c}{\leftarrow} q_0 \overset{w_0}{\longleftarrow} p_0.$$

It ends in $\bar{q}$ if k is odd, and in q_k if k is even. In any case it is an initial accepting run, which proves that $L(\mathcal{B}') = \Gamma^*$.

Now assume that $L(\mathcal{B}) \neq \Sigma^*$, i.e. there exists a word $x \in \Sigma^* \setminus L(\mathcal{B})$. Then one can easily verify that xc is not accepted by $\mathcal{B}$, hence $L(\mathcal{B}') \neq \Gamma^*$.

So, we have established that the log-space construction $\mathcal{B} \mapsto \mathcal{B}'$ reduces the universality problem for rNFAs to NFA$(\log n)$. □

4.5 Conclusion

In this chapter we have identified $\langle \mathbf{LI}, \mathbf{Len} \rangle$ as the class of regular languages with logarithmic space complexity in both sliding window models. Notice that regular left ideals over a sliding window express statements of the form "recently in the stream some regular event happened". Dually, complements of left ideals over a sliding window express statements of the form "at all recent times in the stream some regular event happened".

Open problems

1. It would be desirable to have a logic which precisely captures $\langle \mathbf{LI}, \mathbf{Len} \rangle$ (or some fragment of it) and admits sliding window algorithms with decent space complexity in the formula size.

2. A regular left ideal $L = \Sigma^* K$ can also be specified by a finite automaton $\mathcal{A}$ for K. An interesting question would be whether one can always find a $|\mathcal{A}|^{O(1)} \log n$ space SW-algorithm for L.

3. The lower bound in Theorem 4.45 for the dependence on the DFA size over bounded alphabets is not tight. For NFAs (over unbounded alphabets) the gap between the upper bound $4^{|\mathcal{A}|} \log n$ and the lower bound $2^{|\mathcal{A}|} \log n$ also remains to be closed. In this context one could also look at unambiguous finite automata.

4. Likewise, it is open whether the bounds in Theorem 4.40 are optimal.

Left Cayley graphs In the paper [G1] we originally proved the space trichotomy using the *left Cayley graph* of the syntactic monoid. Suppose $L \subseteq \Sigma^*$ is a regular language. Then the left Cayley graph of the syntactic monoid $M = \Sigma^*/\equiv_L$ of L can be viewed as a particular rDFA: Its state set is M, its initial state is $[\varepsilon]_{\equiv_L}$, its set of final states is $L/\equiv_L$, and its transitions have the form $[aw]_{\equiv_L} \xleftarrow{a} [w]_{\equiv_L}$ for $w \in \Sigma^*$ and $a \in \Sigma$. If the left Cayley graph is well-behaved then one can construct a path summary algorithm for L with $O(\log n)$ space.

The advantage is that it suffices to store the path summary of the initial run on the active window (instead of all runs on the active window starting from any state). The disadvantage is that the hidden O-constant is proportional to the *height* of the reachability order $\preceq$ of the left Cayley graph, i.e. the maximum length of a chain $C_1 \prec C_2 \prec \cdots \prec C_h$. This parameter is also known as the $\mathcal{L}$-height of M and can be a priori as large as $|M|$, which in turn can have $|\mathcal{A}|^{|\mathcal{A}|}$ elements where $\mathcal{A}$ is the minimal DFA for L. Fleischer and Kufleitner presented a sequence of minimal DFAs $\mathcal{A}_m$ over a fixed alphabet such that $\mathcal{A}_m$ has m states and its transition monoid has $\mathcal{L}$-height $\Omega(2^m/m^{9.5})$ [51].

Chapter 5

Rational functions

In this chapter we study the space complexity of rational functions in the sliding window model. We will first prove a closely related dichotomy result, namely that the suffix expansion of any rational functions has a polynomially or an exponentially growing image (Theorem 5.2). This dichotomy result has two applications: Firstly, we can extend the space trichotomy from regular languages to rational functions. For the variable-size model we prove that every rational function has either constant, logarithmic or linear space complexity (Corollary 5.23). For the fixed-size model we show that the space complexity is at most logarithmic or linear (Corollary 5.24). Secondly, in Chapter 10 we will apply this dichotomy theorem to prove a space trichotomy for the class of *visibly pushdown languages*. Parts of this chapter have appeared in [G6].

5.1 Suffix expansions

Let $t\colon \Sigma^* \to \Omega^*$ be a partial function with suffix-closed domain. Recall that its suffix expansion $\overleftarrow{t}\colon \mathrm{dom}(t) \to (\Omega^*)^*$ is defined by

$$\overleftarrow{t}(a_1 \cdots a_n) = t(a_1 \cdots a_n)\, t(a_2 \cdots a_n) \;\cdots\; t(a_{n-1}a_n)\, t(a_n).$$

We emphasize that the range of $\overleftarrow{t}$ is not Ω^* but the free monoid over Ω^*, consisting of all finite sequences of words over Ω. In this chapter we analyze the growth of $\mathrm{im}(\overleftarrow{t})$.

Example 5.1. Consider the transduction $t\colon \{a,b\}^* \to a^*$ defined by

$$t = \{(a^n, a^n) \mid n \in \mathbb{N}\} \cup \{(a^n b w, a^n) \mid n \in \mathbb{N},\, w \in \{a,b\}^*\},$$

which projects a word over $\{a,b\}$ to its leftmost (maximal) a-block and is rational. We can identify $\mathrm{im}(\overleftarrow{t}) \subseteq (a^*)^*$ with the set of all sequences of natural numbers which are concatenations of monotonically decreasing sequences of the form $(k, k-1, \ldots, 0)$. There are exactly 2^n of such sequences of length n and hence $\mathrm{im}(\overleftarrow{t})$ has exponential growth.

Fooling schemes A witness for exponential growth of $\mathrm{im}(\overleftarrow{t})$ can be given by a fooling scheme. A partial function $t\colon \Sigma^* \to Y$ *separates* subsets $X_1, X_2 \subseteq \mathrm{dom}(t)$ if $t(X_1) \cap t(X_2) = \emptyset$. A *fooling scheme* of a partial function $t\colon \Sigma^* \to Y$ in $X \subseteq \mathrm{dom}(t)$ is a tuple $(u_2, v_2, u, v, z) \in (\Sigma^*)^5$ such that

- $\{u_2, v_2\}\{u, v\}^* z \subseteq X$,
- $|u_2| = |v_2|$,
- u_2 is a suffix of u, v_2 is a suffix of v,
- and t separates $u_2\{u, v\}^* z$ and $v_2\{u, v\}^* z$.

We omit X if $X = \mathrm{dom}(t)$. The function t from Example 5.1 has the fooling scheme $(a, b, a, b, \varepsilon)$ since $t(a\{a, b\}^*) = a^+$ and $t(b\{a, b\}^*) = \{\varepsilon\}$ are disjoint. Our main theorem for rational functions in this chapter is the following.

Theorem 5.2 (Dichotomy for rational functions). *Let $t\colon \Sigma^* \to \Omega^*$ be a rational function with suffix-closed domain.*

1. *If $\mathrm{im}(t)$ is a bounded language and t has no fooling scheme then $\mathrm{im}(\overleftarrow{t})$ has polynomial growth.*
2. *In all other cases $\mathrm{im}(\overleftarrow{t})$ has exponential growth.*

Here we will make use of a representation of rational functions due to Reutenauer and Schützenberger, as the composition of a left-subsequential and a right-subsequential function [100]. In fact, the left-subsequential function is represented by a certain syntactic right congruence $\mathcal{R}_t$; the input $a_1 \cdots a_n$ is read deterministically from left to right, and each letter a_i is annotated with the congruence class $[a_1 \cdots a_{i-1}]_{\mathcal{R}_t}$. In total, there are three "barriers" for polynomial growth of $\mathrm{im}(\overleftarrow{t})$.

(i) The image of t has exponential growth.

(ii) The right congruence $\mathcal{R}_t$ has a critical tuple.

(iii) A right transducer for t is not *well-behaved*.

Finally we will prove that the absence of the three properties above ensures polynomial growth.

Lower bounds The second statement in Theorem 5.2 follows from the following two statements.

Proposition 5.3. *If a partial function $t\colon \Sigma^* \to \Omega^*$ with suffix-closed domain has a fooling scheme in $X \subseteq \mathrm{dom}(t)$ then $\overleftarrow{t}(X)$ has exponential growth.*

Proof. Let (u_2, v_2, u, v, z) be a fooling scheme of t in X. Let $n \in \mathbb{N}$ and let $w \neq w' \in u_2\{u, v\}^n z \subseteq X$. There exists a word $w'' \in \{u, v\}^*$ such that w and w' have the suffixes $u_2 w'' z$ and $v_2 w'' z$, respectively. The fooling scheme property states that $t(u_2 w'' z) \neq t(v_2 w'' z)$ and, since $u_2 w'' z$ and $v_2 w'' z$ have equal length, we obtain $\overleftarrow{t}(w) \neq \overleftarrow{t}(w')$. Therefore $\overleftarrow{t}(X)$ contains at least 2^n words of length at most $|u_2| + cn + |z| = O(n)$ where $c = \max\{|u|, |v|\}$. $\square$

Proposition 5.4. *Let* $t \colon \Sigma^* \to \Omega^*$ *be a rational function with suffix-closed domain. If* $\mathrm{im}(t)$ *has exponential growth then the following statements hold:*

(i) $|t(\Sigma^{\leqslant n})| = 2^{\Omega(n)}$.

(ii) $|t(\Sigma^n)|$ *grows exponentially.*

(iii) $\mathrm{im}(\overleftarrow{t})$ *has exponential growth.*

Proof. We claim that every word $y \in \mathrm{im}(t)$ has a preimage under t of length $O(|y|)$. One way to show this is the rational uniformization theorem. Let $t' \colon \Omega^* \to \Sigma^*$ be a rational function which *uniformizes* the inverse relation $t^{-1} = \{(t(x), x) \mid x \in \mathrm{dom}(t)\}$, i.e. $\mathrm{dom}(t') = \mathrm{dom}(t^{-1}) = \mathrm{im}(t)$ and $t(t'(y)) = y$ for all $y \in \mathrm{dom}(t')$. Such a rational uniformization exists by [40, Chapter XI, Proposition 8.2]. Since t' is rational its output length is linearly bounded in the input length. This proves our claim.

We have shown that $\mathrm{im}(t) \cap \Omega^{\leqslant n} \subseteq t(\Sigma^{\leqslant cn})$ for some constant $c > 0$ and sufficiently large n. If $\mathrm{im}(t)$ has exponential growth then its cumulative growth is $2^{\Omega(n)}$ by Theorem 2.5 and hence $|t(\Sigma^{\leqslant n})| = 2^{\Omega(n)}$. Since $|t(\Sigma^n)| \leqslant |t(\Sigma^{\leqslant n})| \leqslant \sum_{i=0}^{n} |t(\Sigma^i)|$ also $|t(\Sigma^n)|$ must grow exponentially by Lemma 2.3.

Since $\overleftarrow{t}(x)$ is a tuple of length $|x|$ whose first entry is $t(x)$, also $\mathrm{im}(\overleftarrow{t})$ has exponential growth. □

5.2 Finite index right congruences

Let $\sim$ be a finite index right congruence on Σ^* and $\approx$ its suffix expansion. We show that $|\Sigma^{\leqslant n}/{\approx}|$ grows polynomially if $\sim$ has no critical tuple, and otherwise it grows exponentially. This can be viewed as a special case of Theorem 5.2 since $\nu_\sim \colon \Sigma^* \to \Sigma^*/{\sim}$ is rational. Implicitly we have already treated the case where $\sim$ is the Myhill-Nerode right congruence $\sim_L$ of a regular language L.

Lemma 5.5. *If* $L \subseteq \Sigma^*$ *is regular, then* $|\Sigma^{\leqslant n}/{\approx}|$ *grows polynomially if and only if* $\sim_L$ *has no critical tuple.*

Proof. If L is empty or universal then $|\Sigma^{\leqslant n}/{\approx}|$ grows linearly and $\sim_L$ is the universal congruence, which has no critical tuple. Otherwise $\log|\Sigma^{\leqslant n}/{\approx_L}|$ is exactly the space complexity $V_L(n)$ by Proposition 3.11. By the trichotomy theorem (Theorem 4.12) the growth of $|\Sigma^{\leqslant n}/{\approx_L}|$ is polynomial or exponential. By Lemma 4.49 the growth is exponential if and only if $\sim_L$ has a critical tuple. □

Proposition 5.6. *If* $\sim$ *has a critical tuple then* $|\Sigma^{\leqslant n}/{\approx}|$ *grows exponentially.*

Proof. Let $n \in \mathbb{N}$ and let $w \neq w' \in \{u, v\}^n$. There exists a word $w'' \in \{u, v\}^*$ such that w and w' have the suffixes $u_2 w''$ and $v_2 w''$ of equal length. By the definition of critical tuples we have $u_2 w'' \not\sim v_2 w''$, which implies $w \not\approx w'$. Therefore $|\Sigma^{\leqslant cn}/{\approx}| \geqslant 2^n$ where $c = \max\{|u|, |v|\}$. □

Lemma 5.7. *The class of right congruences on* Σ^* *which have no critical tuple is closed under coarsening and intersection.*

Proof. Closure under coarsening is clear because the property "$\sim$ has no critical tuple" is *positive* in $\sim$:

$$\forall u = u_1 u_2 \, \forall v = v_1 v_2 (|u_2| = |v_2| \to \exists w \in \{u, v\}^* : u_2 w \sim v_2 w).$$

Consider two right congruences $\sim$, $\sim'$ which have no critical tuples. One can verify that their intersection $\sim \cap \sim'$ is again a right congruence. Towards a contradiction assume that $\sim \cap \sim'$ has a critical tuple (u_2, v_2, u, v). Let us first ensure that $u_2\{u, v\}^*$ and $v_2\{u, v\}^*$ are contained in single $\sim$- and $\sim'$-classes. Let $\equiv$ and $\equiv'$ be any finite index congruences which refine $\sim$ and $\sim'$, respectively. For example, we can define $x \equiv y$ if and only if $\ell x \sim \ell y$ for all $\ell \in \Sigma^*$, and similarly for $\equiv'$, as in the proof of Lemma 4.51. Define the homomorphism

$$\begin{aligned} h \colon \Sigma^* &\to (\Sigma^*/{\equiv}) \times (\Sigma^*/{\equiv'}) \\ x &\mapsto ([x]_\equiv, [x]_{\equiv'}). \end{aligned}$$

By Lemma 4.51 $\sim \cap \sim'$ has a critical tuple (u_2, v_2, u, v) such that $h(u_2\{u, v\}^*)$ and $h(v_2\{u, v\}^*)$ have size one, and hence $u_2\{u, v\}^*$ and $v_2\{u, v\}^*$ are contained in single $\sim$- and $\sim'$-classes. By assumption $\sim$ and $\sim'$ have no critical tuple, and therefore all words in $u_2\{u, v\}^*$ must be $\sim$- and $\sim'$-congruent to all words in $v_2\{u, v\}^*$. This would imply that all words in $u_2\{u, v\}^*$ must be $(\sim \cap \sim')$-congruent to all words in $v_2\{u, v\}^*$, contradicting that (u_2, v_2, u, v) is a critical tuple in $\sim \cap \sim'$. $\square$

Theorem 5.8. *$|\Sigma^{\leqslant n}/{\approx}|$ is polynomially bounded if and only if $\sim$ has no critical tuple.*

Proof. Let $u_1, \dots, u_m$ be representatives from each $\sim$-class. Each equivalence class $[u_i]_\sim$ is regular since it is saturated by the finite right congruence $\sim$. Therefore each Myhill-Nerode right congruence $\sim_{[u_i]_\sim}$ has finite index. Observe that $\sim = \bigcap_{i=1}^m \sim_{[u_i]_\sim}$ because $\sim$ saturates each class $[u_i]_\sim$ and $\bigcap_{i=1}^m \sim_{[u_i]_\sim}$ also saturates each class $[v]_\sim$. Let us write $\sim_i$ instead of $\sim_{[u_i]_\sim}$ and let $\approx_i$ be its suffix expansion $\approx_{[u_i]_\sim}$. Then we have $\sim = \bigcap_{i=1}^m \sim_i$ and $\approx = \bigcap_{i=1}^m \approx_i$. This implies that

$$\max_{1 \leqslant i \leqslant m} |\Sigma^{\leqslant n}/{\approx_i}| \leqslant |\Sigma^{\leqslant n}/{\approx}| \leqslant \prod_{i=1}^m |\Sigma^{\leqslant n}/{\approx_i}|. \tag{5.1}$$

Since $[u_i]_\sim$ is regular the growth of $|\Sigma^{\leqslant n}/{\approx_i}|$ is polynomial if and only if $\sim_i$ has no critical tuple by Lemma 5.5.

($\Rightarrow$): If $|\Sigma^{\leqslant n}/{\approx}|$ is polynomially bounded then the same holds for $|\Sigma^{\leqslant n}/{\approx_i}|$ for all $1 \leqslant i \leqslant k$ by (5.1). By Lemma 5.5 the right congruence $\sim_i$ has no critical tuple for all $1 \leqslant i \leqslant k$ and therefore Lemma 5.7 implies that $\sim = \bigcap_{i=1}^m \sim_i$ has no critical tuple.

($\Leftarrow$): If $\sim$ has no critical tuple then each congruence $\sim_i$ has no critical tuple by Lemma 5.7 because $\sim_i$ is coarser than $\sim$. Lemma 5.5 implies that $|\Sigma^{\leqslant n}/{\approx_i}|$ is polynomially bounded for all $1 \leqslant i \leqslant k$. By (5.1) also $|\Sigma^{\leqslant n}/\approx|$ is polynomially bounded. □

5.3 Regular look-ahead

We follow the notation from the survey paper [47] where the Reutenauer-Schützenberger representation is viewed a sequential machine with regular look-ahead. The only difference is in our setting that the direction is reversed and thus the term "look-ahead" is a slight misnomer. Let $\mathcal{R}$ be a right congruence on Σ^* with finite index. The *look-ahead extension* is the injective function $e_{\mathcal{R}} \colon \Sigma^* \to (\Sigma \times \Sigma^*/\mathcal{R})^*$ defined by

$$e_{\mathcal{R}}(a_1 \cdots a_n) = (a_1, [\varepsilon]_{\mathcal{R}})(a_2, [a_1]_{\mathcal{R}})(a_3, [a_1 a_2]_{\mathcal{R}}) \cdots (a_n, [a_1 \cdots a_{n-1}]_{\mathcal{R}}).$$

Let $t \colon \Sigma^* \to \Omega^*$ be a partial function. We define

$$t[\mathcal{R}] \colon (\Sigma \times \Sigma^*/\mathcal{R})^* \to \Omega^*$$

to be the unique partial function with $\mathrm{dom}(t[\mathcal{R}]) = e_{\mathcal{R}}(\mathrm{dom}(t))$ and

$$t[\mathcal{R}](e_{\mathcal{R}}(x)) = t(x), \quad \text{for all } x \in \mathrm{dom}(t).$$

Furthermore we define a canonical right congruence $\mathcal{R}_t$ on Σ^*. For this we need the distance function $\|x, y\| = |x| + |y| - 2|x \wedge y|$ where $x \wedge y$ is the longest common suffix of x and y. Equivalently, $\|x, y\|$ is the length of the reduced word of xy^{-1} in the free group generated by Σ. Notice that $\|\cdot,\cdot\|$ satisfies the triangle inequality. We define $u \; \mathcal{R}_t \; v$ if and only if

(i) $u \sim_{\mathrm{dom}(t)} v$, and

(ii) $\sup\{\|t(uw), t(vw)\| : w \in u^{-1}\mathrm{dom}(t)\} < \infty$.

One can verify that $\mathcal{R}_t$ is a right congruence on Σ^*. As an example, recall the rational transduction t from Example 5.1. The induced right congruence $\mathcal{R}_t$ has two classes, which are a^* and its complement.

Theorem 5.9 ([100]). *A partial function* $t \colon \Sigma^* \to \Omega^*$ *is rational if and only if* $\mathcal{R}_t$ *has finite index and* $t[\mathcal{R}_t]$ *is right-subsequential.*

For the rest of this chapter let $t \colon \Sigma^* \to \Omega^*$ be a rational function with suffix-closed domain. Let $\mathcal{B} = (Q, \Sigma \times \Sigma^*/\mathcal{R}_t, \Omega, F, \Delta, q_0, o)$ be a trim right-subsequential transducer for $t[\mathcal{R}_t]$. One obtains a real-time right transducer $\mathcal{A}$ for t by projection to the first component, i.e. $\mathcal{A} = (Q, \Sigma, \Omega, F, \Lambda, q_0, o)$ where $\Lambda = \{(q, a, y, p) \mid (q, (a, b), y, p) \in \Delta\}$. Notice that $\mathcal{A}$ is *strongly unambiguous*, i.e. for every word $x \in \Sigma^*$ and all states $p, q \in Q$ there exists at most one run from p to q with input word x. Therefore, the state pair (p, q) and the input word x uniquely determine the run (if it exists) and we can simply write $q \xleftarrow{x} p$. Notice that every run $q \xleftarrow{x|y} p$ in $\mathcal{A}$ induces a corresponding run $q \xleftarrow{(x,z)|y} p$ in $\mathcal{B}$

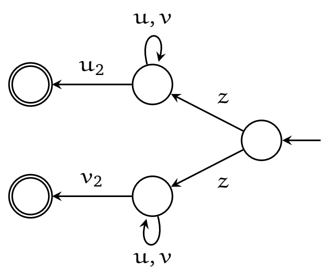

Figure 5.1: If $\mathcal{R}_t$ has a critical tuple then the right-transducer $\mathcal{B}$ contains this pattern where $\|t(u_2z), t(v_2z)\|$ can be made arbitrarily large.

for some $z \in (\Sigma^*/\mathcal{R}_t)^*$ and that this correspondence is a bijection between the sets of all runs in $\mathcal{A}$ and $\mathcal{B}$.

5.4 Critical tuples in $\mathcal{R}_t$

In the following we prove that if $\mathcal{R}_t$ has a critical tuple then t has a fooling scheme. First we identify the structure displayed in Figure 5.1.

Lemma 5.10. *If $\mathcal{R}_t$ has a critical tuple then there exists a critical tuple (u_2, v_2, u, v) in $\mathcal{R}_t$ with the following property: For every $m \in \mathbb{N}$ there exists a word $z \in \Sigma^*$ such that $\|t(u_2z), t(v_2z)\| \geqslant m$ and for each $s \in \{u_2, v_2\}$ there exists a successful run in $\mathcal{A}$ of the form $r_s \xleftarrow{s} q_s \xleftarrow{z} q_0$ and runs $q_s \xleftarrow{u} q_s$ and $q_s \xleftarrow{v} q_s$.*

Proof. By Lemma 4.51 we can assume that $u_2\{u, v\}^*$ and $v_2\{u, v\}^*$ are contained in $\mathcal{R}_t$-classes, and in particular in $\sim_{\mathrm{dom}(t)}$-classes. Furthermore we claim that $u_2x \sim_{\mathrm{dom}(t)} v_2y$ for all $x, y \in \{u, v\}^*$. Let $x, y \in \{u, v\}^*$, $z \in \Sigma^*$, and assume that $u_2xz \in \mathrm{dom}(t)$ (the other direction is analogous). Since $u_2\{u, v\}^*$ is contained in a single $\mathcal{R}_t$-class we have $u_2x \mathrel{\mathcal{R}_t} u_2xvy$ and therefore $u_2xz \mathrel{\mathcal{R}_t} u_2xvyz$. In particular we have $u_2xz \sim_{\mathrm{dom}(t)} u_2xvyz$, and thus $u_2xvyz \in \mathrm{dom}(t)$. Since $\mathrm{dom}(t)$ is suffix-closed we conclude $v_2yz \in \mathrm{dom}(t)$. The claim implies that

$$\sup_{z \in Z} \|t(u_2xz), t(v_2yz)\| = \infty \tag{5.2}$$

for all $x, y \in \{u, v\}^*$ where $Z = u_2^{-1}\mathrm{dom}(t) = v_2^{-1}\mathrm{dom}(t)$. In particular Z is nonempty.

Next we define look-ahead extensions where the congruence classes are shifted by some word $s \in \Sigma^*$. Given $s \in \Sigma^*$ we define $e_s \colon \Sigma^* \to (\Sigma \times \Sigma^*/\mathcal{R}_t)^*$ where $e_s(w)$ is the unique word such that $e_{\mathcal{R}_t}(sw) = e_{\mathcal{R}_t}(s)e_s(w)$. Explicitly written, for $w = a_1 \cdots a_n \in \Sigma^*$ it is defined as

$$e_s(a_1 \cdots a_n) = (a_1, [s]_{\mathcal{R}_t})(a_2, [sa_1]_{\mathcal{R}_t}) \cdots (a_n, [sa_1 \cdots a_{n-1}]_{\mathcal{R}_t}).$$

We have $e_s(xy) = e_s(x)e_{sx}(y)$ for all $s, x, y \in \Sigma^*$. If $s \in \{u_2, v_2\}$, $x \in \{u, v\}^*$ and $y \in \Sigma^*$ then $e_{sx}(y) = e_s(y)$ because sx and s are $\mathcal{R}_t$-congruent. Therefore

$$e_s|_{\{u,v\}^*} : \{u, v\}^* \to (\Sigma \times \Sigma^*/\mathcal{R}_t)^*$$

is a monoid homomorphism for all $s \in \{u_2, v_2\}$. Hence for all $s \in \{u_2, v_2\}$, $z \in Z$ and $x = x_1 \cdots x_n$ where $x_1, \ldots, x_n \in \{u, v\}$ we have

$$e_{\mathcal{R}_t}(sxz) = e_{\mathcal{R}_t}(sx)e_{sx}(z) = e_{\mathcal{R}_t}(s)e_s(x_1) \cdots e_s(x_n)e_s(z).$$

Let M be the monoid of partial functions $\tau\colon Q \to Q$ with equipped with the composition $(\tau \circ \sigma)(q) = \tau(\sigma(q))$ as multiplication. Let $h\colon (\Sigma \times \Sigma^*/\mathcal{R}_t)^* \to M$ be the homomorphism where $h(w)(q)$ is the state p reached on the unique run $p \xleftarrow{w} q$ in $\mathcal{B}$, if it exists. Hence we have the following homomorphism

$$\begin{aligned}\varphi\colon \{u, v\}^* &\to M \times M \\ w &\mapsto (h(e_{u_2}(w)), h(e_{v_2}(w))).\end{aligned}$$

By Lemma 4.50 there exist words $u' \in \{u, v\}^*u$ and $v' \in \{u, v\}^*v$ such that $\{\varphi(u'), \varphi(v')\}$ is a right zero semigroup. Let us replace u by u' and v by v'. Notice that (5.2) still holds.

The right zero property allows us to cycle in the transducer. Suppose that there is a run $q \xleftarrow{e_s(x)} p$ in $\mathcal{B}$ where $s \in \{u_2, v_2\}$ and $x \in \{u, v\}$. For any $x' \in \{u, v\}$ we have $\varphi(x')\varphi(x) = \varphi(x)$ and therefore $q \xleftarrow{e_s(x')e_s(x)} p$. Since $\mathcal{B}$ is right-subsequential the run $q \xleftarrow{e_s(x')} q$ exists in $\mathcal{B}$.

Now let $m \in \mathbb{N}$ be any bound. By (5.2) there exists $z \in Z$ such that $\|t(u_2uz), t(v_2uz)\| \geqslant m$. Let $s \in \{u_2, v_2\}$ and consider the successful run

$$r_s \xleftarrow{e_{\mathcal{R}_t}(s)} q_s \xleftarrow{e_s(u)} p_s \xleftarrow{e_s(z)} q_0$$

in $\mathcal{B}$. As remarked above we also have runs $q_s \xleftarrow{e_s(u)} q_s$ and $q_s \xleftarrow{e_s(v)} q_s$. By replacing z by uz we conclude the proof. □

We need a simple lemma which states how distinct arithmetic progressions can be made disjoint by synchronous pumping.

Lemma 5.11. *Given two arithmetic progressions $p_1(n) = d_1n + c_1$ and $p_2(n) = d_2n + c_2$ where $d_i, c_i \in \mathbb{N}$ such that $c_1 \neq c_2$. Then there exists an arithmetic progression $p(n) = dn + c$ where $d \geqslant 1$ such that $\{p_1(p(n)) \mid n \in \mathbb{N}\}$ and $\{p_2(p(m)) \mid m \in \mathbb{N}\}$ are disjoint.*

Proof. We will do a case distinction on whether the numbers d_1, d_2 are zero or not.

Case 1. If $d_1 = d_2 = 0$ then $p_1(n) = c_1 \neq c_2 = p_2(m)$ for all $m, n \in \mathbb{N}$.

Case 2. If $d_1 \geqslant 1$ and $d_2 = 0$ then $p_1(n + c_2 + 1) \geqslant d_1(c_2 + 1) + c_1$ is strictly bigger than $c_2 = p_2(m + c_2 + 1)$ for all $m, n \in \mathbb{N}$. The case $d_2 \geqslant 1$ and $d_1 = 0$ is analogous.

Case 3. Assume $d_1, d_2 \geqslant 1$. Let $d > |c_2 - c_1| \geqslant 1$. We claim that $p_1(dn) \neq p_2(dm)$ for all $m, n \in \mathbb{N}$. Otherwise there exist $m, n \in \mathbb{N}$ such that

$$dd_1n + c_1 = dd_2m + c_2$$

and hence

$$d(d_1n - d_2m) = c_2 - c_1,$$

which implies that d divides $|c_2 - c_1|$ and therefore contradicts $d > |c_2 - c_1|$. □

Similarly to [112], we define the parameter

$$\mathsf{iml}(\mathcal{A}) = \max\left(\{|y| : (q, a, y, p) \in \Delta\} \cup \{|o(q)| : q \in Q\}\right).$$

For every run π on a word $x \in \Sigma^*$ we have $|\mathsf{out}(\pi)| \leqslant \mathsf{iml}(\mathcal{A}) \cdot |x|$ and $|\mathsf{out}_F(\pi)| \leqslant \mathsf{iml}(\mathcal{A}) \cdot (|x| + 1)$.

Lemma 5.12. *If $\mathcal{R}_t$ has a critical tuple then t has a fooling scheme.*

Proof. Let (u_2, v_2, u, v) be the critical tuple from Lemma 5.10. We use the same state names as in the lemma. Let $m = \mathsf{iml}(\mathcal{A}) \cdot (|u_2| + 1)$, which is an upper bound on $|\mathsf{out}_F(q \xleftarrow{s} p)|$ for any run $q \xleftarrow{s} p$ on $s \in \{u_2, v_2\}$. By Lemma 5.10 we can choose $z \in \Sigma^*$ such that $\|t(u_2z), t(v_2z)\| \geqslant 2m + 1$. We distinguish two cases.

Case 1. First assume that $|t(u_2z)| = |t(v_2z)|$. Let $y = t(u_2z) \wedge t(v_2z)$ be the longest common suffix. We have

$$t(sz) = \mathsf{out}_F(r_s \xleftarrow{s} q_s)\, \mathsf{out}(q_s \xleftarrow{z} q_0)$$

for $s \in \{u_2, v_2\}$. Assume that $\mathsf{out}(q_s \xleftarrow{z} q_0)$ is a suffix of y for some $s \in \{u_2, v_2\}$. This would imply that

$$\begin{aligned} \|t(u_2z), t(v_2z)\| &= |t(u_2z)| + |t(v_2z)| - 2|y| \\ &\leqslant 2|t(sz)| - 2|\mathsf{out}(q_s \xleftarrow{z} q_0)| \\ &\leqslant 2|\mathsf{out}_F(r_s \xleftarrow{s} q_s)| \leqslant 2m, \end{aligned}$$

which contradicts the assumption that $\|t(u_2z), t(v_2z)\| \geqslant 2m+1$. This proves that y is a proper suffix of both $\mathsf{out}(q_{u_2} \xleftarrow{z} q_0)$ and $\mathsf{out}(q_{v_2} \xleftarrow{z} q_0)$. By maximality of $|y|$ the words $\mathsf{out}(q_{u_2} \xleftarrow{z} q_0)$ and $\mathsf{out}(q_{v_2} \xleftarrow{z} q_0)$ have distinct suffixes of length $|y| + 1$. Since for all $s \in \{u_2, v_2\}$ every word in $t(s\{u, v\}^*z)$ has the suffix $\mathsf{out}(q_s \xleftarrow{z} q_0)$, the sets $u_2\{u, v\}^*z$ and $v_2\{u, v\}^*z$ are separated by t.

Case 2. Now assume that $|t(u_2z)| \neq |t(v_2z)|$. For $s \in \{u_2, v_2\}$ define

$$d_s = |\mathsf{out}(q_s \xleftarrow{u} q_s)| + |\mathsf{out}(q_s \xleftarrow{v} q_s)|$$

and

$$c_s = |\mathsf{out}_F(r_s \xleftarrow{s} q_s \xleftarrow{z} q_0)|.$$

These parameters satisfy

$$|t(sxz)| = d_s n + c_s, \quad \text{for all } x \in \{uv, vu\}^n,$$

We have $c_{u_2} = |t(u_2z)| \neq |t(v_2z)| = c_{v_2}$. By Lemma 5.11 there exist numbers $d \geqslant 1$, $c \in \mathbb{N}$ such that

$$\{d_{u_2}(dn + c) + c_{u_2} \mid n \in \mathbb{N}\} \cap \{d_{v_2}(dm + c) + c_{v_2} \mid m \in \mathbb{N}\} = \emptyset.$$

This implies that t separates

$$u_2\{(uv)^d, (vu)^d\}^*(uv)^c z \quad \text{and} \quad v_2\{(uv)^d, (vu)^d\}^*(uv)^c z.$$

This concludes the proof. □

5.5 Well-behaved transducers

Let $(Q, \preceq)$ be the quasi-order defined by $q \preceq p$ iff there exists a run from p to q in $\mathcal{A}$, or equivalently in $\mathcal{B}$. Its equivalence classes are the *strongly connected components (SCCs)* of $\mathcal{A}$ and $\mathcal{B}$. A run π in $\mathcal{A}$ is *internal* if all states in π are contained in single SCC. A word $w \in \Sigma^*$ is *guarded* by a state $p \in Q$ if there exists an internal run $q \xleftarrow{w} p$ in $\mathcal{A}$. Notice that the set of all words which are guarded by a fixed state p is suffix-closed. A run $q \xleftarrow{w} p$ in $\mathcal{A}$ is *guarded* if w is guarded by p. We say that $\mathcal{A}$ is *well-behaved* if for all $p \in Q$ and all guarded accepting runs π, π' from p with $|\pi| = |\pi'|$ we have $\mathsf{out}_F(\pi) = \mathsf{out}_F(\pi')$.

Lemma 5.13. *If $\mathcal{A}$ is not well-behaved then t has a fooling scheme.*

Proof. If $\mathcal{A}$ is not well-behaved then it contains the structure from Figure 5.1 where the two z-runs are identical, and the u_2- and v_2-runs have distinct output words. Assume there exist states $p, q, r \in Q$, and guarded accepting runs $q \xleftarrow{u_2} p$ and $r \xleftarrow{v_2} p$ with $|u_2| = |v_2|$ and $\mathsf{out}_F(q \xleftarrow{u_2} p) \neq \mathsf{out}_F(r \xleftarrow{v_2} p)$. The latter property also shows that $|u_2| = |v_2| \geqslant 1$. Furthermore let $q', r' \in Q$ be states such that $p \xleftarrow{u_1} q' \xleftarrow{u_2} p$, $p \xleftarrow{v_1} r' \xleftarrow{v_2} p$ and $p \xleftarrow{z} q_0$. Finally define $u = u_1u_2$ and $v = v_1v_2$.

Case 1. If $\mathsf{out}(p \xleftarrow{u} p) = \mathsf{out}(p \xleftarrow{v} p) = \varepsilon$ then $t(u_2xz) = t(u_2z) \neq t(v_2z) = t(v_2yz)$ for all $x, y \in \{u, v\}^*$. Hence t separates $u_2\{u, v\}^*z$ and $v_2\{u, v\}^*z$.

