Algorithmica manuscript No.
(will be inserted by the editor)

Counting and generating permutations in regular classes.

Nicolas Basset

Received: date / Accepted: date

Abstract The signature of a permutation σ is a word $\mathtt{sg}(\sigma) \subseteq \{\mathfrak{a}, \mathfrak{d}\}^*$ with ith letter $\mathfrak{d}$ when σ has a descent (i.e. $\sigma(i) > \sigma(i+1)$) and $\mathfrak{a}$ when σ has an ascent (i.e. $\sigma(i) < \sigma(i+1)$). The combinatorics of permutations with a prescribed signature is quite well explored. Here we introduce regular classes of permutations, the sets $\Lambda(L)$ of permutations with signature in regular languages $L \subseteq \{\mathfrak{a}, \mathfrak{d}\}^*$. Given a regular class of permutations we (i) count the permutations of a given length within the class; (ii) compute a closed form formula for the exponential generating function; and (iii) sample uniformly at random the permutation of a given length. We first recall how (i) is solved in the literature for the case of a single signature. We then explain how to extend these methods to regular classes of permutations using language equations from automata theory. We give two methods to solve (ii) in terms of exponentials of matrices. For the third problem we provide both discrete and continuous recursive methods as well as an extension of Boltzmann sampling to uncountable union of sets parametrised by a variable ranging over an interval. Last but not least, a part of our contributions are based on a geometric interpretation of a subclass of regular timed languages (that is, recognised by timed automata specific to our problem).

Keywords Regular class of permutations · Signature of a permutation · Uniform random sampling · Exponential generating function · Timed automata · Boltzmann sampling

Contents

This research is supported in part by ERC Advanced Grant VERIWARE and was also supported by the ANR project EQINOCS (ANR-11-BS02-004).

N. Basset
University of Oxford, Department of Computer Science.
E-mail: basset@cs.ox.ac.uk

1 Introduction

Counting the permutations with a prescribed signature (described in the abstract) is a classical combinatorial topic (see [Luc14] and reference therein).

A very well studied example of permutations given by their signatures are the so-called alternating (or zig-zag, or down-up) permutations (see [Sta10] for a survey). Their signatures belong to the language expressed by the regular expression $(\mathfrak{d}\mathfrak{a})^*(\mathfrak{d}+\varepsilon)$ (in other words they satisfy $\sigma_1 > \sigma_2 < \sigma_3 > \sigma_4 ...$).

We associate to a language $L \subseteq \{\mathfrak{a},\mathfrak{d}\}^*$ the class $\Lambda(L)$ of permutations with signature in L. When the language L is regular (namely, recognised by a finite state automaton), we say that the class of permutations $\Lambda(L)$ is *regular*. Many classes of permutations can be expressed in that way; for instance, alternating permutations or those with an even number of descents.

In this paper, we study the combinatorics of regular classes of permutations. We are thus interested in the sequence $(\alpha_n(L))_{n\in\mathbb{N}}$ of the cardinalities of the sets of permutations of length $n \in \mathbb{N}$ with signature in L. We address the problem of

Problem 1: computing elements of the sequence $(\alpha_n(L))_{n\in\mathbb{N}}$;
Problem 2: computing its exponential generating function (EGF), that is, the formal power series $G_L(z) \stackrel{\text{def}}{=} \sum \alpha_n(L)\frac{z^n}{n!}$;
Problem 3: generating uniformly at random permutations in $\Lambda(L)$ so that permutations of length n in $\Lambda(L)$ have all the same probability $1/\alpha_n(L)$ to be returned.

We propose two main approaches to solve these problems.

The first approach is an extension of previous results on the subject designed for the particular problem of counting and randomly generating permutations with a prescribed signature. Within this approach, there are two sub-approaches: one remains in the discrete world of permutations while the other consider vectors $(\nu_1, \nu_2, \ldots, \nu_n)$ of the hyper-cube $[0,1]^n$. To a permutation corresponds the *order polytope* of all the vectors having the same ordering of its coordinates as the permutation. Such order polytopes are also defined for signatures and are particular cases of Stanley's poset polytopes [Sta86]. The main novelty of our approach is to introduce the dynamics of automata into the recursive equation defining the order polytopes or the set of permutations. We consider families of polytopes called ordered sets, that are parametrised by the state of an automaton recognising the regular language under consideration. Then, we write systems of equations on these ordered sets mimicking the equation on languages of automata theory. This allows us to compute the coefficient $\alpha_n(L)$ recursively and to characterise the generating functions as the solution of a system of integral equations.

The second approach is based on a connection between the regular classes of permutations and the volumetry of regular timed languages (the languages recognised by the so-called timed automata). Timed automata were introduced in [AD94] to model and verify properties of real-time systems. The volumetry of timed language is a more recent theory initiated by Asarin and Degorre. We refer the reader to [ABD15] and to our PhD thesis [Bas13] for an overview of results of this theory. The connection is in two steps. First, we recall the link between the order polytopes of the first approach and the *chain polytopes* which are a second type of Stanley's poset polytopes [Sta86]; then we interpret the chain polytopes of a signature w as the set of delays that together with w forms a timed word of a well chosen timed language.

A note on asymptotic behaviours and bit complexity. Another meaningful problem is to study the asymptotic behaviour of $\alpha_n(L)$. A first information on the asymptotic growth rate of $(\alpha_n(L))_{n\in\mathbb{N}}$ can be obtained by knowing the radius of convergence R of the EGF:

$$1/R = \limsup_{n\to+\infty} \left(\frac{\alpha_n(L)}{n!}\right)^{1/n} \tag{1}$$

This result is known as the Cauchy-Hadamard Theorem. Note that as $\frac{\alpha_n(L)}{n!} \geq 1$, the quantity $1/R$ belongs to $[0,1]$. The logarithm of this quantity is often called the *entropy*. In the second approach sketched above, we interpret the sequence $(\alpha_n(L))_{n\in\mathbb{N}}$, in terms of the volume sequence associated to a timed language. Hence, one can use the theory of volume and entropy of timed languages presented in [ABD15] to analyse the growth rate of $(\alpha_n(L))_{n\in\mathbb{N}}$ (without need for generating function). More precisely the quantity $1/R$ is the spectral radius of a functional operator akin to the integral operators considered in the present paper. We explained the link between the operator approach and generating function for timed languages in [ABDP12].

To get finer measures, one can study more precisely the generating function around the convergence radius or alternatively study more precisely the spectral theory of the integral operator under consideration. The former option is extensively described in [FS09] while the latter has been explored in [EKP11, EJ12] for classes of permutations defined by consecutive descent pattern avoidance.

In the present paper, we establish complexity results in terms of elementary arithmetic operations (we speak of arithmetic complexity). In practice, the bit complexity comes into play. Indeed the numbers handled such as $\alpha_n(L)$ needs $\Theta(n \log n)$ bits to be stored as they have order of magnitude $n!$. An idea to reduce this complexity is to handle and store directly numbers of the form $\frac{\alpha_n(L)}{n!}$. According to (1), to store these numbers, one needs asymptotically $\Theta(n \log R)$ bits assuming that $R < +\infty$.

Contribution summary We propose the following contributions. In the following m is the size of an automaton that recognises the regular languages of signature of interest, and n is the length of permutation one wants to count or generate.

- In the first approach we extend the discrete and the continuous recursive methods for counting permutations of a single signature to the case of a regular class of signatures (Corollary 1 and Proposition 6). This method has an arithmetic time complexity $O(n^2 m)$. These recursive methods of counting transfers naturally to a recursive method of uniform sampling for regular classes of permutations (Algorithm 3). We further characterise in Proposition 7 the generating functions in terms of a system of integral equations (derived from a system of language equations) and give explicit solutions in terms of (integrals of) exponentials of matrices (Theorem 1).
- We describe in the second approach, the link between regular classes of permutations and the volumetry of regular timed languages. In particular, the wanted EGF is equal to a volume generating function associated to a well chosen timed language. We characterise the generating function using a new system of integral equations (Theorem 3) that yields a characterisation in terms of exponential of a matrix (Theorem 4). This second approach also allows one to describe signatures of permutations directly in terms of straights (aka. double-ascents and double-descents) and turns (aka. peaks and valleys).
- From the characterisation of the generating function of Theorem 4, one can compute a closed form formula in exponential time complexity with respect to m.
- Given an automaton of size m, Algorithm 2 allows one to find the first n terms of the generating function in arithmetic time complexity $O(n \log(n) m^3)$.
- We show with Theorem 8 how random generation of timed words can be used for random generation of permutations. Then we describe a continuous recursive method to generate timed words and hence to give another solution to Problem 3.
- We extend Boltzmann sampling to our framework involving uncountable union of sets parametrised by a real valued variable, giving a third solution to Problem 3.

– We have implemented a part of the algorithms using the computer algebra system Sage [S+15] and illustrate them on a running example.

1.1 Related works

A part of the present paper is the chapter 8 of the PhD thesis [Bas13] and was presented in [Bas14]. Our work is mainly inspired by our previous work on volumetry, entropy and generating functions for timed languages [ABDP12, ABD15, Bas13]. In particular, the link between enumerative combinatorics and timed languages that we establish here was foreseen in [ABDP12]. No particular knowledge of the (timed) automata theory or the combinatorics of permutations is required to read the paper.

Other classes of permutations considered. The random generation of permutations with signature following an arbitrary signature has been addressed very recently by Philippe Marchal [Mar14]. In this work the computation of the exponential generating function for classes of permutations that follow a periodic pattern is also addressed. In another recent work [Luc14], this latter problem is also solved. This paper also establishes a link between the discrete and the continuous approach of previous works and the study of the entropy for classes of permutations following a periodic pattern. Another interest of this paper is its motivation from statistical physics.

Particular regular languages of signatures are considered in [EJ12] under the name of consecutive descent pattern avoidance. Numerous other works treat more general cases of (consecutive) pattern avoidance (see [EN03], [Kit11]) and are quite incomparable to our work. Indeed, certain classes of permutations avoiding a finite set of patterns cannot be described as a language of signatures while some classes of permutations involving regular languages cannot be described by finite pattern avoidance (for instance, the permutations with an even number of descents).

About the recursive method for uniform random sampling. We use the so-called recursive method introduced by [NW78] and developed by [FZVC94]. This method has been improved for the particular case of uniform random generation of words in regular languages [BG12, ODG13] (see the latter reference for experimental comparison). Bit complexity issues were already discussed above. It is known that one can decrease the bit complexity of the recursive method by using floating point arithmetic, even without introducing a bias in the sampling [DZ99, BG12]. The random sampler of timed words (Algorithm 4) is an adaptation of this method to the timed case. We give related works on Boltzmann sampling in Section 6.

1.2 Paper structure

In Section 2, we give some preliminary definitions and discuss how certain classes of permutations considered in the literature can be seen as regular classes of permutations. In Section 3 and 4 we describe the first and the second approaches sketched above. Section 5 is devoted to uniform random sampling. We discuss the results and perspectives as well as further related works in Section 6.

2 Preliminaries

Throughout this paper we use two *alphabets*, $\{\mathfrak{a}, \mathfrak{d}\}$ and $\{\mathfrak{s}, \mathfrak{t}\}$, whose elements should be respectively read as "ascent", "descent", "straight" and "turn". A *signature* is a word on the alphabet $\{\mathfrak{a}, \mathfrak{d}\}$. The empty word (the unique 0-length word) is denoted by ε. The concatenation of two languages A and B is denoted by $AB = \{uw \mid u \in A \text{ and } w \in B\}$. If $A = l$ where l is a single letter, we write lB instead of $\{l\}B$. Given a language A and a natural number k, we denote by A^k the set $\{u_1 \dots u_k \mid u_i \in A\}$, in particular $A^0 = \{\varepsilon\}$. The *Kleene star closure* of A is defined by $A^* = \cup_{k \in \mathbb{N}} A^k$. In particular, the set of all words over the alphabet Σ is denoted by Σ^*.

Example 1 We consider as a running example the regular language

$$L^{(\text{run})} = (\{\mathfrak{aa}, \mathfrak{dd}\})^* \{\mathfrak{a}, \mathfrak{d}\}.$$

It is composed of words that are concatenation of blocks of consecutive ascents (or descents) of even length followed by an odd length block of ascents (or descents). For example the following three words are in $L^{(\text{run})}$: $\mathfrak{aaa}$, $\mathfrak{aad}$ and $\mathfrak{ddddaaddd}$.

Automata and regular languages A deterministic finite state automaton (hereafter simply called automaton) is a tuple $\mathcal{T} = (\Sigma, Q, q_0, F, \delta)$ where Σ is a finite alphabet; Q is a finite set of *states*; $q_0 \in Q$ is the *initial state*; $F \subseteq Q$ is the set of *final states*; and $\delta : Q \times \Sigma \to Q$ is a partial *transition function*. For a state p and a letter $l \in \Sigma$, we let $p.l$ stand for $\delta(p, l)$ whenever it is defined, otherwise we use the convention that sets and real-valued functions parametrised by $p.l$ (e.g. in (2) or (30)) are respectively empty and null. This notation extends inductively on words as follows $p.\varepsilon = p$ and $p.ua = (p.u).a$ for $u \in \Sigma^*$ and $a \in \Sigma$.

We denote by $[\mathcal{T}_p]_n$ the set of words of length $n \in \mathbb{N}$ recognised by $\mathcal{T}$ from a state p defined recursively as follows: $[\mathcal{T}_p]_0 = \{\varepsilon\}$ if p is a final state, i.e. $p \in F$, and $[\mathcal{T}_p]_0 = \emptyset$ otherwise; and for $n \in \mathbb{N}$

$$[\mathcal{T}_p]_{n+1} = \bigcup_{l \in \Sigma} l[\mathcal{T}_{p.l}]_n \tag{2}$$

The language recognised by $\mathcal{T}$ from the state $p \in Q$ is $[\mathcal{T}_p] \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}} [\mathcal{T}_p]_n$. The language recognised by $\mathcal{T}$ is the language $[\mathcal{T}_{q_0}]$, that is, recognised from the initial state q_0. The languages recognised by automata are called *regular languages*.

Alternatively one can define directly the languages $([\mathcal{T}_p])_{p \in Q}$ as the unique solution of the following language equation:

$$[\mathcal{T}_p] = \bigcup_{l \in \Sigma} l[\mathcal{T}_{p.l}] \quad (\cup\{\varepsilon\} \text{ if } p \in F). \tag{3}$$

We assume without loss of generality that all the states of automata considered in this paper are reachable, that is, of the form $q_0.u$ for some $u \in \Sigma^*$ (non reachable states can be deleted without changing the language of the automaton).

In this paper, automata will be denoted by $\mathcal{A}$ when their alphabet is $\{\mathfrak{a}, \mathfrak{d}\}$ and by $\mathcal{B}$ when their alphabet is $\{\mathfrak{s}, \mathfrak{t}\}$.

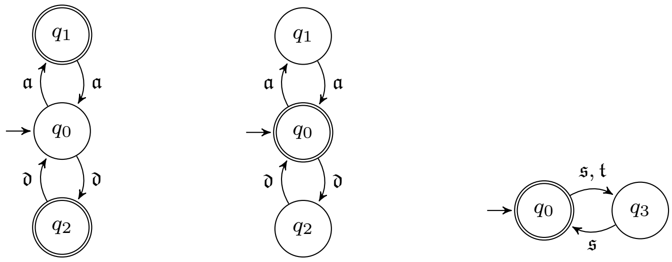

Fig. 1 From left to right: automata for $L^{(\text{run})}$, $L^{(\text{run})'} \cup \{\varepsilon\}$ and $\mathtt{st}_{\mathfrak{a}}(L^{(\text{run})'}) \cup \{\varepsilon\}$. The final states are marked with double circle, the initial state q_0 by an incoming arrow and labelled arrows $q_i \xrightarrow{l} q_j$ mean that $q_i.l = q_j$

Example 2 Consider the automaton $\mathcal{A}$ on Figure 1. (2) and (3) gives

$$\begin{cases} [\mathcal{T}_{q_0}]_{n+1} = \mathfrak{a}[\mathcal{T}_{q_1}]_n \ \bigcup \mathfrak{d}[\mathcal{T}_{q_2}]_n; \\ [\mathcal{T}_{q_1}]_{n+1} = \mathfrak{a}[\mathcal{T}_{q_0}]_n; \\ [\mathcal{T}_{q_2}]_{n+1} = \mathfrak{d}[\mathcal{T}_{q_0}]_n. \end{cases} \quad \text{and} \quad \begin{cases} [\mathcal{T}_{q_0}] = \mathfrak{a}[\mathcal{T}_{q_1}] \bigcup \mathfrak{d}[\mathcal{T}_{q_2}]; \\ [\mathcal{T}_{q_1}] = \mathfrak{a}[\mathcal{T}_{q_0}] \bigcup \{\varepsilon\}; \\ [\mathcal{T}_{q_2}] = \mathfrak{d}[\mathcal{T}_{q_0}] \bigcup \{\varepsilon\}. \end{cases}$$

It can be seen that the language recognised by $\mathcal{A}$ is exactly the language $L^{(\text{run})}$ described in Example 1.