Case 2. Assume $\mathsf{out}(p \xleftarrow{u} p) = \varepsilon$ and $\mathsf{out}(p \xleftarrow{v} p) \neq \varepsilon$. First we can ensure that $|u| = |v|$ by replacing u by $u^{|v|}$, and v by $v^{|u|}$. Next we can ensure that $|t(u_2z)| <$

$|t(v_2z)|$ by replacing (u_2, v_2, u, v) by $(u_2u^i, v_2v^i, u^{i+1}, v^{i+1})$ for sufficiently large i. Notice that $|u_2u^i| = |v_2v^i|$ because $|u_2| = |v_2|$ and $|u| = |v|$ by assumption. This shows that $|t(u_2xz)| = |t(u_2z)| < |t(v_2z)| \leqslant |t(v_2yz)|$ for all $x, y \in \{u, v\}^*$. Hence t separates $u_2\{u, v\}^*z$ and $v_2\{u, v\}^*z$.

The case $\mathsf{out}(p \xleftarrow{u} p) \neq \varepsilon$ and $\mathsf{out}(p \xleftarrow{v} p) = \varepsilon$ is analogous.

Case 3. Finally assume that both $\mathsf{out}(p \xleftarrow{u} p)$ and $\mathsf{out}(p \xleftarrow{v} p)$ are nonempty. We can establish $|\mathsf{out}(p \xleftarrow{u} p)| = |\mathsf{out}(p \xleftarrow{v} p)|$ by replacing u and v with suitable powers u^k and v^ℓ. By picking k, ℓ large enough we can further assume that

$$|\mathsf{out}(p \xleftarrow{u} p)| = |\mathsf{out}(p \xleftarrow{v} p)| > \max\{\mathsf{out}_F(q \xleftarrow{u_2} p), \mathsf{out}_F(r \xleftarrow{v_2} p)\}. \qquad (5.3)$$

Now assume that there exist $x, y \in \{u, v\}^*$ such that $t(u_2xz) = t(v_2yz)$. We can compute these values as

$$t(u_2xz) = \mathsf{out}_F(q \xleftarrow{u_2} p)\mathsf{out}(p \xleftarrow{x} p)\mathsf{out}(p \xleftarrow{z} q_0)$$

and

$$t(v_2yz) = \mathsf{out}_F(r \xleftarrow{v_2} p)\mathsf{out}(p \xleftarrow{y} p)\mathsf{out}(p \xleftarrow{z} q_0).$$

By (5.3) the factors $\mathsf{out}(p \xleftarrow{x} p)$ and $\mathsf{out}(p \xleftarrow{y} p)$ must have equal length. Therefore $\mathsf{out}_F(q \xleftarrow{u_2} p) = \mathsf{out}_F(r \xleftarrow{v_2} p)$, which contradicts the assumption. Hence t separates $u_2\{u, v\}^*z$ and $v_2\{u, v\}^*z$. □

Run keys If π is a nonempty run $p \xleftarrow{a_1 \cdots a_n} q$ in $\mathcal{A}$ and $p \xleftarrow{(a_1,\rho_1)\cdots(a_n,\rho_n)} q$ is the corresponding run in $\mathcal{B}$ then we call ρ_1 the *key* of π, denoted by $\rho_1 = \mathsf{key}(\pi)$. The following lemma justifies the name, stating that π is determined by the state q, the word $a_1 \cdots a_n$ and the key ρ_1.

Lemma 5.14. *If* $p \xleftarrow{w} q$ *and* $p' \xleftarrow{w} q$ *are nonempty runs in* $\mathcal{A}$ *with the same key then the runs must be identical.*

Proof. Assume that $w = a_1 \cdots a_n$ and let

$$\pi \colon p \xleftarrow{(a_1,\rho_1)\cdots(a_n,\rho_n)} q$$

be a run in $\mathcal{B}$. Since $\mathcal{B}$ is trim there exist an accepting run π' on $e_{\mathcal{R}_t}(u)$ from p for some word u. By definition of $t[\mathcal{R}_t]$ the run $\pi'\pi$ is on the word

$$e_{\mathcal{R}_t}(uw) = e_{\mathcal{R}_t}(u)(a_1, [u]_{\mathcal{R}_t})(a_2, [ua_1]_{\mathcal{R}_t}) \cdots (a_n, [ua_1 \cdots a_{n-1}]_{\mathcal{R}_t}).$$

This shows that $\rho_i = [ua_1 \cdots a_i]_{\mathcal{R}_t}$ and that ρ_1 determines all classes $\rho_2, \dots, \rho_n$. Hence, any two nonempty runs in $\mathcal{A}$ on the same word starting from q with the same key must be identical. □

Tree summaries We define a data structure, called a tree summary, on the transducer $\mathcal{A}$ similar to the path summary from Section 4.1. For each word

$w \in \Sigma^*$ and each state $q \in Q$ we define the *tree summary* $T_{q,w}$ recursively, which is a node- and edge-labeled tree. The root is labeled by the tuple $(q, |w|, \bar{v}_{\mathcal{R}_t}(w))$. If w is guarded by q then the root is the only node in the tree. Otherwise let $w = uv$ such that v is the shortest suffix of w which is not guarded by q. For each run $p \xleftarrow{v} q$ in $\mathcal{A}$ we attach $T_{p,u}$ to the root as a direct subtree. The edge is labeled by the pair $(\mathsf{key}(p \xleftarrow{v} q), \mathsf{out}(p \xleftarrow{v} q))$. By Lemma 5.14 distinct outgoing edges from the root are labeled by distinct keys.

Lemma 5.15. *The tree summary* $T_{q,w}$ *indicates whether* w *is guarded by* q*, and if not, it determines the length of the shortest suffix of* w *which is not guarded by* q.

Proof. The word w is guarded by q if and only if $T_{q,w}$ has size one. If the root does have children, then one can factor $w = uv$ such that v is the shortest suffix that is unguarded by q and all children are all labeled by tuples of the form $(p, |u|, \bar{v}_{\mathcal{R}_t}(u))$ for some state $p \in Q$. Hence we can determine $|v| = |w| - |u|$ where $|w|$ can be determined from the root label of $T_{q,w}$. □

If $T_{p,u}$ is a subtree of $T_{q,w}$ then $p \prec q$. Therefore the tree $T_{q,w}$ has height at most $|Q|$ and size at most $|Q|^{|Q|}$, which are constants in $|w|$.

Lemma 5.16. *Assume that* $\mathrm{im}(t)$ *is bounded and* $\mathcal{R}_t$ *has no critical tuple. Then the number of tree summaries* $T_{q,w}$ *with* $q \in Q$ *and* $w \in \mathrm{dom}(t) \cap \Sigma^{\leqslant n}$ *is polynomially bounded in* n.

Proof. All occurring numbers have at most magnitude n, the state set Q has constant size, and the set of possible keys $\Sigma^*/\mathcal{R}_t$ has constant size. The output words $\mathsf{out}(p \xleftarrow{v} q)$ are factors of words from the bounded language $\mathrm{im}(t)$ and have length at most $\mathsf{iml}(\mathcal{A}) \cdot |v| = O(n)$. Thus, by Lemma 3.14 there are at most polynomially many of such output words. By Theorem 5.8 there are also at most polynomially many node labels $\bar{v}_{\mathcal{R}_t}(u)$ where $|u| \leqslant n$. In conclusion, the number of tree summaries is polynomially bounded. □

Guarded factorizations Let π be any run on a word $w \in \Sigma^*$. If π is not guarded, we can factorize $\pi = \pi'\pi''$ such that π'' is the shortest suffix of π which is unguarded, and then repeat this process on π'. This yields unique factorizations $\pi = \pi_0\pi_1 \cdots \pi_m$ and $w = w_0w_1 \cdots w_m$ where π_i is a run on w_i from a state p_{i+1} to a state p_i such that w_i is the shortest suffix of $w_0 \cdots w_i$ which is not guarded by p_i for all $1 \leqslant i \leqslant m$, and π_0 is guarded. The factorization $\pi = \pi_0\pi_1 \cdots \pi_m$ is the *guarded factorization* of π.

Proposition 5.17. *If* $\mathcal{A}$ *is well-behaved and* π *is an accepting run on* w *from* q *then* $T_{q,w}$ *determines* $\mathsf{out}_F(\pi)$*. In particular, if* $w \in \mathrm{dom}(t)$ *then* $T_{q_0,w}$ *determines* $t(w)$.

Proof. Let $\pi = \pi_0\pi_1 \cdots \pi_m$ be the guarded factorization of π and $w_0w_1 \cdots w_m$ be the corresponding factorization of w. We prove the statement by induction on m.

The root of $T_{q,w}$ is labeled by $(q, |w|, \overleftarrow{v}_{\mathcal{R}_t}(w))$. If $m = 0$ then w is guarded by q and $T_{q,w}$ has size one. Since $\mathcal{A}$ is well-behaved $\mathsf{out}_F(\pi)$ is determined by q and $|w|$ only.

Now assume $m \geqslant 1$ and suppose that π_i is a run $p_i \xleftarrow{w_i} p_{i+1}$ for all $1 \leqslant i \leqslant m$ with $p_{m+1} = q$. Then w_m is the shortest suffix of w which is not guarded by q. All children of the root of $T_{q,w}$ are labeled by

$$(p, |w_0 \cdots w_{m-1}|, \overleftarrow{v}_{\mathcal{R}_t}(w_0 \cdots w_{m-1}))$$

for some state $p \in Q$. Hence, we can determine $\overleftarrow{v}_{\mathcal{R}_t}(w_0 \cdots w_{m-1})$, which in turn determines $|w_0 \cdots w_{m-1}|$ and $[w_0 \cdots w_{m-1}]_{\mathcal{R}_t}$. Notice that $[w_0 \cdots w_{m-1}]_{\mathcal{R}_t}$ is the key of π_m. Therefore, the root has an outgoing edge which is labeled by $([w_0 \cdots w_{m-1}]_{\mathcal{R}_t}, \mathsf{out}(\pi_m))$. By Lemma 5.14 it is the unique outgoing edge with key $[w_0 \cdots w_{m-1}]_{\mathcal{R}_t}$. It leads to the direct subtree $T_{p_m, w_0 \cdots w_{m-1}}$ of $T_{q,w}$. By induction hypothesis this subtree determines $\mathsf{out}_F(\pi_0 \cdots \pi_{m-1})$. Finally, we can determine

$$\mathsf{out}_F(\pi_0 \cdots \pi_m) = \mathsf{out}_F(\pi_0 \cdots \pi_{m-1})\, \mathsf{out}(\pi_m),$$

concluding the proof. □

It remains to show that the tree summary $T_{q,w}$ determines the output value of every suffix of w. In fact, we will prove that we can compute the tree summary of every suffix.

Lemma 5.18. *Suppose that* $w = xy \in \Sigma^*$. *Given the tree summary* $T_{q,w}$ *and* $|y|$, *one can determine the tree summary* $T_{q,y}$.

Proof. We proceed by induction on the height of $T_{q,w}$. Its root is labeled by $(q, |w|, \overleftarrow{v}_{\mathcal{R}_t}(w))$. The root of $T_{q,y}$ is labeled by $(q, |y|, \overleftarrow{v}_{\mathcal{R}_t}(y))$. Notice that $\overleftarrow{v}_{\mathcal{R}_t}(y)$ is the suffix of $\overleftarrow{v}_{\mathcal{R}_t}(w)$ of length $|y|$.

If $|T_{q,w}| = 1$ then w and also all its suffixes are guarded by q and thus $T_{q,y}$ also only consists of its root.

Now factorize $w = uv$ where v is the shortest suffix which is not guarded by q. By Lemma 5.15 we can determine $|u|$ and $|v|$ from $T_{q,w}$. The suffix y is guarded by q if and only if $|y| < |v|$. If y is guarded by q, then, as above, $T_{q,y}$ only consists of its root. Otherwise assume $|y| \geqslant |v|$, and hence $y = sv$ for some word s. We can determine $\overleftarrow{v}_{\mathcal{R}_t}(s)$ from $\overleftarrow{v}_{\mathcal{R}_t}(u)$ as its suffix of length $|s| = |y| - |v|$. It remains to construct the subtrees. For this, observe the following.

- The direct subtrees of $T_{q,w}$ are of the form $T_{p,u}$ where $p \xleftarrow{v} q$ is any run.
- The direct subtrees of $T_{q,y}$ are of the form $T_{p,s}$ where $p \xleftarrow{v} q$ is any run.

By induction hypothesis we can determine each subtree $T_{p,s}$ from $T_{p,u}$ and the length $|s|$. The edge labels are directly copied: The edge (in either tree) induced by the run $p \xleftarrow{v} q$ is labeled by $(\mathsf{key}(p \xleftarrow{v} q), \mathsf{out}(p \xleftarrow{v} q))$. □

Proposition 5.19. *Assume that* $\mathrm{im}(t)$ *is bounded,* $\mathcal{A}$ *is well-behaved and* $\mathcal{R}_t$ *has no critical tuple. Then* $\mathrm{im}(\overleftarrow{t})$ *has polynomial growth.*

Proof. Consider the function ψ which maps a word $w \in \mathrm{dom}(t)$ to $T_{q_0,w}$. By Lemma 5.16 we know that $|\psi(\mathrm{dom}(t) \cap \Sigma^{\leqslant n})|$ is polynomially bounded in n. By Lemma 5.18 the tree summary $T_{q_0,w}$ determines the tree summaries $T_{q_0,v}$ of all suffixes v of w. Hence, by Proposition 5.17 the values $t(v)$ of all suffixes v of w are also determined.

□

Proof of Theorem 5.2. If t has no fooling scheme then $\mathcal{A}$ must be well-behaved by Lemma 5.13 and $\mathcal{R}_t$ has no critical tuple by Lemma 5.12. If additionally $\mathrm{im}(t)$ is a bounded language then $\mathrm{im}(\overleftarrow{t})$ has polynomial growth by Proposition 5.19. Otherwise t has a fooling scheme or $\mathrm{im}(t)$ has exponential growth. Then $\mathrm{im}(\overleftarrow{t})$ must have exponential growth by Proposition 5.3 and Proposition 5.4. □

5.6 Space trichotomy

Finally, we will prove the space trichotomy for rational functions in the sliding window model. So far we have shown that the suffix complexity $S_\varphi(n)$ of any rational function φ is either $O(\log n)$ or $\Omega(n)$ infinitely often. In fact, we will see that, as for regular languages, these complexity bounds can be transferred to $V_\varphi(n)$ and $F_\varphi(n)$. Notice that this holds although $\sim_\varphi$ does not have finite index for arbitrary rational functions φ.

First we observe that one can maintain the set of all tree summaries on the active window w. By Lemma 5.18 we already know that the $\downarrow$-operation is supported.

Proposition 5.20. *Given the set of all tree summaries* $\{T_{q,w} \mid q \in Q\}$ *for a word* $w \in \Sigma^*$ *and a letter* $a \in \Sigma$*, one can determine* $\{T_{q,wa} \mid q \in Q\}$.

Proof. Observe that a word va is guarded from p if and only if there exists an internal transition $q \xleftarrow{a} p$ such that v is guarded from q. Suppose we want to compute $T_{p,wa}$ for some state $p \in Q$. We distinguish three cases:

Case 1. Assume that a is not guarded from p, i.e. there exists no internal transition $q \xleftarrow{a} p$. The root of $T_{p,wa}$ is labeled by $(p, |wa|, \bar{v}_{\mathcal{R}_t}(wa))$, which can be computed from the label $(p, |w|, \bar{v}_{\mathcal{R}_t}(w))$ of the root of $T_{p,w}$ because $\mathcal{R}_t$ is a right congruence. For every transition $q \xleftarrow{a} p$ we attach $T_{q,w}$ as a direct subtree with the edge label $(\mathsf{key}(q \xleftarrow{a} p), \mathsf{out}(q \xleftarrow{a} p))$. Since all tree summaries $T_{q,w}$ are given, we are done in this case.

Case 2. Assume that a is guarded from p. Further assume that there is an internal transition $q \xleftarrow{a} p$ such that q guards w, and hence p guards wa. Using Lemma 5.15 we can determine whether this case holds. In this case $T_{p,wa}$ only consists of a root labeled by $(p, |wa|, \bar{v}_{\mathcal{R}_t}(wa))$.

Case 3. Otherwise for every internal transition $q \xleftarrow{a} p$ there exists a shortest suffix v_q of w which is not guarded by q. Choose q such that $|v_q|$ is maximal. We claim that $v_q a$ is not guarded by p. Otherwise there exists an internal run

$$q'' \xleftarrow{v_q} q' \xleftarrow{a} p$$

and therefore q' guards v_q. This implies that $|v_{q'}| > |v_q|$, which contradicts the maximality of $|v_q|$. Furthermore every proper suffix of $v_q a$ is guarded by p since every proper suffix of v_q is guarded by q. In conclusion $v_q a$ is the shortest suffix of wa which is not guarded by p.

Hence we can compute $T_{p,wa}$ from $T_{q,w}$ as follows. First, we change the root label from $(q, |w|, \bar{v}_{\mathcal{R}_t}(w))$ to $(p, |wa|, \bar{v}_{\mathcal{R}_t}(wa))$, similar to case 1. Next, take an outgoing edge from the root which is induced by a run of the form $r \xleftarrow{v_q} q$. We change its edge label from

$$(\mathsf{key}(r \xleftarrow{v_q} q), \mathsf{out}(r \xleftarrow{v_q} q)) \quad \text{to} \quad (\mathsf{key}(r \xleftarrow{v_q a} p), \mathsf{out}(r \xleftarrow{v_q a} p)).$$

Notice that the two keys above are actually equal and $\mathsf{out}(r \xleftarrow{v_q a} p)$ can be determined since

$$\mathsf{out}(r \xleftarrow{v_q a} p) = \mathsf{out}(r \xleftarrow{v_q} q)\,\mathsf{out}(q \xleftarrow{a} p).$$

This concludes the case distinction and the proof of the statement. □

Theorem 5.21. *Let $\varphi\colon \Sigma^* \to \Omega^*$ be a total rational function. If* $\mathrm{im}(\varphi)$ *is a bounded language and φ has no fooling scheme then $V_\varphi(n) = O(\log n)$.*

Proof. Assume that $\mathrm{im}(\varphi)$ is a bounded language and φ has no fooling scheme. Let $\mathcal{B}$ be a trim right-subsequential transducer for $\varphi[\mathcal{R}_\varphi]$ with state set Q, and let $\mathcal{A}$ be the right transducer for φ obtained by projecting the transitions in $\mathcal{B}$ to the first component. Lemma 5.12 and Lemma 5.13 state that $\mathcal{R}_\varphi$ has no critical tuple and that $\mathcal{A}$ is well-behaved. By Lemma 5.18 and Proposition 5.20 a streaming algorithm for φ can maintain the set of all tree summaries $\{T_{q,w} \mid q \in Q\}$ for the active window $w \in \Sigma^*$. By Proposition 5.17 this information suffices to compute the value $\varphi(w)$. By Lemma 5.16 the space complexity of the algorithm is $O(\log n)$. □

Theorem 5.22. *Let $\varphi\colon \Sigma^* \to \Omega^*$ be a total rational function. If* $\mathrm{im}(\varphi)$ *is not a bounded language or φ has a fooling scheme then $V_\varphi(n) = \Omega(n)$, and $F_\varphi(n) = \Omega(n)$ for infinitely many n.*

Proof. Assume that $\mathrm{im}(\varphi)$ is not bounded. By Proposition 5.4 the growth of $|\varphi(\Sigma^n)|$ is exponential and $|\varphi(\Sigma^{\leqslant n})| = 2^{\Omega(n)}$. Hence for infinitely many window sizes n any SW-algorithm $\mathcal{P}_n$ for φ must output an exponential number of values and thus $\mathcal{P}_n$ must have space complexity $\Omega(n)$. Analogously, every variable-size SW-algorithm $\mathcal{P}$ for φ must output an exponential number of values on streams of maximum window length $\leqslant n$, and thus its space complexity is $\Omega(n)$.

Assume that (u_2, v_2, u, v, z) is a fooling scheme of φ. We can assume $|u| = |v|$ by replacing u by $u^{|v|}$ and replacing v by $v^{|u|}$. Consider an SW-algorithm $\mathcal{P}_n$ for φ and window length $n = |u_2| + |u| \cdot (m-1) + |z|$ for some $m \geqslant 1$. Read two distinct words $x, y \in \{u, v\}^m$ into two instances of $\mathcal{P}_n$. Consider the last $\{u, v\}$-block where x and y differ, say $x = x_1 x_2 s$ and $y = y_1 y_2 s$ for some $x_1, y_1, s \in \{u, v\}^*$ and $\{x_2, y_2\} = \{u, v\}$. By reading $x_1 z$ into both instances the windows become $u_2 s x_1 z$ and $v_2 s x_1 z$. Since $\varphi(u_2 s x_1 z) \neq \varphi(v_2 s x_1 z)$ the algorithm $\mathcal{P}_n$ must distinguish the streams x and y. Therefore $\mathcal{P}_n$ has space complexity $\Omega(n)$. Since $V_\varphi(n) \geqslant F_\varphi(n)$ and the linear lower bound for $F_\varphi(n)$ holds over an arithmetic progression (namely $n = |u_2| + |u| \cdot (m-1) + |z|$) we get $V_\varphi(n) = \Omega(n)$ by monotonicity of $V_\varphi(n)$. □

Together with the $\Omega(\log n)$ lower bound from Lemma 3.7 for every nontrivial problem we obtain the following space trichotomy.

Corollary 5.23. *Every total rational function has space complexity* $\Theta(1)$, $\Theta(\log n)$, *or* $\Omega(n)$ *infinitely often in the variable-size sliding window model.*

In the fixed-size sliding window model we only obtain a dichotomy theorem and leave it as an open problem to analyze rational functions with sublogarithmic complexity.

Corollary 5.24. *Every total rational function has space complexity* $O(\log n)$*, or* $\Omega(n)$ *infinitely often in the fixed-size sliding window model.*

We conjecture that one can extend the techniques from Section 4.1.2 to rational functions and prove that $F_\varphi(n)$ is either $O(1)$ or $\Omega(\log n)$ infinitely often for every rational function φ.

5.7 Conclusion

In this chapter we have studied the suffix expansions of rational functions and have shown a dichotomy for their growth functions. From this dichotomy we have derived a space trichotomy for rational functions in the variable-size sliding window model.

Open problems

1. Is the space complexity of every rational function in the fixed-size model either $O(1)$ or $\Omega(\log n)$ infinitely often?
2. Find a simple characterization of those rational functions t where $\mathrm{im}(\overleftarrow{t})$ has polynomial growth. This class of rational functions could be related to the class of (right) multisubsequential functions [29]. A function is multisubsequential if its domain can be decomposed into regular subsets such that the restriction to each subset is subsequential.

Related work Filiot et al. have studied the question which word functions can be computed by a deterministic streaming algorithm using small memory [46]. In contrast to our model the output symbols are written left-to-right to an output tape. They observe that a rational function can be computed using constant space in the input length if and only if it is left-sequential. Furthermore, they consider so-called *nested word transductions*, which are computed by finite state transducers equipped with a visibly pushdown. The authors prove that it is decidable in coNP, whether a nested word transduction is streamable in height bounded memory, i.e. the memory only depends on the height of the input word.

Chapter 6

Randomized sliding window algorithms

Most of the work in the context of streaming use randomness and/or approximation to design space- and time-efficient algorithms. For example, the AMS-algorithm [5] approximates the number of distinct elements in a stream with high probability in $O(\log m)$ space where m is the size of the universe. Furthermore, it is proved that any deterministic approximation algorithm and any randomized exact algorithm must use $\Omega(n)$ space [5]. On the other hand, the exponential histogram algorithm by Datar et al. [36] is a *deterministic* sliding window approximation algorithm using $O(\frac{1}{\epsilon}\log^2 n)$ bits. It is proven that $\Omega(\frac{1}{\epsilon}\log^2 n)$ bits are necessary even for randomized (Monte Carlo or Las Vegas) sliding window algorithms.

In this chapter we will study if and how randomness helps for testing membership to regular languages over sliding windows. The main result of this chapter is a space quatrochotomy in the fixed-size sliding window model, stating that every regular language has space complexity $O(1)$, $\Theta(\log\log n)$, $\Theta(\log n)$ or $\Theta(n)$ if the algorithms may have a two-sided error. For algorithms with one-sided error we obtain the same trichotomy as in the deterministic setting.

The results of this chapter appeared in [G3].

6.1 Randomized streaming algorithms

Probabilistic automata with output In the following we will introduce probabilistic automata [97, 99] as a model of randomized streaming algorithms which produce an output after each input symbol. A *randomized streaming algorithm* or a *probabilistic automaton with output* $\mathcal{P} = (Q, \Sigma, \iota, \rho, o)$ consists of

- a (possibly infinite) set of states Q,
- a finite alphabet Σ,
- an initial state distribution $\iota \colon Q \to [0, 1]$,

- a transition probability function $\rho\colon Q \times \Sigma \times Q \to [0,1]$,
- and an output function $o\colon Q \to Y$,

such that

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

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

If ι and ρ map into $\{0,1\}$, then $\mathcal{P}$ can be viewed as a deterministic automaton with output. We call $\mathcal{P}$ a *probabilistic automaton* if the output set is $Y = \{0,1\}$; in this case we specify the set $F \subseteq Q$ of final states instead of the output function o. A *run* on a word $a_1 \cdots a_m \in \Sigma^*$ in $\mathcal{P}$ from q_0 to q_m is a sequence $\pi = q_0 a_1 q_1 a_2 \dots a_m q_m \in Q(\Sigma Q)^*$ where $\rho(q_{i-1}, a_i, q_i) > 0$ for all $1 \leqslant i \leqslant m$. We write runs in the usual way

$$\pi\colon q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} q_m$$

or also omit the intermediate states: $\pi\colon q_0 \xrightarrow{a_1 \cdots a_m} q_m$. We extend ρ to runs in the natural way: If $\pi\colon q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} q_m$ is a run in $\mathcal{A}$ then $\rho(\pi) = \prod_{i=1}^{n} \rho(q_{i-1}, a_i, q_i)$. Furthermore we define $\rho_\iota(\pi) = \iota(q_0) \cdot \rho(\pi)$. We denote by $\mathrm{Runs}(\mathcal{P}, w)$ the set of all runs on w in $\mathcal{P}$. Furthermore $\mathrm{Runs}(\mathcal{P}, p, w)$ contains those runs on w that start in $p \in Q$; similarly, $\mathrm{Runs}(\mathcal{P}, w, q)$ contains those runs on w that end in $q \in Q$. Also we define $\mathrm{Runs}(\mathcal{P}, p, w, q) = \mathrm{Runs}(\mathcal{P}, p, w) \cap \mathrm{Runs}(\mathcal{P}, w, q)$. Usually we omit the automaton $\mathcal{P}$ in this notation. Notice that for each $w \in \Sigma^*$ the function ρ_ι is a probability distribution on $\mathrm{Runs}(w)$, and for each $p \in Q$ the restriction of ρ to $\mathrm{Runs}(p, w)$ is a probability distribution on $\mathrm{Runs}(p, w)$. Given a word $w \in \Sigma^*$ and a state $p \in Q$ we define the random variable $p \cdot w$ to be the state reached from p on input w, i.e. the distribution is given by

$$\Pr[p \cdot w = q] = \sum_{\pi \in \mathrm{Runs}(\mathcal{P}, p, w, q)} \rho(\pi).$$

The random variable $\mathcal{P}(w)$ is the state reached on input w, i.e. for a state $q \in Q$ we have

$$\Pr[\mathcal{P}(w) = q] = \sum_{\pi \in \mathrm{Runs}(\mathcal{P}, w, q)} \rho_\iota(\pi).$$

Hence the random output of $\mathcal{P}$ on w is $o(\mathcal{P}(w))$. The *space* of $\mathcal{P}$ (or *number of bits used by* $\mathcal{P}$) is given by $s(\mathcal{P}) = \log |Q| \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$. We say that $\mathcal{P}$ is a randomized streaming algorithm for $\varphi\colon \Sigma^* \to Y$ with error probability $0 \leqslant \lambda \leqslant 1$ if

$$\Pr[o(\mathcal{P}(x)) = \varphi(x)] \geqslant 1 - \lambda$$

for all $x \in \Sigma^*$. If φ has binary output then we view λ as the *two-sided error*. If we omit λ we choose $\lambda = 1/3$.

Probability amplification We need the following *Chernoff bound*, which can be found in [89, Theorem 4.5].

Theorem 6.1 (Chernoff bound). *Let $X_1, \dots, X_k$ be independent Bernoulli variables, and let $X = \sum_{i=1}^{k} X_i$ with expectation $\mu = \mathbf{E}[X]$. For any $0 < \delta < 1$ we have*

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

For a randomized streaming algorithm $\mathcal{P} = (Q, \Sigma, \iota, \rho, o)$ and a number $k \geqslant 1$ let $\mathcal{P}^{(k)}$ be the randomized streaming algorithm which simulates k instances of $\mathcal{P}$ in parallel with independent random bits and outputs the majority vote. Formally the states of $\mathcal{P}^{(k)}$ are multisets of size k over Q. Therefore $s(\mathcal{P}^{(k)}) \leqslant k \cdot s(\mathcal{P})$.

Lemma 6.2 (probability amplification). *For all $0 \leqslant \lambda < 1/2$ and $0 < \lambda' < 1/2$ there exists a number $k = O(\log\left(\frac{1}{\lambda'}\right) \cdot \left(\frac{1}{2} - \lambda\right)^{-2})$ such that the following holds: If $\mathcal{P}$ is a randomized streaming algorithm and $\Pr[o(\mathcal{P}(x)) = y] \geqslant 1 - \lambda$ then $\Pr[o(\mathcal{P}^{(k)}(x)) = y] \geqslant 1 - \lambda'$.*

Proof. We will choose k later. Let $X_1, \dots, X_k$ be independent Bernoulli random variables with $\Pr[X_i = 0] = \lambda$ and $\Pr[X_i = 1] = 1 - \lambda$. Since λ is an upper bound on the error probability of $\mathcal{P}$ on x, the algorithm $\mathcal{P}^{(k)}$ errs on x with probability at most $\Pr[X \leqslant k/2]$ where $X = \sum_{i=1}^{k} X_i$. By choosing $\mu = \mathbf{E}[X] = k(1-\lambda)$ and $\delta = 1 - \frac{1}{2(1-\lambda)} \in (0, \frac{1}{2}]$ we get $(1-\delta)\mu = k/2$. The Chernoff bound gives the following estimate:

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

By choosing

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

we can bound the error probability of $\mathcal{P}^{(k)}$ on x by λ'. □

Derandomization Rabin proved that any probabilistic finite automaton with a so-called isolated cut-point can be made deterministic with an exponential size increase [99]. Let $\mathcal{P} = (Q, \Sigma, \iota, \rho, F)$ be a probabilistic finite automaton. We call $\lambda \in [0, 1]$ an *isolated cut-point* with radius $\delta > 0$ if $|\Pr[\mathcal{P} \text{ accepts } x] - \lambda| \geqslant \delta$ for all $x \in \Sigma^*$.

Theorem 6.3 ([99, Theorem 3]). *Let $\mathcal{P}$ be a probabilistic finite automaton with m states. Let $L = \{x \in \Sigma^* \mid \Pr[\mathcal{P} \text{ accepts } x] \geqslant \lambda\}$ where λ is an isolated cut-point of $\mathcal{P}$ with radius $\delta > 0$. Then there exists a DFA $\mathcal{A}$ for L with at most $(1 + m/\delta)^{m-1} = 2^{O(m \log m)}$ states.*

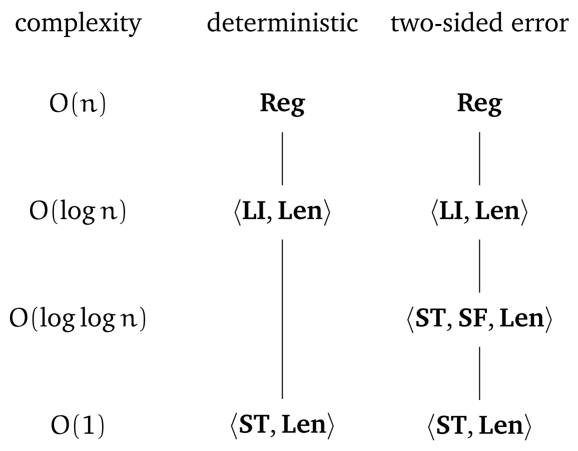
complexity
deterministic
two-sided error
$O(n)$
Reg
Reg
$O(\log n)$
$\langle \mathbf{LI}, \mathbf{Len} \rangle$
$\langle \mathbf{LI}, \mathbf{Len} \rangle$
$O(\log \log n)$
$\langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$
$O(1)$
$\langle \mathbf{ST}, \mathbf{Len} \rangle$
$\langle \mathbf{ST}, \mathbf{Len} \rangle$

Figure 6.1: The deterministic and randomized space complexity of regular languages.

6.2 Space quatrochotomy

A *randomized sliding window algorithm* for a problem φ and window size n is a randomized streaming algorithm for $\mathsf{SW}_n(\varphi)$. The *randomized space complexity* $F^r_\varphi(n)$ of φ in the fixed-size sliding window model is the minimal space complexity $s(\mathcal{P}_n)$ of a randomized sliding window algorithm $\mathcal{P}_n$ for φ and window size n. Clearly we have $F^r_\varphi(n) \leqslant F_\varphi(n)$. Furthermore, we prove that for decision problems randomness can reduce the space complexity at most exponentially using Theorem 6.3. Notice by our definition any randomized SW-algorithm for a language L has 1/2 as an isolated cutpoint with radius 1/3.

Lemma 6.4. *For any language* L *we have* $F_L(n) = 2^{O(F^r_L(n))}$.

Proof. Let $\mathcal{P}_n$ be a minimal probabilistic finite automaton for $\mathsf{SW}_n(L)$ with m states, and let $\mathcal{Q}_n$ be an equivalent DFA $\mathcal{Q}_n$ with $|\mathcal{Q}_n| \leqslant 2^{O(m \log m)}$ states. The statement follows from $F_L(n) \leqslant \log|\mathcal{Q}_n| = O(m \log m) = O(2^{F^r_L(n)} \cdot F^r_L(n))$, which is bounded by $2^{O(F^r_L(n))}$. □

Let us now state the main result of this chapter, which is a quatrochotomy for the randomized space complexity of regular languages in the fixed-size sliding window model. A language L is *suffix-free* if $xy \in L$ and $x \neq \varepsilon$ implies $y \notin L$. We denote by **SF** the class of all regular suffix-free languages.

Theorem 6.5. *Let* $L \subseteq \Sigma^*$ *be a regular language.*

(1) If $L \in \langle \mathbf{ST}, \mathbf{Len} \rangle$ *then* $F^r_L(n) = O(1)$.

(2) If $L \notin \langle \mathbf{ST}, \mathbf{Len} \rangle$ *then* $F^r_L(n) = \Omega(\log \log n)$ *infinitely often.*

(3) If $L \in \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$ *then* $F^r_L(n) = O(\log \log n)$.

(4) If $L \notin \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$ *then* $F^r_L(n) = \Omega(\log n)$ *infinitely often.*

(5) *If* $L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$ *then* $F^r_L(n) = O(\log n)$.

(6) *If* $L \notin \langle \mathbf{LI}, \mathbf{Len} \rangle$ *then* $F^r_L(n) = \Omega(n)$ *infinitely often.*

Figure 6.1 compares the deterministic and the randomized space complexity. Points (1) and (5) already hold in the deterministic setting, see Theorem 4.35. In the next sections we prove points (2), (3), (4), and (6). Observe that we can transfer Corollary 3.18 to the randomized setting.