Matrix notation for system of equations parametrised by states of an automaton. Let $\mathcal{T} = (\Sigma, Q, q_0, F, \delta)$ be an automaton. In several places in this paper, we will consider the family of functions $f_p : x, z \mapsto f_p(x, z)$ indexed by states $p \in Q$ of the automaton under consideration. We denote by $\mathbf{f}(x, z)$, the column vector with qth component $f_q(x, z)$. Integration and derivation of vectors of functions are taken componentwise; for instance, $\int_0^1 \mathbf{f}(y, z)dy$ is the vector with qth component $\int_0^1 f_q(y, z)dy$. We denote by $\mathbf{F}$ the column vector with qth component 1 if $q \in F$ and 0 otherwise. For $l \in \Sigma$, the $Q \times Q$-matrix M_l for $l \in \Sigma$ is the adjacency matrix corresponding to letter l. That is, for $p, q \in Q$, $M_l(p, q) = 1$ if $p.l = q$ and 0 otherwise.

The signature of a permutation and regular classes of permutations. For $n \in \mathbb{N}$, $[n]$ denotes $\{1, \ldots, n\}$ and $\mathfrak{S}_n$ the set of permutations of $[n]$. We use the one line notation for permutations. For example, $\sigma = 231$ means that $\sigma(1) = 2$, $\sigma(2) = 3$, $\sigma(3) = 1$.

Let n be a positive integer. The *signature* of a permutation $\sigma = \sigma_1 \cdots \sigma_n$ is the word $u = u_1 \cdots u_{n-1} \in \{\mathfrak{a}, \mathfrak{d}\}^{n-1}$ denoted by $\mathtt{sg}(\sigma)$ such that for $i \in [n]$, $\sigma_i < \sigma_{i+1}$ iff $u_i = \mathfrak{a}$ (we speak of an "ascent") and $\sigma_i > \sigma_{i+1}$ iff $u_i = \mathfrak{d}$ (we speak of a "descent"), for instance $\mathtt{sg}(1342) = \mathtt{sg}(2341) = \mathfrak{aad}$. We can also define the signature of a word of natural numbers. For example, $\mathtt{sg}(10\ 13\ 2) = \mathfrak{ad}$.

The notion of signature appears in the literature under several different names and forms such as descent word, descent set, ribbon diagram, etc. A *regular class of permutation* is a set of the form $\Lambda(L) \stackrel{\text{def}}{=} \{\sigma \mid \mathtt{sg}(\sigma) \in L\}$, where $L \subseteq \{\mathfrak{a}, \mathfrak{d}\}^*$ is a regular language. For positive integers $k \leq n$, we let $\Lambda_n(L) \stackrel{\text{def}}{=} \Lambda(L) \cap \mathfrak{S}_n$ and $\Lambda_{n,k}(L) \stackrel{\text{def}}{=} \{\sigma \in \Lambda_n(L) \mid \sigma_1 = k\}$.

Cardinalities and exponential generating function for regular classes of permutations. For a signature $u \in \{\mathfrak{a}, \mathfrak{d}\}^*$, we denote by Λ_u the set of permutations with signature u and by α_u its cardinality. Given a language L we denote by L_n the sub-language of L restricted to its n-length words. We denote by $\alpha_n(L)$ the number of n-length permutations with signature in L, that is $\alpha_n(L) = \sum_{w \in L_{n-1}} \alpha_w = |\Lambda_n(L)|$. Similarly we define $\alpha_{n,k}(L) = |\Lambda_{n,k}(L)|$ for $k \leq n$. The exponential generating function of $\Lambda(L)$ is

$$G_L(z) \stackrel{\text{def}}{=} \sum_{\sigma \in \Lambda(L)} \frac{z^{|\sigma|}}{|\sigma|!} = \sum_{u \in L} \alpha_u \frac{z^{|u|+1}}{(|u|+1)!} = \sum_{n \geq 1} \alpha_n(L) \frac{z^n}{n!}.$$

Example 3 For the running example, the theory developed in the paper allows us to find the exponential generating function of $\Lambda(L^{(\text{run})})$:

$$G_{L^{(\text{run})}}(z) = \frac{2\sqrt{2}z(e^{\sqrt{2}z} - 1)}{2 + \sqrt{2}z + (2 - \sqrt{2}z)e^{\sqrt{2}z}} \tag{4}$$

$$= 2f\left(\frac{\sqrt{2}}{2}z\right) \text{ with } f(X) = \frac{1}{1 - X \tanh(X)} - 1. \tag{5}$$

Its Taylor expansion is

$$2\frac{z^2}{2!} + 8\frac{z^4}{4!} + 84\frac{z^6}{6!} + 1632\frac{z^8}{8!} + 51040\frac{z^{10}}{10!} + 2340480\frac{z^{12}}{12!} + \cdots$$

For instance, there are 1632 permutations of length 8 in the regular class considered.

Convergence radii of generating functions. It is often useful to consider a generating function $T(z) = \sum_{n \geq 0} a_n z^n$ as a function of the complex variable (see [FS09]). For a non-negative integer r, we denote by $D(0, r)$ the following set $\{z \in \mathbb{C} \mid |z| < r\}$. The disc of convergence of $T(z) = \sum_{n \geq 0} a_n z^n$ is the largest disc $D(0, r)$ containing the complex numbers z for which $\sum_{n \geq 0} a_n z^n$ converges. The *convergence radius* $\texttt{Rconv}(T)$ is the radius r of such a disc of convergence.

In this paper, we also consider generating functions on two variables $V(x, z) = \sum_{n \in \mathbb{N}} v_n(x) z^n$ where v_n is a polynomial that is non-negative on the interval $[0, 1]$. We define the convergence radius of V as $\inf_{x \in [0,1]} \texttt{Rconv}(z \mapsto V(x, z))$.

For a vector $\mathbf{V}$ of generating functions $V_q(x, z)$ (indexed by states $q \in Q$ of an automaton) we define $\texttt{Rconv}(\mathbf{V}) \stackrel{\text{def}}{=} \min_{q \in Q} \texttt{Rconv}(V_q)$.

Exponential of a matrix. In several places of this article, we will use the matrix exponential. The exponential of a matrix M is the matrix defined by $\exp(M) = \sum_{n \in \mathbb{N}} M^n / n!$. We often use exponentials of matrices of the form $\exp(zM) = \sum_{n \in \mathbb{N}} M^n z^n / n!$ where z is either a formal variable or a complex number and M has real or complex entries. The matrix $\exp(zM)$ is invertible with inverse $\exp(-zM)$.

2.1 Particular regular languages of signatures considered in the literature

In this paper, we introduce for the first time regular languages of signatures. However previous works on the subject can be encoded using our framework.

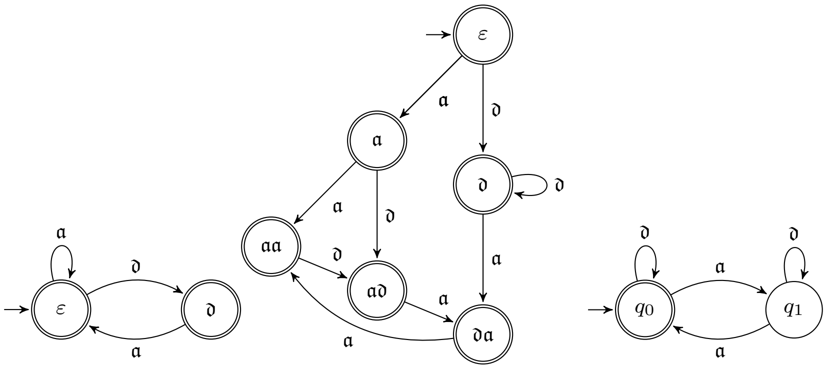

Fig. 2 From left to right automata for $X_{\{\mathfrak{dd}\}}$, $X_{\{\mathfrak{aaa},\mathfrak{add},\mathfrak{dad}\}}$ and L_{even}.

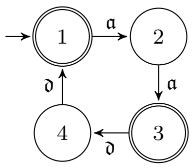

Fig. 3 Automaton for signatures of even length following the periodic pattern $\mathfrak{aadd}$.

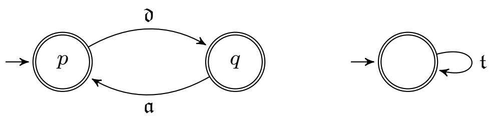

Fig. 4 An automaton for the language $\{\mathfrak{da}\}^*\{\mathfrak{d},\varepsilon\}$ and its $\mathfrak{s}$-$\mathfrak{t}$ encoding of type $\mathfrak{a}$

2.1.1 Consecutive descent pattern avoidance

Works on consecutive descent pattern avoidance (such as [EKP11, EJ12]) consider finite set of words Forb and the permutations whose signature avoid words of this set as factors. In other words, the underlying language of signature X_{Forb} contains exactly the words w such that for all $0 < i < j \leq |w|$ it holds that $w_i \dots w_j \notin \text{Forb}$. Such a language X_{Forb} can be recognised by an automaton with a number of states upper bounded by $\sum_{w \in \text{Forb}} |w|$ by using a prefix tree (aka. trie[1]). We depict two examples of automata in the left and middle of Figure 2. They recognise respectively $X_{\{\mathfrak{dd}\}} = \{\mathfrak{a}, \mathfrak{da}\}^*\{\varepsilon, \mathfrak{d}\}$ and $X_{\{\mathfrak{aaa},\mathfrak{add},\mathfrak{dad}\}}$. We name the state after the longest prefix of a forbidden word that is currently seen.

It is also well known that some regular languages cannot be described using sets of forbidden patterns. Consider for instance the language L_{even} of signatures that contain an even number of ascents and an arbitrary number of descents, this language is regular as recognised by the automaton depicted in the right of Figure 2. Assume for a contradiction that it is of the form X_{Forb} for some set of words

[1] We refer the reader to [Lot05] for definition of such data structure.

Forb. For every $w \in$ Forb, either $w\mathfrak{a}$ or $w\mathfrak{a}\mathfrak{a}$ contains an even number of $\mathfrak{a}$ and does not avoid w, a contradiction. A similar argument holds for the regular language $L^{(\mathrm{run})}$ which cannot be defined with a finite set of forbidden patterns.

2.1.2 Periodic pattern

In [Mar14, Luc14], the authors consider classes of permutations that match so-called periodic patterns. Such classes are regular classes that can be described by using automata with a single cycle in which every state is a final state. One can make some states non-final to discard some cardinalities, for instance the class of permutations of odd length with signature matching the periodic pattern $\mathfrak{a}\mathfrak{a}\mathfrak{d}\mathfrak{d}$ is depicted in the left of Figure 3.

Alternating permutations of positive length have signatures that match the periodic pattern $\mathfrak{d}\mathfrak{a}$. The automaton depicted in the left of Figure 4 recognises the language of such signatures $\{\mathfrak{d}\mathfrak{a}\}^*\{\mathfrak{d}, \varepsilon\}$.

3 The first approach

In Section 3.1, we recall some known results for counting permutations with a single prescribed signature. In Section 3.2, we extend these results to counting permutations of fixed length with signature in a regular language. In Section 3.3, we characterize the generating function as the solution of a linear algebra problem involving the integral of the exponential of a matrix.

3.1 Number of permutations with a prescribed signature

Here we review known results needed in the rest of Section 3. Such a review with historical notes can also be found in the beginning of [Luc14]. We give a somewhat different presentation. For instance, we invoke geometry rather than probability for the continuous case and emphasize the use of Bernstein polynomials and their nice algebraic properties.

3.1.1 The discrete approach

The aim of this section is to describe a set of recursive equations which, given a signature $u \in \{\mathfrak{a}, \mathfrak{d}\}^*$, will allow one to compute the number α_u of permutations with signature u. For this we classify the permutations according to their first element σ_1. Given a signature $u \in \{\mathfrak{a}, \mathfrak{d}\}^*$ and an integer $k \in [n]$ with $n = |u| + 1$, we define $\Lambda_{u,k} \stackrel{\mathrm{def}}{=} \{\sigma \in \mathfrak{S}_n \mid \mathtt{sg}(\sigma) = u \text{ and } \sigma_1 = k\}$ and $\alpha_{u,k} \stackrel{\mathrm{def}}{=} |\Lambda_{u,k}|$.

Proposition 1 *The following equalities hold for every signature $u \in \{\mathfrak{a}, \mathfrak{d}\}^*$ (with $n = |u| + 1$)*

$$\alpha_u = \sum_{k=1}^{n} \alpha_{u,k}; \tag{6}$$

$$\alpha_{\varepsilon,1} = 1; \tag{7}$$

$$\alpha_{\mathfrak{a}u,k} = \sum_{i=k}^{n} \alpha_{u,i}; \tag{8}$$

$$\alpha_{\mathfrak{d}u,k} = \sum_{i=1}^{k-1} \alpha_{u,i}. \tag{9}$$

Proof (7) follows from the definition of signatures, the 0-length word ε is the signature of the unique permutation of $\mathfrak{S}_1 = \{1\}$. (6) comes from the set equation

$$\Lambda_u = \{\sigma \in \mathfrak{S}_n \mid \mathtt{sg}(\sigma) = u\} = \bigcup_{k=1}^{n} \Lambda_{u,k}$$

Note that $\Lambda_{\mathfrak{a}u,k} = \{k\} \times \{\sigma_2 \dots \sigma_{n+1} \mid k < \sigma_2 \text{ and } \mathtt{sg}(\sigma_2 \dots \sigma_{n+1}) = u\}$ and, hence, $\Lambda_{\mathfrak{a}u,k} = \{k\} \times \bigcup_{j=k+1}^{n+1} \{\sigma_2 \dots \sigma_{n+1} \mid \mathtt{sg}(\sigma_2 \dots \sigma_{n+1}) = u \text{ and } \sigma_2 = j\}$.

By subtracting 1 from the values greater or equal to $k+1$ we establish a bijection between the set $\{\sigma_2 \dots \sigma_{n+1} \mid \mathtt{sg}(\sigma_2 \dots \sigma_{n+1}) = u \text{ and } \sigma_2 = j\}$ and $\Lambda_{u,j-1}$.

Thus $\Lambda_{\mathfrak{a}u,k}$ is in bijection with $\bigcup_{j=k+1}^{n+1} \Lambda_{u,j-1} = \bigcup_{i=k}^{n} \Lambda_{u,i}$. Passing to cardinalities we obtain (8). (9) is obtained similarly. □

One can use a system of "local" recursive equations instead of (8) and (9):

$$\alpha_{\mathfrak{a}u,k} = \alpha_{\mathfrak{a}u,k+1} + \alpha_{u,k}; \tag{10}$$

$$\alpha_{\mathfrak{d}u,k} = \alpha_{\mathfrak{d}u,k-1} + \alpha_{u,k-1}; \tag{11}$$

together with boundary conditions

$$\alpha_{\mathfrak{a}u,n+1} = 0; \; \alpha_{\mathfrak{d}u,1} = 0. \tag{12}$$

obtained by setting $k = n + 1$ and $k = 1$ in (8) and (9) respectively.

Remark 1 Equations (10), (11) and (12) are slightly different than the recursive equations that appeared in [DB70, Vie79, Luc14]. Indeed, previous works usually consider classification of the permutations according to their last element σ_n and hence the action of a new ascent or descent is done at the end of the word (on the right). Here we change the order and operate on the left to be consistent with the rest of the article that rely on language equations[2] (2) and (3).

[2] In fact, one could also write language equations with letters added on the right; but, this would introduce inconsistency with timed language and volume equations of our previous work [ABD15, ABDP12] (used in Section 4) that can only be written with operations on the left.

3.1.2 The continuous approach

We say that a collection of polytopes $(S_1, \cdots, S_n)$ is an *almost disjoint partition* of a set A if A is the union of S_i and they have pairwise a null volume intersection. In this case we write $S = \bigsqcup_{i=1}^n S_i$.

The set $\{(\nu_1, \ldots, \nu_n) \in [0,1]^n \mid 0 \leq \nu_{\sigma_1^{-1}} \leq \ldots \leq \nu_{\sigma_n^{-1}} \leq 1\}$ is called the *order simplex*[3] of σ and denoted by $\mathcal{O}(\sigma)$. For instance $\boldsymbol{\nu} = (0.1, 0.3, 0.4, 0.2)$ belongs to $\mathcal{O}(1342)$ since $\nu_1 \leq \nu_4 \leq \nu_2 \leq \nu_3$ and $(1342)^{-1} = 1423$. The set $\mathcal{O}(\sigma)$ for $\sigma \in \mathfrak{S}_n$ forms an almost disjoint partition of $[0,1]^n$. By symmetry all the order simplices of permutations have the same volume which is $1/n!$.