Lemma 6.6. *For any function* $s(n)$*, the class* $\{L \subseteq \Sigma^* \mid F^r_L(n) = O(s(n))\}$ *forms a Boolean algebra.*

Proof. Let $n \in \mathbb{N}$ be a window size. If $\mathcal{P}_n$ is an SW-algorithm for L then $\overline{\mathcal{P}}_n$ is an SW-algorithm for $\Sigma^* \setminus L$ where $\overline{\mathcal{P}}_n$ simulates $\mathcal{P}_n$ and returns the negated output. Let $\mathcal{P}_n$ and $\mathcal{Q}_n$ be SW-algorithms for K and L. By Lemma 6.2 we can reduce error probability to 1/6 with a constant space increase. Then the algorithm which simulates $\mathcal{P}_n$ and $\mathcal{Q}_n$ in parallel, and returns the disjunction of the outputs is an SW-algorithm for $K \cup L$. Its error probability is at most 1/3 by the union bound. □

6.3 The Bernoulli counter

Let us start by defining a simple probabilistic counter. It is inspired by the approximate counter by Morris [49, 90], which uses $O(\log \log n)$ bits. For our purposes, it suffices to detect whether the counter has exceeded a certain threshold, which can be accomplished using only $O(1)$ bits.

Formally, a probabilistic counter is a probabilistic automaton with output $\mathcal{Z} = (C, \{\texttt{inc}\}, \iota, \rho, o)$ over the unary alphabet $\{\texttt{inc}\}$ where $o \colon C \to \{\bot, \top\}$ distinguishes *low* and *high* states. The random state reached after k increments is $\mathcal{Z}(\texttt{inc}^k)$, for which we also write $\mathcal{Z}(k)$. Given numbers $0 \leqslant \ell < h$ we say that $\mathcal{Z}$ is an (h, ℓ)*-counter with error probability* $\lambda < \frac{1}{2}$ if for all $k \in \mathbb{N}$ we have:

- If $k \leqslant \ell$, then $\Pr[\mathcal{Z}(k) \text{ is high}] \leqslant \lambda$.
- If $k \geqslant h$, then $\Pr[\mathcal{Z}(k) \text{ is low}] \leqslant \lambda$.

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

$$\Pr[\mathcal{Z}_p(k) \text{ is low}] = (1-p)^k \quad \text{and} \quad \Pr[\mathcal{Z}_p(k) \text{ is high}] = 1 - (1-p)^k.$$

Let us first show the following claim.

Lemma 6.7. *For all* $h, \ell, \xi, > 0$ *with* $\xi < 1$ *and* $\ell \leqslant (1-\xi)h$ *there exists* $0 < p < 1$ *such that* $\mathcal{Z}_p$ *is an* (h, ℓ)*-counter with error probability* $1/2 - \xi/8$.

Proof. We need to choose p such that (i) $1-(1-p)^{(1-\xi)h} \leqslant 1/2-\xi/8$, or equivalently, $1/2+\xi/8 \leqslant (1-p)^{(1-\xi)h}$, and (ii) $(1-p)^h \leqslant 1/2-\xi/8$, or equivalently, $(1-p)^{(1-\xi)h} \leqslant (1/2-\xi/8)^{1-\xi}$. It suffices to show

$$\frac{1}{2}+\frac{\xi}{8} \leqslant \left(\frac{1}{2}-\frac{\xi}{8}\right)^{1-\xi}, \tag{6.1}$$

then one can pick $p = 1-(1/2-\xi/8)^{1/h}$. Note that (ii) holds automatically for this value of p. Taking logarithms shows that (6.1) is equivalent to $\ln(4+\xi)-\ln 8 \leqslant (1-\xi)\cdot(\ln(4-\xi)-\ln 8)$, and by rearranging we obtain $\ln(4+\xi) \leqslant \ln(4-\xi)+\xi(\ln 8-\ln(4-\xi))$. Since $\ln 8-\ln(4-\xi) \geqslant \ln 8-\ln 4 = \ln 2$, it suffices to prove

$$\ln(4+\xi) \leqslant \ln(4-\xi)+\xi\ln 2. \tag{6.2}$$

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

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

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

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

This proves (6.2) and hence (6.1), and thus the statement. □

Proposition 6.8. *For all* $h, \ell, \xi > 0$ *and* $0 < \lambda' < 1/2$ *with* $\ell \leqslant (1-\xi)h$ *there exists an* (h,ℓ)*-counter* $\mathcal{Z}$ *with error probability* λ' *which uses* $O(\log\log(1/\lambda')+\log(1/\xi))$ *bits.*

Proof. We apply Lemma 6.2 to Lemma 6.7 with $\lambda = 1/2-\xi/8$, which states that we need to run $k = O(\log(\frac{1}{\lambda'})\cdot\frac{1}{\xi^2})$ copies to reduce the error probability to λ'. The states of $\mathcal{Z}_p^{(k)}$ are multisets over $\{0,1\}$ of size k, which can be encoded by $O(\log k) = O(\log\log\frac{1}{\lambda'}+\log\frac{1}{\xi})$ bits by specifying the number of 1-bits in the multiset. □

6.4 Suffix-free languages

In this section we prove Theorem 6.5(3). Since languages in **ST** ∪ **Len** have constant space SW-algorithms it suffices to show:

Theorem 6.9. *If* L *is regular and suffix-free then* $F_L^r(n) = O(\log\log n)$.

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

$$\ell_w(q) = \inf\{k \in \mathbb{N} \mid \mathrm{last}_k(w) \cdot q = q_F\}, \tag{6.3}$$

where we set $\inf(\emptyset) = \infty$. Notice that $\mathrm{last}_n(w) \in L$ if and only if $\ell_w(q_0) = n$. A *deterministic* streaming algorithm can maintain the value ℓ_w where $w \in \Sigma^*$ is the stream prefix read so far: If a symbol $a \in \Sigma$ is read, we can determine

$$\ell_{wa}(q) = \begin{cases} 0, & \text{if } q = q_F, \\ 1 + \ell_w(a \cdot q), & \text{otherwise,} \end{cases} \tag{6.4}$$

where $1 + \infty = \infty$. Since this would require storing $O(\log n)$ bit numbers we use probabilistic counters instead.

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

Lemma 6.10 (Threshold counting). *There exists a randomized streaming algorithm $\mathcal{T}_n$ with $O(1)$ bits such that for all $w \in \Sigma^*$ we have:*

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

Proof. By Proposition 6.8 there is a $(2n, n)$-counter $\mathcal{Z}$ with error probability 1/3 which uses $O(1)$ space. Let C be its state space and let $c_\infty \in C$ be an arbitrary high state. The algorithm $\mathcal{T}_n$ maintains a random function $c \colon Q \to C$ such that on input stream $w \in \Sigma^*$ we have

$$c(q) = \begin{cases} \mathcal{Z}(\ell_w(q)), & \ell_w(q) < \infty, \\ c_\infty, & \ell_w(q) = \infty. \end{cases}$$

Initially we set the states accordingly, i.e. for every state $q \in Q$, if $\ell_\varepsilon(q) < \infty$ then we initialize an instance of $\mathcal{Z}$ in $c(q)$ and increment it $\ell_\varepsilon(q)$ times, and otherwise we set $c(q) = c_\infty$. Given an input symbol $a \in \Sigma$, we compute the new function c' from c. First we set $c'(q_F) = c_\infty$, since no nonempty run from q_F is accepting. For all $q \in Q \setminus \{q_F\}$ we set $c'(q) = c(a \cdot q) \cdot \mathtt{inc}$ since

$$c'(q) = c(a \cdot q) \cdot \mathtt{inc} = \mathcal{Z}(1 + \ell_w(a \cdot q)) = \mathcal{Z}(\ell_{wa}(q)).$$

The algorithm accepts if and only if $c(q_0)$ is low. Correctness follows from the properties of Proposition 6.8. □

Lemma 6.11 (Modulo counting). *There exists a randomized streaming algorithm $\mathcal{M}_n$ with $O(\log\log n)$ bits such that for all $w \in \Sigma^*$ we have:*

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

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

Proof. Let p_i be the i-th prime number and let $s(m)$ be the product of all prime numbers $\leqslant m$. It is known that $\ln(s(m)) > m \cdot (1 - 1/\ln m)$ for $m \geqslant 41$ [101, p. 3.16] and $p_i < i \cdot (\ln i + \ln \ln i)$ for $i \geqslant 6$ [101, p. 3.13]. Let k be the first natural number such that $\prod_{i=1}^{k} p_i \geqslant n$. By the above bounds we get $k = O(\log n)$ and $p_{3k} = O(\log n \cdot \log \log n)$. The algorithm $\mathcal{M}_n$ initially picks a random prime $p \in \{p_1, \dots, p_{3k}\}$, which is stored throughout the run using $O(\log \log n)$ bits. Then, after reading $w \in \Sigma^*$, $\mathcal{M}_n$ stores for every $q \in Q$ a bit telling whether $\ell_w(q) < \infty$ and, if the latter holds, the values $\ell_w(q) \bmod p$ using $O(|Q| \cdot \log \log n)$ bits. This can be maintained according to (6.4). The algorithm accepts if and only if $\ell_w(q_0) \equiv n \bmod p$.

If $\ell_w(q_0) = n$ then the algorithm always accepts. Now assume $\ell_w(q_0) < 2n$ and $\ell_w(q_0) < n$. Since $-n \leqslant \ell_w(q_0) - n \leqslant n$ and any product of at least $k+1$ pairwise distinct primes exceeds n, the number $\ell_w(q_0) - n \neq 0$ has at most k prime factors. Therefore, $\mathcal{M}_n$ rejects with probability at least 2/3. $\square$

By combining both algorithms above we can prove Theorem 6.9. The algorithm $\mathcal{P}_n$ is the conjunction of the threshold algorithm $\mathcal{T}_n$ and the modulo counting algorithm $\mathcal{M}_n$. Recall that $\text{last}_n(w) \in L$ if and only if $\ell_w(q_0) = n$. If $\ell_w(q_0) = n$ then both algorithms accept with probability 2/3. If $\ell_w(q_0) \neq n$ then either $\mathcal{M}_n$ or $\mathcal{T}_n$ rejects with probability 2/3.

6.5 Lower bounds with two-sided error

In this section, we prove the lower bounds from Theorem 6.5. Point (2) from Theorem 6.5 follows easily from the relation $F_L(n) = 2^{O(F^r_L(n))}$ (Lemma 6.4). Since every language $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$ satisfies $F_L(n) = \Omega(\log n)$ infinitely often by Section 4.2.4 it also satisfies $F^r_L(n) = \Omega(\log \log n)$ infinitely often.

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

6.5.1 Communication complexity

We present the necessary background from communication complexity; see [82] for a detailed introduction. We only need the one-way setting where Alice sends a single message to Bob. Consider a function $f\colon X \times Y \to \{0,1\}$ for some finite sets X and Y. A *randomized one-way (communication) protocol* $P = (a, b)$ consists of functions $a\colon X \times R_a \to \{0,1\}^*$ and $b\colon \{0,1\}^* \times Y \times R_b \to \{0,1\}$, where R_a and R_b are finite sets of random choices of Alice and Bob, respectively. The *cost* of P is the maximum number of bits transmitted by Alice, i.e.

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

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

$$\Pr_{r_a \in R_a, r_b \in R_b} [P(x, y) \neq f(x, y)] \leqslant \frac{1}{3}. \tag{6.5}$$

where $P(x, y)$ is the random variable $b(a(x, r_a), y, r_b)$. The *randomized one-way communication complexity* $C(f)$ of f is the minimal cost among all one-way randomized protocols that compute f (with an arbitrary number of random bits). The choice of the constant 1/3 in (6.5) is arbitrary in the sense that changing the constant to any $\lambda < 1/2$ only changes the cost $C(f)$ by a fixed constant (depending on λ), see [82, p. 30]. We will use established lower bounds on the following functions:

Theorem 6.12 ([81, Theorem 3.7 and 3.8]). *Let* $n \in \mathbb{N}$.

- *The* index function

$$\begin{aligned} \mathrm{IDX}_n \colon \{0,1\}^n \times \{1, \dots, n\} &\to \{0,1\} \\ (a_1 \cdots a_n, i) &\mapsto a_i. \end{aligned}$$

  has randomized one-way communication complexity $\Theta(n)$.

- *The* greater-than function

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

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

The bounds above also hold for the deterministic one-way communication complexity as witnessed by the trivial deterministic protocols. We also define the *equality function* $\mathrm{EQ}_n \colon \{1, \dots, n\}^2 \to \{0,1\}$ by $\mathrm{EQ}_n(i, j) = 1$ if and only if $i = j$. Its randomized one-way communication complexity is $\Theta(\log \log n)$ whereas its deterministic one-way communication complexity is $\Theta(\log n)$ [82].

6.5.2 Linear lower bound

We start with the proof of (6) from Theorem 6.5, which extends our linear space lower bound from the deterministic setting to the randomized setting.

Proposition 6.13. *If* $L \in \mathbf{Reg} \setminus \langle \mathbf{LI}, \mathbf{Len} \rangle$ *then* $F^r_L(n) = \Omega(n)$ *infinitely often.*

Proof. By Theorem 4.24 any rDFA for L is not well-behaved and by Lemma 4.7 there exist words $u = u_1u_2, v = v_1v_2, z \in \Sigma^*$ such that $|u_1| = |v_1|$, $|u_2| = |v_2|$ and L separates $u_2\{u, v\}^*z$ and $v_2\{u, v\}^*z$. Let $\eta \colon \{0,1\}^* \to \{u, v\}^*$ be the homomorphism defined by $\eta(0) = u$ and $\eta(1) = v$.

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

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

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

where $s = u_2$ if $\alpha_i = 0$ and $s = v_2$ if $\alpha_i = 1$. Hence from the output of $\mathcal{P}_n$ Bob can determine whether $\alpha_i = 1$. The cost of the protocol P_m is bounded by $O(s(\mathcal{P}_n))$ and must be at least $\Omega(m) = \Omega(n)$ by Theorem 6.12 we conclude that $s(\mathcal{P}_n) = \Omega(n)$. Therefore $F_L^r(n) = \Omega(n)$ infinitely often. □

6.5.3 Logarithmic lower bound

Next we prove point (4) from Theorem 6.5. For that, we need the following automaton property. In the following let $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ be an rDFA.

A pair $(p, q) \in Q \times Q$ of states is called *synchronized* if there exist words $x, y, z \in \Sigma^*$ with $|x| = |y| = |z| \geqslant 1$ such that $q \xleftarrow{x} q$, $q \xleftarrow{y} p$, $p \xleftarrow{z} p$. A pair (p, q) is called *reachable* if p and q are reachable from q_0. A state pair (p, q) is called F-*consistent* if either $\{p, q\} \cap F = \emptyset$. or $\{p, q\} \subseteq F$. We remark that synchronized state pairs have no connection to the notion of synchronizing words.

Lemma 6.14. *A state pair (p, q) is synchronized if and only if p and q are nontransient and there exists a nonempty run $q \xleftarrow{y} p$ whose length is divided by $|Q|!$.*

Proof. Let $x, y, z \in \Sigma^+$ with $|x| = |y| = |z| = k$ such that $q \xleftarrow{x} q$, $q \xleftarrow{y} p$, $p \xleftarrow{z} p$. Then p and q are nontransient and we have $q \xleftarrow{x^{|Q|!-1}y} p$ where $x^{|Q|!-1}y$ has length $(|Q|! - 1) \cdot k + k = |Q|! \cdot k$.

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

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

Lemma 6.15. *Assume that every reachable synchronized pair in $\mathcal{B}$ is F-consistent. Then for every word $v \in \Sigma^*$ of length at least $|Q|! \cdot (|T|+1)$ there exists a $|Q|!$-periodic function $\beta_v \colon \mathbb{N} \to \{0, 1\}$ such that the following holds: If $w \in \Sigma^* v$ and $w \cdot q_0 \in N$, then we have $w \in L$ iff $\beta(|w|) = 1$.*

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

$$q_k \xleftarrow{a_k} \cdots \xleftarrow{a_2} q_1 \xleftarrow{a_1} q_0 \tag{6.6}$$

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

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

is nontransient because otherwise the set would contain $|T| + 1$ pairwise distinct transient states. Furthermore, we claim that the nontransient states in Q_i are either all final or all nonfinal: Take two nontransient states $q_{i+j_1|Q|!}$ and $q_{i+j_2|Q|!}$ with $j_1 < j_2$. Since we have a run of length $(j_2 - j_1)|Q|!$ from $q_{i+j_1|Q|!}$ to $q_{i+j_2|Q|!}$, the states form a synchronized pair by Lemma 6.14. Hence, by assumption the two states are F-consistent.

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

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

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

Let $w = a_m \cdots a_2 a_1 \in \Sigma^* v$ be a word of length $m \geqslant k$. The initial run of $\mathcal{B}$ on w prolongs the run in (6.6):

$$q_m \xleftarrow{a_m} \cdots \xleftarrow{a_2} q_1 \xleftarrow{a_1} q_0$$

Assume that $q_m \in N$. As argued above, there is a position $0 \leqslant i \leqslant k$ such that $i \equiv m \pmod{|Q|!}$ and $q_i \in N$. Hence (q_i, q_m) is a synchronized pair by Lemma 6.14, which is F-consistent by assumption. Therefore $w \in L$ iff $q_m \in F$ iff $q_i \in F$ iff $\beta_v(|w|) = 1$. □

Lemma 6.16. *Assume that every reachable synchronized pair in $\mathcal{B}$ is F-consistent. Then $L(\mathcal{B})$ belongs to $\langle$**ST**, **SF**, **Len**$\rangle$.*

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

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

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

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

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

The first part $L(\mathcal{B}, F_N) \cap \Sigma^{\leqslant k-1}$ is finite and thus suffix testable. To finish the proof, we will show that $L(\mathcal{B}, F_N) \cap \Sigma^* v \in \langle$**ST**, **SF**, **Len**$\rangle$ for each $v \in \Sigma^k$. Let $v \in \Sigma^k$ and let $\beta_v \colon \mathbb{N} \to \{0, 1\}$ be the $|Q|!$-periodic function from Lemma 6.15.

The lemma implies that

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

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

The following lemma is an immediate consequence of Lemma 6.16.

Lemma 6.17. *If* $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$ *then there exist* $u, x, y, z \in \Sigma^*$ *with* $|x| = |y| = |z| \geqslant 1$ *such that* L *separates* $u^* x y^* z$ *and* $y^* z$*.*

Proposition 6.18. *If* $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$ *then* $F^r_L(n) = \Omega(\log n)$ *infinitely often.*

Proof. Consider the words $u, x, y, z \in \Sigma^*$ described in Lemma 6.17. Let $n = |y| \cdot m + |z|$ for some $m \geqslant 1$ and let $\mathcal{P}_n$ be a randomized SW-algorithm for L. We describe a randomized one-way protocol for GT_m: Let $1 \leqslant i \leqslant m$ be the input of Alice and $1 \leqslant j \leqslant m$ be the input of Bob. Alice starts reads $u^m x y^{m-i}$ into $\mathcal{P}_n$ and she sends the reached state to Bob using $O(s(\mathcal{P}_n))$ bits. Bob then continues the run of $\mathcal{P}_n$ from the transmitted state with the word $y^j z$. Hence $\mathcal{P}_n$ is simulated on the word $w := u^m x y^{m-i} y^j z = u^m x y^{m-i+j} z$. We have

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

By Lemma 6.17, $\mathrm{last}_n(w)$ belongs to L in exactly one of the two cases $i > j$ and $i \leqslant j$. Hence Bob can distinguish these two cases with probability at least 2/3. It follows that the protocol computes GT_m and its cost is bounded by $s(\mathcal{P}_n)$. By Theorem 6.12 we can conclude that $s(\mathcal{P}_n) = \Omega(\log m) = \Omega(\log n)$, and therefore $F^r_L(n) = \Omega(\log n)$ infinitely often. □

6.6 Lower bounds in the variable-size model

In the following we look at randomized algorithms in the variable-size model. First we transfer the definitions from Section 3.3 randomized setting in a straightforward way. A *variable-size sliding window algorithm* $\mathcal{P}$ for $\varphi \colon \Sigma^* \to Y$ is a randomized streaming algorithm for $\mathrm{SW}(\varphi)$. Its *space complexity* is $v(\mathcal{P}, n) = |\log M_{\leqslant n}| \in \mathbb{N} \cup \{\infty\}$ where $M_{\leqslant n}$ contains all memory states in $\mathcal{P}$ which are reachable with positive probability in $\mathcal{P}$ on inputs $w \in \Sigma^*_{\downarrow}$ with $\mathrm{mwl}(w) \leqslant n$. Since the variable-size sliding window model subsumes the fixed-size model we have $F^r_\varphi(n) \leqslant v(\mathcal{P}, n)$ for every randomized variable-size sliding window algorithm $\mathcal{P}$ for φ.

Again we raise the question if randomness can improve the space complexity in the variable-size model. We claim that, in contrast to the fixed-size model, here the space complexity is unaltered. First, the upper bounds are inherited from the deterministic setting, i.e. languages in $\langle \mathbf{LI}, \mathbf{Len} \rangle$ have $O(\log n)$ space complexity, and trivial languages have $O(1)$ space complexity. For every regular language L which is not contained in $\langle \mathbf{LI}, \mathbf{Len} \rangle$ we proved a linear lower bound on $F^r_L(n)$ (Proposition 6.13), which is also a lower bound on the space complexity of any randomized variable-size sliding window algorithm for L. It remains to look at nontrivial languages, for which we have proved a logarithmic lower bound in the deterministic setting (Lemma 3.7).

Lemma 6.19. *If $\mathcal{P}$ is a randomized variable-size SW-algorithm for a nontrivial problem then* $v(\mathcal{P}, n) = \Omega(\log n)$.

Proof. Let $\varphi\colon \Sigma^* \to Y$ be a nontrivial problem. Hence there is a length-minimal word $a_1 \cdots a_k \in \Sigma^*$ such that $\varphi(\varepsilon) \neq \varphi(a_1 \cdots a_k)$. By minimality we have $\varphi(a_1 \cdots a_k) \neq \varphi(a_2 \cdots a_k)$. Let $\mathcal{P}$ be a randomized variable-size SW-algorithm for φ. By Lemma 6.2 we can assume that the error probability of $\mathcal{P}$ is at most $1/6$, which increases its space complexity $v(\mathcal{P}, n)$ by a constant factor.

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

$$y_1 = \varphi(\mathrm{wnd}(a_1^i \downarrow^j a_2 \cdots a_k)) \text{ and } y_2 = \varphi(\mathrm{wnd}(a_1^i \downarrow^{j+1} a_1 \cdots a_k)).$$

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

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

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

and hence Bob returns true. If $i \leqslant j$ then

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

and

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

By assumption these values are distinct and therefore Bob returns false. □

6.7 Lower bounds with one-sided error

So far, we have only considered randomized SW-algorithms with a two-sided error (analogously to the complexity class BPP). Randomized SW-algorithms with a one-sided error (analogously to the classes RP and coRP) can be motivated by applications, where all "yes"-outputs or all "no"-outputs, respectively, have to be correct. We distinguish between true-biased and false-biased algorithms.

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

- If $w \in L$ then $\Pr[\mathcal{P}$ accepts $w] \geqslant 2/3$.
- If $w \notin L$ then $\Pr[\mathcal{P}$ rejects $w] = 1$.

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

- If $w \in L$ then $\Pr[\mathcal{P}$ accepts $w] = 1$.
- If $w \notin L$ then $\Pr[\mathcal{P}$ rejects $w] \geqslant 2/3$.

Let $F_L^0(n)$ (and $F_L^1(n)$) be the minimal space complexity $s(\mathcal{P}_n)$ of any true-biased (false-biased) streaming algorithm $\mathcal{P}_n$ for L and window size n. Since algorithms with one-sided error are stronger than algorithms with two-sided error, but weaker than deterministic algorithms we have the relations $F_L^r(n) \leqslant F_L^i(n) \leqslant F_L(n)$ for $i \in \{0,1\}$, and $F_L^0(n) = F^1_{\Sigma^* \setminus L}(n)$.

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

Theorem 6.20 (One-sided error). *For every regular language L we have:*

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

(2) If $L \notin \langle \mathbf{ST}, \mathbf{Len} \rangle$ then $F_L^0(n)$ and $F_L^1(n)$ are $\Omega(\log n)$ infinitely often.

(3) If $L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$ then $F_L^0(n)$ and $F_L^1(n)$ are $O(\log n)$.

(4) If $L \notin \langle \mathbf{LI}, \mathbf{Len} \rangle$ then $F_L^0(n)$ and $F_L^1(n)$ are $\Omega(n)$ infinitely often.

The upper bounds in (1) and (3) already hold for deterministic SW-algorithms (Theorem 4.35). Moreover, the lower bound in (4) already holds for SW-algorithms with two-sided error (Theorem 6.5(5)). It remains to prove point (2) of the theorem. In fact we show that any *nondeterministic* SW-algorithm for a regular language $L \notin \langle \mathbf{ST}, \mathbf{Len} \rangle$ requires $\Omega(\log n)$ space. A nondeterministic SW-algorithm for a language L and window size n is an NFA $\mathcal{P}_n$ with $L(\mathcal{P}_n) = \mathrm{SW}_n(L)$, and its space complexity is $s(\mathcal{P}_n) = \log |\mathcal{P}_n|$. If we have a true-biased randomized SW-algorithm for L we can turn it into a nondeterministic SW-algorithm by keeping only those transitions with positive probabilities and making all states q initial which have a positive initial probability $\iota(q) > 0$. Therefore, it suffices to show the following statement:

Proposition 6.21. *Let $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$. Then for infinitely many n every nondeterministic SW-algorithm $\mathcal{P}_n$ for L has $\Omega(\sqrt{n})$ many states.*

For the proof of Proposition 6.21 we need the following lemma.

Lemma 6.22. *Let $L \subseteq a^*$ and $n \in \mathbb{N}$ such that L separates $\{a^n\}$ and $\{a^k \mid k > n\}$. Then every NFA for L has at least $\sqrt{n}$ many states.*

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

Now assume $a^n \notin L$ and $a^k \in L$ for all $k > n$. The proof is essentially the same as for [70, Lemma 6], where the statement of the lemma is shown for $L = a^* \setminus \{a^n\}$. Let us give the proof for completeness. It is known that every unary NFA has an equivalent NFA in so-called Chrobak normal form. A unary NFA in Chrobak normal form consists of path starting in the unique initial state. From the last state of the path, edges go to a collection of disjoint cycles. In [53] it is shown that an m-state unary NFA has an equivalent NFA in Chrobak normal form whose initial path consists of $m^2 - m$ states. Now assume that L is accepted by an NFA with m states and let $\mathcal{A}$ be the equivalent Chrobak normal form NFA, whose initial path consists of $m^2 - m$ states. If $n \geqslant m^2 - m$ then all states that are reached in $\mathcal{A}$ from the initial state via a^n belong to a cycle and every cycle contains such a state. Since $a^n \notin L$, all these states are rejecting. Hence, $a^{n+x\cdot d} \notin L$ for all $x \geqslant 0$, where d is the product of all cycle lengths. This contradicts the fact that $a^k \in L$ for all $k > n$. Hence, we must have $n < m^2 - m$ and therefore $m > \sqrt{n}$. □

Proof of Proposition 6.21. Let $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$. By Theorem 4.19 and the results from Section 4.1.2 there are words $x, y, z \in \Sigma^*$ such that $|x| = |y|$ and L separates xy^*z and y^*z. Note that we must have $x \neq y$.

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

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

It recognizes the language $L(\mathcal{A}) = \{a^k \mid xy^k z \in \mathrm{SW}_n(L)\}$, and therefore $L(\mathcal{A})$ separates $\{a^m\}$ and $\{a^k \mid k > m\}$. By Lemma 6.22, $\mathcal{A}$ and thus $\mathcal{P}_n$ has at least $\sqrt{m} = \Omega(\sqrt{n})$ many states. □

This proves $F^0_L(n) = \Omega(\log n)$ infinitely often for $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$. Since $\mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$ is closed under complement this also implies a $\Omega(\log n)$ lower bound on $F^1_L(n)$ for all $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{Len} \rangle$.

6.8 Conclusion

We proved that most of the space lower bounds for deterministic sliding window algorithms for regular languages also hold for randomized algorithms. The only exception are the languages in $\langle \mathbf{SF}, \mathbf{ST}, \mathbf{Len} \rangle$, which have deterministic space complexity $O(\log n)$ but randomized space complexity $O(\log \log n)$. The situation is similar to the one-way communication complexity of index ($\Theta(n)$), greater-than ($\Theta(\log n)$), and equality function (deterministic $\Theta(\log n)$ and randomized $\Theta(\log \log n)$).

Uniform setting One can again consider the setting where the language $L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$ is given as an automaton. In Theorem 4.42 we showed that an exponential dependence of the automaton size is unavoidable. We conjecture that the same holds for randomized algorithms.

Streaming algorithms with small failure ratio In [G3] we proposed a more relaxed definition of correctness of randomized sliding window algorithms. A randomized SW-algorithm is said to have *failure ratio* ϕ if the portion of all time instants where the algorithm gives a correct answer with probability $> 1/3$ is at most ϕ. Let us only summarize the results.

Theorem 6.23. *For every regular language $L \subseteq \Sigma^*$ we have:*

- *If $L \in \langle \mathbf{LI}, \mathbf{PF}, \mathbf{Len} \rangle$ and $0 < \phi \leqslant 1$ then L has a randomized SW-algorithm which has failure ratio ϕ and uses $O(1)$ space.*
- *If $L \notin \langle \mathbf{LI}, \mathbf{PF}, \mathbf{Len} \rangle$ then there exist $0 < \phi \leqslant 1$ and infinitely many window sizes n for which any randomized SW-algorithm for L uses $\Omega(n)$ space.*

Theorem 6.24. *For every regular language $L \subseteq \Sigma^*$ we have:*

- *If $L \in \langle \mathbf{LB}, \mathbf{PF}, \mathbf{SF}, \mathbf{Len} \rangle$ and $0 < \phi \leqslant 1$ then L has a deterministic SW-algorithm which has failure ratio ϕ and uses $O(\log n)$ space.*
- *If $L \notin \langle \mathbf{LB}, \mathbf{PF}, \mathbf{SF}, \mathbf{Len} \rangle$ then there exist $0 < \phi \leqslant 1$ and infinitely many window sizes n for which any deterministic SW-algorithm for L uses $\Omega(\log n)$ space.*
- *If $L \in \langle \mathbf{LI}, \mathbf{PF}, \mathbf{Len} \rangle$ and $0 < \phi \leqslant 1$ then L has a deterministic SW-algorithm which has failure ratio ϕ and uses $O(\log n)$ space.*
- *If $L \notin \langle \mathbf{LI}, \mathbf{PF}, \mathbf{Len} \rangle$ then there exist $0 < \phi \leqslant 1$ and infinitely many window sizes n for which any deterministic SW-algorithm for L uses $\Omega(n)$ space.*

Here **PF** denotes the class of regular *prefix-free* languages, and **LB** denotes the class of all languages $\Sigma^* L$ where L is a regular language which is both prefix-free and suffix-free.

Chapter 7

Sliding window property testing

7.1 Introduction

The results presented so far show that simple languages as $a\{a,b\}^*$ require linear space in the sliding window model, even for randomized algorithms. This gives the motivation to seek for alternative approaches in order to achieve efficient algorithms for all regular languages. We take our inspiration from the property testing model introduced by Goldreich et al. [64]. In this model, the task is to decide (with high probability) whether the input has a particular property P, or is “far” from any input satisfying it. The distance measure on words that is commonly used is the *Hamming distance*, which counts the positions at which two words differ. For a function $\gamma(n)$, we say that a word w of length n is $\gamma(n)$-far from satisfying P, if the Hamming distance between w and any word w' satisfying P is at least $\gamma(n)$. We will call the function $\gamma(n)$ the Hamming gap of the tester. We must make the decision by querying as few symbols of the input as possible, and the query complexity of the algorithm is defined to be equal to the number of inspected symbols. The motivation is that when working with large-scale data, accessing a data item is a very time-expensive operation. The membership problem for a regular language in the property testing model was studied by Alon et al. [4] who showed that for every regular language L and every $\epsilon > 0$, there is a property tester with Hamming gap $\gamma(n) = \epsilon n$ and two-sided error for deciding membership in L that makes only a constant number of queries.

In this chapter we introduce a class of algorithms called *sliding window (property) testers*, which must accept if the active window has the property P and reject if it is far from satisfying P. We consider deterministic sliding window property testers and randomized sliding window property testers with one-sided and two-sided error. A similar but simpler model of streaming property testers, where the whole stream is considered, was introduced by Feigenbaum et al. [44].

François et al. [52] continued the study of this model in the context of language membership problems and came up with a streaming property tester for visibly pushdown languages that uses polylogarithmic space.

While at first sight the only connection between property testers and sliding window property testers is that we must accept the input if it satisfies P and reject if it is far from satisfying P, there is, in fact, a deeper link. In particular, the above mentioned result of Alon et al. [4] combined with an optimal sampling algorithm for sliding windows [23], immediately yields a $O(\log n)$-space, two-sided error sliding window property tester with Hamming gap $\gamma(n) = \epsilon n$ for every regular language. We will improve on this observation. Our main contribution are tight complexity bounds for deterministic and randomized sliding window property testers for regular languages.

The results of this chapter appeared in [G8].

7.2 Sliding window testers for regular languages

The *Hamming distance* between two words $u = a_1 \cdots a_n$ and $v = b_1 \cdots b_n$ of equal length is the number of positions where u and v differ, i.e. $\text{dist}(u, v) = |\{i \mid a_i \neq b_i\}|$. If $|u| \neq |v|$ we set $\text{dist}(u, v) = \infty$. The distance of a word u to a language L is defined as $\text{dist}(u, L) = \inf\{\text{dist}(u, v) \mid v \in L\} \in \mathbb{N} \cup \{\infty\}$. In this chapter $\gamma \colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ is always a function with $\gamma(n) \leqslant n$ for all $n \in \mathbb{N}$.

Sliding window testers with one- and two-sided error A *deterministic sliding window (property) tester* for a language $L \subseteq \Sigma^*$ and window size n with Hamming gap $\gamma(n)$ is a deterministic streaming algorithm $\mathcal{P}_n$ over the alphabet Σ with the following properties:

- If $\text{last}_n(w) \in L$, then $\mathcal{P}_n$ accepts w.
- If $\text{dist}(\text{last}_n(w), L) > \gamma(n)$, then $\mathcal{P}_n$ rejects w.

A *randomized sliding window tester* for a language $L \subseteq \Sigma^*$ and window size n with Hamming gap $\gamma(n)$ is a randomized streaming algorithm $\mathcal{P}_n$ over the alphabet Σ with the following properties. It has *two-sided error* if for every input stream $w \in \Sigma^*$ we have:

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

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

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

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

- If $\text{last}_n(w) \in L$, then $\Pr[\mathcal{P}_n \text{ accepts } w] = 1$.

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

We remark that all known property testers with one-sided error are false-biased, cf. [4, 52, 64]. Again, the success probability 2/3 is an arbitrary choice in light of Lemma 6.2. The case of Hamming gap $\gamma(n) = 0$ corresponds to exact membership testing to L from the previous chapters. We will focus on the two cases $\gamma(n) = O(1)$ and $\gamma(n) = \epsilon n$ for some $\epsilon > 0$.

Main results Our first main result is a deterministic logspace sliding window tester for every regular language.

Theorem 7.1. *For every regular language* L *there exists a deterministic sliding window tester for* L *with constant Hamming gap which uses* $O(\log n)$ *space.*

Again we use the path summary algorithm for (possibly not well-behaved) rDFAs. If the path summary of a run is accepting then the run itself might not be accepting but it can be made accepting by modifying a short prefix.

Theorem 7.1 cannot be improved whenever L is a *nontrivial* regular language. A language is $L \subseteq \Sigma^*$ is $\gamma(n)$-*trivial* if there exists a number n_0 such that for all $n \geqslant n_0$ with $L \cap \Sigma^n \neq \emptyset$ and all $w \in \Sigma^n$ we have $\mathrm{dist}(w, L) \leqslant \gamma(n)$. Hence for large enough n, whenever $L \cap \Sigma^n \neq \emptyset$, then there is a trivial sliding window tester, which always accepts, and has Hamming gap $\gamma(n)$. If L is $O(1)$-trivial we say that L is *trivial*. Note that Alon et al. [4] call a language L trivial if L is (ϵn)-trivial for all $\epsilon > 0$ according to our definition. In fact, we will prove that both definitions coincide for regular languages (Corollary 7.10).

Theorem 7.2. *For every nontrivial regular language* L *there exist* $\epsilon > 0$ *and infinitely many window sizes* $n \in \mathbb{N}$ *for which every true-biased sliding window tester with Hamming gap* ϵn *uses space* $\Omega(\log n)$.

Next we consider randomized sliding window property testers. With the help of Bernoulli counters we can construct a constant-space randomized sliding window property tester with two-sided error for any regular language.

Theorem 7.3. *For every regular language* L *and every* $\epsilon > 0$ *there exists a randomized sliding window tester for* L *with two-sided error and Hamming gap* ϵn *that uses space* $O(1/\epsilon)$.

While the randomized setting with two-sided error allows ultra-efficient testers, one could argue that allowing a two-sided error is a strong relaxation. To this end, we study the randomized setting with one-sided error. In this setting only unions of trivial and suffix-free languages admit sliding window testers working in space $o(\log n)$.

Theorem 7.4. *If* L *is a finite union of trivial regular languages and suffix-free regular languages then there exists a false-biased randomized sliding window tester for* L *with constant Hamming gap which uses* $O(\log \log n)$ *space.*

Theorem 7.5. *Let* L *be a regular language.*

(1) *If* L *is not a finite union of trivial regular languages and suffix-free regular languages there exist* $\epsilon > 0$ *and infinitely many window sizes* n *for which every false-biased sliding window tester for* L *with Hamming gap* ϵn *uses space* $\Omega(\log n)$.

(2) *If* L *is nontrivial then there exist* $\epsilon > 0$ *and infinitely many window sizes* n *for which every false-biased sliding window tester for* L *with Hamming gap* ϵn *uses space* $\Omega(\log \log n)$.

(Co-)nondeterministic sliding window algorithms The lower bounds from Theorem 7.2 and Theorem 7.5 will in fact be proven for (co-)nondeterministic algorithms, which include algorithms with one-sided error and deterministic algorithms. A *co-nondeterministic finite automaton (coNFA)* $\mathcal{P}$ has the same format as an NFA but accepts a word w if and only if all initial runs of $\mathcal{P}$ on w are accepting. A *(co-)nondeterministic sliding window tester* $\mathcal{P}_n$ for a language L with Hamming gap $\gamma(n)$ is an (co-)NFA $\mathcal{P}_n$ which accepts a stream w if $\mathrm{last}_n(w) \in L$, and rejects w if $\mathrm{dist}(\mathrm{last}_n(w), L) > \gamma(n)$. Its space complexity is $s(\mathcal{P}_n) = \log |\mathcal{P}_n|$. Every true-biased sliding window tester can be turned into a nondeterministic sliding window tester; every false-biased sliding window tester can be turned into a co-nondeterministic one.

7.3 Trivial languages

Let us start by analyzing trivial regular languages. The reason we introduce trivial languages the way we do (and a justification to call them "trivial") is stated in the following theorem:

Theorem 7.6. *If* L *is a trivial language (not necessarily regular), then for every window size* n *there is a deterministic sliding window tester for* L *with constant Hamming gap which uses constant space. The converse is also true: Let* L *be a language which has a deterministic constant-space sliding window tester with Hamming gap* $\gamma(n)$ *for every window size* n*. Then there exists a constant* c *such that* L *is* $(\gamma(n) + c)$*-trivial.*

Proof. Assume first that L is trivial. Let $n \in \mathbb{N}$ be a window size. If $L \cap \Sigma^n = \emptyset$, then the algorithm always rejects, which is obviously correct since any active window of length n has infinite Hamming distance to L. Otherwise, the algorithm always accepts. In this case, we use the fact that L is trivial, i.e. there is a constant c such that the Hamming distance between an arbitrary active window of length n and L is at most c.

We now show the converse statement. For each window size $n \in \mathbb{N}$ let $\mathcal{P}_n$ be a deterministic sliding window tester for L with Hamming gap $\gamma(n)$ with a constant number of states, say s states. Let $N \subseteq \mathbb{N}$ be the set of all n such that $L \cap \Sigma^n \neq \emptyset$. Note that every $\mathcal{P}_n$ with $n \in N$ accepts a nonempty language. The number of deterministic sliding window testers (DFAs) with at most s states over the input alphabet Σ is bounded by a fixed constant d (up to isomorphism).

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

In the rest of the section we show that every nontrivial regular language L is already not ϵn-trivial for some $\epsilon > 0$. For this we first show some auxiliary results that will be also used in Section 7.5. Given $i, j \geqslant 0$ and a word w of length at least $i + j$ we define $\text{cut}_{i,j}(w) = y$ such that $w = xyz$, $|x| = i$ and $|z| = j$. If $|w| < i + j$, then $\text{cut}_{i,j}(w)$ is undefined. For a language L we define the *cut-language* $\text{cut}_{i,j}(L) = \{\text{cut}_{i,j}(w) \mid w \in L\}$.

Lemma 7.7. *If* L *is regular, then there are finitely many languages* $\text{cut}_{i,j}(L)$.

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

Lemma 7.8. *If* $\text{cut}_{i,j}(L)$ *is a length language for some* $i, j \geqslant 0$*, then* L *is trivial.*

Proof. Assume that $\text{cut}_{i,j}(L)$ is a length language. Let $n \in \mathbb{N}$ such that $L \cap \Sigma^n \neq \emptyset$ and $n \geqslant i + j$. We claim that $\text{dist}(w, L) \leqslant i + j$ for all $w \in \Sigma^n$. Let $w \in \Sigma^n$ and $w' \in L \cap \Sigma^n$. Then $\text{cut}_{i,j}(w') \in \text{cut}_{i,j}(L)$ and hence also $\text{cut}_{i,j}(w) \in \text{cut}_{i,j}(L)$. Therefore there exist $x \in \Sigma^i$ and $z \in \Sigma^j$ such that $x \, \text{cut}_{i,j}(w) \, z \in L$ satisfying $\text{dist}(w, x \, \text{cut}_{i,j}(w) \, z) \leqslant i + j$. □

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

Proposition 7.9. *Let* L *be regular. If* $\text{cut}_{i,j}(L)$ *is not a length language for all* $i, j \geqslant 0$*, then* L *has an infinite restriction* $L|_N$ *to an arithmetic progression* $N = d + e\mathbb{N}$ *which excludes a factor.*

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

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

Let u be an arbitrary word which contains for every remainder $0 \leqslant r \leqslant h-1$ an occurrence of y as a factor starting at a position which is congruent to r mod h. We claim that $L|_N$ excludes $a^i u a^j$ as a factor where a is an arbitrary symbol. Assume that there exists a word $w \in L|_N$ which contains $a^i u a^j$ as a factor. Then $\mathrm{cut}_{i,j}(w)$ contains u as a factor, has length $k + hn$ for some $n \geqslant 0$, and belongs to $\mathrm{cut}_{i,j}(L)$. Therefore $\mathrm{cut}_{i,j}(w)$ also contains h many occurrences of y, one per remainder $0 \leqslant r \leqslant h-1$. Consider the occurrence of y in $\mathrm{cut}_{i,j}(w)$ which starts at a position which is divisible by h, i.e. we can factorize $\mathrm{cut}_{i,j}(w) = xyz$ such that $|x|$ is a multiple of h. Since $\mathrm{cut}_{i,j}(w)$ has length $k + hn$ also $|z|$ is a multiple of h. Therefore $y \in \mathrm{cut}_{i+|x|,j+|z|}(L) = \mathrm{cut}_{i,j}(L)$, which is a contradiction. □

Corollary 7.10. *For every regular language* L*, the following statements are equivalent:*

(i) L *is trivial.*

(ii) L *is* ϵn*-trivial for every* $\epsilon > 0$.

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

Proof. If $\mathrm{cut}_{i,j}(L)$ is a length language then L is trivial by Lemma 7.8, and thus also ϵn-trivial for any $\epsilon > 0$. It remains to show the direction (ii) to (iii). If (iii) would not hold then some infinite restriction $L|_N$ of L excludes a factor w by Proposition 7.9. Hence if $n \in N$ and v is any word of length n, which contains at least $\lfloor n/|w| \rfloor$ many disjoint occurrences of w, then $\mathrm{dist}(v, L) \geqslant \lfloor n/|w| \rfloor$. □

Examples of trivial languages include all prefix testable and all suffix testable languages, and also the set of all words over $\{a, b\}$ which contain an even number of a's.

7.4 Upper bounds

For this section we fix a regular language L and an rDFA $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$. All presented sliding window testers are based on the path summary algorithm from Section 4.1.

7.4.1 Preliminary results

First observe that in sliding window property testing, space complexity is preserved under finite unions since $\mathrm{dist}(w, L_1 \cup L_2) = \min(\mathrm{dist}(w, L_1), \mathrm{dist}(w, L_2))$.

Lemma 7.11. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be randomized sliding window testers for L_1 and L_2, respectively, for window size n with Hamming gap $\gamma(n)$ and two-sided error. Then there exists a sliding window tester for $L_1 \cup L_2$ for window size n with Hamming gap $\gamma(n)$ and two-sided error using $O(s(\mathcal{P}_1) + s(\mathcal{P}_2))$ bits. The same holds for randomized algorithms with one-sided error and deterministic algorithms.*

Proof. Using Lemma 6.2 we reduce the error of $\mathcal{P}_1$ and $\mathcal{P}_2$ to $1/6$. Then we run both algorithms in parallel and accept if and only if one of them accepts.

If the window belongs to $L_1 \cup L_2$ then either $\mathcal{P}_1$ or $\mathcal{P}_2$ accepts with probability $5/6$. If the window w satisfies $\mathrm{dist}(w, L_1 \cup L_2) > \gamma(n)$ then $\mathrm{dist}(w, L_i) > \gamma(n)$ for both $i = 1, 2$. Hence both algorithms falsely accept with probability at most $1/6 + 1/6 = 1/3$. □

Strongly connected graphs With the rDFA $\mathcal{B}$ we associate the directed graph (Q, E) with edge set $E = \{(p, \delta(a, p)) \mid p \in Q, a \in \Sigma\}$. The *period* $g(G)$ of a directed graph G is the greatest common divisor of all cycle lengths in G. If G is acyclic we define the period to be ∞. We will apply the following lemma to the SCCs of $\mathcal{B}$.

Lemma 7.12 ([4]). *Let $G = (V, E)$ be a strongly connected directed graph with $E \neq \emptyset$ and finite period g. Then there exist a partition $V = \bigcup_{i=0}^{g-1} V_i$ and a constant $m(G) \leqslant 3|V|^2$ with the following properties:*

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

If $G = (V, E)$ is strongly connected with $E \neq \emptyset$ and finite period g, and $V_0, \dots, V_{g-1}$ satisfy the properties from Lemma 7.12, then we define the *shift* from $u \in V_i$ to $v \in V_j$ by

$$\mathrm{shift}(u, v) = (j - i) \bmod g \in \{0, \dots, g-1\}. \tag{7.1}$$

Notice that this definition is independent of the partition $\bigcup_{i=0}^{g-1} V_i$ since any path from u to v has length $\ell \equiv \mathrm{shift}(u, v) \pmod g$ by Lemma 7.12. Also note that $\mathrm{shift}(u, v) + \mathrm{shift}(v, u) \equiv 0 \pmod g$.

Lemma 7.13 (Uniform period). *There exists an rDFA $\mathcal{B}'$ for L and a number g such that every nontransient SCC C in $\mathcal{B}'$ has period $g(C) = g$.*

Proof. Let g be the product of all periods $g(C)$ over all nontransient SCCs C in $\mathcal{B}$. In the following we compute in the additive group $\mathbb{Z}_g = \{0, \dots, g-1\}$. We define

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

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

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

Clearly, $\mathcal{B} \times \mathbb{Z}_g$ is equivalent to $\mathcal{B}$. We show that every nontransient SCC of $\mathcal{B} \times \mathbb{Z}_g$ has period g. The nontransient SCCs of $\mathcal{B} \times \mathbb{Z}_g$ are the sets $C \times \mathbb{Z}_g$, where C is a nontransient SCC of $\mathcal{B}$. Let C be a nontransient SCC of $\mathcal{B}$. Clearly, every cycle length in $C \times \mathbb{Z}_g$ is a multiple of g. Moreover, by Lemma 7.12 the SCC C contains a cycle of length $k \cdot g(C)$ for every sufficiently large $k \in \mathbb{N}$ ($k \geqslant m(C)$ suffices). Since g is a multiple of $g(C)$, C also contains a cycle of length $k \cdot g$ for every sufficiently large k. But every such cycle induces a cycle of the same length $k \cdot g$ in $C \times \mathbb{Z}_g$. Hence there exists $k \in \mathbb{N}$ such that $C \times \mathbb{Z}_g$ contains cycles of length $k \cdot g$ and $(k+1) \cdot g$. It follows that the period of $C \times \mathbb{Z}_g$ divides $\gcd(k \cdot g, (k+1) \cdot g) = g$. This proves that the period of $C \times \mathbb{Z}_g$ is exactly g. ☐

Henceforth, we assume that $\mathcal{B}$ has property from Lemma 7.13 and fix the number g such that all nontransient SCCs have period g.

Periodic acceptance sets For $a \in \mathbb{N}$ and $X \subseteq \mathbb{N}$ we use the standard notation $X + a = \{a + x \mid x \in X\}$. For a state $q \in Q$ we define $\mathrm{Acc}(q) = \{n \in \mathbb{N} \mid \exists w \in \Sigma^n \colon \delta(w, q) \in F\}$. A set $X \subseteq \mathbb{N}$ is *eventually d-periodic*, where $d \geqslant 1$ is an integer, if there exists a *threshold* $t \in \mathbb{N}$ such that for all $x \geqslant t$ we have $x \in X$ if and only if $x + d \in X$. If X is eventually d-periodic for some $d \geqslant 1$, then X is *eventually periodic*.

Lemma 7.14. *For every $q \in Q$ the set $\mathrm{Acc}(q)$ is eventually g-periodic.*

Proof. It suffices to show that for all $0 \leqslant r \leqslant g-1$ the set $S_r = \{i \in \mathbb{N} \mid r + i \cdot g \in \mathrm{Acc}(q)\}$ is either finite or co-finite. Consider a remainder $0 \leqslant r \leqslant g-1$ where S_r is infinite. We need to show that S_r is indeed co-finite. Let $i \in S_r$ with $i \geqslant |Q|$, i.e. there exists an accepting run π from q of length $r + i \cdot g$. Since π has length at least $|Q|$ it must traverse a state q in a nontransient SCC C. Choose j_0 such that $j_0 \cdot g \geqslant m(C)$ where $m(C)$ is the reachability constant from Lemma 7.12. By Lemma 7.12 for all $j \geqslant j_0$ there exists a cycle from q to q of length $j \cdot g$. Therefore we can prolong π to a longer accepting run by $j \cdot g$ symbols for any $j \geqslant j_0$. This proves that $x \in S_r$ for every $x \geqslant i + j_0$ and that S_r is co-finite. ☐

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

Lemma 7.15. *A set $X \subseteq \mathbb{N}$ is eventually d-periodic iff X and $X + d$ are almost equal.*

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

Lemma 7.16. *Let C be a nontransient SCC in $\mathcal{B}$, $p, q \in C$ and $s = \mathrm{shift}(p, q)$. Then $\mathrm{Acc}(p)$ and $\mathrm{Acc}(q) + s$ are almost equal.*

Proof. Let $k \in \mathbb{N}$ such that $k \cdot g \geqslant m(C)$ where $m(C)$ is the large enough constant from Lemma 7.12. By Lemma 7.12 there exists a run from p to q of length $s + k \cdot g$, and a run from q to p of length $(k + 1) \cdot g - s$ (the latter number is congruent to $\mathrm{shift}(q, p)$ modulo g). By prolonging accepting runs we obtain

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

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

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

By Lemma 7.14 and Lemma 7.15 the three sets above are almost equal. Also $\mathrm{Acc}(q) + s + k \cdot g$ is almost equal to $\mathrm{Acc}(q) + s$ by Lemma 7.14 and Lemma 7.15. Since almost equality is a transitive relation, this proves the statement. □

Corollary 7.17. *There exists a threshold $t \in \mathbb{N}$ such that*

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

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

We fix the threshold t from Corollary 7.17 in the following. The following lemma is the main tool to prove the correctness of our sliding window testers. It states that if a word of length n is accepted from p and ρ is any internal run from p of length at most n, then, up to a bounded length prefix, ρ can be extended to an accepting run of length n. Formally, a run π k-*simulates* a run ρ if one can factorize $\rho = \rho_1\rho_2$ and $\pi = \pi'\rho_2$ where $|\rho_1| \leqslant k$.

Lemma 7.18. *If ρ is an internal run starting from p of length at most n and $n \in \mathrm{Acc}(p)$, then there exists an accepting run π from p of length n which t-simulates ρ.*

Proof. If $|\rho| \leqslant t$, then we choose any accepting run π from p of length $n \in \mathrm{Acc}(p)$. Otherwise, if $|\rho| > t$, then the SCC C containing p is nontransient and we can factor $\rho = \rho_1\rho_2$ such that $|\rho_1| = t$ where ρ_2 leads from p to q. Set $s := \mathrm{shift}(q, p)$, which satisfies $s + |\rho_2| \equiv 0 \pmod g$ by the properties in Lemma 7.12. Since $\mathrm{Acc}(q) =_t \mathrm{Acc}(p) + s$ by Corollary 7.17, $n > t$ and $n \in \mathrm{Acc}(p)$, we have $n+s \in \mathrm{Acc}(q)$. Finally since $n+s \equiv n-|\rho_2| \pmod g$ and $n-|\rho_2| = n-|\rho|+t \geqslant t$ we know $n - |\rho_2| \in \mathrm{Acc}(q)$. This yields an accepting run π' from q of length $n - |\rho_2|$. Then ρ is t-simulated by $\pi = \pi'\rho_2$. □

7.4.2 Deterministic sliding window tester

We are now ready to prove Theorem 7.1.

Proof of Theorem 7.1. Let $n \in \mathbb{N}$ be a window size. If $n < |Q|$ we use a trivial streaming algorithm which stores the window explicitly. By Lemma 4.2 we can maintain the set of all path summaries $\mathrm{PS}_{\mathcal{B}}(w) = \{\mathrm{ps}(\pi_{w,q}) \mid q \in Q\}$ for the active window $w \in \Sigma^n$. In fact, the path summary algorithm works for variable-size windows but we do not need this here. By Proposition 4.3 the path summary algorithm requires $O(\log n)$ bits.

It remains to define a proper acceptance condition. Consider the SCC-factorization of π_{w,q_0}, say $\pi_{w,q_0} = \pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_1 \pi_1$ and its path summary $(\ell_m, q_m) \cdots (\ell_1, q_1)$. The algorithm accepts if and only if the path summary is accepting, i.e. $\ell_m = |\pi_m| \in \mathrm{Acc}(q_m)$. If $w \in L$ then clearly $|\pi_m| \in \mathrm{Acc}(q_m)$. If $|\pi_m| \in \mathrm{Acc}(q_m)$ then the internal run π_m can be t-simulated by an accepting run π'_m of equal length by Lemma 7.18. The run $\pi'_m \tau_{m-1} \pi_{m-1} \cdots \tau_1 \pi_1$ is accepting and witnesses that $\mathrm{pdist}(w, L) \leqslant t$. □

7.4.3 Sliding window tester with two-sided error

Fix a parameter $0 < \epsilon < 1$ and a window length $n \in \mathbb{N}$. The goal is to construct a randomized sliding window tester for L with two-sided error and Hamming gap ϵn that uses $O(1/\epsilon)$ bits. If $\epsilon n/4 < t + 1$ then we use a trivial sliding window tester that stores the window explicitly with $n = O(1/\epsilon)$ bits. Now assume $t + 1 \leqslant \epsilon n/4$ and define the parameters $h = n - t$ and $\ell = (1 - \epsilon)n + t + 1$, which satisfy

$$\begin{aligned} \frac{\ell}{h} &= \frac{(1-\epsilon)n + t + 1}{n - t} \leqslant \frac{(1-\epsilon)n + \frac{\epsilon}{4}n}{n - \frac{\epsilon}{4}n} \\ &= \frac{1 - \frac{3}{4}\epsilon}{1 - \frac{1}{4}\epsilon} = \frac{1 - \frac{1}{4}\epsilon}{1 - \frac{1}{4}\epsilon} - \frac{\frac{1}{2}\epsilon}{1 - \frac{1}{4}\epsilon} \leqslant 1 - \frac{1}{2}\epsilon. \end{aligned} \tag{7.2}$$

Let $\mathcal{Z}$ be the (h, ℓ)-counter with error probability $1/(3|Q|)$ from Proposition 6.8, which uses $O(\log(1/\epsilon))$ space by (7.2). The counter is used to define so-called compact summaries of runs.

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

(C1) π_i starts in q_i;

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

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

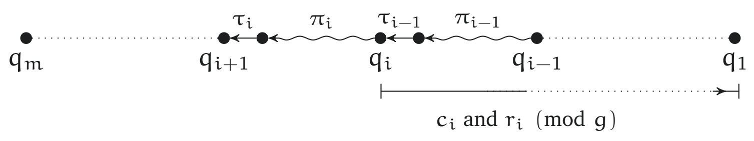

Figure 7.1: A compact summary of a run π.

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

The idea of a compact summary is visualized in Figure 7.1. If $m > |Q|$ then the above compact summary cannot represent a run. Therefore, we can assume that $m \leqslant |Q|$. For every triple (q_i, r_i, c_i), the entries q_i and r_i only depend on the rDFA $\mathcal{B}$, and hence can be stored with $O(1)$ bits. Every state c_i of the probabilistic counter needs $O(\log(1/\epsilon))$ bits. Hence, a compact summary can be stored in $O(\log(1/\epsilon))$ bits. In contrast to Theorem 7.1, we maintain a set of compact summaries which represent all runs of $\mathcal{B}$ on the *complete* stream read so far (not only on the active window) with high probability.

Proposition 7.19. *For a given input stream $w \in \Sigma^*$, we can maintain a set of compact summaries S containing for each $q \in Q$ a compact summary $\kappa_q \in S$ starting in q such that κ_q represents the unique run $\pi_{w,q}$ with probability at least 2/3.*

Proof. For each state in Q, we initialize the compact summary so that it represents the run $\pi_{\varepsilon,q}$. Consider a compact summary $\kappa = (q_m, r_m, c_m)\cdots(q_1, r_1, c_1)$, which represents a run π_{x,q_1}. We prolong κ by a transition $q_1 \overset{a}{\leftarrow} p$ in $\mathcal{B}$ as follows:

- if p and q are not in the same SCC, then we increment all counter states c_i, increment all remainders r_i mod g, and append a new triple $(p, 0, c_1)$;
- if p and q belong to the same SCC, then we increment all counter states c_i for $2 \leqslant i \leqslant m$, increment the remainder r_i mod g for $2 \leqslant i \leqslant m$, and replace q_1 by p.

If $a \in \Sigma$ is the next input symbol of the stream, then S is updated to the new set S' of compact summaries by iterating over all transitions $q \overset{a}{\leftarrow} p$ in $\mathcal{B}$ and prolonging the compact summary starting in q by that transition.

To verify correctness, consider a compact summary κ computed by the algorithm. Properties (C1) and (C2) are satisfied by construction. Furthermore, since the length of κ is bounded by $|Q|$ and each instance of $\mathcal{Z}$ has error probability $1/(3|Q|)$ the probability that property (C3) or (C4) is violated is at most 1/3 by the union bound. □

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

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

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

Proposition 7.20. *Assume that* $\epsilon n \geqslant t$. *Let* $w \in \Sigma^*$ *with* $|w| \geqslant n$ *and let* κ *be a compact summary which represents* π_{w,q_0}.

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

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

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

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

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

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

and therefore κ is accepting.

For point 2 assume that κ is accepting, i.e.

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

Recall that $|\pi_k' \tau_{k-1} \cdots \pi_i| > t$. By definition of $\mathrm{Acc}_{\mathit{mod}}(q_i)$ there exists an accepting run from q_i whose length is congruent to $|\pi_k' \tau_{k-1} \cdots \pi_i| \bmod g$ and at least t. By point (i) from Corollary 7.17 we derive that $|\pi_k' \tau_{k-1} \cdots \pi_i| \in \mathrm{Acc}(q_i)$. We claim that $|\pi_i \tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1| \geqslant (1 - \epsilon)n + t$ by a case distinction. If $i = m$, then clearly $|\pi_i \tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1| \geqslant n \geqslant (1 - \epsilon)n + t$. If $i < m$, then c_{i+1} is high by maximality of i, which implies $|\tau_i \pi_i \cdots \tau_1 \pi_1| > (1 - \epsilon)n + t + 1$ by (C3). Since τ_i has length one, we have $|\pi_i \tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1| > (1 - \epsilon)n + t$.

Since $|\pi_k' \tau_{k-1} \cdots \pi_i| \in \mathrm{Acc}(q_i)$, we can apply Lemma 7.18 and obtain an accepting run ρ of length $|\pi_k' \tau_{k-1} \cdots \pi_i| \in \mathrm{Acc}(q_i)$ starting in q_i which t-simulates the internal run π_i. The prefix distance from ρ to $\pi_k' \tau_{k-1} \cdots \pi_i$ is at most

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

Therefore the prefix distance from the accepting run $\rho \tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1$ to the run $\pi_k' \tau_{k-1} \pi_{k-1} \cdots \tau_1 \pi_1$ is also at most ϵn. This implies $\mathrm{pdist}(\mathrm{last}_n(w), L) \leqslant \epsilon n$. □

We are now ready to prove Theorem 7.3.

Proof of Theorem 7.3. As mentioned above we use a trivial sliding window algorithm whenever $\epsilon n/4 < t + 1$, using $O(1/\epsilon)$ bits. Otherwise, we use the algorithm from Proposition 7.19, which is initialized by reading $\square^n$. It maintains a compact summary which represents π_{w,q_0} with probability 2/3 for the read stream prefix w. The algorithm accepts if that compact summary is accepting. From Proposition 7.20 we get:

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

This concludes the proof of Theorem 7.3. $\square$

7.4.4 Sliding window tester with one-sided error

Let L be a finite union of trivial regular languages and suffix-free regular languages. In this section, we present a randomized sliding window tester for L with one-sided error and Hamming gap $\gamma(n) = \epsilon n$ that uses space $O(\log \log n)$. By Lemma 7.11 and Theorem 7.6, it suffices to consider the case when L is a suffix-free regular language. Since L is suffix-free, $\mathcal{B}$ has the property that no final state can be reached from a final state by a nonempty run. We decompose $\mathcal{B}$ into a finite union of *partial rDFAs*, cf. Section 4.2.2.

A *path description* is a sequence

$$(q_k, a_k, p_{k-1}), C_{k-1}, \dots, (q_2, a_2, p_1), C_1, (q_1, a_1, p_0), C_0, q_0$$

where $C_{k-1}, \dots, C_0$ is a chain (read from right to left) in the SCC-ordering of $\mathcal{B}$, $p_i, q_i \in C_i$, $q_{i+1} \xleftarrow{a_{i+1}} p_i$ is a transition in $\mathcal{B}$ for all $0 \leqslant i \leqslant k-1$, and $q_k \in F$.

Each path description defines a *partial rDFA* $\mathcal{B}_P = (Q_P, \Sigma, \{q_k\}, \delta_P, q_0)$ by restricting $\mathcal{B}$ to the state set $Q_P = \bigcup_{i=0}^{k-1} C_i \cup \{q_k\}$, restricting the transitions of $\mathcal{B}$ to internal transitions from the SCCs C_i and the transitions $q_{i+1} \xleftarrow{a_{i+1}} p_i$, and declaring q_k to be the only final state. The rDFA is partial since for every state p_i and every symbol $a \in \Sigma$ there exists at most one transition $q \xleftarrow{a} p_i$. Since the number of path descriptions P is finite and $L(\mathcal{B}) = \bigcup_P L(\mathcal{B}_P)$, we can fix a single path description P and provide a sliding window tester for $L(\mathcal{B}_P)$ (we again use Lemma 7.11 here).

From now on, we fix a path description

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

and the partial automaton $\mathcal{B}_P = (Q_P, \Sigma, \{q_k\}, \delta_P, q_0)$ corresponding to it. The acceptance sets $\mathrm{Acc}(q)$ are defined with respect to B_P. If all C_i are transient, then $L(\mathcal{B}_P)$ is a singleton and we can use a trivial sliding window tester with space complexity $O(1)$. Now assume the contrary and let $0 \leqslant e \leqslant k-1$ be maximal such that C_e is nontransient.

Lemma 7.21. *There exist numbers* $r_0, \dots, r_{k-1}, s_0, \dots, s_e \in \mathbb{N}$ *such that the following holds:*

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

(ii) Every run from q_i *to* q_{i+1} *has length* $r_i \pmod g$.

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

Proof. Point (i) follows immediately from the definition of transient SCCs. Let us now show the second part of the claim.

Let $0 \leqslant i \leqslant k-1$ and let N_i be the set of lengths of runs of the form $q_{i+1} \xleftarrow{a_{i+1}} p_i \xleftarrow{w} q_i$ in B_P. If C_i is transient, then $N_i = \{1\}$. Otherwise, by Lemma 7.12 there exist a number $r_i \in \mathbb{N}$ and a cofinite set $D_i \subseteq \mathbb{N}$ such that $N_i = r_i + gD_i$. We can summarize both cases by saying that there exist a number $r_i \in \mathbb{N}$ and a set $D_i \subseteq \mathbb{N}$ which is either cofinite or $D_i = \{0\}$ such that $N_i = r_i + gD_i$. This implies point (ii). Then the acceptance sets in B_P satisfy

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

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

Let p be a random prime with $\Theta(\log\log n)$ bits. We choose p as in the proof of Lemma 6.11 such that $\Pr[\ell \equiv n \pmod p] \leqslant 1/3$ for all $0 \leqslant \ell < 2n$, $\ell \neq n$. Define a threshold $s = \max\{k, \sum_{j=0}^{k-1} r_j, s_0, \dots, s_e\}$ and for a word $w \in \Sigma^*$ define the function $\ell_w \colon Q \to \mathbb{N} \cup \{\infty\}$ where

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

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

Lemma 7.22. *Let* $n \in \mathrm{Acc}(q_0)$ *be a window size with* $n \geqslant s + |Q_P|$ *and* $w \in \Sigma^*$ *with* $|w| \geqslant n$*. There exists a constant* $c > 0$ *such that:*

(i) if $\mathrm{last}_n(w) \in L(\mathcal{B}_P)$*, then* w *is accepted with probability* 1*;*

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

Proof. Assume first that $\mathrm{last}_n(w) \in L(\mathcal{B}_P)$. Since $L(\mathcal{B}_P) \subseteq L$ is suffix-free, $\ell_w(q_0) = n \pmod p$ and w is accepted with probability 1.

Consider now the case when $\mathrm{last}_n(w) \notin L(\mathcal{B}_P)$. By definition, in this case $\ell_w(q_0) \neq n$. In other words, only two cases are possible: either $\ell_w(q_0) < n$, or $\ell_w(q_0) > n$. If $\ell_w(q_0) < n$, then by the choice of p we have $\ell_w(q_0) \not\equiv n \pmod p$ with probability at least 2/3.

We finally consider the case $\ell_w(q_0) > n$. We will show that in this case the prefix distance between $\mathrm{last}_n(w)$ and $L(\mathcal{B}_P)$ is bounded by a constant c, which means that we can either accept or reject. Let π be the initial run of $\mathcal{B}_P$ on $\mathrm{last}_n(w)$, and let $\pi = \pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_0\pi_0$ be its SCC-factorization. We have $|\pi| = n$. Since $\ell_w(q_0) > n$, the run π can be strictly prolonged to a run to q_k and hence we must have $m < k$. For all $0 \leqslant i \leqslant m$, the run π_i is an internal run in the SCC C_i from q_i to p_i. For all $0 \leqslant i \leqslant m-1$ we have $\tau_i = (q_{i+1} \xleftarrow{a_{i+1}} p_i)$ and $|\tau_i\pi_i| \equiv r_i \pmod g$, by point (ii) from Lemma 7.21. We claim that there exists an index $0 \leqslant i_0 \leqslant m$ such that the following three properties hold:

1. q_{i_0} is nontransient,
2. $|\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_{i_0}\pi_{i_0}| \geqslant s$,
3. $|\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_{i_0+1}\pi_{i_0+1}| \leqslant s + |Q_P|$.

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

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

Now let $0 \leqslant i_0 \leqslant m$ be the largest integer satisfying properties 1 and 2. If $\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_{i_0+1}\pi_{i_0+1}$ only passes transient states, then its length is bounded by $m - i_0 \leqslant s + m$, and we are done. Otherwise, let $i_0 + 1 \leqslant j \leqslant m$ be the smallest integer such that q_j is nontransient. The run $\tau_{j-1}\pi_{j-1}\cdots\tau_{i_0+1}\pi_{i_0+1}$ only passes transient states and therefore it has length $j - i_0 - 1$. By maximality of i_0, we have $|\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_j\pi_j| < s$ and hence property 3 holds:

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

Let $0 \leqslant i_0 \leqslant m$ be the index satisfying properties 1-3. Since q_{i_0} is nontransient, we have $i_0 \leqslant e$ and therefore $\mathrm{Acc}(q_{i_0}) =_s \sum_{j=i_0}^{k-1} r_j + g\mathbb{N}$. We have $|\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_{i_0}\pi_{i_0}| \in \mathrm{Acc}(q_{i_0})$ because it is larger than s (by property 2) and

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

where the last congruence follows from the fact that $n \in \mathrm{Acc}(q_0) =_s \sum_{j=0}^{k-1} r_j + g\mathbb{N}$.

By Lemma 7.18 there exists an accepting run π' which t-simulates π_{i_0} and has length $|\pi_m\tau_{m-1}\pi_{m-1}\cdots\tau_{i_0}\pi_{i_0}|$. The prefix distance between the runs $\pi =$

$\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_0 \pi_0$ and $\pi' \tau_{i-1} \pi_{i_0-1} \cdots \tau_0 \pi_0$ is at most

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

by property 3. □

Proof of Theorem 7.4. Let $n \in \mathbb{N}$ be the window size. From the discussion above, it suffices to give a sliding window tester for a fixed partial automaton $\mathcal{B}_P$. Assume $n \geqslant s + |Q|$, otherwise a trivial tester can be used. If $n \notin \mathrm{Acc}(q_0)$, the tester always rejects. Otherwise, the tester picks a random prime p with $\Theta(\log \log n)$ bits and maintains $\ell_w(q) \pmod p$ for all $q \in Q_P$, where w is the stream read so far, which requires $O(\log \log n)$ bits. When a symbol $a \in \Sigma$ is read, we can update ℓ_{wa} using ℓ_w: If $q = q_k$, then $\ell_{wa}(q) = 0$, otherwise $\ell_{wa}(q) = 1 + \ell_w(\delta_P(a, q)) \pmod p$ where $1 + \infty = \infty$. The tester accepts if $\ell_w(q_0) \equiv n \pmod p$. Lemma 7.22 guarantees that the tester is false-biased. □

7.5 Lower bounds

7.5.1 Tradeoff between Hamming gap and space

Comparing Theorem 7.1 and Theorem 7.3 leads to the question whether one can replace the Hamming gap $\gamma(n) = \epsilon n$ in Theorem 7.3 by a sublinear function $\gamma(n)$ while retaining constant space at the same time. We show that this is not the case:

Lemma 7.23. *Let* $L = a^* \subseteq \{a, b\}^*$. *Every randomized sliding window tester with two-sided error for* L *and window size* n *with Hamming gap* $\gamma(n)$ *needs space* $\Omega(\log n - \log \gamma(n))$.