If $\boldsymbol{\nu}$ is uniformly sampled in $[0,1]^n$, then it falls in any $\mathcal{O}(\sigma)$ with probability $1/n!$. To retrieve σ from $\boldsymbol{\nu}$ it suffices to use a sorting algorithm. We denote by $\Pi(\boldsymbol{\nu})$ the permutation σ returned by the sorting algorithm on $\boldsymbol{\nu}$, that is such that $0 \leq \nu_{\sigma_1^{-1}} \leq \ldots \leq \nu_{\sigma_n^{-1}} \leq 1$. Moreover with probability 1, $\boldsymbol{\nu}$ has pairwise distinct coordinates and one can define its *signature*[4] $\mathtt{sg}(\boldsymbol{\nu}) = u_1 \ldots u_{n-1}$ by $u_i = \mathfrak{a}$ if $\nu_i < \nu_{i+1}$ and $u_i = \mathfrak{d}$ if $\nu_i > \nu_{i+1}$. For instance $\mathtt{sg}(0.1, 0.3, 0.4, 0.2) = \mathfrak{aad}$.

The *order polytope* $\mathcal{O}(u)$ [Sta86] of a signature $u \in \{\mathfrak{a}, \mathfrak{d}\}^{n-1}$ is the set of vectors $\boldsymbol{\nu}$ such that for all $i \leq n-1$, if $u_i = \mathfrak{a}$ then $\nu_i \leq \nu_{i+1}$, and $\nu_i \geq \nu_{i+1}$ otherwise. This set is the topological closure of $\{\boldsymbol{\nu} \in [0,1]^n \mid \mathtt{sg}(\boldsymbol{\nu}) = u\}$. It is clear that the collection of order simplices $\mathcal{O}(\sigma)$ with all σ having the same signature u form an almost disjoint partition of the order polytope $\mathcal{O}(u)$: $\mathcal{O}(u) = \bigsqcup_{\sigma \in \mathtt{sg}^{-1}(u)} \mathcal{O}(\sigma)$. For instance $\mathcal{O}(\mathfrak{aad}) = \mathcal{O}(1243) \sqcup \mathcal{O}(1342) \sqcup \mathcal{O}(2341)$. Passing to volume we get:

$$\mathtt{Vol}(\mathcal{O}(u)) = \sum_{\sigma \in \Lambda_u} \mathtt{Vol}(\mathcal{O}(\sigma)) = \frac{\alpha_u}{n!}. \tag{13}$$

One can classify vectors according to their first coordinate as the permutations were in Section 3.1.1. We denote by $\mathcal{O}_u(x) \stackrel{\text{def}}{=} \{(\nu_2, \ldots, \nu_n) \mid (x, \nu_2, \ldots, \nu_n) \in \mathcal{O}(u)\}$ and by $\mathrm{vo}_u(x) \stackrel{\text{def}}{=} \mathtt{Vol}(\mathcal{O}_u(x))$.

The following proposition is the continuous counterpart of Proposition 1.

Proposition 2 *The following equalities hold for every signature $u \in \{\mathfrak{a}, \mathfrak{d}\}^*$ (with $n = |u| + 1$)*

$$\mathtt{Vol}(\mathcal{O}(u)) = \int_0^1 \mathrm{vo}_u(x) dx; \tag{14}$$

$$\mathrm{vo}_\varepsilon(x) = 1; \tag{15}$$

$$\mathrm{vo}_{\mathfrak{a}u}(x) = \int_x^1 \mathrm{vo}_u(y) dy; \tag{16}$$

$$\mathrm{vo}_{\mathfrak{d}u}(x) = \int_0^x \mathrm{vo}_u(y) dy. \tag{17}$$

Proof (16) can be proved using the following identity on sets:

$$\mathcal{O}_{\mathfrak{a}u}(x) = \{(y, \nu_2 \ldots, \nu_n) \mid x \leq y \text{ and } (y, \nu_2, \ldots, \nu_n) \in \mathcal{O}(u)\} = \bigcup_{y \in (x,1)} \{y\} \times \mathcal{O}_u(y).$$

A similar argument holds for descents.

[3] Order simplices, order and chain polytopes of signatures defined here are particular cases of Stanley's order and chain polytopes of posets [Sta86].

[4] Alternatively $\mathtt{sg}(\boldsymbol{\nu}) \stackrel{\text{def}}{=} \mathtt{sg}(\Pi(\boldsymbol{\nu}))$ (defined also when some coordinates are equal).

3.1.3 The link between the two approaches

The link between the coefficients $\alpha_{u,k}$ and the functions vo_u is made in Proposition 3 below using Bernstein polynomials. This proposition is essentially the equation (15) of [Luc14] written with different indices convention and with the order reversed as explained in Remark 1. The *Bernstein polynomials* $b_{k,n}$ for integers $k \leq n$ are defined as follows:

$$b_{k,n}(x) = \binom{n}{k} x^k (1-x)^{n-k}.$$

In particular, $b_{0,0}(x) = 1$ and $b_{0,n}(x) = (1-x)^n$ for $n \in \mathbb{N}$.

Proposition 3 *For every signature w of a given length m,*

$$\mathrm{vo}_w(x) = \frac{1}{m!} \sum_{k=0}^{m} \alpha_{w,k+1} b_{k,m}(x). \tag{18}$$

This proposition can be proved using the following algebraic identities on the integral operators $\int_x^1 dy$ and $\int_0^x dy$ applied to Bernstein polynomials:

$$\int_x^1 b_{i,n}(y)dy = \frac{1}{n+1} \sum_{k=0}^{i} b_{k,n+1}(x); \tag{19}$$

$$\int_0^x b_{i,n}(y)dy = \frac{1}{n+1} \sum_{k=i+1}^{n} b_{k,n+1}(x). \tag{20}$$

Proof (of Proposition 3) We prove the result by induction. The base case is $\mathrm{vo}_\varepsilon(x) = 1 = \alpha_{\varepsilon,1} b_{0,0}(x)$. For the induction step, we suppose that (18) holds for some u of a given length m and prove that it holds for $\mathfrak{a}u$ (the case $\mathfrak{d}u$ is omitted as the proof is similar). We start with (16),

$$\mathrm{vo}_{\mathfrak{a}w}(x) = \int_x^1 \mathrm{vo}_w(y)dy = \int_x^1 \frac{1}{m!} \sum_{i=0}^{m} \alpha_{u,i+1} b_{i,m}(x)dx = \sum_{i=1}^{m+1} \frac{\alpha_{u,i}}{m!} \int_x^1 b_{i-1,m}(x)dx.$$

Applying (19) we get:

$$\mathrm{vo}_{\mathfrak{a}w}(x) = \sum_{i=1}^{m+1} \frac{\alpha_{u,i}}{(m+1)!} \sum_{k=0}^{i-1} b_{k,m+1}(x) = \frac{1}{(m+1)!} \sum_{k=0}^{m} \left(\sum_{i=k+1}^{m+1} \alpha_{u,i} \right) b_{k,m+1}(x).$$

Now it suffices to use (8) to conclude:

$$\mathrm{vo}_{\mathfrak{a}w}(x) = \frac{1}{(m+1)!} \sum_{k=0}^{m} \alpha_{\mathfrak{a}u,k+1} b_{k,m+1}(x).$$

□

3.2 Counting permutations in a regular class

For the rest of Section 3 we consider an arbitrary regular language L recognised by an automaton $\mathcal{A} = (\{\mathfrak{a}, \mathfrak{d}\}, Q, q_0, F, \delta)$.

3.2.1 The discrete approach

For an arbitrary language U and $n \geq 1$ we let $\alpha_{n,k}(U)$ be the number of permutations of $\mathfrak{S}_n$ with signature in U and first element k. The following proposition is obtained from Proposition 1, by using the fact that $\alpha_{n,k}(U) = \sum_{u \in U_{n-1}} \alpha_u$.