Proof. We reduce from the greater-than-function GT_m, whose randomized one-way communication complexity is $\Omega(\log m)$ (Theorem 6.12). Consider a randomized sliding window tester $\mathcal{P}_n$ for a^* and window size n with Hamming gap $\gamma(n)$. Let $k := \gamma(n) + 1$ and define $m = \lfloor n/k \rfloor$. Hence we have $n = mk + r$ for some $r < k$. We divide the window into m blocks of length k. We then obtain a randomized one-way protocol for the greater-than-function on the interval $\{1, \dots, m\}$: Alice produces from her input $i \in \{1, \dots, m\}$ the word $w_i = b^k a^{r+(m-i)k}$. She then runs $\mathcal{P}_n$ on w_i and sends the memory state to Bob. Bob continues the run of the randomized sliding window tester, starting from the transferred memory state, with the input stream a^{jk}. He obtains the memory state reached after the input $b^k a^{r+(m-i+j)k}$. Finally, Bob outputs the negated answer given by the randomized sliding window tester. If $i \leqslant j$, then $\mathrm{last}_n(b^k a^{r+(m-i+j)k}) = a^n \in L$. On the other hand, if $i > j$, then

$$|b^k a^{r+(m-i+j)k}| = k + r + (m - i + j)k \leqslant k + r + (m-1)k = r + mk = n$$

and hence $\mathrm{last}_n(b^k a^{r+(m-i+j)k})$ contains at least k many b's. Since the Hamming distance between this window and a^* is at least $k = \gamma(n) + 1$, the algorithm must reject with high probability. Therefore $s(\mathcal{P}_n) = \Omega(\log m) = \Omega(\log n - \log \gamma(n))$. □

For example if $\gamma(n) \leqslant n^c$ for some $0 < c < 1$ then the space complexity must be at least $\Omega(\log n)$.

7.5.2 Nontrivial languages

Next we prove Theorem 7.2 and Theorem 7.5(2). Since every true-biased sliding window tester can be turned into a nondeterministic one (by forgetting probabilities), we obtain Theorem 7.2 from the following proposition.

Proposition 7.24. *Let* L *be regular and nontrivial. Then there exists* $0 < \epsilon \leqslant 1$ *and infinitely many window sizes* n *such that every nondeterministic sliding window tester for* L *with Hamming gap* ϵn *uses at least* $\log n - O(1)$ *space.*

Proof. By Lemma 7.8, $\mathrm{cut}_{i,j}(L)$ is not a length language for all $i, j \geqslant 0$. Let N be the set of lengths from Proposition 7.9 such that $L|_N$ is infinite and excludes some factor w_f. Let $c = |w_f| > 0$ and choose $0 < \epsilon < 1/c$. Since N is an arithmetic progression, $L|_N$ is regular. Recall that every word v that contains k disjoint occurrences of w_f has Hamming distance at least k from any word in $L|_N$. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ be a DFA for $L|_N$. Since $L(\mathcal{A})$ is infinite, there must exist words x, y, z such that $y \neq \lambda$ and for $\delta(q_0, x) = q$ we have $\delta(q, y) = q$ and $\delta(q, z) \in F$. Let $d = |xz|$ and $e = |y| > 0$, which satisfy $d + e\mathbb{N} \subseteq N$.

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

$$u_k = w_f^n x y^k \text{ and } v_k = u_k z = w_f^n x y^k z.$$

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

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

many disjoint occurrences of w_f. Hence, $\mathcal{P}_n$ must reject any of the input streams v_k for $0 \leqslant k \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$.

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

Assume now that $m \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$. Then there must exist numbers i and j with $0 \leqslant i < j \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$ such that $p_i = p_j =: p$. By cutting off cycles at p from the run π and repeating this, we finally obtain a run of $\mathcal{P}_n$ on an input stream $v_k = w_f^n x y^k z$ with $k \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$. This run is still successful, and hence $\mathcal{P}_n$ accepts v_k with $k \leqslant \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$. This contradicts our previous observation. Hence, $\mathcal{P}_n$ must have more than $\lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$ states. This implies

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

which proves the theorem. □

Proposition 7.25. *Let* L *be regular and nontrivial. Then there exists* $0 < \epsilon \leqslant 1$ *and infinitely many window sizes* n *such that every co-nondeterministic sliding window tester for* L *with Hamming gap* ϵn *uses at least* $\log\log n - O(1)$ *space.*

Proof. We can transform any coNFA $\mathcal{P} = (Q, \Sigma, I, \Delta, F)$ into an equivalent DFA over the state set 2^Q using the powerset construction. The only difference to the powerset construction for NFAs is that a state $P \subseteq Q$ is declared final if and only if $P \subseteq F$. Since the lower bound from Proposition 7.24 also holds for deterministic sliding window testers we obtain $\log\log n - O(1)$ as a lower bound for the space complexity of co-nondeterministic sliding window testers for nontrivial regular languages. □

7.5.3 Finite unions of suffix-free and trivial languages

Finally, we need to show the $\Omega(\log n)$ lower bound in Theorem 7.5(1), which will be proven for co-nondeterministic SW-testers. We start with the following observation.

Lemma 7.26. *Every regular suffix-free language excludes a factor.*

Proof. Let $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ be an rDFA for L. Since L is suffix-free, we can assume that there is a single minimal SCC that consists of a single state $q_{\mathit{fail}} \notin F$ (if a minimal SCC would contain a final state, then L would not be suffix-free). We have $\delta(a, q_{\mathit{fail}}) = q_{\mathit{fail}}$ for all $a \in \Sigma$. We construct a word $w_f \in \Sigma^*$ such that $\delta(p, w_f) = q_{\mathit{fail}}$ for all $p \in Q$. Let $p_1, \dots, p_m$ be an enumeration of all states in $Q \setminus \{q_{\mathit{fail}}\}$. We then construct inductively words $w_0, w_1, \dots, w_m \in \Sigma^*$ such that for all $0 \leqslant i \leqslant m$: $\delta(w_i, p) = q_{\mathit{fail}}$ for all $p \in \{p_1, \dots, p_i\}$. We start with $w_0 = \varepsilon$. Assume that w_i has been constructed for some $i < m$. There is a word x such that that $\delta(x, \delta(w_i, p_{i+1})) = q_{\mathit{fail}}$. We set $w_{i+1} = x w_i$. Then $\delta(w_{i+1}, p_{i+1}) = \delta(x w_i, p_{i+1}) = q_{\mathit{fail}}$ and $\delta(w_{i+1}, p_j) = \delta(w_i x, p_j) = \delta(x, q_{\mathit{fail}}) = q_{\mathit{fail}}$ for $1 \leqslant j \leqslant i$. We finally define $w_f = w_m$. □

Lemma 7.27. *Every regular language* L *satisfies one of the following properties:*

- L *is a finite union of regular trivial languages and regular suffix-free languages.*
- L *has a restriction* $L|_N$ *which excludes some factor and contains* $y^* z$ *for some* $y, z \in \Sigma^*$, $|y| > 0$.

Proof. Let $\mathcal{B} = (Q, \Sigma, F, \delta, q_0)$ be an rDFA for L. Let $\mathcal{B}_r = (Q, \Sigma, F_r, \delta, q_0)$ where F_r is the set of nontransient final states and $\mathcal{B}_q = (Q, \Sigma, \{q\}, \delta, q_0)$ for $q \in Q$. We can decompose L as a union of $L_r = L(\mathcal{B}_r)$ and all languages $L(\mathcal{B}_q)$ over all transient states $q \in F$. Notice that $L(\mathcal{B}_q)$ is suffix-free for all transient $q \in F$ since any run to q cannot be prolonged to another run to q. If L_r is trivial, then L satisfies the first property. If L_r is nontrivial, then by Lemma 7.8 and Proposition 7.9 there exists an arithmetic progression $N = a + b\mathbb{N}$ such that $L_r|_N$ is infinite and excludes some word $w \in \Sigma^*$ as a factor. Let $z \in L_r|_N$ be any word. Since $\mathcal{B}_r$ reaches some nontransient final state p on input z there exists a word y which leads from p back to p. We can ensure that $|y|$ is a multiple of b by replacing y by a suitable power y^i. Then $y^*z \subseteq L_r|_N \subseteq L|_N$. Furthermore since each language $L(\mathcal{B}_q)$ excludes some factor w_q by Lemma 7.26 the language $L|_N \subseteq L_r|_N \cup \bigcup_q L(\mathcal{B}_q)$ excludes any concatenation of w and all words w_q as a factor. □

Theorem 7.28. *Let* L *be a regular language that is not a finite union of regular trivial languages and regular suffix-free languages. Then there exist* $0 < \epsilon \leqslant 1$ *and infinitely many window sizes* n *for which every co-nondeterministic sliding window tester for* L *with Hamming gap* ϵn *uses at least* $\log n - O(1)$ *space.*

Proof. By Lemma 7.27, L has a restriction $L|_N$ which excludes some factor w_f and contains y^*z for some $y, z \in \Sigma^*$, $|y| > 0$. Let $c = |w_f| \geqslant 1$. We choose $\epsilon < 1/c$. Let $d = |z|$ and $e = |y|$. Fix a window length $n \in N$ and define for $k \geqslant 0$ the input streams $u_k = w_f^n y^k$ and $v_k = u_k z = w_f^n y^k z$. Consider a co-nondeterministic sliding window tester $\mathcal{P}_n$ for L and window size n with Hamming gap ϵn. Let $\alpha = c\epsilon < 1$ and $r = \lfloor \frac{(1-\alpha)n-c-d}{e} \rfloor$. If $0 \leqslant k \leqslant r$, then the suffix of v_k of length n contains at least

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

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

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

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

This proves the theorem. □

7.6 Conclusion

In this chapter we proved that every regular language has a randomized (deterministic) sliding window tester using constant (logarithmic) space. We described the regular languages that are trivial in the context of property testing. Furthermore, we characterized the regular languages that admit sliding window testers with one-sided error and $o(\log n)$ space.

Open problems

1. In the introduction we mentioned the sampling algorithm over sliding windows [23], which uses $O(1)$ memory words, i.e. $O(\log n)$ bits (to store positions in the window). Using this result, any sampling-based property tester which makes $q(n)$ queries can be turned into a sliding window property tester using $O(q(n) \log n)$ bits of space, see [23, Theorem 5.1]. We pose the question whether one can rule out a $o(\log n)$-space sampling algorithm over sliding windows.

2. Does every context-free language have a deterministic sliding window tester with Hamming gap ϵn using $O(\log n)$ space (or at least space $o(n)$)?

3. Does every context-free language have a randomized sliding window tester with Hamming gap ϵn that uses constant space (or at least space $o(n)$)?

Related work The third open problem is related to the question whether every context-free language has a property tester making $o(n)$ many queries. While the general case seems to be open, some progress has been done for Dyck languages [48, 96]. Alon et al. [4] presented a two-sided error property tester for D_1 with $O(1)$ queries. For D_m where $m \geqslant 2$ there currently best upper bound on the query complexity is $O(n^{2/5+\delta})$ for any $\delta > 0$ and the best lower bound is $\Omega(n^{1/5})$ [48].

Lachish, Newman and Shapira studied the relation between (offline) space complexity and the query complexity of languages [83]. They construct languages with space complexity $O(s(n))$ and query complexity $2^{\Omega(s(n))}$ for every $\log\log n \leqslant s(n) \leqslant \frac{1}{10}\log n$. Furthermore, the authors conjecture that any language with space complexity $s(n)$ can be tested with $2^{O(s(n))}$ many queries.

Chapter 8

Strict correctness

8.1 Introduction

In this chapter we look at a stronger correctness definition of randomized sliding window algorithms. Let $\Phi \subseteq \Sigma^* \times Y$ be an *approximation problem*, which associates with each word $w \in \Sigma^*$ a set of admissible or correct outputs values in Y. This generalizes the previous definition of computational problems as functions $\varphi \colon \Sigma^* \to Y$. For example, let $\varphi \colon \Sigma^* \to \mathbb{N}$ be a statistical function, say counting the occurrences of a certain symbol or the number of distinct symbols. We can define the ϵ-approximation problem of φ to be the set $\Phi_\epsilon \subseteq \Sigma^* \times \mathbb{N}$ of all pairs (w, k) such that $(1-\epsilon)\varphi(w) \leqslant k \leqslant (1+\epsilon)\varphi(w)$. To every approximation problem Φ we associate the sliding window problem $\mathsf{SW}_n(\Phi) = \{(x, y) \mid (\mathrm{last}_n(x), y) \in \Phi\}$ for window length n.

Let $\mathcal{P}$ be a randomized streaming algorithm, modelled as a probabilistic automaton over an alphabet Σ with an output set Y. We look at two correctness definitions that can be found in the literature.

- For every input stream $w = a_1 \cdots a_m$ the probability that after reading w, $\mathcal{P}$ outputs a value y with $(w, y) \notin \Phi$ is at most λ. In this case, we say that $\mathcal{P}$ is λ-*correct* for Φ. This extends the definition of randomized streaming algorithms from Chapter 6 to approximation problems.
- For every input stream $w = a_1 \cdots a_m$ the probability that on input w, there exists a time instant $0 \leqslant t \leqslant m$ where $\mathcal{P}$ outputs a value y with $(a_1 \cdots a_t, y) \notin \Phi$ is at most λ. In this case, we say that $\mathcal{P}$ is *strictly* λ-*correct* for Φ.

If an algorithm is λ-correct, then a priori it is strictly correct, i.e. it produces a correct output at every time instant, only with probability $(1-\lambda)^m$ where m is the length of the stream, which tends to 0 for $m \to \infty$.

A *(strictly)* λ-*correct* randomized sliding window algorithm $\mathcal{P}_n$ for Φ and window size n is a *(strictly)* λ-*correct* randomized streaming algorithm for $\mathsf{SW}_n(\Phi)$. Hence a strictly λ-correct sliding window algorithm produces correct outputs for all windows during a (potentially long) run with high probability.

Strict correctness is used, without being explicitly mentioned, for instance in [17, 36]. For instance, Ben-Basat et al. write “We say that algorithm A is ϵ-correct on a input instance S if it is able to approximate the number of 1’s in the last W bits, at every time instant while reading S, to within an additive error of $W\epsilon$”. In these papers, the lower bounds shown for deterministic sliding window algorithms are extended with the help of Yao’s minimax principle [113] to strictly λ-correct randomized sliding window algorithms. The main result of this chapter states that this is a general phenomenon: we show that every strictly λ-correct sliding window algorithm for an approximation problem can be derandomized without increasing the space usage. The results of this chapter have appeared in [G7].

8.2 Derandomization

Let $\Phi \subseteq \Sigma^* \times Y$ be an approximation problem and $\mathcal{P} = (Q, \Sigma, \iota, \rho, o)$ be a randomized streaming algorithm. A run $\pi = q_0 a_1 \cdots a_m q_m$ in $\mathcal{P}$ is *strictly correct* for Φ if for all $0 \leqslant i \leqslant m$ we have $(a_1 \cdots a_i, o(q_i)) \in \Phi$, i.e. if the value returned by the algorithm after reading the prefix $a_1 \cdots a_i$ is correct. We say that $\mathcal{P}$ is *strictly λ-correct* for Φ if for all $w \in \Sigma^*$ we have

$$\Pr_{\pi \in \mathrm{Runs}(\mathcal{P}, w)} [\pi \text{ is strictly correct for } \Phi] \geqslant 1 - \lambda.$$

Recall that the probability for a run $\pi \in \mathrm{Runs}(\mathcal{P}, w)$ is given by $\iota(q_0) \cdot \rho(\pi)$ where q_0 is the initial state of π.

Fix an approximation problem $\Phi \subseteq \Sigma^* \times Y$, a window size $n \in \mathbb{N}$ and a randomized SW-algorithm $\mathcal{P}_n = (Q, \Sigma, \iota, \rho, o)$ which is strictly λ-correct for Φ where $0 \leqslant \lambda < 1$. In this section we will prove that one can extract a deterministic SW-algorithm $\mathcal{D}_n$ for Φ from $\mathcal{P}_n$ such that $s(\mathcal{D}_n) \leqslant s(\mathcal{P}_n)$.

Fix a window size $n \geqslant 0$ and let $\mathcal{P}_n = (Q, \Sigma, \iota, \rho, o)$. Consider a run

$$\pi\colon q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_m} q_m$$

in $\mathcal{P}_n$. A *subrun* of π is a run of the form

$$q_i \xrightarrow{a_{i+1}} q_{i+1} \xrightarrow{a_{i+2}} \cdots q_{j-1} \xrightarrow{a_j} q_j.$$

The run π is *simple* if $q_i \neq q_j$ for $0 \leqslant i < j \leqslant m$. Consider a nonempty subset $S \subseteq Q$ and a function $\delta\colon Q \times \Sigma \to Q$ such that S is closed under δ, i.e. $\delta(S \times \Sigma) \subseteq S$. We say that the run π is *δ-conform* if $\delta(q_{i-1}, a_i) = q_i$ for all $1 \leqslant i \leqslant m$. We say that π is *(S, δ)-universal* if for all $q \in S$ and $x \in \Sigma^n$ there exists a δ-conform subrun $\pi'\colon q \xrightarrow{x} q'$ of π. Finally, π is *δ-universal* if it is (S, δ)-universal for some nonempty subset $S \subseteq Q$ which is closed under δ.

Lemma 8.1. *Let π be a strictly correct run in $\mathcal{P}_n$ for Φ, let $S \subseteq Q$ be a nonempty subset and let $\delta\colon Q \times \Sigma \to Q$ be a function such that S is closed under δ. If π*

is (S,δ)*-universal, then there exists* $q_0 \in S$ *such that* $\mathcal{D}_n = (Q,\Sigma,q_0,\delta,o)$ *is a deterministic sliding window algorithm for* Φ *and window size* n.

Proof. Let $q_0 = \delta(p,\Box^n) \in S$ for some arbitrary state $p \in S$ and define $\mathcal{D}_n = (Q,\Sigma,q_0,\delta,o)$. Let $w \in \Sigma^*$ and consider the run $\sigma\colon p \xrightarrow{\Box^n} q_0 \xrightarrow{w} q$ in $\mathcal{D}_n$ of length $\geqslant n$. We have to show that $(\mathrm{last}_n(w), o(q)) \in \Phi$. We can write $\Box^n w = x\ \mathrm{last}_n(w)$ for some $x \in \Sigma^*$. Thus, we can rewrite the run σ as $\sigma\colon p \xrightarrow{x} q' \xrightarrow{\mathrm{last}_n(w)} q$. We know that $q' \in S$ because S is closed under δ. Since π is (S,δ)-universal, it contains a subrun $q' \xrightarrow{\mathrm{last}_n(w)} q$. By strict correctness of π we obtain $(\mathrm{last}_n(w), o(q)) \in \Phi$. □

For the rest of this section we fix an arbitrary function $\delta\colon Q \times \Sigma \to Q$ such that for all $q \in Q$, $a \in \Sigma$,

$$\rho(q,a,\delta(q,a)) = \max\{\rho(q,a,p) \mid p \in Q\}.$$

Note that

$$\rho(q,a,\delta(q,a)) \geqslant \frac{1}{|Q|}.$$

for all $q \in Q$, $a \in \Sigma$. Furthermore, let $\mathcal{D}_n = (Q,\Sigma,q_0,\delta,o)$ where the initial state q_0 will be defined later. We define for each $i \geqslant 1$ a state p_i, a run π_i^* in $\mathcal{D}_n$ on some word w_i, and a set $S_i \subseteq Q$. We abbreviate $\mathrm{Runs}(\mathcal{P}_n, w_1 \cdots w_m)$ by R_m. For $1 \leqslant i \leqslant m$ let H_i denote the event that for a random run $\pi = \pi_1 \cdots \pi_m \in R_m$, where each π_j is a run on w_j, the subrun π_i is (S_i,δ)-universal. Notice that H_i is independent of $m \geqslant i$.

First, we choose for p_1 a state that maximizes

$$\Pr_{\pi \in R_{i-1}}[\pi \text{ ends in } p_i \mid \forall j \leqslant i-1 : \overline{H_j}],$$

which is at least $1/|Q|$. Note that p_1 is a state such that $\iota(p_1)$ is maximal, since R_0 only consists of empty runs. Let $\preceq$ be the preorder on Q defined by $p \preceq q$ if and only if there exists a run q is reachable from p in $\mathcal{D}_n$. For S_i we take any maximal SCC of D_n which is reachable from p_i. Finally, we define the run π_i^*. It starts in p_i. Then, for each state $q \in S_i$ and each word $x \in \Sigma^n$ the run π_i^* leads from the current state to q via a simple run and reads the word x from q. Since S_i is a minimal SCC of $\mathcal{D}_n$ such a run exists. Hence, π_i^* is a run on a word of the form

$$w_i = \prod_{q \in S_i} \prod_{x \in \Sigma^n} y_{q,x}\, x.$$

Since we choose the runs on the words $y_{q,x}$ to be simple, the lengths of the words w_i are bounded independently of i. More precisely, we have $|w_i| \leqslant |Q| \cdot |\Sigma|^n \cdot (|Q| + n)$. Let us define

$$\mu = \frac{1}{|Q|^{|Q|\cdot|\Sigma|^n\cdot(|Q|+n)+1}}.$$

Lemma 8.2. *For all* $m \geqslant 0$ *we have* $\Pr_{\pi \in R_m}[H_m \mid \forall i \leqslant m-1 : \overline{H_i}] \geqslant \mu$.

Proof. In the following, let π be a random run from R_m and let π_i be the subrun on w_i. Notice that under the assumption that the event [π_{m-1} ends in p_m] holds, the events [$\pi_m = \pi^*_m$] and [$\forall i \leqslant m-1 : \overline{H_i}$] are conditionally independent.[1] Thus, we have

$$\begin{aligned} & \Pr_{\pi \in R_m}[\pi_m = \pi^*_m \mid \pi_{m-1} \text{ ends in } p_m \wedge \forall i \leqslant m-1 : \overline{H_i}] \\ = & \Pr_{\pi \in R_m}[\pi_m = \pi^*_m \mid \pi_{m-1} \text{ ends in } p_m]. \end{aligned}$$

Since the event [$\pi_m = \pi^*_m$] implies the event [π_{m-1} ends in p_m], we obtain:

$$\begin{aligned} & \Pr_{\pi \in R_m}[H_m \mid \forall i \leqslant m-1\colon \overline{H_i}] \\ \geqslant & \Pr_{\pi \in R_m}[\pi_m = \pi^*_m \mid \forall i \leqslant m-1\colon \overline{H_i}] \\ = & \Pr_{\pi \in R_m}[\pi_m = \pi^*_m \wedge \pi_{m-1} \text{ ends in } p_m \mid \forall i \leqslant m-1\colon \overline{H_i}] \\ = & \Pr_{\pi \in R_m}[\pi_m = \pi^*_m \mid \pi_{m-1} \text{ ends in } p_m \wedge \forall i \leqslant m-1\colon \overline{H_i}] \cdot \\ & \Pr_{\pi \in R_m}[\pi_{m-1} \text{ ends in } p_m \mid \forall i \leqslant m-1\colon \overline{H_i}] \\ = & \Pr_{\pi \in R_m}[\pi_m = \pi^*_m \mid \pi_{m-1} \text{ ends in } p_m] \cdot \\ & \Pr_{\pi \in R_m}[\pi_{m-1} \text{ ends in } p_m \mid \forall i \leqslant m-1\colon \overline{H_i}] \\ \geqslant & \Pr_{\pi_m \in \mathrm{Runs}(p_m, w_m)}[\pi_m = \pi^*_m] \cdot \frac{1}{|Q|} \\ \geqslant & \frac{1}{|Q|^{|w_m|+1}} \geqslant \mu \end{aligned}$$

This proves the lemma. □

Lemma 8.3. $\Pr_{\pi \in R_m}[\pi$ *is* δ*-universal*$] \geqslant \Pr_{\pi \in R_m}[\exists i \leqslant m\colon H_i] \geqslant 1 - (1-\mu)^m$.

Proof. The first inequality follows from the definition of the event H_i. Moreover, we have

$$\begin{aligned} \Pr_{\pi \in R_m}[\exists i \leqslant m\colon H_i] = & \Pr_{\pi \in R_m}[\exists i \leqslant m-1\colon H_i] + \\ & \Pr_{\pi \in R_m}[H_m \mid \forall i \leqslant m-1\colon \overline{H_i}] \cdot \Pr_{\pi \in R_m}[\forall i \leqslant m-1\colon \overline{H_i}] \\ = & \Pr_{\pi \in R_{m-1}}[\exists i \leqslant m-1\colon H_i] + \\ & \Pr_{\pi \in R_m}[H_m \mid \forall i \leqslant m-1\colon \overline{H_i}] \cdot \Pr_{\pi \in R_{m-1}}[\forall i \leqslant m-1\colon \overline{H_i}] \\ \geqslant & \Pr_{\pi \in R_{m-1}}[\exists i \leqslant m-1\colon H_i] + \mu \cdot \Pr_{\pi \in R_{m-1}}[\forall i \leqslant m-1\colon \overline{H_i}]. \end{aligned}$$

[1] Two events A and B are conditionally independent assuming event C if $\Pr[A \wedge B \mid C] = \Pr[A \mid C] \cdot \Pr[B \mid C]$, which is equivalent to $\Pr[A \mid B \wedge C] = \Pr[A \mid C]$.

Define $r_m = \Pr_{\pi \in R_m}[\exists i \leqslant m \colon H_i]$. We get

$$r_m \geqslant r_{m-1} + \mu \cdot (1 - r_{m-1}) = (1-\mu) \cdot r_{m-1} + \mu.$$

Since $r_0 = 0$, we get $r_m \geqslant 1 - (1-\mu)^m$ by induction. □

Theorem 8.4. *There exists* $q_0 \in Q$ *such that* $\mathcal{D}_n = (Q, \Sigma, q_0, \delta, o)$ *is a deterministic SW-algorithm for* Φ *and window size* n.

Proof. We use the probabilistic method. With Lemma 8.3 we get

$$\begin{aligned} & \Pr_{\pi \in R_m}[\pi \text{ is strictly correct for } \Phi \text{ and } \delta\text{-universal}] \\ = \; & 1 - \Pr_{\pi \in R_m}[\pi \text{ is not strictly correct for } \Phi \text{ or is not } \delta\text{-universal}] \\ \geqslant \; & 1 - \Pr_{\pi \in R_m}[\pi \text{ is not strictly correct for } \Phi] - \Pr_{\pi \in R_m}[\pi \text{ is not } \delta\text{-universal}] \\ \geqslant \; & \Pr_{\pi \in R_m}[\pi \text{ is } \delta\text{-universal}] - \lambda \\ \geqslant \; & 1 - (1-\mu)^m - \lambda. \end{aligned}$$

We have $1 - (1-\mu)^m - \lambda > 0$ for $m > \log(1-\lambda)/\log(1-\mu)$ (note that $\lambda < 1$ and $0 < \mu < 1$). Hence there exists an $m \geqslant 0$ and a strictly correct run $\pi \in R_m$ which is δ-universal. The statement follows directly from Lemma 8.1. □

Corollary 8.5. *There exists a deterministic sliding window algorithm* $\mathcal{D}_n$ *for* Φ *and window size* n *such that* $s(\mathcal{D}_n) \leqslant s(\mathcal{P}_n)$.

8.3 Polynomially long streams

The word $w_1 w_2 \cdots w_m$ (with $m > \log(1-\lambda)/\log(1-\mu)$), for which there exists a strictly correct and δ-universal run has a length that is exponential in the window size n. In other words: We need words of length exponential in n in order to transform a strictly λ-correct randomized SW-algorithm into an equivalent deterministic SW-algorithm. We remark that this is unavoidable: if we restrict to inputs of polynomial length then strictly λ-correct SW-algorithms can yield a proper space improvement over deterministic SW-algorithms.

Take the language $K_{\text{pal}} = \{ww^R \mid w \in \{a,b\}^n\}$ of all palindromes of even length, which belongs to the class **DLIN** of deterministic linear context-free languages, and let $L = \$K_{\text{pal}}$.

Lemma 8.6. *Any deterministic sliding window algorithm for* L *and window size* $2n+1$ *uses* $\Omega(n)$ *space.*

Proof. Let $\mathcal{D}_{2n+1}$ be a deterministic SW-algorithm for L and window size $2n+1$, and take two distinct words $\$x$ and $\$y$ where $x, y \in \{a,b\}^n$. Since $\mathcal{D}_{2n+1}$ accepts $\$xx^R$ and rejects $\$yx^R$, the algorithm reaches two different states on the inputs $\$x$ and $\$y$. Therefore, $\mathcal{D}_{2n+1}$ must have at least $|\{a,b\}^n| = 2^n$ states. □

Let us now fix a polynomial $p(n)$.

Lemma 8.7. *Let $n \in \mathbb{N}$ be a window size. There is a randomized streaming algorithm $\mathcal{P}_n$ with $s(\mathcal{P}_n) = O(\log n)$ such that*

$$\Pr_{\pi \in \mathrm{Runs}(\mathcal{P}_n, w)} [\pi \textit{ is strictly correct for } L] \geqslant 1 - 1/n$$

for all $w \in \Sigma^*$ with $|w| \leqslant p(n)$.

Proof. Babu et al. [14] have shown that for every language $K \in$ **DLIN** there exists a randomized streaming algorithm using space $O(\log n)$ which, given an input w of length n,

- accepts with probability 1 if $w \in K$,
- and rejects with probability at least $1 - 1/n$ if $w \notin K$.

We remark that the algorithm needs to know the length of w in advance. To stay consistent with our definition, we view the algorithm above as a family $(\mathcal{S}_n)_{n \geqslant 0}$ of randomized streaming algorithms $\mathcal{S}_n$. Furthermore, it is easy to see that the error probability $1/n$ can be further reduced to $1/n^d$ where $p(n) \leqslant n^d$ for sufficiently large n (by picking random primes of size $\Theta(n^{d+1})$ in the proof from [14]).

Now we prove our claim for $L = \$K_{\mathrm{pal}}$. The streaming algorithm $\mathcal{P}_n$ for window size n works as follows: After reading a \$-symbol, the algorithm $\mathcal{S}_{n-1}$ from above is simulated on the longest factor from $\{a, b\}^*$ that follows. Simultaneously we maintain the length ℓ of the maximal suffix over $\{a, b\}$, up to n, using $O(\log n)$ bits. If ℓ reaches $n - 1$, then $\mathcal{P}_n$ accepts if and only if $\mathcal{S}_{n-1}$ accepts. Notice that R_n only errs if the stored length is $n - 1$ (with probability $1/n^d$), which happens at most once in every n steps. Therefore the number of time instants where $\mathcal{P}_n$ errs on w is bounded by $|w|/n \leqslant n^d/n = n^{d-1}$. By the union bound we have for every stream $w \in \{\$, a, b\}^{\leqslant p(n)}$:

$$\Pr_{\pi \in \mathrm{Runs}(\mathcal{P}_n, w)} [\pi \text{ is not strictly correct for } L] \leqslant n^{d-1} \cdot \frac{1}{n^d} = \frac{1}{n}.$$

This concludes the proof. □

8.4 Conclusion

To the best of our knowledge, this is the first investigation on the general power of randomness in sliding window algorithms. We emphasize that our proof does not utilize Yao's minimax principle, which would require the choice of a "hard" distribution of input streams specific to the problem. It remains open, whether such a hard distribution exists for every approximation problem.

The "easy" direction of Yao's minimax principle can be viewed as a weak derandomization result: For any randomized algorithm (e.g. streaming algorithm, communication protocol, Boolean circuit) which computes a function f with high probability there exists a deterministic algorithm with essentially the same complexity which computes f on almost all inputs. However, this argument does not give a way to construct this deterministic algorithm. Shaltiel [103] proved that, amongst others, for a given polynomial-time computable family of randomized streaming algorithms (one for each stream length) computing functions f_n there is a polynomial-time computable family of deterministic streaming algorithms which compute f_n on almost all inputs.

In the proof we construct for each randomized SW-algorithm a strictly correct "worst case" stream (the δ-universal run), from which we can extract a deterministic SW-algorithm. It would be interesting whether such a worst case stream can be constructed independently from the randomized SW-algorithm.

Let us also remark that it is crucial for our proof that the input alphabet (i.e. the set of data values in the input stream) is finite. This is for instance the case when counting the number of 1's in a 0/1-sliding window. On the other hand, the problem of computing the sum of all data values in a sliding window of arbitrary numbers (a problem that is considered in [36] as well) is not covered by our setting, unless one puts a bound on the size of the numbers in the input stream.

Chapter 9

Context-free languages

9.1 Introduction

In this chapter we investigate to which extent the results for regular languages can be generalized to context-free languages. Our first main result (Theorem 9.1) states that any context-free language L with space complexity $F_L(n) \leqslant \log n - \omega(1)$ must be regular. By Theorem 4.16 the space complexity $F_L(n)$ is indeed constant and hence L is a Boolean combination of suffix-testable languages and regular length languages by Theorem 4.19. Our proof uses a variant of the classical pumping lemma. The crucial observation is that taking a reversed Greibach normal form grammar for L, we can ensure that pumping in a word of length n does not affect a suffix of length $o(n)$.

Theorem 9.1 shows that, analogous to regular languages, there is a gap between $O(1)$ and $O(\log n)$ in the space complexity spectrum for context-free languages. This leads to the question whether there is also a gap between $O(\log n)$ and $O(n)$ (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 this way we obtain for every $c \in \mathbb{N}$ a context-free language, whose optimal sliding window algorithm uses space $O(n^{1/c})$ (Theorem 9.10). This result holds for both the fixed-size and the variable-size model.

The context-free languages from the proof of Theorem 9.9 are nondeterministic. 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 between $\log n$ and n. We answer this question positively by constructing deterministic one-counter languages with sliding window space complexity $(\log n)^2$. Again this result holds for both fixed-size windows (Corollary 9.13) and variable-size windows (Theorem 9.15).

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. Since the arguments for our space trichotomy result for regular languages mainly use a DFA for the reverse language, one might think that these arguments extend to reversals of deterministic context-free languages.

The results of this chapter have appeared in [G5].

9.2 Below logarithmic space

In this section we prove the following result:

Theorem 9.1. *If* L *is a context-free language with* $F_L(n) = \log n - \omega(1)$*, then* L *is regular and* $F_L(n) = O(1)$.

We start with some definitions. Recall that a language is k-suffix testable if for all $u, v, w \in \Sigma^*$ where $|w| = k$ we have

$$uw \in L \iff vw \in L.$$

Let $f\colon \mathbb{N} \to \mathbb{N}$ 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

$$uw \in L \iff vw \in L.$$

If L is k-suffix definable for a constant $k \in \mathbb{N}$ then it is also k-suffix definable, but not vice versa. For example, if L is the union of $\{a, b\}^* a$ and the set of all words over $\{a, b\}$ with even length, then L is 1-suffix definable but not 1-suffix testable. Similarly, one defines prefix testable and f-prefix definable languages. We prove Theorem 9.1 in two steps:

Theorem 9.2. *Every language* $L \subseteq \Sigma^*$ *is* $2^{F_L(n)}$*-suffix definable.*

Proof. By Lemma 4.21 for all $n \in \mathbb{N}$ the language $SW_n(L) = \{w \in \Sigma^* \mid \text{last}_n(w) \in L\}$ is $2^{F_L(n)}$-suffix testable For all words $u, v, w \in \Sigma^*$ such that $|uw| = |vw| = n$ and $|w| = 2^{F_L(n)}$ we have $\text{last}_n(uw) = uw$, $\text{last}_n(vw) = vw$ and

$$uw \in L \iff uw \in SW_n(L) \iff vw \in SW_n(L) \iff vw \in L,$$

which shows that L is $2^{F_L(n)}$-suffix definable. □

Theorem 9.3. *If a context-free language* L *is* f*-suffix definable for a function* $f(n) = o(n)$*, then* L *is a finite Boolean combination of suffix testable languages and regular length languages.*

We remark that the requirement $f(n) = o(n)$ above cannot be relaxed: For every $k \geqslant 1$, the language $\{xay \mid x, y \in \{a, b\}^*, |x| = k|ay|\}$ is context-free and $\lceil n/(k+1) \rceil$-suffix definable but not even regular.

Combining Theorem 9.2 and Theorem 9.3 yields Theorem 9.1: If a context-free language L satisfies $F_L(n) = \log n - \omega(1)$ then L is f-suffix definable for a function $f(n) = o(n)$ by Theorem 9.2. Theorem 9.3 implies that L is a finite Boolean combination of suffix testable languages and regular length languages. Hence L is regular and $F_L(n) = O(1)$. The rest of this section is devoted to the proof of Theorem 9.3.

We prove the variant of Theorem 9.3 that talks about prefix-definability. This makes no difference, since the reversal of a context-free language is again context-free. First, we show that in the proof of Theorem 9.3 we can restrict ourselves to functions f with the following property: A monotonic function $f\colon \mathbb{N} \to \mathbb{N}$ has the *increasing plateau property* if for every $k \geqslant 1$ there exists an n_0 such that for all $n \geqslant n_0$ we have: $f(n+k) - f(n) \leqslant 1$.

Lemma 9.4. *If a monotonic function* f *has the increasing plateau property then* $f(n) = o(n)$.

Proof. Let $k \geqslant 1$. The increasing plateau property implies that there exists an n_0 such that for all $n \geqslant n_0$ and $t \in \mathbb{N}$ we have $f(n + tk) - f(n) \leqslant t$ and hence

$$\begin{aligned} f(n) = f(n_0 + \frac{n - n_0}{k} k) &\leqslant f(n_0 + \lceil \frac{n - n_0}{k} \rceil k) \\ &\leqslant f(n_0) + \lceil \frac{n - n_0}{k} \rceil \leqslant \frac{n}{k} + f(n_0) - \frac{n_0}{k} + 1. \end{aligned}$$

By choosing $n_0' \geqslant n_0$ such that $f(n_0) - n_0/k + 1 \leqslant n_0'/k$ we can bound

$$f(n) \leqslant \frac{n}{k} + \frac{n_0'}{k} \leqslant \frac{2n}{k}$$

for all $n \geqslant n_0'$. This proves that $f(n) = o(n)$. □

Lemma 9.5. *Let* $f\colon \mathbb{N} \to \mathbb{R}_{\geqslant 0}$. *If* $f(n) = o(n)$ *then there exists a monotonic function* $g\colon \mathbb{N} \to \mathbb{N}$ *with the increasing plateau property and such that* $f(n) \leqslant g(n)$ *for all* $n \in \mathbb{N}$.

Proof. For a linear function $g\colon \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ of the form $g(x) = \alpha \cdot x + \beta$ we call α 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 $h\colon \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ 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 $g\colon \mathbb{N} \to \mathbb{N}$ with $g(n) = \lceil h(n) \rceil$ has the properties from the lemma.

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

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

For the induction step, assume that n_i and the linear function $h_i : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ (of slope $1/i$) are defined such that $f(n) \leqslant h_i(n)$ for all $n \geqslant n_i$. 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 number n_{i+1} such that $n_{i+1} > n_i$ and $u_{i+1}(n) \geqslant f(n)$ for each $n \geqslant n_{i+1}$. This holds because $f(n) = o(n)$, and hence for any constants $\alpha > 0, \beta \in \mathbb{R}$ we have $f(n) \leqslant \alpha \cdot n + \beta$ 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 $n \geqslant n_{i+1}$ we have

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

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

We now define the function $h\colon \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$:

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

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 $f(n) \leqslant h_i(n)$ for all $n \geqslant n_i$ and $f(n) \leqslant f(n_1) \leqslant n_1$ for all $n \leqslant n_1$, we have $f(n) \leqslant h(n) \leqslant g(n)$ 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 $k \geqslant 1$ and $n \geqslant n_k$. Since h is continuous and piecewise linear with slopes $\leqslant 1/k$ on $[n_k, \infty)$, we have $h(n+k) - h(n) \leqslant (n+k-n)/k = 1$. This implies $g(n+k) - g(n) \leqslant 1$. □

Consider the following variant of the pumping lemma for context-free languages (see also [66, 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.

Lemma 9.6. *Given a context-free grammar $\mathcal{G} = (N, \Sigma, S, P)$ and let $L(A) = \{w \in \Sigma^* \mid A \overset{*}{\Rightarrow}_{\mathcal{G}} w\}$. Then there exists a natural number c_1 depending only on $\mathcal{G}$ and not on A, such that every word $w \in L(A)$ with $|w| \geqslant c_1$ can be written as $w = w_1w_2w_3w_4w_5$ with:*

(i) $w_1w_2^kw_3w_4^kw_5 \in L(A)$ *for every* $k \geqslant 0$,

(ii) $|w_2w_3w_4| \leqslant c_1$,

(iii) and $|w_2w_4| > 0$.

In particular, the word $w_1 w_2^{1+c_1!/|w_2w_4|} w_3 w_4^{1+c_1!/|w_2w_4|} w_5$ *of length* $|w| + c_1!$ *belongs to* $L(A)$.

Lemma 9.7. *Let* L *be a context-free language and* $f\colon \mathbb{N} \to \mathbb{N} \setminus \{0\}$ *be monotonic with* $f(n) = o(n)$. *There are constants* n_0 *and* $c > 0$ *(only depending on* L *and* f*) such that the following hold for every* $n \geqslant n_0$*:*

- $n \geqslant f(n) + c$ *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. Let $\mathcal{G} = (N, \Sigma, P, S)$ be a grammar for L in Greibach normal form, i.e. all productions are of the form $A \to aA_1 \cdots A_k$ for $k \geqslant 0$, nonterminals $A, A_1, \ldots, A_k$ and a terminal a (such a grammar exists for every context-free language); see also [66, Chapter 4.6]. Let $r = \max_{A \to \alpha \in P} |\alpha|$ be the maximal length of the productions' right-hand sides and let c_1 be the constant from the above pumping lemma for $\mathcal{G}$. We can assume that $r \geqslant 2$, otherwise L is finite and the lemma holds. Define $c = c_1!$ and choose an n_0 such that for all $n \geqslant n_0$ the following three inequalities hold:

$$\frac{n}{f(n)} > 1 + (r-1)r^{2|N| \cdot (rc_1|N|+1)!} \tag{9.1}$$

$$\frac{n}{f(n)} > 1 + c_1(r-1) \tag{9.2}$$

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

As the right-hand sides are constant and $f(n) = o(n)$, such an n_0 exists. Hence, for all $n \geqslant n_0$ the following two inequalities hold ((9.1) is equivalent to (9.3) and (9.2) is equivalent to (9.4)):

$$\log_r \left(\frac{n - f(n)}{f(n)(r-1)} \right) > 2|N|(rc_1|N| + 1)! \tag{9.3}$$

$$\frac{n - f(n)}{f(n)(r-1)} > c_1 \tag{9.4}$$

Consider a string uv of length $n \geqslant n_0$ generated by $\mathcal{G}$, 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 $\mathcal{G}$ is in Greibach normal form. Apart from the prefix u, the rest of the sentential form has length at most $1 + f(n)(r-2) \leqslant f(n)(r-1)$ 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

$$|x| \geqslant \frac{n - f(n)}{f(n)(r-1)} \overset{(9.4)}{>} c_1. \tag{9.5}$$

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 $uv'' \in L$. Since $|x| \geqslant c_1$, 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 $uv' \in L$, where $|v'| = |v| - c$. Again, consider the nonterminal A that generates a string x satisfying (9.5). Since the length of each right-hand side is at most r, there is a path Π 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 (9.3). We are going to color some nodes on the path Π black or gray: if a node v on Π has a child that does not belong to Π 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 Π (v above v') such that (i) v and v' are labeled 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' gray.

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 gray 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 Π.

- For the second kind of black nodes, let v and v' be the corresponding nodes colored black and gray, respectively. We can delete the subtree rooted in v and replace it with the one rooted in v'. The length of the path is $\leqslant |N|$, the arity of the rules $\leqslant r$ and each deleted nonterminal derived a string of length $\leqslant c_1$ (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 $uv' \in L$ 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 $uz \in L$ such that $|uz| = n + (rc_1|N|)! - c$ and then use the argument above to obtain a word $uv' \in L$ 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. $\square$

Lemma 9.8. *Let L be a context-free language that is f-prefix definable for a function $f(n) = o(n)$. Then there exists a constant α such that L is α-prefix definable.*

Proof. By Lemma 9.5 there exists a monotonic function $g(n) = o(n)$ having the increasing plateau property and such that $f(n) \leqslant g(n)$ for all $n \geqslant 0$. Hence, L is still g-prefix definable. Moreover, let $f' = o(n)$ be defined by $f'(n) = g(n) + 1$ for all n. Take the constants n_0 and c from Lemma 9.7 for L and f' (instead of f). Choose m such that (i) $m \geqslant n_0 + c$ and (ii) $g(n) - g(n - c) \leqslant 1$ for all $n \geqslant m$, 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. Assume $n \leqslant m$, which implies $g(n) \leqslant g(m) = |u|$. 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. Assume $n > m$, and thus $n > n_0 + c$ and $g(n) \geqslant g(m) = |u|$. Since $n - g(m) \geqslant n - g(n) = n - f'(n) + 1 > c > 0$, we can write $v = v_1 a v_2$ and $w = w_1 b w_2$ such that $a \neq b \in \Sigma$ and $|uv_1| = |uw_1| = g(n)$. Thus, $|uv_1 a| = |uw_1 b| = f'(n)$. By Lemma 9.7 there exists a word v_2' with $|v_2'| = |v_2| - c$ and $uv_1 a v_2' \in L$. Take any word w_2' of length $|w_2'| = |w_2| - c$. By the length-minimality of v and w we must have $uw_1 b w_2' \in L$ (note that $c > 0$). Note that $|uw_1 b w_2'| = |uw| - c = n - c > n_0$. Therefore, we can apply Lemma 9.7 to the word $uw_1 b w_2'$. Note that $g(n) - g(n - c) \leqslant 1$ since $n \geqslant m$. Thus, $f'(n - c) = g(n - c) + 1 \geqslant g(n)$ and the prefix of $uw_1 b w_2'$ of length $f'(n - c)$ starts with uw_1. We can conclude with Lemma 9.7 that there is a word w_2'' such that $uw_1 w_2'' \in L$ and $|uw_1 w_2''| = n$. But since $|uw_1| = g(n)$ and $|uw_1 w_2''| = n$, the g-prefix definability of L implies that $uw_1 y \in L$ for all words y of length $n - g(n)$. In particular, we get $uw_1 b w_2 = uw \in L$, which is a contradiction. □

We can now prove (the prefix version of) Theorem 9.3:

Proof of Theorem 9.3. Let L be a f-prefix definable context-free language with $f(n) = o(n)$. Let α be the constant from Lemma 9.8. For every word u of length α the language $u^{-1}L$ is context-free and by Lemma 9.8 it is a length language. It is a direct consequence of Parikh's theorem (or the fact that every unary context-free language is regular) that a context-free length language is regular. Hence, every $u^{-1}L$ (for $|u| = \alpha$) is a regular length language. We can now decompose L as follows:

$$L = (L \cap \Sigma^{<\alpha}) \cup \bigcup_{u \in \Sigma^{\alpha}} u(u^{-1}L) = (L \cap \Sigma^{<\alpha}) \cup \bigcup_{u \in \Sigma^{\alpha}} (u\Sigma^* \cap \Sigma^{\alpha}(u^{-1}L)).$$

Since $u^{-1}L$ is a regular length language, also $\Sigma^{\alpha}(u^{-1}L)$ 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. □

9.3 Above logarithmic space

In this section, we show that the trichotomy result for regular languages does not carry over to the context-free languages. More precisely, we show that for every natural number $c \geqslant 1$ there exists a one-counter language L_c such that $F_{L_c}(n) = O(n^{1/c})$ and $F_{L_c}(n) = \Omega(n^{1/c})$ infinitely often. Furthermore it satisfies $V_{L_c}(n) = \Theta(n^{1/c})$. 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 [66, Chapter 9.3]. We first show the following technical result:

Theorem 9.9. *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*

$$f(n) = \begin{cases} k, & \textit{if } n = k(t(k)+3) \textit{ for some } k,^1 \\ 0, & \textit{else,} \end{cases}$$

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

Proof. Let Γ 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 $\Delta := \Gamma \cup (Q \times \Gamma)$. We reserve a binary alphabet $\{0,1\}$ which is disjoint from Δ. A computation of M on an input a^k ($k \geqslant 1$) is a sequence of configurations $c_0 \vdash_M \cdots \vdash_M c_{t(k)}$ where $|c_i| = k$ for all $1 \leqslant i \leqslant t(k)$, $c_0 = (q_0, a)a^{k-1}$ is the start configuration on input a^k, every c_{i+1} is obtained from c_i by applying a transition of M for $0 \leqslant i \leqslant t(k)-1$, and $c_{t(k)}$ is an accepting configuration. Let $\overline{\Delta} = \{\overline{x} \mid x \in \Delta\}$ be a disjoint copy of Δ and define the function

$$\overline{\cdot} \colon \Delta^* \to \overline{\Delta}^*, \quad \overline{a_1 \cdots a_n} = \overline{a_n} \cdots \overline{a_1}.$$

Finally, let K be the set of all words

$$c_0\, \overline{c_1}\, c_2\, \overline{c_3} \cdots c_{t(k)}\, s\, s^R \text{ or} \tag{9.6}$$

$$c_0\, \overline{c_1}\, c_2\, \overline{c_3} \cdots \overline{c_{t(k)}}\, s\, s^R \tag{9.7}$$

such that $k \geqslant 1$, $c_0 \vdash_M \cdots \vdash_M c_{t(k)}$ is a computation on input a^k, $s \in \{0,1\}^*$ is an arbitrary word of length k, and $t(k)$ even in case (9.6) and odd in case (9.7). Notice that the words in (9.6) and (9.7) have length $k(t(k)+3)$. We can assume

[1]Since $t(k)$ is monotonically increasing, this number k is unique.

that M never goes back to the initial state q_0. This ensures that every word has at most one nonempty 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. Factorize $w = u_1 \cdots u_m$ into blocks u_i which is a maximal factor from $\Delta^+ \cup \overline{\Delta}^+ \cup \{0,1\}^+$ in w. Possible errors are the following:

1. $m < 2$,
2. u_1 is not an initial configuration, i.e. of the form $(q_0, a)a^{k-1}$ for some $k \geqslant 1$,
3. for some odd $i < m$, u_i is not a configuration,
4. for some even $i < m$, $\overline{u_i}$ is not a configuration,
5. u_{m-1} is not an accepting configuration,
6. there exists $1 \leqslant i < m-1$ with $|u_i| \neq |u_{i+1}|$,
7. $|u_m| \neq 2|u_{m-1}|$,
8. there exists $1 \leqslant i < m-1$ odd such that $u_i \vdash_M \overline{u_{i+1}}$ does not hold,
9. there exists $1 \leqslant i < m-1$ even such that $\overline{u_i} \vdash_M u_{i+1}$ does not hold,
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 $O(g(n))$. Assume that $w = a_1 \cdots a_n$ is the active window. The algorithm stores the following data n, i, t, k, ℓ, s:

- $n = |w|$ is the length of the active window.
- i is the smallest position $1 \leqslant i \leqslant n+1$ such that $a_i \cdots a_n$ is a prefix of a word from K. If this prefix is empty, then $i = n+1$.
- t is the number of blocks in $a_i \cdots a_n$ 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 such that $a_i \cdots a_n$ starts with $(q_0, a)a^{k-1}$; hence, k tells us the input length.
- In case $1 \leqslant t \leqslant t(k)$, ℓ is the length of the last block of $a_i \cdots a_n$ (if $t = 0$ or $t = t(k)+1$ we store some dummy value in ℓ).
- In case $t = t(k)+1$, s is the maximal suffix of $a_i \cdots a_n$ 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 $1 \leqslant t \leqslant t(k)$ 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 overlined) $(\ell+1)$-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 ℓ need $O(\log n)$ bits. Recall that s has maximal length k. But we only store symbols in s if $n \geqslant k(t(k)+1) \geqslant \lfloor k/3 \rfloor (t(\lfloor k/3 \rfloor)+3)$, since the window must already contain a complete computation on input a^k before s becomes nonempty. We get $\lfloor k/3 \rfloor = f(\lfloor k/3 \rfloor (t(\lfloor k/3 \rfloor)+3)) \leqslant g(n)$, i.e. $k \leqslant 3g(n)+3$. Finally, since $g(n)$ is the maximal value k such that $k(t(k)+3) \leqslant n$ and $t(k) \in 2^{O(k)}$, we get $g(n) = \Omega(\log n)$. This shows that the algorithm works in space $O(g(n))$.

To show that $F_K(n) = O(f(n))$ 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 memory is needed. Otherwise, since $t(k)$ is monotonically increasing, there is a unique k with $n = k(t(k)+3)$.

Finally, we show that $F_K(n) \geqslant f(n)$ for all $n \in \mathbb{N}$, which implies that $V_K(n) \geqslant g(n)$ for all $n \in \mathbb{N}$ since $V_K(n) \geqslant F_K(n)$ 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

$$w(s) := 0^k \, c_0 \, \overline{c_1} \, c_2 \, \overline{c_3} \cdots c_{t(k)} \, s$$

for $s \in \{0,1\}^k$. The length of these words is $n = k(t(k)+3)$, which is the window size.

Consider a sliding window algorithm $\mathcal{P}_n$ for the language K and window size n. We have $\mathcal{P}_n(w(s)) \neq \mathcal{P}_n(w(u))$ for all $s, u \in \{0,1\}^k$ with $s \neq u$ because $\mathrm{last}_n(w(s)s^R) \in K$ and $\mathrm{last}_n(w(u)s^R) \notin K$. Hence, $\mathcal{P}_n$ must distinguish 2^k many streams, and thus $F_K(n) \geqslant k = f(n)$. □

Theorem 9.9 yields quite a 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 $c \in \mathbb{N}$, $c \geqslant 1$:

Theorem 9.10. *For every $c \in \mathbb{N}$, $c \geqslant 1$, there exists a one-counter language L_c such that $F_{L_c}(n) = O(n^{1/c})$, $F_{L_c}(n) = \Omega(n^{1/c})$ infinitely often, and $V_{L_c}(n) = \Theta(n^{1/c})$.*

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 9.9 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 $f(n) = O(n^{1/c})$, $f(n) = \Omega(n^{1/c})$ infinitely often, and $g(n) = \Theta(n^{1/c})$ for the mapping $g(n)$ from Theorem 9.9. Hence, by Theorem 9.9 there is a one-counter language L_c with the properties stated in the theorem. □

To fully exploit Theorem 9.9 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.

9.4 Deterministic one-counter languages

The context-free language L_c from Theorem 9.10 is not deterministic context-free and it is open whether the deterministic context-free languages also have such a dense spectrum of space complexity functions. In this section we exhibit a deterministic one-turn one-counter language with space complexity $\Theta((\log n)^2)$ in the variable-size and in the 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 [58].

We start with the variable-size model. In the following a maximal factor β in a word $w \in \{a, b\}^*$ of the form $\beta = ab^i$ is called a *block* of length $i + 1$ in w.[2] Define the language

$$L = \{ab^k av \mid k \geqslant 0,\ v \in \{a, b\}^{\leqslant k}\} \cup ab^*, \tag{9.8}$$

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 $\geqslant |w|/2$.

Lemma 9.11. *We have* $V_L(n) = O((\log n)^2)$.

Proof. Any word $w \in \{a, b\}^*$ can be uniquely factorized as $w = b^s \beta_m \cdots \beta_2 \beta_1$ where $s, m \geqslant 0$ and each β_i is a block. A block β_i is *relevant* if it is at least as long as the remaining suffix, i.e. $|\beta_i| \geqslant \sum_{j=1}^{i-1} |\beta_j|$. For an active window $w \in \{a, b\}^*$ our variable-size algorithm maintains the window size and for each relevant block β_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-symbol arrives, we create a new relevant block of length 1. If a b-symbol arrives, we prolong the rightmost relevant

[2]This notion is not related to the blocks used in the proof of Theorem 9.9).

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 $O((\log n)^2)$, it suffices to show that each word $w \in \{a,b\}^n$ has $O(\log n)$ 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 $|\gamma_i| \geqslant \sum_{j=1}^{i-1} |\gamma_j|$ for all $1 \leqslant i \leqslant k$. Inductively, we show that $|\gamma_i| \geqslant 2^{i-2}|\gamma_1|$ for all $2 \leqslant i \leqslant k$. This is immediate for $i = 2$ and for the induction step, observe that $|\gamma_i| \geqslant |\gamma_1| + \sum_{j=2}^{i-1} |\gamma_j| \geqslant |\gamma_1| + |\gamma_1| \sum_{j=2}^{i-1} 2^{j-2} = 2^{i-2}|\gamma_1|$ for all $i \geqslant 3$. This proves $k = O(\log n)$, which concludes the proof. $\square$

We remark that the structure of relevant blocks resembles the *exponential histogram* from [36], which stores exponentially growing buckets.

Lemma 9.12. *We have* $V_L(n) = \Omega((\log n)^2)$.

Proof. For each $k \geqslant 0$ 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, then $k \geqslant 1$ and there exists the maximal common suffix α_3 of α_1 and α_2 that is again an arrangement. Consider the suffixes of α_1, α_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 α_1 and α_2 differ, there exists a number $\ell \geqslant |\alpha_3|$ such that α_1 has the suffix $ab^\ell\alpha_3$ and α_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 α_1 and α_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 $|\alpha_3| \leqslant \ell$, 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. $\square$

Corollary 9.13. *There exists a deterministic one-turn one-counter language* L *such that* $V_L(n) = \Theta((\log n)^2)$.

The language L from (9.8) is an example where the space complexity in the fixed-size model is strictly below the space complexity in the variable-size model:

Lemma 9.14. *We have* $F_L(n) = O(\log n)$.

Proof. Let $n \geqslant 0$ 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 $O(\log n)$ 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 9.13 for the fixed-size model: For the language L from (9.8) let $K = Lc^*$, which is a deterministic one-turn one-counter language.

Theorem 9.15. *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 $O(\log n)$ 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 9.11 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 $O((\log n)^2)$.

The proof for the lower bound is similar to the proof of Lemma 9.12. Let k be maximal such that $3^k \leqslant n$. 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 9.12; we only have to replace every $\downarrow$-operation by the insertion of a c-symbol. □

Finally we will look at reversals of deterministic context-free languages since the proof of the trichotomy for regular languages also uses right-deterministic automata. For the language L from (9.8) let

$$L' = \{\#^j u^R v \$^i \mid u \in L,\ i \geqslant 0,\ v \in \{a, b\}^i,\ j \geqslant 1\}.$$

Its reversal L'^R is accepted by a deterministic one-counter three-turn automaton: The one-counter automaton first stores the number i of \$-symbols and ignores the next i symbols over $\{a, b\}$. Then it simulates the one-counter automaton for L, and finally reads a sequence of #-symbols.

Theorem 9.16. *We have* $V_{L'}(n) = O((\log n)^2)$*, and* $F_{L'}(n) = \Omega((\log n)^2)$ *for infinitely many* n.

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 $i, j \geqslant 0$. Using $O(\log n)$ 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 $j \geqslant 1$ we maintain for each relevant block in w^R its starting position and its length, which requires $O((\log n)^2)$ bits (see the proof of Lemma 9.11). 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 w^R contains no blocks. If $j \geqslant 1$ 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 w^R 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 w^R 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 9.12, there exists a number $0 \leqslant \ell < 3^k$ and an arrangement α_3 of length at most ℓ such that α_1 and α_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, $\alpha_1 = v_1ab^\ell\alpha_3$ and $\alpha_2 = v_2b^{\ell+1}\alpha_3$ 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

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

This shows that the algorithm must distinguish the words $\#^{3^k}\alpha_1^R$ and $\#^{3^k}\alpha_2^R$. Note that adding a \$ at the right end of the word removes the rightmost symbol (a or b) in the factor which has to belong to L^R in order to have a word from L'. The rest of the proof follows the arguments from the proof of Lemma 9.12. □

9.5 Conclusion

The results in this chapter indicate that, to a wide extent, the class of context-free languages is too rich for classification results in the sliding window model. We suggest to consider context-free languages in approximate settings (see Chapter 7) and deterministic context-free languages. A particular open question is: Is there a (reverse) deterministic context-free (one-counter) language whose space complexity (either $F_L(n)$ or $V_L(n)$) is between between $(\log n)^2$ and n, or between $\log n$ and $(\log n)^2$?

Aaronson, Grier and Schaeffer [1] study the *quantum query complexity* of formal languages and prove that every regular language has quantum query complexity $\Theta(1)$, $\tilde{\Theta}(\sqrt{n})$ or $\Theta(n)$. Furthermore they construct for every *limit computable* number $1/2 \leqslant c \leqslant 1$ a context-free language with quantum query complexity $\Theta(n^c)$. Here c is limit computable if there is a Turing machine which on input k computes a rational number c_k such that c_k tends to c for $k \to \infty$. Their proof technique could be used to extend Theorem 9.9.

Chapter 10

Visibly pushdown languages

In this chapter we extend the trichotomy for regular languages in the variable-size sliding window model to the class of *visibly pushdown languages*. This language class has already been considered under the name of *input-driven languages* in the 80's [20, 39] and was reintroduced by Alur and Madhusudan [9]. They are recognized by *visibly pushdown automata* where the alphabet is partitioned into *call letters*, *return letters* and *internal letters*, which determine the behavior of the stack height. Since visibly pushdown automata can be determinized, the class of visibly pushdown languages turns out to be very robust (it is closed under Boolean operations and other standard language operations) and more tractable in many algorithmic questions than the class of context-free languages [9]. The main theorem of this chapter is the following:

Theorem 10.1 (Trichotomy for VPL). *If L is a visibly pushdown language then $V_L(n)$ is either $O(1)$, $\Theta(\log n)$ or $\Theta(n)$ for infinitely many n.*

The proof uses the dichotomy theorem for rational functions (Theorem 5.2). A simple characterization of the $O(\log n)$-class as well as a study of the fixed-size model are left as open problems. The results of this chapter have appeared in [G6].

10.1 Visibly pushdown automata

A *pushdown alphabet* is a triple $\tilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_{int})$ consisting of three pairwise disjoint alphabets: a set of *call letters* Σ_c, a set of *return letters* Σ_r and a set of *internal letters* Σ_{int}. We identify $\tilde{\Sigma}$ with the union $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$. A *visibly pushdown automaton (VPA)* has the form $\mathcal{A} = (Q, \tilde{\Sigma}, \Gamma, \bot, q_0, \delta, F)$ where Q is a finite state set, $\tilde{\Sigma}$ is a pushdown alphabet, Γ is the finite stack alphabet containing a special symbol $\bot$ (representing the empty stack), $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta = \delta_c \cup \delta_r \cup \delta_{int}$ is the transition function where $\delta_c \colon Q \times \Sigma_c \to (\Gamma \setminus \{\bot\}) \times Q$, $\delta_r \colon Q \times \Sigma_r \times \Gamma \to Q$ and $\delta_{int} \colon Q \times \Sigma_{int} \to Q$. The set of *configurations* Conf is the set of all words αq where $q \in Q$ is a state and $\alpha \in \bot(\Gamma \setminus \{\bot\})^*$ is the *stack content*. We define $\delta \colon \mathrm{Conf} \times \Sigma \to \mathrm{Conf}$ for all $p \in Q$, $a \in \Sigma$, $\alpha \in \bot(\Gamma \setminus \{\bot\})^*$, $\gamma \in \Gamma$ as follows:

- If $a \in \Sigma_c$ and $\delta(p, a) = (\gamma, q)$ then $\delta(\alpha p, a) = \alpha\gamma q$.
- If $a \in \Sigma_{int}$ and $\delta(p, a) = q$ then $\delta(\alpha p, a) = \alpha q$.
- If $a \in \Sigma_r$, $\delta(p, a, \gamma) = q$ and $\gamma \in \Gamma \setminus \{\bot\}$ then $\delta(\alpha\gamma p, a) = \alpha q$.
- If $a \in \Sigma_r$ and $\delta(p, a, \bot) = q$ then $\delta(\bot p) = \bot q$.

As usual we inductively extend δ to a function $\delta \colon \mathrm{Conf} \times \Sigma^* \to \mathrm{Conf}$ where $\delta(c, \varepsilon) = c$ and $\delta(c, wa) = \delta(\delta(c, w), a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$. The *initial* configuration is $\bot q_0$ and a configuration c is *final* if $c \in \Gamma^* F$. A word $w \in \Sigma^*$ is *accepted* from a configuration c if $\delta(c, w)$ is final. The VPA $\mathcal{A}$ *accepts* w if w is accepted from the initial configuration. The set of all words accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$; the set of all words accepted from c is denoted by $L(c)$. A language L is a *visibly pushdown language (VPL)* if $L = L(\mathcal{A})$ for some VPA $\mathcal{A}$.

Example 10.2. Let $\tilde{\Sigma} = (\Sigma_c, \Sigma_r, \Sigma_{int})$ where $\Sigma_c = \{a\}$, $\Sigma_r = \{b\}$, $\Sigma_{int} = \emptyset$. Let L be the set of all words $w \in \{a, b\}^*$ whose maximal suffix of the form $a^i b^j$ satisfies $i \leqslant j$. We claim that L is a VPL. One can recognize L by a deterministic one-counter automaton which resets the counter if a new a-block is read. Since VPAs cannot clear the stack we need to push a special marker to the stack whenever new block of a-symbols starts. Let $\mathcal{A} = (Q, \tilde{\Sigma}, \Gamma, \bot, q_F, \delta, \{q_F\})$ be the following VPA: The state set is $Q = \{q_a, q_b, q_F\}$, q_F is the initial state and the only final state, the stack alphabet is $\Gamma = \{\bot, A, \underline{A}\}$, and the transitions are

$$\delta(q_F, a) = (q_a, \underline{A}) \qquad \delta(q_a, a) = (q_a, A) \qquad \delta(q_b, a) = (q_a, \underline{A})$$

and

$$\delta(q, b, \gamma) = \begin{cases} q_b, & \text{if } q \in \{q_a, q_b\} \text{ and } \gamma = A, \\ q_F, & \text{otherwise.} \end{cases}$$

One can also define nondeterministic visibly pushdown automata in the usual way, which can always be converted into deterministic ones [9]. This leads to good closure properties of the class of all VPLs.

Theorem 10.3 ([9]). *Nondeterministic VPAs can be made deterministic. The class of VPLs is closed under Boolean operations, concatenation and Kleene star.*

To exclude some pathological cases we assume that $\Sigma_c \neq \emptyset$ and $\Sigma_r \neq \emptyset$. In fact, if $\Sigma_c = \emptyset$ or $\Sigma_r = \emptyset$ then any VPL over that pushdown alphabet would be regular. The set of *well-matched* words W over Σ is defined as the smallest set which contains $\{\varepsilon\} \cup \Sigma_{int}$, is closed under concatenation, and if w is well-matched, $a \in \Sigma_c$, $b \in \Sigma_r$ then also awb is well-matched. The set W of well-matched words forms a submonoid of Σ^*. The *stack height function* $\mathsf{sh} \colon \Sigma^* \to \mathbb{Z}$ is the homomorphic extension of

$$\mathsf{sh}(a) = \begin{cases} +1, & \text{if } a \in \Sigma_c, \\ 0, & \text{if } a \in \Sigma_{int}, \\ -1, & \text{if } a \in \Sigma_r. \end{cases}$$

We emphasize that $\mathsf{sh}(w)$ does *not* specify the stack height in the reached configuration $\delta(\bot q_0, w)$ since sh does not truncate at stack height 0. A word w is well-matched if $\mathsf{sh}(w) = 0$ and $\mathsf{sh}(v) \geqslant 0$ for every prefix v of w. The black plot in Figure 10.1 displays the stack height function over all prefixes of an example word w. Furthermore, the gray lines display for all factors v of w the stack height of $\delta(\bot q_0, v)$: We start at stack height 0 (the gray dots) and follow the gray lines to the right.

Notice that a VPA can only see the top of the stack when reading return symbols. Therefore, the behavior of a VPA on a well-matched word is determined by the current state and independent of the current stack content. More precisely, there exists a monoid homomorphism $\theta \colon W \to Q^Q$ into the finite monoid of all state transformations $Q \to Q$ such that $\delta(\alpha p, w) = \alpha\theta(w)(p)$ for all $w \in W$ and $\alpha p \in \mathrm{Conf}$.

Monotonic factorizations A word is called *descending (ascending)* if it can be factorized into well-matched factors and return (call) letters. The set of descending words is denoted by D.

A factorization of $w = w_0 w_1 \cdots w_m \in \Sigma^*$ into factors $w_i \in \Sigma^*$ is *monotonic* if w_0 is descending (possibly empty) and for each $1 \leqslant i \leqslant m$ the factor w_i is either a call letter $w_i \in \Sigma_c$ or a nonempty well-matched factor. The descending prefix w_0 can be further factorized in well-matched factors and all return letters which do not have a matching call letter in w. The factors w_i which are call letters are precisely those call letters which do not have a matching return letter. The number of such unmatched call letters specifies the stack height in $\delta(\bot q_0, w)$.

Lemma 10.4. *Every word $w \in \Sigma^*$ has a monotonic factorization.*

Proof. Consider the set of nonempty maximal well-matched factors in w (maximal with respect to inclusion). Observe that two distinct maximal well-matched factors in a word cannot overlap because the union of two overlapping well-matched factors is again well-matched. Since every internal letter is well-matched the remaining positions contain only return and call letters. Furthermore, every remaining call letter must be to the right of every remaining return letter, which yields a monotonic factorization of w. □

Figure 10.1 shows a monotonic factorization $w = w_0 w_1 \cdots w_m$ where the descending prefix w_0 is colored red and call letters w_i are colored blue. If $w_0 w_1 \cdots w_m$ is a monotonic factorization then $w_i' w_{i+1} \cdots w_j$ is a monotonic factorization for any $0 \leqslant i \leqslant j \leqslant m$ and any suffix w_i' of w_i.

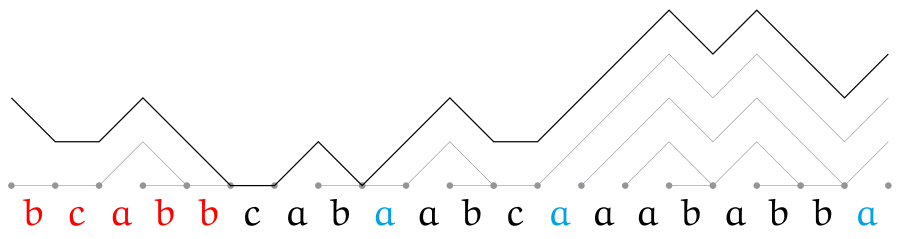

Figure 10.1: The stack height function for a word ($\Sigma_c = \{a\}$, $\Sigma_r = \{b\}$, $\Sigma_{int} = \{c\}$) and a monotonic factorization $bcabb\ cab\ a\ abc\ a\ aababb\ a$.

10.2 Description of the Myhill-Nerode classes

Fix a VPA $\mathcal{A} = (Q, \tilde{\Sigma}, \Gamma, \bot, q_0, \delta, F)$ and let $L = L(\mathcal{A})$ for the rest of this chapter. In the following we show how to describe the Myhill-Nerode classes of L using a regular set of representative configurations. This allows us to derive that the growth of $n \mapsto |\Sigma^{\leqslant n}/{\sim_L}|$ is polynomial or exponential. In the latter case this also yields an exponential lower bound for $|\Sigma^{\leqslant n}/{\approx_L}|$ since $\approx_L$ refines $\sim_L$.

Two configurations $c_1, c_2 \in \mathrm{Conf}$ are *equivalent*, denoted by $c_1 \sim c_2$, if $L(c_1) = L(c_2)$. We follow the approach in [15] of choosing length-lexicographic minimal configurations. Since their definition slightly differs from ours[1] we briefly recall their argument.

Lemma 10.5 ([15]). *The equivalence relation $\sim^R$ is synchronous rational.*

Proof. We present a finite automaton which recognizes the complement of $\sim^R$. It reads two reversed configurations $(\alpha p)^R = p\alpha^R$ and $(\beta q)^R = q\beta^R$ synchronously, which are aligned to the left. The automaton stores a pair of states of $\mathcal{A}$, starting with the pair (p, q). It then guesses a word w by its monotonic factorization which witnesses that w belongs to exactly one of the languages $L(\alpha p)$ and $L(\beta q)$. Notice that it suffices to read the maximal descending prefix of w and test whether the reached state pair (p', q') belongs to some fixed set of state pairs since the remaining ascending suffix cannot access the stack contents of the reached configurations. To simulate $\mathcal{A}$ on a descending prefix in each step the automaton either guesses a return symbol and removes the top most stack symbol of both configurations (or leaves $\bot$ at the top), or guesses a state transformation $\tau \in \theta(W)$ which only modifies the current state pair. □

Let $\mathrm{rConf} = \{\delta(\bot q_0, w) \mid w \in \Sigma^*\}$ be the set of all *reachable* configurations in $\mathcal{A}$, which is known to be regular [19, 26]. By fixing arbitrary linear orders on Γ and Q we can consider the length-lexicographical order $\leqslant_{llex}$ on rConf. It is well-known that $\leqslant_{llex}$ is a synchronous rational relation. Now let $\mathrm{rep}\colon \mathrm{rConf} \to \mathrm{rConf}$ be the function which chooses the minimal representative from each $\sim$-class, i.e. for all $c \in \mathrm{rConf}$ we have $\mathrm{rep}(c) \sim c$ and for any $c' \in \mathrm{rConf}$ with $c \sim c'$ we have $\mathrm{rep}(c) \leqslant_{llex} c'$. The closure properties of synchronous rational relations imply the following fact.