Proposition 4 *The coefficients $\alpha_n([\mathcal{A}_q])$ satisfy the following recursive equations:*

$$\alpha_n([\mathcal{A}_q]) = \sum_{k=1}^{n} \alpha_{n,k}([\mathcal{A}_q]); \tag{21}$$

$$\alpha_{1,1}([\mathcal{A}_q]) = 1_{q \in F}; \tag{22}$$

$$\alpha_{n+1,k}([\mathcal{A}_q]) = \sum_{i=k}^{n} \alpha_{n,i}([\mathcal{A}_{q.\mathfrak{a}}]) + \sum_{i=1}^{k-1} \alpha_{n,i}([\mathcal{A}_{q.\mathfrak{d}}]); \tag{23}$$

One could also use the following system of local equations instead of (23):

$$\alpha_{n+1,k}([\mathcal{A}_q]) = \alpha_{n+1,k}(\mathfrak{a}[\mathcal{A}_{q.\mathfrak{a}}]) + \alpha_{n+1,k}(\mathfrak{d}[\mathcal{A}_{q.\mathfrak{d}}]); \tag{24}$$

$$\alpha_{n+1,k}(\mathfrak{a}[\mathcal{A}_q]) = \alpha_{n+1,k+1}(\mathfrak{a}[\mathcal{A}_q]) + \alpha_{n,k}([\mathcal{A}_q]); \tag{25}$$

$$\alpha_{n+1,k}(\mathfrak{d}[\mathcal{A}_q]) = \alpha_{n+1,k-1}(\mathfrak{d}[\mathcal{A}_q]) + \alpha_{n,k-1}([\mathcal{A}_q]); \tag{26}$$

and the boundary conditions:

$$\alpha_{n+1,n+1}(\mathfrak{a}[\mathcal{A}_q]) = 0; \quad \alpha_{n+1,1}(\mathfrak{d}[\mathcal{A}_q]) = 0. \tag{27}$$

Hence we can state a solution to Problem 1.

Corollary 1 *One can compute $\alpha_n(L)$ in arithmetic time complexity $O(|Q|n^2)$ and arithmetic space complexity $O(|Q|n)$ (and $O(|Q|n^2)$ if all the numbers needed for the computation are kept in memory.).*

3.2.2 The continuous approach

For $n \geq 1$, the family $(\mathcal{O}(u))_{u \in L_{n-1}}$ forms an almost disjoint partition of a subset of $[0,1]^n$ called the nth *order set* of L and denoted by $\mathcal{O}_n(L)$:

$$\mathcal{O}_n(L) = \bigsqcup_{u \in L_{n-1}} \mathcal{O}(u) = \bigsqcup_{\sigma \in \mathtt{sg}^{-1}(L_{n-1})} \mathcal{O}(\sigma) = \overline{\{\boldsymbol{\nu} \in [0,1]^n \mid \mathtt{sg}(\boldsymbol{\nu}) \in L_{n-1}\}}. \tag{28}$$

For instance

$$\begin{aligned} \mathcal{O}_3(L^{(\mathrm{run})}) &= \mathcal{O}(\mathfrak{aaa}) \sqcup \mathcal{O}(\mathfrak{aad}) \sqcup \mathcal{O}(\mathfrak{dda}) \sqcup \mathcal{O}(\mathfrak{ddd}) \\ &= \mathcal{O}(1234) \sqcup \\ &\quad \mathcal{O}(1243) \sqcup \mathcal{O}(1342) \sqcup \mathcal{O}(2341) \sqcup \\ &\quad \mathcal{O}(4312) \sqcup \mathcal{O}(4213) \sqcup \mathcal{O}(3214) \sqcup \\ &\quad \mathcal{O}(4321). \end{aligned}$$

Passing to the volume in (28) we get:

$$\mathtt{Vol}(\mathcal{O}_n(L)) = \sum_{u \in L_{n-1}} \mathtt{Vol}(\mathcal{O}(u)) = \sum_{\sigma \in \Lambda_n(L)} \mathtt{Vol}(\mathcal{O}(\sigma)) = \frac{\alpha_n(L)}{n!}. \tag{29}$$

Measuring the order sets $\mathcal{O}_n([\mathcal{A}_q])$ for $q \in Q$ is done as in the discrete case, by parametrising these sets according to their first component. For all $q \in Q$, $n \in \mathbb{N}$ and $x \in [0,1]$ we let

$$\mathcal{O}_{q,n}(x) \stackrel{\text{def}}{=} \{(\nu_2, \dots, \nu_n) \mid (x, \nu_2, \dots, \nu_n) \in \mathcal{O}_n([\mathcal{A}_q])\} \text{ and } \mathrm{vo}_{q,n}(x) \stackrel{\text{def}}{=} \mathtt{Vol}[\mathcal{O}_{q,n+1}(x)].$$

Proposition 5 *The function* $\mathrm{vo}_{q,n}$ *for* $q \in Q$ *and* $n \in \mathbb{N}$ *are polynomials of degree* $\leq n$ *that satisfy the following recurrence:* $\mathrm{vo}_{q,0}(x) = 1_{q \in F}$ *and*

$$\mathrm{vo}_{q,n+1}(x) = \int_x^1 \mathrm{vo}_{q.\mathfrak{a},n}(y)dy + \int_0^x \mathrm{vo}_{q.\mathfrak{d},n}(y)dy. \tag{30}$$

In addition, for all $n \geq 1$*:*

$$\int_0^1 \mathrm{vo}_{q,n-1}(x)dx = \frac{\alpha_n([\mathcal{A}_q])}{n!}. \tag{31}$$

Proof The sets $\mathcal{O}_{q,n}(x)$ can be recursively defined as follows: $\mathcal{O}_{q,0}(x) = [0,1]$ if $q \in F$ and $\mathcal{O}_{q,0}(x) = \emptyset$ otherwise;

$$\mathcal{O}_{q,n+1}(x) = \bigcup_{y \in (x,1)} \{y\} \times \mathcal{O}_{q.\mathfrak{a},n}(y) \cup \bigcup_{y \in (0,x)} \{y\} \times \mathcal{O}_{q.\mathfrak{d},n}(y)$$

Passing to the volume we get (30). We remark that $\mathcal{O}_{q,n}(x) = \bigcup_{u \in [\mathcal{A}_q]_{n-1}} \mathcal{O}_u(x)$ and hence $\mathrm{vo}_{q,n-1}(x) = \sum_{u \in [\mathcal{A}_q]_{n-1}} \mathrm{vo}_u(x)$. We can conclude using (13) that

$$\int_0^1 \mathrm{vo}_{q,n-1}(x)dx = \sum_{u \in [\mathcal{A}_q]_{n-1}} \int_0^1 \mathrm{vo}_u(x)dx = \sum_{u \in [\mathcal{A}_q]_{n-1}} \frac{\alpha_u}{n!} = \frac{\alpha_n([\mathcal{A}_q])}{n!}.$$

□

One can define and compute the $\mathrm{vo}_{q,n}$ in the Bernstein basis using the following proposition that links the volume function with the numbers $\alpha_{n,k}$, or directly compute these functions in the standard basis $1, x, x^2, \dots$

Proposition 6 *The coefficients of the polynomials* $\mathrm{vo}_{q,n}$ *in the Bernstein basis can be computed in arithmetic time and space complexity* $O(|Q|n^2)$ *using the following characterisation:*

$$\mathrm{vo}_{q,n}(x) = \frac{1}{n!} \sum_{k=0}^{n} \alpha_{n+1,k+1}([\mathcal{A}_q]) b_{k,n}(x) \tag{32}$$

Alternatively, the polynomials $\mathrm{vo}_{q,m}$ *for* $q \in Q$ *and* $m \leq n$ *can be computed in the standard basis with an arithmetic time and space complexity* $O(|Q|n^2)$ *using recursive equation* (30)*.*

We remark that the space complexity can be reduced to $O(|Q|n)$ if one is interested only in $(\mathrm{vo}_{q,n})_{q \in Q}$ (as the computation of $(\mathrm{vo}_{q,m})_{q \in Q}$ needs only to have $(\mathrm{vo}_{q,m-1})_{q \in Q}$ in memory).

3.3 Generating functions

3.3.1 Characterisation of the generating functions

We saw in (31) how to link the counts of permutations with the volume functions. We now show how generating functions for these two kinds of objects can be related. In the rest of this section, we just write $G_q(z)$ for the generating function

$$G_{[\mathcal{A}_q]}(z) = \sum_{n\geq 1} \frac{\alpha_n([\mathcal{A}_q])}{n!} z^n$$

and we write

$$\mathrm{VO}_q(x,z) = \sum_{n\geq 1} \mathrm{vo}_{q,n-1}(x) z^n.$$

By taking $\sum_{n\geq 1} z^n$ in the equations of Proposition 5 we obtain the following proposition.

Proposition 7 *For $z < \mathtt{Rconv}(\mathbf{VO})$, the vector of generating functions $\mathbf{VO}$ is the unique solution of the following system of integral equations:*

$$\mathrm{VO}_q(x,z) = z\int_x^1 \mathrm{VO}_{q.\mathfrak{a}}(y,z)dy + z\int_0^x \mathrm{VO}_{q.\mathfrak{d}}(y,z)dy + z1_{q\in F}. \tag{33}$$

In addition for $q \in Q$, it holds that: $G_q(z) = \int_0^1 \mathrm{VO}_q(x,z)dx$.