[1]According to their definition, a VPA may not read a return letter if the stack is empty.

Corollary 10.6 ([15]). *The function rep is rational.*

The set of *representative* configurations is denoted by $\mathrm{Rep} = \mathrm{rep}(\mathrm{rConf})$. Finally we define $\nu_{\mathcal{A}} \colon \Sigma^* \to \mathrm{Rep}$ by $\nu_{\mathcal{A}}(w) = \mathrm{rep}(\delta(\bot q_0, w))$ for all $w \in \Sigma^*$. It represents $\sim_L$ in the sense that $L(\nu_{\mathcal{A}}(w)) = w^{-1}L(\mathcal{A})$ for all $w \in \Sigma^*$ and hence $\sim_L = \ker(\nu_{\mathcal{A}})$.

Example 10.7. Recall the VPL L over $\{a, b\}$ from Example 10.2. The set of reachable configurations is

$$\mathrm{rConf} = \{\bot q_F\} \cup \bot \underline{A}\{A, \underline{A}\}^* Q.$$

The set of length-lexicographical minimal representatives configurations is given by

$$\mathrm{Rep} = \{\bot q_F\} \cup \bot \underline{A} A^* q_a \cup \bot \underline{A} A^* q_b$$

since any configuration $\bot \alpha \underline{A} A^i q_x$ is equivalent to $\bot \underline{A} A^i q_x$ for $\alpha \in \{A, \underline{A}\}^*$ and $x \in \{a, b\}$.

Since the set of representative configurations Rep is regular it has either polynomial or exponential cumulative growth.

Lemma 10.8. *If Rep has polynomial (exponential) cumulative growth then also $|\Sigma^{\leqslant n}/{\sim_L}|$ has polynomial (exponential) growth.*

Proof. Consider the function which maps a class $[w]_{\sim_L}$ to $\mathrm{rep}(\delta(\bot q_0, w))$. It is well-defined because, if $w \sim_L w'$ are equivalent, then also $\delta(\bot q_0, w) \sim \delta(\bot q_0, w')$ are equivalent configurations. By definition of Rep it is a bijection between $|\Sigma^*/{\sim_L}|$ and Rep.

First, any word of length n can reach a configuration $\delta(\bot q_0, w)$ with stack height at most n. Since rep picks length-minimal representatives the configuration $\mathrm{rep}(\delta(\bot q_0, w))$ has a stack of height at most n. Hence, if $r(n) = |\{c \in \mathrm{Rep} : |c| \leqslant n\}|$ then $|\Sigma^{\leqslant n}/{\sim_L}| \leqslant r(n+2)$. Therefore if $r(n)$ is polynomially bounded then so is $|\Sigma^{\leqslant n}/{\sim_L}|$.

Conversely, let $c \in \mathrm{Rep}$ be a configuration of length n and let $c = \delta(\bot q_0, w)$. Let $w = w_0 w_1 \cdots w_m$ be a monotonic factorization. Notice that there are at most n many factors w_i which are call letters. By choosing the well-matched factors in the monotonic factorization maximal (as in the proof of Lemma 10.4) this implies $m \leqslant 2n+1$. Since w_0 is descending we have $\delta(\bot q_0, w_0) \in \bot Q$ and we can replace it by a length-minimal word w_0' with $\delta(\bot q_0, w_0) = \delta(\bot q_0, w_0')$. Every well-matched factor w_i is replaced by a length-minimal well-matched word w_i' with $\theta(w_i) = \theta(w_i')$. Since the number of states and the number of state transformations is constant, the new word $w' = w_0' w_1' \cdots w_m'$ consists of factors of constant length and hence $|w'| = O(m) = O(n)$. It satisfies $\mathrm{rep}(\delta(\bot q_0, w')) = c$. Hence $r(n) \leqslant |\Sigma^{\leqslant dn}/{\sim_L}|$ for some constant $d > 0$ and sufficiently large n. Hence any exponential lower bound on $r(n)$ transfers to $|\Sigma^{\leqslant n}/{\sim_L}|$. □

As a corollary we obtain a dichotomy for the standard streaming space complexity of VPLs.

Corollary 10.9. *In the standard streaming model the space complexity* $E_L(n) = \log |\Sigma^{\leqslant n}/{\sim_L}|$ *of every VPL* L *is either* $O(\log n)$ *or* $\Omega(n)$ *infinitely often.*

10.3 Proof strategy

The rest of the chapter is dedicated to the proof of Theorem 10.1. If L is either empty or universal then $V_L(n)$ is $O(1)$, and otherwise $\Omega(\log n)$ by Lemma 3.7. Hence we can assume that $\emptyset \subsetneq L \subsetneq \Sigma^*$ and prove that the space complexity is either $O(\log n)$ or $\Omega(n)$ for infinitely many n. Recall the definition of the suffix expansion of an equivalence relation or a function, see Section 3.4. By Proposition 3.11 and the definition of $\nu_{\mathcal{A}}$ we have

$$2^{V_L(n)} = |\Sigma^{\leqslant n}/{\approx_L}| = |\overleftarrow{\nu}_L(\Sigma^{\leqslant n})| = |\overleftarrow{\nu}_{\mathcal{A}}(\Sigma^{\leqslant n})| = |\mathrm{im}(\overleftarrow{\nu}_{\mathcal{A}}) \cap \mathrm{Rep}^{\leqslant n}|, \tag{10.1}$$

and therefore it suffices to prove that $\mathrm{im}(\overleftarrow{\nu}_{\mathcal{A}})$ has polynomial or exponential growth. Notice that $\overleftarrow{\nu}_{\mathcal{A}}$ maps a word over Σ of length n to a sequence of n representative configurations from Rep, which is a “two-dimensional” object.

Example 10.10. Let us compute $\overleftarrow{\nu}_{\mathcal{A}}(w)$ for an example word w and the VPA $\mathcal{A}$ from Example 10.7. For example $\overleftarrow{\nu}_{\mathcal{A}}(abaaaaabb)$ is the sequence

$$(\bot\underline{A}AAq_b, \bot\underline{A}AAq_b, \bot\underline{A}AAq_b, \bot\underline{A}Aq_b, \bot\underline{A}q_b, \bot q_F, \bot q_F, \bot q_F, \bot q_F)$$

because applying $\nu_{\mathcal{A}}$ to all suffixes of $w = abaaaaabb$ yields:

$$\begin{aligned}
\nu_{\mathcal{A}}(abaaaaabb) &= \bot\underline{A}AAq_b \\
\nu_{\mathcal{A}}(baaaaabb) &= \bot\underline{A}AAq_b \\
\nu_{\mathcal{A}}(aaaaabb) &= \bot\underline{A}AAq_b \\
\nu_{\mathcal{A}}(aaaabb) &= \bot\underline{A}Aq_b \\
\nu_{\mathcal{A}}(aaabb) &= \bot\underline{A}q_b \\
\nu_{\mathcal{A}}(aabb) &= \bot q_F \\
\nu_{\mathcal{A}}(abb) &= \bot q_F \\
\nu_{\mathcal{A}}(bb) &= \bot q_F \\
\nu_{\mathcal{A}}(b) &= \bot q_F
\end{aligned}$$

Notice that $\nu_{\mathcal{A}}$ need not be rational in general and hence we cannot directly apply Theorem 5.2. Intuitively one would need a stack to compute the configuration $\delta(\bot q_0, w)$. In the example above we have $\nu_{\mathcal{A}}(a^i b^j) = \bot\underline{A}A^{i-j-1}q_b$ for all $1 \leqslant j < i$ and a simple pumping argument shows that $\nu_{\mathcal{A}}$ cannot be rational. Using a different representation of the input word we will describe $\nu_{\mathcal{A}}$ by a rational function and apply Theorem 5.2.

Let us give an overview of the proof of Theorem 10.1. Figure 10.2 gives an overview of the used sets and abstractions. We identify three sufficient conditions for the linear growth of $V_L(n)$.

Condition 1 The context-free set $\bar{\nu}_{\mathcal{A}}(D)$ has exponential growth.

Since $\bar{\nu}_{\mathcal{A}}(D) \subseteq \bar{\nu}_{\mathcal{A}}(\Sigma^*)$ this condition immediately implies a linear lower bound on $V_L(n)$. In order to simulate $\mathcal{A}$ by a finite transducer we will "flatten" the input word in the following way. The input word w is factorized $w = w_0 w_1 \cdots w_m$ into a descending prefix w_0, and call letters and well-matched factors $w_1, \dots, w_m$. The descending prefix and each well-matched factor w_i are replaced by so-called D- and W-sequences which describe the behavior of $\mathcal{A}$ on the factor w_i and on each of its suffixes. The set of all (syntactically correct) flattenings is denoted by AllFlat, which is regular; the set Flat of all realized flattenings is only context-free. There exists a rational function $\nu_f \colon \text{AllFlat} \to \text{Rep}$ such that, if s is a flattening of a word $w \in \Sigma^*$ then $\nu_f(s)$ is a configuration representing the Myhill-Nerode class $\nu_L(w)$ (Proposition 10.15). Hence, we can reduce proving the main theorem to the question whether the growth of $\bar{\nu}_f(\text{Flat})$ is always either polynomial or exponential.

This question is resolved positively as follows. If Condition 1 does not hold, we can approximate Flat by a regular superset RegFlat with $\nu_f(\text{RegFlat}) = \nu_f(\text{Flat}) = \text{Rep}$ and some other properties. If $\bar{\nu}_f(\text{RegFlat})$ has polynomial growth then the same holds for the subset $\bar{\nu}_f(\text{Flat})$ and therefore $V_L(n) = O(\log n)$. Otherwise, the dichotomy theorem (Theorem 5.2), applied to the rational restriction $\bar{\nu}_f|_{\text{RegFlat}}$, states that $\bar{\nu}_f(\text{RegFlat})$ has exponential growth if one of the following two conditions hold:

Condition 2 The regular set Rep has exponential growth.

Condition 3 ν_f has a fooling scheme in RegFlat.

If Condition 2 holds then Lemma 10.8 states that $|\nu_{\mathcal{A}}(\Sigma^{\leqslant n})| = |\Sigma^{\leqslant n}/{\sim_L}|$ grows exponentially, which is a lower bound on $|\bar{\nu}_{\mathcal{A}}(\Sigma^{\leqslant n})|$. If Condition 3 holds we can ensure that the fooling scheme is already contained in Flat. In both cases we obtain a linear lower bound on $V_L(n)$.

10.4 Reduction to transducers

D- and W-sequences We first define abstractions for the descending prefix and well-matched factors. A D-*sequence* is any sequence $q_1 \cdots q_n \in Q^*$, and a W-*sequence* is any sequence $\tau q_2 \cdots q_n \in Q^Q Q^*$.

If $x = a_1 \cdots a_n \in D$ is a descending word then $\delta(\bot q_0, x) = \bot p$ for some $p \in Q$. Since rep chooses length-minimal configurations there exists a state $q \in Q$ with $\nu_{\mathcal{A}}(x) = \bot q$. Since each suffix of x is also descending we have $\bar{\nu}_{\mathcal{A}}(x) = (\bot q_1, \bot q_2, \dots, \bot q_n)$ for some $q_1, \dots, q_n \in Q$. The D-*sequence* of x is defined as $\sigma_D(x) = q_1 \cdots q_n \in Q^*$, i.e. we remove the redundant $\bot$-symbols from $\bar{\nu}_{\mathcal{A}}(x)$. If x is nonempty and well-matched with $\sigma_D(x) = q_1 \cdots q_n$ and $n \geqslant 1$ we additionally define its W-*sequence* as $\sigma_W(x) = \tau q_2 \cdots q_n \in Q^Q Q^*$ where $\tau = \theta(x)$.

We denote by $S_D = \sigma_D(D)$ and $S_W = \sigma_W(W \setminus \{\varepsilon\})$ the set of all realized D- and W-sequences, respectively. Since descending words are exactly the (proper)

$$\begin{tikzcd}
\Sigma^* \arrow[r, "\delta(\bot q_0,\cdot)"] \arrow[rr, bend left, "\nu_{\mathcal{A}}"] \arrow[d, dash] & \mathrm{rConf} \arrow[r, "\mathrm{rep}"] & \mathrm{Rep} \\
\mathrm{Flat} & & \\
\mathrm{RegFlat} \arrow[u, phantom, sloped, "\subseteq"] & & \\
\mathrm{AllFlat} \arrow[u, phantom, sloped, "\subseteq"] \arrow[uuur, "t_f"] \arrow[uuurr, "\nu_f"'] & &
\end{tikzcd}$$

Figure 10.2: On input $w \in \Sigma^*$ the VPA $\mathcal{A}$ reaches configuration $\delta(\bot q_0, w)$. Its representative configuration is $\nu_{\mathcal{A}}(w)$. We abstract the words from Σ^* by flattenings from Flat.

suffixes of well-matched words S_D is exactly the set of proper suffixes of words from S_W. Notice that σ_D and σ_W preserve suffixes in the following sense: If $\sigma_D(a_1 \cdots a_n) = q_1 \cdots q_n$ or $\sigma_W(a_1 \cdots a_n) = \tau q_2 \cdots q_n$ then $\sigma_D(a_i \cdots a_n) = q_i \cdots q_n$ for all $2 \leqslant i \leqslant n$.

Lemma 10.11. *The languages* S_D *and* S_W *are context-free.*

Proof. Since S_D is the set of all proper suffixes of words from S_W it suffices to consider S_W. We will prove that $\{w \otimes \sigma_W(w) \mid w \in W \setminus \{\varepsilon\}\}$ is a VPL over the pushdown alphabet $(\Sigma_c \times \Sigma_f, \Sigma_r \times \Sigma_f, \Sigma_{int} \times \Sigma_f)$. Since the class of context-free languages is closed under projections it then follows that S_W is context-free. A VPA can test whether the first component $w = a_1 \cdots a_n$ is well-matched and whether the second component has the form $\tau q_2 \cdots q_n \in Q^Q Q^*$. Since VPLs are closed under Boolean operations, it suffices to test whether $\tau \neq \theta(w)$ or there exists a state q_i with $\nu_{\mathcal{A}}(a_i \cdots a_n) \neq \bot q_i$. To guess an incorrect state we use a VPA whose stack alphabet contains all stack symbols of $\mathcal{A}$ and a special symbol # representing the stack bottom. We guess and read a prefix of the input word and push/pop only the special symbol # on/from the stack. Then at some point we store the second component q_i in the next symbol and simulate $\mathcal{A}$ on the remaining suffix. Finally, we accept if and only if the reached state is q and $\mathrm{rep}(\bot q) \neq \bot q_i$. Similarly, we can verify τ by testing whether there exists a state $p \in Q$ with $\theta(w)(p) \neq \tau(p)$. □

Lemma 10.12. *The language* S_D *is bounded if and only if* S_W *is bounded. If* S_D *is not bounded then* $\mathrm{im}(\vec{\nu}_{\mathcal{A}})$ *grows exponentially.*

Proof. Assume that $S_D \subseteq s_1^* \cdots s_k^*$ is bounded. Since $S_W \subseteq \bigcup\{\tau S_D \mid \tau \in Q^Q\}$ we have $S_W \subseteq \tau_1^* \cdots \tau_m^* s_1^* \cdots s_k^*$ for any enumeration $\tau_1, \ldots, \tau_m$ of Q^Q. Conversely, if S_W is bounded then each word in S_D is a factor, namely a proper suffix, of a word from S_W. By Lemma 3.14 then also S_D must be bounded.

If the context-free language $S_D = \sigma_D(D) \subseteq Q^*$ is not bounded then its growth must be exponential. Recall that $\bar{\nu}_{\mathcal{A}}(w)$ and $\sigma_D(w)$ are equal for all $w \in D$ up to the $\perp$-symbol. Hence $|\bar{\nu}_{\mathcal{A}}(\Sigma^{\leqslant n})| \geqslant |\bar{\nu}_{\mathcal{A}}(D \cap \Sigma^{\leqslant n})| = |\sigma_D(D \cap \Sigma^{\leqslant n})| = |S_D \cap Q^{\leqslant n}|$, which proves the growth bound. □

Flattenings Define the alphabet $\Sigma_f = \Sigma_c \cup Q \cup Q^Q$. A *flattening* is a word $s_0 s_1 \cdots s_m \in \Sigma_f^*$ where $s_0 \in Q^*$ is a D-sequence and $s_i \in \Sigma_c \cup Q^Q Q^*$ is a call letter or a W-sequence for all $1 \leqslant i \leqslant m$. The factorization into the initial D-sequence, the call letters and the W-sequences is unique. The set of all flattenings is AllFlat $= Q^*(\Sigma_c \cup Q^Q Q^*)^*$, which is suffix-closed. We say that $s = s_0 s_1 \cdots s_m \in$ AllFlat is a *flattening of* a word $w \in \Sigma^*$ if there exists a monotonic factorization $w = w_0 w_1 \cdots w_m \in \Sigma^*$ such that $s_0 = \sigma_D(w_0)$, and for all $1 \leqslant i \leqslant m$ if w_i is well-matched, then $s_i = \sigma_W(w_i)$, and if $w_i \in \Sigma_c$ then $s_i = w_i$. The set of *realized flattenings* is Flat $= S_D(\Sigma_c \cup S_W)^*$. Since a word may have different monotonic factorizations, it may also have many flattenings.

Lemma 10.13. *If* $s = b_1 \cdots b_n \in \Sigma_f^*$ *is a flattening of a word* $w = a_1 \cdots a_n \in \Sigma^n$, *then* $b_i \cdots b_n$ *is a flattening of* $a_i \cdots a_n$ *for all* $1 \leqslant i \leqslant n$. *In particular* Flat *is suffix-closed.*

Proof. Let $w = w_0 w_1 \cdots w_m$ be a monotonic factorization and let $s = s_0 s_1 \cdots s_m$ be the associated flattening. This means $\sigma_D(w_0) = s_0$, and for all $1 \leqslant i \leqslant m$, if w_i is a call letter then $w_i = s_i$, and if w_i is a well-matched factor then $\sigma_W(w_i) = s_i$. Now take any suffix w' of w, which has a monotonic factorization of the form $w'_k w_{k+1} \cdots w_m$ where w'_k is a possibly empty suffix of w_k. Let $s' = s'_k s_{k+1} \cdots s_m$ where s'_k is the length-$|w'_k|$ suffix of s_k. One can see that s' is the associated flattening of w' since $\sigma_D(w_k) = s_k$ implies $\sigma_D(w'_k) = s'_k$. □

We define a function $t_f \colon$ AllFlat $\to$ rConf as follows. Let $s = s_0 s_1 \cdots s_m \in \Sigma_f^*$ be a flattening and we define $t_f(s)$ by induction on m:

- If $s_0 = \varepsilon$ then $t_f(s_0) = \perp q_0$. If $s_0 = q_1 \cdots q_n \in Q^+$ then $t_f(s_0) = \perp q_1$.
- If $s_m \in \Sigma_c$ and $m \geqslant 1$ then $t_f(s) = \delta(t_f(s_0 \cdots s_{m-1}), s_m)$.
- If $s_m = \tau q_2 \cdots q_m \in Q^Q Q^*$ and $m \geqslant 1$ and $t_f(s_0 \cdots s_{m-1}) = \alpha q$ then $t_f(s) = \alpha\tau(q)$.

Define the function $\nu_f \colon$ AllFlat $\to$ Rep by $\nu_f = \mathrm{rep} \circ t_f$.

Lemma 10.14. *The functions* t_f *and* ν_f *are rational.*

Proof. We first define a transducer $\mathcal{A}_1$ which handles flattenings where the initial D-sequence is empty. Let $\mathcal{A}_1 = (Q, \Sigma_f, Q \cup \Gamma, q_0, \Delta', Q, o)$ with the following transitions:

- $p \xrightarrow{q|\varepsilon} p$ for all $p, q \in Q$
- $p \xrightarrow{a|\gamma} q$ for all $\delta(p, a) = (\gamma, q)$ where $a \in \Sigma_c$

- $p \xrightarrow{\tau|\varepsilon} \tau(p)$ for all $p \in Q, \tau \in Q^Q$

and $o(q) = q$. For each $p \in Q$ let t_p be the rational function defined by $\mathcal{A}_1$ with the only initial state p. One can easily show that for all $s \in (Q^Q Q^* \cup \Sigma_c)^*$ we have $t_f(s) = \bot t_{q_0}(s)$ and $t_f(q_1 \cdots q_k s) = \bot t_{q_1}(s)$ for all $q_1 \cdots q_k \in Q^+$. Hence we can prove that t_f is rational by providing a transducer for t_f: First it verifies whether the input word s belongs to the regular language AllFlat $\subseteq \Sigma_f^*$. Simultaneously, it verifies whether s begins with a nonempty D-sequence $q_1 \cdots q_k$. If so, it memorizes q_1 and simulates $\mathcal{A}_1$ on the remaining suffix s' starting from state q_1. Otherwise $\mathcal{A}_1$ is directly simulated on s from q_0. Since rep is rational by Corollary 10.6, ν_f is also rational. □

Proposition 10.15. *If $s \in$ AllFlat is a flattening of $w \in \Sigma^*$ then $\nu_f(s) = \nu_{\mathcal{A}}(w)$. Therefore $\nu_f(\text{Flat}) =$ Rep and $|\bar{\nu}_{\mathcal{A}}(\Sigma^{\leqslant n})| = |\bar{\nu}_f(\text{Flat}) \cap \text{Rep}^{\leqslant n}|$.*

Proof. Let $w = w_0 w_1 \cdots w_m \in \Sigma^*$ be a monotonic factorization and let $s = s_0 s_1 \cdots s_m \in$ Flat be the associated flattening. We prove $t_f(s) \sim \delta(\bot q_0, w)$ by induction on m.

- If $m = 0$ and $s_0 = \varepsilon$ then $t_f(s) = \bot q_0 = \delta(\bot q_0, \varepsilon)$.
- If $m = 0$ and $s_0 = q_1 \cdots q_k \in Q^+$ then $t_f(s) = \bot q_1$ and $\nu_{\mathcal{A}}(w) = \text{rep}(\delta(\bot q_0, w)) = \bot q_1$.
- If $m \geqslant 1$ and $s_m \in \Sigma_c$ then $s_m = w_m$. By induction hypothesis we know that $t_f(s_0 \cdots s_{m-1}) \sim \delta(\bot q_0, w_0 \cdots w_{m-1})$. Since $\delta(\bot q_0, w) = \delta(\delta(\bot q_0, w_0 \cdots w_{m-1}), w_m)$ and $t_f(s) = \delta(t_f(s_0 \cdots s_{m-1}), s_m)$ we obtain $\delta(\bot q_0, w) \sim t_f(s)$.
- If $m \geqslant 1$ and $s_m = \tau q_2 \cdots q_k \in Q^Q Q^*$ then w_m is well-matched and $\theta(w_m) = \tau$. Assume that $t_f(s_0 \cdots s_{m-1}) = \alpha p$ and $\delta(\bot q_0, w_0 \cdots w_{m-1}) = \beta q$. By induction hypothesis we know that $\alpha p \sim \beta q$. Since $t_f(s) = \alpha\tau(p) = \delta(\alpha p, w_m)$ and $\delta(\bot q_0, w) = \delta(\beta q, w_m)$ we obtain $t_f(s) \sim \delta(\bot q_0, w)$.

Since $\nu_f = \text{rep} \circ t_f$ and $\nu_{\mathcal{A}}(w) = \text{rep}(\delta(\bot q_0, w))$ we have $\nu_f(s) = \nu_{\mathcal{A}}(w)$. The consequence follows from the suffix preservation (Lemma 10.13). □

10.5 Bounded overapproximation

In this section we will assume that S_D and S_W are bounded languages. Otherwise Lemma 10.12 implies that $\text{im}(\bar{\nu}_{\mathcal{A}})$ grows exponentially and hence L has linear sliding window space complexity by (10.1). To apply Theorem 5.2 we will approximate Flat by a regular language RegFlat. Define the function

$$\Phi(a_1 \cdots a_n) = \{(a_1, n), (a_2, n-1) \ldots, (a_n, 1)\}$$

and $\Phi(K) = \bigcup_{w \in K} \Phi(w)$. It summarizes which symbols occur at which positions across all words in a language K (read from right to left).

Lemma 10.16. *Let K be a bounded context-free language. Then there exists a bounded regular superset $R \supseteq K$ such that $\{|w| : w \in K\} = \{|w| : w \in R\}$ and $\Phi(K) = \Phi(R)$, called a* bounded overapproximation *of* K.

Proof. We use Parikh's theorem, which implies that for every context-free language $K \subseteq \Sigma^*$ the set $\{|w| : w \in K\}$ is semilinear, i.e. a finite union of arithmetic progressions, and hence $\{v \in \Sigma^* \mid \exists w \in K\colon |v| = |w|\}$ is a regular language. Assume that $K \subseteq w_1^* \cdots w_k^*$ for some $w_1, \ldots, w_k \in \Sigma^*$. We define

$$R = (w_1^* \cdots w_k^*) \cap \{v \in \Sigma^* \mid \exists w \in K : |v| = |w|\} \cap \{w \in \Sigma^* \mid \Phi(w) \subseteq \Phi(K)\}.$$

Clearly, K is contained in R and it remains to verify that the third part is regular. It suffices to show that for each $a \in \Sigma$ the set $P_a = \{i \mid (a, i) \in \Phi(K)\}$ is semilinear because then an automaton can verify the property $\Phi(w) \subseteq \Phi(K)$. Consider the transduction

$$T_a = \{(a_1 \cdots a_n, \square^{n-i+1}) \mid a_1 \cdots a_n \in \Sigma^*,\ a_i = a\}.$$

It is easy to see that T_a is rational and $T_a K = \{\square^i \mid i \in P_a\}$. The claim follows again from Parikh's theorem. □

For each $\tau \in Q^Q$ let R_τ be a bounded overapproximation of $\tau^{-1} S_W$ and let $R_W = \bigcup_{\tau \in Q^Q} (\tau R_\tau)$. Let $R_D = \bigcup_{\tau \in Q^Q} \mathrm{Suf}(R_\tau)$, which is the set of all proper suffixes of words in R_W. Both R_D and R_W are also bounded languages. Finally, set $\mathrm{RegFlat} = R_D(\Sigma_c \cup R_W)^*$, which is a suffix-closed subset of AllFlat, since $\mathrm{RegFlat} = \mathrm{Suf}((\Sigma_c \cup R_W)^*)$. According to the definition of bounded overapproximations we can approximate a W-sequence $v = \tau q_2 \cdots q_k \in R_W$ in two possible ways:

- There exists a W-sequence of the form $\tau p_2 \cdots p_k \in S_W$ of length $|v|$. It is called a *length approximation of* v.
- For all $2 \leqslant i \leqslant k$ there exists a W-sequence of the form $\tau s' q_i p_{i+1} \cdots p_k \in S_W$. It is called an *approximation of* v *along* q_i.

If $r = r_0 r_1 \cdots r_m \in \mathrm{RegFlat}$ then we can replace any W-sequence $r_i \in R_W$ by an approximation of r_i (of either type) without changing the value of $\nu_f(r)$ since the state transformations are preserved.

Proposition 10.17. $\nu_f(\mathrm{Flat}) = \nu_f(\mathrm{RegFlat})$.

Proof. We clearly have $\nu_f(\mathrm{Flat}) \subseteq \nu_f(\mathrm{RegFlat})$ and it remains to show the other inclusion. Consider a flattening $r = r_0 r_1 \cdots r_m \in \mathrm{RegFlat}$. First, we replace any W-sequence $r_i \in R_W$ by a length approximation $r_i' \in S_W$ of r_i, which preserves the value of $\nu_f(r)$. If r_0 is empty, we are done. Otherwise suppose that $r_0 = q_1 \cdots q_k \in R_D$ is nonempty. By definition $q_1 \cdots q_k$ is a proper suffix of some word $x = \tau p_2 \cdots p_{i-1} q_1 \cdots q_k \in R_W$. Take an approximation $y \in S_W$ of x along q_1, which has a proper suffix of the form $q_1 q_2' \cdots q_k'$ belonging to S_D. We can replace $q_1 \cdots q_k$ by $q_1 q_2' \cdots q_k'$ in r without changing the value of $\nu_f(r)$. □

Lemma 10.18. *If* ν_f *has a fooling scheme in* RegFlat *then it also has a fooling scheme* (u_2, v_2, u, v, z) *in* Flat.

Proof. Let (u_2, v_2, u, v, z) be a fooling scheme of ν_f in RegFlat. First we ensure that $u, v, z \in (\Sigma_c \cup R_W)^*$.

Since u_2 and v_2 are distinct words of equal length $\{u_2, v_2\}\{u, v\}^n$ contains 2^n distinct words and therefore $\{u_2, v_2\}\{u, v\}^*$ is not bounded. We claim that $\{u, v\} \not\subseteq Q^*$. Otherwise $\{u_2, v_2\}\{u, v\}^* \subseteq Q^*$ would be contained in the set of prefixes of words in R_D. This is a contradiction because R_D is bounded by assumption and $\{u_2, v_2\}\{u, v\}^*$ has exponential growth.

Without loss of generality, assume that $u = u_3u_4$ such that u_4 either starts with a call letter $a \in \Sigma_c$ or a transformation $\tau \in Q^Q$. We claim that

$$(u_2u_3, v_2u_3, u_4uu_3, u_4vu_3, u_4z)$$

is a fooling scheme for ν_f. It has the following properties:

- $\{u_2u_3, v_2u_3\}\{u_4uu_3, u_4vu_3\}^*u_4z \subseteq \{u_2, v_2\}\{u, v\}^*z \subseteq$ RegFlat,
- u_2u_3 is a suffix of u_4uu_3,
- v_2u_3 is a suffix of u_4vu_3,
- $|u_2u_3| = |v_2u_3|$,
- ν_f separates $u_2u_3\{u_4uu_3, u_4vu_3\}^*u_4z$ and $v_2u_3\{u_4uu_3, u_4vu_3\}^*u_4z$.

Since $(u_2u_3)(u_4uu_3)(u_4vu_3)(u_4z)$ belongs to RegFlat $= R_D(\Sigma_c \cup R_W)^*$ and u_4 starts with a call letter or a transformation, the words u_4uu_3, u_4vu_3, u_4z must belong to $(\Sigma_c \cup R_W)^*$.

Now, let (u_2, v_2, u, v, z) be a fooling scheme of ν_f in RegFlat such that $u, v, z \in (\Sigma_c \cup R_W)^*$. We replace occurring factors from R_W by factors from S_W while maintaining the values $\nu_f(swz)$ for all $s \in \{u_2, v_2\}$ and $w \in \{u, v\}^*$.

1. First we replace each R_W-factor in z by a length approximation, which ensures that $z \in (\Sigma_c \cup S_W)^*$.
2. Next consider u and v, and assume that $u = u_1u_2$ and $v = v_1v_2$ for some $u_1, v_1 \in \Sigma_f^*$. Let us consider R_W-factors which cross the factorization $u = u_1u_2$ or $v = v_1v_2$, respectively. If u_2 starts with some state we can factorize u_1 and u_2 as $u_1 = u_3\tau q_2 \cdots q_{i-1}$ and $u_2 = q_i \cdots q_k u_4$ where $u_3, u_4 \in (\Sigma_c \cup R_W)^*$ and $\tau q_2 \cdots q_k \in R_W$. Let $y \in S_W$ be an approximation of $\tau q_2 \cdots q_k$ along q_i, say $y = \tau s' q_i p_{i+1} \cdots p_k$. We replace u_1 by $u_3\tau s'$ and u_2 by $q_i p_{i+1} \cdots p_k u_4$. Notice that the length of u_2 has not changed (this maintains $|u_2| = |v_2|$) and the first state of u_2 has not changed either (this maintains the values $\nu_f(u_2wz)$). If v_2 starts with some state we do the analogous replacements for v_1 and v_2.
3. Finally, each R_W-factor in u_1, u_2, v_1 and v_2 is replaced by a length approximation.

One can verify that the new tuple (u_2, v_2, u, v, z) is a fooling scheme of ν_f in Flat. $\square$

Now we are ready to prove the main theorem:

Proof of Theorem 10.1. If $L = \emptyset$ or $L = \Sigma^*$ then $V_L(n) = O(1)$. Now assume $\emptyset \subsetneq L \subsetneq \Sigma^*$, in which case we have $V_L(n) = \Omega(\log n)$. Furthermore we know that $2^{V_L(n)}$ is the cumulative growth of $\mathrm{im}(\overleftarrow{\nu}_{\mathcal{A}})$ by (10.1).

If the constructed languages S_D and S_W are not bounded then $\mathrm{im}(\overleftarrow{\nu}_{\mathcal{A}})$ grows exponentially by Lemma 10.12 (Condition 1). Now assume that S_D and S_W are bounded, in which case we can construct RegFlat. We apply Theorem 5.2 to the rational restriction $\nu_f|_{\mathrm{RegFlat}}$. The latter theorem states that the growth of $\overleftarrow{\nu}_f(\mathrm{RegFlat})$ is either polynomial or exponential. If $\overleftarrow{\nu}_f(\mathrm{RegFlat})$ has polynomial growth then the same holds for its subset $\overleftarrow{\nu}_f(\mathrm{Flat})$. Otherwise, either the image $\nu_f(\mathrm{RegFlat})$ is not bounded (Condition 2) or ν_f has a fooling scheme in RegFlat (Condition 3).

First assume Condition 2 holds. Then the regular set Rep has exponential growth because $\nu_f(\mathrm{RegFlat}) = \nu_f(\mathrm{Flat}) = \mathrm{Rep}$ by Proposition 10.15 and Proposition 10.17. By Lemma 10.8 we know that $|\Sigma^{\leqslant n}/{\sim_L}| = |\nu_{\mathcal{A}}(\Sigma^{\leqslant n})|$ grows exponentially and hence also $|\overleftarrow{\nu}_{\mathcal{A}}(\Sigma^{\leqslant n})|$.

If Condition 3 holds then ν_f has a fooling scheme in Flat by Lemma 10.18. Then $\overleftarrow{\nu}_f(\mathrm{Flat})$ has exponential growth by Proposition 5.3. By Proposition 10.15 also $|\overleftarrow{\nu}_{\mathcal{A}}(\Sigma^{\leqslant n})|$ grows exponentially. This concludes the proof. $\square$

10.6 Conclusion

We have proved that every visibly pushdown language has constant, logarithmic or linear space complexity in the variable-size sliding window model. Let us summarize the open questions.

Open problems

1. Does every VPL have constant, logarithmic or linear space complexity in the fixed-size model? If L is a VPL with $F_L(n) = o(\log n)$ then $F_L(n)$ must be constant by Theorem 9.1. We conjecture that the linear lower bounds for $V_L(n)$ can be transferred to $F_L(n)$. To do so, one would have to reexamine the three conditions from Section 10.3.

2. It also remains open to give a simple characterization of the visibly pushdown languages with logarithmic complexity. This question is related to characterizing the rational functions whose suffix expansions have polynomial image growth. We remark that the example language L from Proposition 3.12 is a visibly pushdown language and a left ideal which has linear space complexity in the fixed-size model.

3. Finally, we pose the question whether the ideas from this chapter can be used to analyze the sliding window space complexity of other language classes. In general, one would need to find a suitable “flattened” representation of the window on which a finite state transducer can compute representative configurations.