In matrix notation, the system of equations (33) can be written as

$$\mathbf{VO}(x,z) = zM_{\mathfrak{a}}\int_x^1 \mathbf{VO}(y,z)dy + zM_{\mathfrak{d}}\int_0^x \mathbf{VO}(y,z)dy + z\mathbf{F}. \tag{34}$$

A similar equation was obtained in [EJ12], to find the growth rate of classes of permutations that avoid a set of consecutive descent patterns (as described in Section 2.1.1).

Equation (34) is equivalent to the following differential equation (35) and boundary condition (36):

$$\frac{\partial}{\partial x}\mathbf{VO}(x,z) = z\left(M_{\mathfrak{d}} - M_{\mathfrak{a}}\right)\mathbf{VO}(x,z); \text{ and} \tag{35}$$

$$\mathbf{VO}(0,z) = zM_{\mathfrak{a}}\int_0^1 \mathbf{VO}(y,z)dy + z\mathbf{F} = zM_{\mathfrak{a}}\mathbf{G}(z) + z\mathbf{F}. \tag{36}$$

The solution of these equations involves the following matrix:

$$I(z) \overset{\text{def}}{=} \int_0^1 \exp\left[xz\left(M_{\mathfrak{d}} - M_{\mathfrak{a}}\right)\right]dx. \tag{37}$$

Theorem 1 *There exists $r \leq \mathtt{Rconv}(\mathbf{G})$ such that for all z within the disc $D(0,r)$ the matrix $I_{|Q|} - zI(z)M_{\mathfrak{a}}$ is invertible and $\mathbf{G}(z)$ satisfies:*

$$\mathbf{G}(z) = \left(I_{|Q|} - zI(z)M_{\mathfrak{a}}\right)^{-1} zI(z)\mathbf{F}. \tag{38}$$

If $M_{\mathfrak{d}} - M_{\mathfrak{a}}$ is invertible (38) is equivalent to:

$$\mathbf{G}(z) = \left(M_{\mathfrak{d}} - \exp\left[z(M_{\mathfrak{d}} - M_{\mathfrak{a}})\right]M_{\mathfrak{a}}\right)^{-1}\left(\exp\left[z(M_{\mathfrak{d}} - M_{\mathfrak{a}})\right] - I_{|Q|}\right)\mathbf{F}. \tag{39}$$

Proof Solutions of (35) are of the form:

$$\mathbf{VO}(x,z) = \exp\left[xz\left(M_{\mathfrak{d}} - M_{\mathfrak{a}}\right)\right]\mathbf{VO}(0,z). \tag{40}$$

Now, we integrate x over $[0,1]$ to make $I(z)$ appear and use the boundary condition (36) to obtain:

$$\mathbf{G}(z) = I(z)\left(zM_{\mathfrak{a}}\mathbf{G}(z) + z\mathbf{F}\right).$$

which yields

$$\left(I_{|Q|} - zI(z)M_{\mathfrak{a}}\right)\mathbf{G}(z) = zI(z)\mathbf{F}.$$

When $z = 0$, the continuous function $z \mapsto \det\left(I_{|Q|} - zI(z)M_{\mathfrak{a}}\right)$ is equal to $\det(I_{|Q|}) = 1$ and thus non null on a neighbourhood of 0. Thus, the matrix $\left(I_{|Q|} - zI(z)M_{\mathfrak{a}}\right)$ is invertible around 0 and the first result (38) follows.

Now, we prove (39). Note that $z\left(M_{\mathfrak{d}} - M_{\mathfrak{a}}\right)I(z) = \exp\left[z(M_{\mathfrak{d}} - M_{\mathfrak{a}})\right] - I_{|Q|}$. Indeed,

$$z\left(M_{\mathfrak{d}} - M_{\mathfrak{a}}\right)I(z) = \int_0^1 \sum_{n\geq 0}\left[z\left(M_{\mathfrak{d}} - M_{\mathfrak{a}}\right)\right]^{n+1}\frac{1}{n!}x^n dx = \sum_{n\geq 0}\left(M_{\mathfrak{d}} - M_{\mathfrak{a}}\right)^{n+1}\frac{z^{n+1}}{(n+1)!}.$$

Then

$$\left(M_{\mathfrak{d}} - M_{\mathfrak{a}}\right)\left(I_{|Q|} - zI(z)M_{\mathfrak{a}}\right) = M_{\mathfrak{d}} - \exp\left[z(M_{\mathfrak{d}} - M_{\mathfrak{a}})\right]M_{\mathfrak{a}}.$$

Hence when $M_{\mathfrak{d}} - M_{\mathfrak{a}}$ is invertible we have:

$$\begin{aligned}\mathbf{G}(z) &= \left(I_{|Q|} - zI(z)M_{\mathfrak{a}}\right)^{-1}\left(M_{\mathfrak{d}} - M_{\mathfrak{a}}\right)^{-1}(M_{\mathfrak{d}} - M_{\mathfrak{a}})zI(z)\mathbf{F}\\ &= \left(M_{\mathfrak{d}} - \exp\left[z(M_{\mathfrak{d}} - M_{\mathfrak{a}})\right]M_{\mathfrak{a}}\right)^{-1}\left(\exp\left[z(M_{\mathfrak{d}} - M_{\mathfrak{a}})\right] - I_{|Q|}\right)\mathbf{F}.\end{aligned}$$

□

As a corollary of the proof of Theorem 1 one can obtain $\mathbf{VO}(x,z)$ by plugging (36) in (40).

Proposition 8 *For every $x \in [0,1]$ it holds that* $\texttt{Rconv}(\mathbf{G}) \leq \texttt{Rconv}(z \mapsto \mathbf{VO}(x,z))$*; and that for every $z < \texttt{Rconv}(\mathbf{G})$,*

$$\mathbf{VO}(x,z) = z\exp\left[xz\left(M_{\mathfrak{d}} - M_{\mathfrak{a}}\right)\right]\left(M_{\mathfrak{a}}\mathbf{G}(z) + \mathbf{F}\right). \tag{41}$$

3.3.2 About generating functions for periodic patterns

It is worth mentioning that a class of examples where (39) can be used to derive explicit formulas of $G_L(z)$ are the periodic patterns described in Section 2.1.2. In that case it is not hard to compute the matrix $\exp\left[z(M_{\mathfrak{d}} - M_{\mathfrak{a}})\right]$ in terms of generalised hyperbolic or generalised trigonometric functions (see Appendix B of [Luc14] for definitions). So, the computation time of the generating function is essentially due to the inversion of the matrix $(M_{\mathfrak{d}} - \exp\left[z(M_{\mathfrak{d}} - M_{\mathfrak{a}})\right]M_{\mathfrak{a}})$, that can be done for instance in a cubic time using basic Gauss–Jordan elimination. We shall not go into the details here as there is already a method provided in [Luc14] with a complexity better than ours (where the main part of the computation is to evaluate an $m \times m$ determinant where m is the minimum between the number of ascents and the number of descents in the pattern considered). Another method was also given recently [Mar14], but it is not as simple since it is based on evaluation of an $nm \times nm$ determinant (where n is the length of the pattern and m is as above).

In [Bas14], we considered as a running example the language recognised by the automaton depicted in Figure 3. However, this example does not reflect all the power of regular classes of permutations as it can be described with a periodic pattern[5].

4 The second approach

We describe here a second approach to study the combinatorics of regular classes of permutations. In the first approach ascents and descents play a symmetric role. This redundancy of information is handled by Stanley's chain polytopes described in Section 4.1 below: they do not distinguish chains of ascents and chains of descents. In the same section, we show that chain polytopes turn out to be polytopes associated to timed languages, and hence, we establish a connection between the combinatorics of permutations and the theory of timed automata. The timed automata associated to regular languages of signatures still suffer from a redundant role played by ascents and descents. In Section 4.2, this redundancy is treated by introducing an encoding of signatures in terms of "straights" and "turns" and their corresponding timed semantics. This allows us to characterise generating functions of permutations in Section 4.4 after having described recursive equations on volume functions in Section 4.3.

4.1 Timed languages and chain polytopes

4.1.1 Chain polytopes of signatures

The *chain polytope* [Sta86] of a signature u is the set $\mathcal{C}(u)$ of vectors $\mathbf{t} \in [0,1]^n$ such that for all $i < j \leq n$ and $l \in \{\mathfrak{a}, \mathfrak{d}\}$, $w_i \cdots w_{j-1} = l^{j-i} \Rightarrow t_i + \ldots + t_j \leq 1$.