Related work In [8] Alur et al. define a natural syntactic congruence for languages over a pushdown alphabet, and prove that a language containing only well-matched words is a VPL if and only if its syntactic congruence has finite index. Subclasses of visibly pushdown languages with low “complexity” were also studied in other contexts. Bárány, Löding and Serre [15] consider visibly counter automata, which can use the stack to count up to a constant threshold. They prove that it is decidable whether a given visibly pushdown automaton is equivalent to a visibly counter automaton. One can easily see that a visibly counter automaton can be viewed a VPA whose set of configurations has constant growth (or linear cumulative growth). Therefore we have $|\Sigma^{\leqslant n}/{\sim_L}| = O(n)$ for every visibly counter language L. Krebs, Lange and Ludwig [79] show that it is decidable whether a given visibly counter language is contained in the circuit complexity class AC^0.

Chapter 11

Conclusion

In this thesis we have analyzed the sliding window streaming model, with an emphasis on the space complexity of monitoring properties over sliding windows. We have pursued the question, which properties, viewed as formal languages, admit space efficient sliding window algorithms and have provided a complete picture for the case of regular languages. The sliding window model and, more generally, the streaming model yield interesting automata theoretic questions, which can give new insights into language classes and their properties.

Let us give an outlook of possible research directions.

Languages classes A widely open question is which other language classes admit a structured analysis in the sliding window model. Besides the already mentioned class of deterministic context-free languages, one could consider languages accepted by constrained automata [27] in the sliding window model, which are regular languages with semilinear constraints. In a preceding step one should study such language classes in the standard streaming model.

Going in the other direction, one should also restrict the class of regular languages and find appropriate input representations that capture queries from real-world applications.

Quantitative queries and data streams Instead of qualitative statements over the sliding window one is often interested in computing quantitative values. Our investigation of rational functions is a starting point for that. In the literature there is a large variety of quantitative models which compute values from words. One of the standard models of quantitative systems are weighted automata [38], which compute a value over a semiring. Other models include cost register automata [6] and quantitative regular expressions [7]. The latter form the core of StreamQRE [87], which is a specification language for streaming queries.

A related topic concerns the fact that streams usually consist of complex data values, e.g. tuples of real numbers, which may represent sensor values or timestamps. Such streams can be modeled by so-called data words. A popular formalism for specifying set of data words (so-called data languages) are register automata, which are also known as finite memory automata [71]; see also [93] for a comparison with other models.

Time-based windows In some streaming applications the data items may arrive at irregular intervals and it may be more interesting to consider time-based windows. The path summary algorithm can be easily adapted to work over time-based windows whenever the given rDFA is weak, i.e. each SCC contains only final or only nonfinal states. Recall that weak rDFAs capture precisely the class $\langle \mathbf{LI} \rangle$ (Theorem 4.32). It would be interesting to compare the space complexity of languages under the fixed-size, variable-size and the time-based window model.

Resulting publications

Relevant for this thesis

[G1] M. Ganardi, D. Hucke, and M. 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)*. Ed. by A. Lal, S. Akshay, S. Saurabh, and S. Sen. Vol. 65. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 18:1–18:14. DOI: 10.4230/LIPIcs.FSTTCS.2016.18.

[G2] M. Ganardi, D. Hucke, D. König, M. Lohrey, and K. Mamouras. “Automata Theory on Sliding Windows”. In: *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*. Ed. by R. Niedermeier and B. Vallée. Vol. 96. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 31:1–31:14. DOI: 10.4230/LIPIcs.STACS.2018.31.

[G3] M. Ganardi, D. Hucke, and M. Lohrey. “Randomized Sliding Window Algorithms for Regular Languages”. In: *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Ed. by I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 127:1–127:13. DOI: 10.4230/LIPIcs.ICALP.2018.127.

[G4] M. Ganardi, D. Hucke, and M. Lohrey. “Sliding Window Algorithms for Regular Languages”. In: *Proceedings of the 12th International Conference on Language and Automata Theory and Applications (LATA 2018)*. Ed. by S. T. Klein, C. Martín-Vide, and D. Shapira. Vol. 10792. Lecture Notes in Computer Science. Springer, 2018, pp. 26–35. DOI: 10.1007/978-3-319-77313-1_2.

[G5] M. Ganardi, A. Jeż, and M. Lohrey. “Sliding Windows over Context-Free Languages”. In: *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Ed. by I. Potapov, P. G. Spirakis, and J. Worrell. Vol. 117. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 15:1–15:15. DOI: 10.4230/LIPIcs.MFCS.2018.15.

[G6] M. Ganardi. “Visibly Pushdown Languages over Sliding Windows”. In: *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*. Ed. by R. Niedermeier and C. Paul. Vol. 126. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, 29:1–29:17. DOI: 10.4230/LIPIcs.STACS.2019.29.

[G7] M. Ganardi, D. Hucke, and M. Lohrey. “Derandomization for Sliding Window Algorithms with Strict Correctness”. In: *Proceedings of the 14th International Computer Science Symposium in Russia (CSR 2019)*. Ed. by R. van Bevern and G. Kucherov. Vol. 11532. Lecture Notes in Computer Science. Springer, 2019, pp. 237–249. DOI: 10.1007/978-3-030-19955-5_21.

[G8] M. Ganardi, D. Hucke, M. Lohrey, and T. Starikovskaya. “Sliding window property testing for regular languages”. In: *To appear in Proceedings of the 30th International Symposium on Algorithms and Computation (ISAAC 2019)*. 2019.

Other publications

[O1] M. Ganardi. “Parity Games of Bounded Tree- and Clique-Width”. In: *Proceedings of the 18th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2015)*. Ed. by A. M. Pitts. Vol. 9034. Lecture Notes in Computer Science. Springer, 2015, pp. 390–404. DOI: 10.1007/978-3-662-46678-0_25.

[O2] M. Ganardi, S. Göller, and M. Lohrey. “On the Parallel Complexity of Bisimulation on Finite Systems”. In: *Proceedings of the 25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*. Ed. by J. Talbot and L. Regnier. Vol. 62. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 12:1–12:17. DOI: 10.4230/LIPIcs.CSL.2016.12.

[O3] M. Ganardi, D. Hucke, M. Lohrey, and E. Noeth. “Tree Compression Using String Grammars”. In: *Proceedings of the 12th Latin American Symposium on Theoretical Informatics (LATIN 2016)*. Ed. by E. Kranakis, G. Navarro, and E. Chávez. Vol. 9644. Lecture Notes in Computer Science. Springer, 2016, pp. 590–604. DOI: 10.1007/978-3-662-49529-2_44.

[O4] M. Ganardi, D. Hucke, A. Jeż, M. Lohrey, and E. Noeth. “Constructing small tree grammars and small circuits for formulas”. In: *J. Comput. Syst. Sci.* 86 (2017), pp. 136–158. DOI: 10.1016/j.jcss.2016.12.007.

[O5] M. Ganardi, D. Hucke, D. König, and M. Lohrey. “Circuit Evaluation for Finite Semirings”. In: *Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*. Ed. by H. Vollmer and B. Vallée. Vol. 66. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 35:1–35:14. DOI: 10.4230/LIPIcs.STACS.2017.35.

[O6] M. Ganardi, S. Göller, and M. Lohrey. “The Complexity of Bisimulation and Simulation on Finite Systems”. In: *Log. Methods Comput. Sci.* 14.4 (2018). DOI: 10.23638/LMCS-14(4:5)2018.

[O7] M. Ganardi, D. Hucke, D. König, and M. Lohrey. “Circuits and Expressions over Finite Semirings”. In: *ACM Trans. Comput. Theory* 10.4 (2018), 15:1–15:30. DOI: 10.1145/3241375.

[O8] M. Ganardi, D. Hucke, M. Lohrey, and E. Noeth. “Tree Compression Using String Grammars”. In: *Algorithmica* 80.3 (2018), pp. 885–917. DOI: 10.1007/s00453-017-0279-3.

[O9] M. Ganardi, D. König, M. Lohrey, and G. Zetzsche. “Knapsack Problems for Wreath Products”. In: *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*. Ed. by R. Niedermeier and B. Vallée. Vol. 96. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 32:1–32:13. DOI: 10.4230/LIPIcs.STACS.2018.32.

[O10] M. Ganardi, D. Hucke, M. Lohrey, and L. Seelbach Benkner. “Universal Tree Source Coding Using Grammar-Based Compression”. In: *IEEE Trans. Inf. Theory* 65.10 (2019), pp. 6399–6413. DOI: 10.1109/TIT.2019.2919829.

[O11] M. Ganardi, A. Jeż, and M. Lohrey. “Balancing Straight-Line Programs”. In: *To appear in Proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2019)*. 2019.

[O12] M. Ganardi and M. Lohrey. “A Universal Tree Balancing Theorem”. In: *ACM Trans. Comput. Theory* 11.1 (2019), 1:1–1:25. DOI: 10.1145/3278158.

[O13] M. Ganardi and B. Khoussainov. “Automatic equivalence structures of polynomial growth”. In: *To appear in Proceedings of the 28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*. 2020.

Bibliography

[1] S. Aaronson, D. Grier, and L. Schaeffer. "A Quantum Query Complexity Trichotomy for Regular Languages". In: *Electronic Colloquium on Computational Complexity (ECCC)* 26 (2019), p. 61. URL: https://eccc.weizmann.ac.il/report/2019/061.

[2] C. C. Aggarwal, ed. *Data Streams - Models and Algorithms*. Vol. 31. Advances in Database Systems. Springer, 2007. DOI: 10.1007/978-0-387-47534-9.

[3] A. V. Aho and M. J. Corasick. "Efficient String Matching: An Aid to Bibliographic Search". In: *Commun. ACM* 18.6 (1975), pp. 333–340. DOI: 10.1145/360825.360855.

[4] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. "Regular Languages are Testable with a Constant Number of Queries". In: *SIAM J. Comput.* 30.6 (2000), pp. 1842–1862. DOI: 10.1137/S0097539700366528.

[5] N. Alon, Y. Matias, and M. Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *J. Comput. Syst. Sci.* 58.1 (1999), pp. 137–147. DOI: 10.1006/jcss.1997.1545.

[6] R. Alur, L. D'Antoni, J. V. Deshmukh, M. Raghothaman, and Y. Yuan. "Regular Functions and Cost Register Automata". In: *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013)*. IEEE Computer Society, 2013, pp. 13–22. DOI: 10.1109/LICS.2013.65.

[7] R. Alur, D. Fisman, and M. Raghothaman. "Regular Programming for Quantitative Properties of Data Streams". In: *Proceedings of the 25th European Symposium on Programming (ESOP 2016)*. Ed. by P. Thiemann. Vol. 9632. Lecture Notes in Computer Science. Springer, 2016, pp. 15–40. DOI: 10.1007/978-3-662-49498-1_2.

[8] R. Alur, V. Kumar, P. Madhusudan, and M. Viswanathan. "Congruences for Visibly Pushdown Languages". In: *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*. Ed. by L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung. Vol. 3580. Lecture Notes in Computer Science. Springer, 2005, pp. 1102–1114. DOI: 10.1007/11523468_89.

[9] R. Alur and P. Madhusudan. “Visibly pushdown languages”. In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*. Ed. by L. Babai. ACM, 2004, pp. 202–211. DOI: `10.1145/1007352.1007390`.

[10] A. Arasu and G. S. Manku. “Approximate Counts and Quantiles over Sliding Windows”. In: *Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2004)*. Ed. by C. Beeri and A. Deutsch. ACM, 2004, pp. 286–296. DOI: `10.1145/1055558.1055598`.

[11] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. “Models and Issues in Data Stream Systems”. In: *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2002)*. Ed. by L. Popa, S. Abiteboul, and P. G. Kolaitis. ACM, 2002, pp. 1–16. DOI: `10.1145/543613.543615`.

[12] B. Babcock, M. Datar, and R. Motwani. “Sampling from a moving window over streaming data”. In: *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*. Ed. by D. Eppstein. ACM/SIAM, 2002, pp. 633–634. URL: `http://dl.acm.org/citation.cfm?id=545381.545465`.

[13] B. Babcock, M. Datar, R. Motwani, and L. O’Callaghan. “Maintaining variance and k-medians over data stream windows”. In: *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2003)*. Ed. by F. Neven, C. Beeri, and T. Milo. ACM, 2003, pp. 234–243. DOI: `10.1145/773153.773176`.

[14] A. Babu, N. Limaye, J. Radhakrishnan, and G. Varma. “Streaming algorithms for language recognition problems”. In: *Theor. Comput. Sci.* 494 (2013), pp. 13–23. DOI: `10.1016/j.tcs.2012.12.028`.

[15] V. Bárány, C. Löding, and O. Serre. “Regularity Problems for Visibly Pushdown Languages”. In: *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS 2006)*. Ed. by B. Durand and W. Thomas. Vol. 3884. Lecture Notes in Computer Science. Springer, 2006, pp. 420–431. DOI: `10.1007/11672142_34`.

[16] R. Ben-Basat, G. Einziger, and R. Friedman. “Give Me Some Slack: Efficient Network Measurements”. In: *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Ed. by I. Potapov, P. G. Spirakis, and J. Worrell. Vol. 117. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 34:1–34:16. DOI: `10.4230/LIPIcs.MFCS.2018.34`.

[17] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner. “Efficient Summing over Sliding Windows”. In: *Proceedings of the 15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*. Ed. by R. Pagh. Vol. 53. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 11:1–11:14. DOI: `10.4230/LIPIcs.SWAT.2016.11`.

[18] J. Berstel. *Transductions and context-free languages*. Vol. 38. Teubner Studienbücher : Informatik. Teubner, 1979. URL: http://www.worldcat.org/oclc/06364613.

[19] A. Bouajjani, J. Esparza, and O. Maler. “Reachability Analysis of Pushdown Automata: Application to Model-Checking”. In: *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR ’97)*. Ed. by A. W. Mazurkiewicz and J. Winkowski. Vol. 1243. Lecture Notes in Computer Science. Springer, 1997, pp. 135–150. DOI: 10.1007/3-540-63141-0_10.

[20] B. von Braunmühl and R. Verbeek. “Input-Driven Languages are Recognized in log n Space”. In: *Proceedings of the 1983 International FCT-Conference (FCT 1983)*. Ed. by M. Karpinski. Vol. 158. Lecture Notes in Computer Science. Springer, 1983, pp. 40–51. DOI: 10.1007/3-540-12689-9_92.

[21] V. Braverman, E. Grigorescu, H. Lang, D. P. Woodruff, and S. Zhou. “Nearly Optimal Distinct Elements and Heavy Hitters on Sliding Windows”. In: *Proceedings of the 21st International Conference on Approximation Algorithms for Combinatorial Optimization Problems, and the 22nd International Conference on Randomization and Computation (APPROX/RANDOM 2018)*. Ed. by E. Blais, K. Jansen, J. D. P. Rolim, and D. Steurer. Vol. 116. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 7:1–7:22. DOI: 10.4230/LIPIcs.APPROX-RANDOM.2018.7.

[22] V. Braverman and R. Ostrovsky. “Smooth Histograms for Sliding Windows”. In: *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007)*. IEEE Computer Society, 2007, pp. 283–293. DOI: 10.1109/FOCS.2007.55.

[23] V. Braverman, R. Ostrovsky, and C. Zaniolo. “Optimal sampling from sliding windows”. In: *J. Comput. Syst. Sci.* 78.1 (2012), pp. 260–272. DOI: 10.1016/j.jcss.2011.04.004.

[24] D. Breslauer and Z. Galil. “Real-Time Streaming String-Matching”. In: *ACM Trans. Algorithms* 10.4 (2014), 22:1–22:12. DOI: 10.1145/2635814.

[25] N. G. de Bruijn. “A combinatorial problem”. English. In: *Nederl. Akad. Wetensch. Proc.* 49.7 (1946), pp. 758–764.

[26] J. R. Büchi. “Regular canonical systems”. In: *Arch. Math. Logik Grundlagenforschung* 6.3-4 (1964), pp. 91–111. DOI: 10.1007/BF01969548.

[27] M. Cadilhac, A. Finkel, and P. McKenzie. “Unambiguous constrained Automata”. In: *Int. J. Found. Comput. Sci.* 24.7 (2013), pp. 1099–1116. DOI: 10.1142/S0129054113400339.

[28] S. Cho and D. T. Huynh. “The Parallel Complexity of Finite-State Automata Problems”. In: *Inf. Comput.* 97.1 (1992), pp. 1–22. DOI: 10.1016/0890-5401(92)90002-W.

[29] C. Choffrut and M. P. Schützenberger. “Décomposition de Fonctions Rationnelles”. In: *Proceedings of the 3rd Annual Symposium on Theoretical Aspects of Computer Science (STACS 86)*. Ed. by B. Monien and G. Vidal-Naquet. Vol. 210. Lecture Notes in Computer Science. Springer, 1986, pp. 213–226. DOI: 10.1007/3-540-16078-7_78.

[30] R. Clifford, A. Fontaine, E. Porat, B. Sach, and T. A. Starikovskaya. “Dictionary Matching in a Stream”. In: *Proceedings of the 23rd Annual European Symposium (ESA 2015)*. Ed. by N. Bansal and I. Finocchi. Vol. 9294. Lecture Notes in Computer Science. Springer, 2015, pp. 361–372. DOI: 10.1007/978-3-662-48350-3_31.

[31] R. Clifford, A. Fontaine, E. Porat, B. Sach, and T. A. Starikovskaya. “The k-mismatch problem revisited”. In: *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*. Ed. by R. Krauthgamer. SIAM, 2016, pp. 2039–2052. DOI: 10.1137/1.9781611974331.ch142.

[32] R. Clifford, T. Kociumaka, and E. Porat. “The streaming k-mismatch problem”. In: *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*. Ed. by T. M. Chan. SIAM, 2019, pp. 1106–1125. DOI: 10.1137/1.9781611975482.68.

[33] R. Clifford and T. A. Starikovskaya. “Approximate Hamming Distance in a Stream”. In: *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Ed. by I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi. Vol. 55. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 20:1–20:14. DOI: 10.4230/LIPIcs.ICALP.2016.20.

[34] E. Cohen and M. J. Strauss. “Maintaining time-decaying stream aggregates”. In: *J. Algorithms* 59.1 (2006), pp. 19–36. DOI: 10.1016/j.jalgor.2005.01.006.

[35] G. Cormode and S. Muthukrishnan. “An improved data stream summary: the count-min sketch and its applications”. In: *J. Algorithms* 55.1 (2005), pp. 58–75. DOI: 10.1016/j.jalgor.2003.12.001.

[36] M. Datar, A. Gionis, P. Indyk, and R. Motwani. “Maintaining Stream Statistics over Sliding Windows”. In: *SIAM J. Comput.* 31.6 (2002), pp. 1794–1813. DOI: 10.1137/S0097539701398363.

[37] M. Datar and S. Muthukrishnan. “Estimating Rarity and Similarity over Data Stream Windows”. In: *Proceedings of the 10th European Symposium on Algorithms (ESA 2002)*. Ed. by R. H. Möhring and R. Raman. Vol. 2461. Lecture Notes in Computer Science. Springer, 2002, pp. 323–334. DOI: 10.1007/3-540-45749-6_31.

[38] M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer, 2009.

[39] P. W. Dymond. “Input-Driven Languages are in log n Depth”. In: *Inf. Process. Lett.* 26.5 (1988), pp. 247–250. DOI: 10.1016/0020-0190(88)90148-2.

[40] S. Eilenberg. *Automata, languages, and machines*. Vol. A. Pure and applied mathematics. Academic Press, 1974.

[41] S. Eilenberg. *Automata, languages, and machines*. Vol. B. Pure and applied mathematics. Academic Press, 1976.

[42] S. Eilenberg and M.-P. Schützenberger. “Rational sets in commutative monoids”. In: *J. Algebra* 13.2 (1969), pp. 173–191.

[43] C. C. Elgot and J. E. Mezei. “On Relations Defined by Generalized Finite Automata”. In: *IBM J. Res. Dev.* 9.1 (1965), pp. 47–68. DOI: 10.1147/rd.91.0047.

[44] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. “Testing and Spot-Checking of Data Streams”. In: *Algorithmica* 34.1 (2002), pp. 67–80. DOI: 10.1007/s00453-002-0959-4.

[45] J. Feigenbaum, S. Kannan, and J. Zhang. “Computing Diameter in the Streaming and Sliding-Window Models”. In: *Algorithmica* 41.1 (2005), pp. 25–41. DOI: 10.1007/s00453-004-1105-2.

[46] E. Filiot, O. Gauwin, P. Reynier, and F. Servais. “Streamability of nested word transductions”. In: *Log. Meth. Comput. Sci.* 15.2 (2019). URL: https://lmcs.episciences.org/5348.

[47] E. Filiot and P. Reynier. “Transducers, logic and algebra for functions of finite words”. In: *SIGLOG News* 3.3 (2016), pp. 4–19. URL: https://dl.acm.org/citation.cfm?id=2984453.

[48] E. Fischer, F. Magniez, and T. A. Starikovskaya. “Improved bounds for testing Dyck languages”. In: *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*. Ed. by A. Czumaj. SIAM, 2018, pp. 1529–1544. DOI: 10.1137/1.9781611975031.100.

[49] P. Flajolet. “Approximate Counting: A Detailed Analysis”. In: *BIT* 25.1 (1985), pp. 113–134.

[50] P. Flajolet and G. N. Martin. “Probabilistic Counting Algorithms for Data Base Applications”. In: *J. Comput. Syst. Sci.* 31.2 (1985), pp. 182–209. DOI: 10.1016/0022-0000(85)90041-8.

[51] L. Fleischer and M. Kufleitner. “Green’s Relations in Deterministic Finite Automata”. In: *Theory Comput. Syst.* 63.4 (2019), pp. 666–687. DOI: 10.1007/s00224-018-9847-4.

[52] N. François, F. Magniez, M. de Rougemont, and O. Serre. “Streaming Property Testing of Visibly Pushdown Languages”. In: *Proceedings of the 24th Annual European Symposium on Algorithms (ESA 2016)*. Ed. by P. Sankowski and C. D. Zaroliagis. Vol. 57. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 43:1–43:17. DOI: 10.4230/LIPIcs.ESA.2016.43.

[53] P. Gawrychowski. "Chrobak Normal Form Revisited, with Applications". In: *Proceedings of the 16th International Conference on Implementation and Application of Automata (CIAA 2011)*. Ed. by B. Bouchou-Markhoff, P. Caron, J. Champarnaud, and D. Maurel. Vol. 6807. Lecture Notes in Computer Science. Springer, 2011, pp. 142–153. DOI: 10.1007/978-3-642-22256-6_14.

[54] P. Gawrychowski and A. Jeż. "Hyper-minimisation Made Efficient". In: *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science 2009 (MFCS 2009)*. Ed. by R. Královic and D. Niwinski. Vol. 5734. Lecture Notes in Computer Science. Springer, 2009, pp. 356–368. DOI: 10.1007/978-3-642-03816-7_31.

[55] P. Gawrychowski, D. Krieger, N. Rampersad, and J. Shallit. "Finding the Growth Rate of a Regular or Context-Free Language in Polynomial Time". In: *Int. J. Found. Comput. Sci.* 21.4 (2010), pp. 597–618. DOI: 10.1142/S0129054110007441.

[56] P. B. Gibbons and S. Tirthapura. "Distributed Streams Algorithms for Sliding Windows". In: *Theory Comput. Syst.* 37.3 (2004), pp. 457–478. DOI: 10.1007/s00224-004-1156-4.

[57] R. H. Gilman. "Formal languages and infinite groups". In: *DIMACS Ser. Discrete Math. Theoret. Comput. Sci* 25 (1996), pp. 27–51.

[58] S. Ginsburg and E. Spanier. "Finite-Turn Pushdown Automata". In: *SIAM J. Control* 4.3 (1966), pp. 429–453. DOI: 10.1137/0304034.

[59] S. Ginsburg and E. H. Spanier. "Bounded ALGOL-like languages". In: *Trans. Amer. Math. Soc.* 113.2 (1964), pp. 333–368. DOI: 10.1090/S0002-9947-1964-0181500-1.

[60] L. Golab, D. DeHaan, E. D. Demaine, A. López-Ortiz, and J. I. Munro. "Identifying frequent items in sliding windows over on-line packet streams". In: *Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference (IMC 2003)*. ACM, 2003, pp. 173–178. DOI: 10.1145/948205.948227.

[61] S. Golan, T. Kopelowitz, and E. Porat. "Streaming Pattern Matching with d Wildcards". In: *Proceedings of the 24th Annual European Symposium on Algorithms (ESA 2016)*. Ed. by P. Sankowski and C. D. Zaroliagis. Vol. 57. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 44:1–44:16. DOI: 10.4230/LIPIcs.ESA.2016.44.

[62] S. Golan, T. Kopelowitz, and E. Porat. "Towards Optimal Approximate Streaming Pattern Matching by Matching Multiple Patterns in Multiple Streams". In: *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Ed. by I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 65:1–65:16. DOI: 10.4230/LIPIcs.ICALP.2018.65.

[63] S. Golan and E. Porat. “Real-Time Streaming Multi-Pattern Search for Constant Alphabet”. In: *Proceedings of the 25th Annual European Symposium on Algorithms (ESA 2017)*. Ed. by K. Pruhs and C. Sohler. Vol. 87. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 41:1–41:15. DOI: 10.4230/LIPIcs.ESA.2017.41.

[64] O. Goldreich, S. Goldwasser, and D. Ron. “Property Testing and its Connection to Learning and Approximation”. In: *J. ACM* 45.4 (1998), pp. 653–750. DOI: 10.1145/285055.285060.

[65] J. Hartmanis and H. Shank. “Two Memory Bounds for the Recognition of Primes by Automata”. In: *Math. Syst. Theory* 3.2 (1969), pp. 125–129. DOI: 10.1007/BF01746518.

[66] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[67] N. Immerman. “Nondeterministic Space is Closed Under Complementation”. In: *SIAM J. Comput.* 17.5 (1988), pp. 935–938. DOI: 10.1137/0217058.

[68] P. Indyk. “Stable distributions, pseudorandom generators, embeddings, and data stream computation”. In: *J. ACM* 53.3 (2006), pp. 307–323. DOI: 10.1145/1147954.1147955.

[69] F. Jahn, M. Kufleitner, and A. Lauser. “Regular Ideal Languages and Their Boolean Combinations”. In: *Proceedings of the 17th International Conference on Implementation and Application of Automata (CIAA 2012)*. Ed. by N. Moreira and R. Reis. Vol. 7381. Lecture Notes in Computer Science. Springer, 2012, pp. 205–216. DOI: 10.1007/978-3-642-31606-7_18.

[70] G. Jirásková and P. Mlynárcik. “Complement on Prefix-Free, Suffix-Free, and Non-Returning NFA Languages”. In: *Proceedings of the 16th International Workshop on Descriptional Complexity of Formal Systems (DCFS 2014)*. Ed. by H. Jürgensen, J. Karhumäki, and A. Okhotin. Vol. 8614. Lecture Notes in Computer Science. Springer, 2014, pp. 222–233. DOI: 10.1007/978-3-319-09704-6_20.

[71] M. Kaminski and N. Francez. “Finite-Memory Automata”. In: *Theor. Comput. Sci.* 134.2 (1994), pp. 329–363. DOI: 10.1016/0304-3975(94)90242-9.

[72] R. M. Karp. “Some Bounds on the Storage Requirements of Sequential Machines and Turing Machines”. In: *J. ACM* 14.3 (1967), pp. 478–489. DOI: 10.1145/321406.321410.

[73] R. M. Karp and M. O. Rabin. “Efficient Randomized Pattern-Matching Algorithms”. In: *IBM J. Res. Dev.* 31.2 (1987), pp. 249–260. DOI: 10.1147/rd.312.0249.

[74] B. Khoussainov and A. Nerode. “Automatic Presentations of Structures”. In: *International Workshop on Logic and Computational Complexity (LCC ’94)*. Ed. by D. Leivant. Vol. 960. Lecture Notes in Computer Science. Springer, 1994, pp. 367–392. DOI: `10.1007/3-540-60178-3_93`.

[75] S. C. Kleene. “Representation of events in nerve nets and finite automata”. In: *Automata Studies*. Ed. by C. Shannon and J. McCarthy. Princeton, N.J.: Princeton University Press, 1956, pp. 3–42.

[76] D. E. Knuth. “Big Omicron and Big Omega and Big Theta”. In: *SIGACT News* 8.2 (Apr. 1976), pp. 18–24. DOI: `10.1145/1008328.1008329`.

[77] D. E. Knuth, J. H. Morris Jr., and V. R. Pratt. “Fast Pattern Matching in Strings”. In: *SIAM J. Comput.* 6.2 (1977), pp. 323–350. DOI: `10.1137/0206024`.

[78] D. Kozen. *Automata and computability*. Undergraduate texts in computer science. Springer, 1997.

[79] A. Krebs, K. Lange, and M. Ludwig. “Visibly Counter Languages and Constant Depth Circuits”. In: *Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*. Ed. by E. W. Mayr and N. Ollinger. Vol. 30. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 594–607. DOI: `10.4230/LIPIcs.STACS.2015.594`.

[80] A. Krebs, N. Limaye, and S. Srinivasan. “Streaming Algorithms for Recognizing Nearly Well-Parenthesized Expressions”. In: *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS 2011)*. Ed. by F. Murlak and P. Sankowski. Vol. 6907. Lecture Notes in Computer Science. Springer, 2011, pp. 412–423. DOI: `10.1007/978-3-642-22993-0_38`.

[81] I. Kremer, N. Nisan, and D. Ron. “On Randomized One-Round Communication Complexity”. In: *Comput. Complex.* 8.1 (1999), pp. 21–49. DOI: `10.1007/s000370050018`.

[82] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1997.

[83] O. Lachish, I. Newman, and A. Shapira. “Space Complexity Vs. Query Complexity”. In: *Comput. Complex.* 17.1 (2008), pp. 70–93. DOI: `10.1007/s00037-008-0239-z`.

[84] P. M. Lewis II, R. E. Stearns, and J. Hartmanis. “Memory bounds for recognition of context-free and context-sensitive languages”. In: *Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*. IEEE Computer Society, 1965, pp. 191–202. DOI: `10.1109/FOCS.1965.14`.

[85] M. Lohrey and C. Mathissen. “Isomorphism of regular trees and words”. In: *Inf. Comput.* 224 (2013), pp. 71–105. DOI: `10.1016/j.ic.2013.01.002`.

[86] F. Magniez, C. Mathieu, and A. Nayak. “Recognizing Well-Parenthesized Expressions in the Streaming Model”. In: *SIAM J. Comput.* 43.6 (2014), pp. 1880–1905. DOI: 10.1137/130926122.

[87] K. Mamouras, M. Raghothaman, R. Alur, Z. G. Ives, and S. Khanna. “StreamQRE: modular specification and efficient evaluation of quantitative queries over streaming data”. In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017)*. Ed. by A. Cohen and M. T. Vechev. ACM, 2017, pp. 693–708. DOI: 10.1145/3062341.3062369.

[88] A. R. Meyer and L. J. Stockmeyer. “The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space”. In: *Proceedings of the 13th Annual Symposium on Switching and Automata Theory (SWAT 1972)*. IEEE Computer Society, 1972, pp. 125–129. DOI: 10.1109/SWAT.1972.29.

[89] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. 2nd. New York, NY, USA: Cambridge University Press, 2017.

[90] R. H. Morris Sr. “Counting Large Numbers of Events in Small Registers”. In: *Commun. ACM* 21.10 (1978), pp. 840–842. DOI: 10.1145/359619.359627.

[91] J. I. Munro and M. Paterson. “Selection and Sorting with Limited Storage”. In: *Theor. Comput. Sci.* 12 (1980), pp. 315–323. DOI: 10.1016/0304-3975(80)90061-4.

[92] S. Muthukrishnan. “Data Streams: Algorithms and Applications”. In: *Found. Trends Theor. Comput. Sci.* 1.2 (2005). DOI: 10.1561/0400000002.

[93] F. Neven, T. Schwentick, and V. Vianu. “Finite state machines for strings over infinite alphabets”. In: *ACM Trans. Comput. Log.* 5.3 (2004), pp. 403–435. DOI: 10.1145/1013560.1013562.

[94] R. Parikh. “On Context-Free Languages”. In: *J. ACM* 13.4 (1966), pp. 570–581. DOI: 10.1145/321356.321364.

[95] F. Parlamento, A. Policriti, and K. Rao. “Witnessing Differences Without Redundancies”. In: *P. Am. Math. Soc.* 125.2 (1997), pp. 587–594. DOI: 10.1090/S0002-9939-97-03630-7.

[96] M. Parnas, D. Ron, and R. Rubinfeld. “Testing membership in parenthesis languages”. In: *Random Struct. Algor.* 22.1 (2003), pp. 98–138. DOI: 10.1002/rsa.10067.

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

[98] B. Porat and E. Porat. “Exact and Approximate Pattern Matching in the Streaming Model”. In: *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*. IEEE Computer Society, 2009, pp. 315–323. DOI: 10.1109/FOCS.2009.11.

[99] M. O. Rabin. “Probabilistic Automata”. In: *Inform. Control* 6.3 (1963), pp. 230–245. DOI: 10.1016/S0019-9958(63)90290-0.

[100] C. Reutenauer and M. P. Schützenberger. “Minimization of Rational Word Functions”. In: *SIAM J. Comput.* 20.4 (1991), pp. 669–685. DOI: 10.1137/0220042.

[101] J. B. Rosser and L. Schoenfeld. “Approximate formulas for some functions of prime numbers”. In: *Illinois J. Math.* 6.1 (1962), pp. 64–94.

[102] J. Shallit and Y. Breitbart. “Automaticity I: Properties of a Measure of Descriptional Complexity”. In: *J. Comput. Syst. Sci.* 53.1 (1996), pp. 10–25. DOI: 10.1006/jcss.1996.0046.

[103] R. Shaltiel. “Weak Derandomization of Weak Algorithms: Explicit Versions of Yao’s Lemma”. In: *Comput. Complex.* 20.1 (2011), pp. 87–143. DOI: 10.1007/s00037-011-0006-4.

[104] T. A. Starikovskaya. “Communication and Streaming Complexity of Approximate Pattern Matching”. In: *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*. Ed. by J. Kärkkäinen, J. Radoszewski, and W. Rytter. Vol. 78. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 13:1–13:11. DOI: 10.4230/LIPIcs.CPM.2017.13.

[105] R. E. Stearns, J. Hartmanis, and P. M. Lewis II. “Hierarchies of memory limited computations”. In: *Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*. IEEE Computer Society, 1965, pp. 179–190. DOI: 10.1109/FOCS.1965.11.

[106] H. Straubing. “Finite semigroup varieties of the form $\mathrm{V} * \mathrm{D}$”. In: *J. Pure Appl. Algebra* 36 (1985), pp. 53–94. DOI: 10.1016/0022-4049(85)90062-3.

[107] A. Szilard, S. Yu, K. Zhang, and J. Shallit. “Characterizing Regular Languages with Polynomial Densities”. In: *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science 1992 (MFCS ’92)*. Ed. by I. M. Havel and V. Koubek. Vol. 629. Lecture Notes in Computer Science. Springer, 1992, pp. 494–503. DOI: 10.1007/3-540-55808-X_48.

[108] K. Tangwongsan, M. Hirzel, and S. Schneider. “Low-Latency Sliding-Window Aggregation in Worst-Case Constant Time”. In: *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS 2017)*. ACM, 2017, pp. 66–77. DOI: 10.1145/3093742.3093925.

[109] D. Thérien. “Classification of Finite Monoids: The Language Approach”. In: *Theor. Comput. Sci.* 14 (1981), pp. 195–208. DOI: 10.1016/0304-3975(81)90057-8.

[110] V. I. Trofimov. “Growth functions of some classes of languages”. In: *Cybernetics* 17.6 (1981), pp. 727–731.

[111] J. S. Vitter. “Random Sampling with a Reservoir”. In: *ACM Trans. Math. Softw.* 11.1 (1985), pp. 37–57. DOI: 10.1145/3147.3165.

[112] A. Weber and R. Klemm. “Economy of Description for Single-Valued Transducers”. In: *Inf. Comput.* 118.2 (1995), pp. 327–340. DOI: 10.1006/inco.1995.1071.

[113] A. C. Yao. “Probabilistic Computations: Toward a Unified Measure of Complexity (Extended Abstract)”. In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*. IEEE Computer Society, 1977, pp. 222–227. DOI: 10.1109/SFCS.1977.24.