Example 4 A vector $(t_1, t_2, t_3, t_4, t_5) \in [0,1]^5$ belongs to $\mathcal{C}(\mathfrak{daad})$ iff $t_1 + t_2 \leq 1, t_2 + t_3 + t_4 \leq 1, t_4 + t_5 \leq 1$ iff $1 - t_1 \geq t_2 \leq t_2 + t_3 \leq 1 - t_4 \geq t_5$ iff $(1 - t_1, t_2, t_2 + t_3, 1 - t_4, t_5) \in \mathcal{O}(\mathfrak{daad})$.

More generally, for $w = ul$ with $u \in \{\mathfrak{a}, \mathfrak{d}\}^*$, $l \in \{\mathfrak{a}, \mathfrak{d}\}$ and $n = |w|$, there is a volume preserving transformation $(t_1, \cdots, t_n) \mapsto (\nu_1, \cdots, \nu_n)$ from the chain polytope $\mathcal{C}(u)$ to the order polytope $\mathcal{O}(u)$ defined as follows. Let $j \in [n]$ and i be the index such that $w_i \cdots w_{j-1}$ is a maximal ascending or descending block, that is, i is minimal such that $w_i \cdots w_{j-1} = l^{j-i}$ with $l \in \{\mathfrak{a}, \mathfrak{d}\}$. If $w_j = \mathfrak{d}$ we define $\nu_j = 1 - \sum_{k=i}^{j} t_k$ and $\nu_j = \sum_{k=i}^{j} t_k$ otherwise.

Proposition 9 (A simple case of Theorem 2.1 of [HL12]) *The mapping ϕ_{ul} : $(t_1, \cdots, t_n) \mapsto (\nu_1, \cdots, \nu_n)$ is a volume preserving transformation from $\mathcal{C}(u)$ to $\mathcal{O}(u)$.*

[5] Indeed, this example can be obtained from an example of [Luc14] by taking only the even-length permutations. We discovered this by looking at the OEIS sequence A005981.

It can be computed in linear time using the recursive definition: $\left| \begin{array}{ll} \nu_1 = t_1 & \text{if } w_1 = \mathfrak{a} \\ \nu_1 = 1 - t_1 & \text{if } w_1 = \mathfrak{d} \end{array} \right.$

and for $i \geq 2$: $\left| \begin{array}{lll} \nu_i = \nu_{i-1} + t_i & \text{if} & w_{i-1}w_i = \mathfrak{aa}; \\ \nu_i = t_i & \text{if} & w_{i-1}w_i = \mathfrak{da}; \\ \nu_i = 1 - t_i & \text{if} & w_{i-1}w_i = \mathfrak{ad}; \\ \nu_i = \nu_{i-1} - t_i & \text{if} & w_{i-1}w_i = \mathfrak{dd}. \end{array} \right.$

As a corollary of (29) and Propostion 9, Problem 2 can be reformulated in geometric terms as follows.

Corollary 2 *For every* $L \in \{\mathfrak{a}, \mathfrak{d}\}^*$ *the following equalities hold:*

$$G_L(z) = \sum_{n \geq 1} \mathtt{Vol}(\mathcal{O}_n(L)) z^n = \sum_{u \in L} \mathtt{Vol}(\mathcal{O}(u)) z^{|u|-1} = \sum_{u \in L} \mathtt{Vol}(\mathcal{C}(u)) z^{|u|-1}.$$

The following three sections (Section 4.1.2, 4.1.3 and 4.1.4) are inspired by timed automata theory and designed for non-experts. We adopt a non-standard[6] and self-contained approach based on the notion of clock languages introduced by [BP02] which was used in our previous work [ABDP12].

4.1.2 Timed languages, their volumes and generating functions

An alphabet of *timed events* is the product $\mathbb{R}^+ \times \Sigma$ where Σ is a finite alphabet. The meaning of a timed event (t_i, w_i) is that t_i is the *time delay* before the *event* w_i. A *timed word* is just a word of timed events and a *timed language* a set of timed words. Adopting a geometric point of view, a timed word is a vector of delays $\mathbf{t} = (t_1, \ldots, t_n) \in \mathbb{R}^n$ together with a word of events $w = w_1 \cdots w_n \in \Sigma^n$. That is why we sometimes write such a timed word $(\mathbf{t}, w)$ instead of $(t_1, w_1) \cdots (t_n, w_n)$. With this convention, given a timed language $\mathbb{L}' \subseteq (\mathbb{R}^+ \times \Sigma)^*$, its restriction to n-length words $\mathbb{L}'_n$ can be seen as a union of sets $\biguplus_{w \in \Sigma^n} \mathbb{L}'_w \times \{w\}$ where $\mathbb{L}'_w = \{\mathbf{t} \in \mathbb{R}^n \mid (\mathbf{t}, w) \in \mathbb{L}'\}$ is the set of delay vectors that together with w form a timed word of $\mathbb{L}'$. In the sequel we will only consider languages $\mathbb{L}'$ for which every $\mathbb{L}'_w$ is Lebesgue measurable. To such $\mathbb{L}'_n$ one can associate a sequence of volumes and a *volume generating function* as follows:

$$\mathtt{Vol}(\mathbb{L}'_n) = \sum_{w \in \Sigma^n} \mathtt{Vol}(\mathbb{L}'_w); \quad T_{\mathbb{L}'}(z) = \sum_{w \in \Sigma^*} \mathtt{Vol}(\mathbb{L}'_w) z^{|w|} = \sum_{n \in \mathbb{N}} \mathtt{Vol}(\mathbb{L}'_n) z^n$$

4.1.3 The clock semantics of a signature

A *clock* is a non-negative real variable. Here we only consider two clocks bounded by 1 and denoted by $x^{\mathfrak{a}}$ and $x^{\mathfrak{d}}$. A *clock word* is a tuple with three components: a starting clock vector $(x_0^{\mathfrak{a}}, x_0^{\mathfrak{d}}) \in [0,1]^2$, a timed word $(t_1, a_1) \cdots (t_n, a_n) \in ([0,1] \times \{\mathfrak{a}, \mathfrak{d}\})^*$ and an ending clock vector $(x_n^{\mathfrak{a}}, x_n^{\mathfrak{d}}) \in [0,1]^2$. It is denoted by $(x_0^{\mathfrak{a}}, x_0^{\mathfrak{d}}) \xrightarrow{(t_1,a_1)\cdots(t_n,a_n)} (x_n^{\mathfrak{a}}, x_n^{\mathfrak{d}})$. Two clock words $\mathbf{x}_0 \xrightarrow{\mathbf{w}} \mathbf{x}_1$ and $\mathbf{x}_2 \xrightarrow{\mathbf{w}'} \mathbf{x}_3$ are said to be compatible if $\mathbf{x}_2 = \mathbf{x}_1$, in this case their product is $(\mathbf{x}_0 \xrightarrow{\mathbf{w}} \mathbf{x}_1) \cdot (\mathbf{x}_2 \xrightarrow{\mathbf{w}'}$

[6] We refer the reader to [AD94] for a standard approach of timed automata theory.

$\mathbf{x}_3) = \mathbf{x}_0 \xrightarrow{\mathbf{ww'}} \mathbf{x}_3$. A *clock language* is a set of clock words. The product of two clock languages $\mathcal{L}$ and $\mathcal{L}'$ is

$$\mathcal{L}\cdot\mathcal{L}' \stackrel{\text{def}}{=} \{c\cdot c' \mid c\in\mathcal{L},\ c'\in\mathcal{L}',\ c \text{ and } c' \text{ compatible}\}. \tag{42}$$

The clock language[7] $\mathcal{L}(\mathfrak{a})$ (resp. $\mathcal{L}(\mathfrak{d})$) of an ascent (resp. a descent) is the set of clock words of the form $(x^{\mathfrak{a}}, x^{\mathfrak{d}}) \xrightarrow{(t,\mathfrak{a})} (x^{\mathfrak{a}}+t, 0)$ (resp. $(x^{\mathfrak{a}}, x^{\mathfrak{d}}) \xrightarrow{(t,\mathfrak{d})} (0, x^{\mathfrak{d}}+t)$) and such that $x^{\mathfrak{a}}+t\in[0,1]$ and $x^{\mathfrak{d}}+t\in[0,1]$ (and by definition of clocks and delays $x^{\mathfrak{a}}\geq 0,\ x^{\mathfrak{d}}\geq 0,\ t\geq 0$). These definitions extend inductively to all signatures: $\mathcal{L}(u_1\cdots u_n) = \mathcal{L}(u_1)\cdots\mathcal{L}(u_n)$ (with product (42)).

Example 5 $(0,0) \xrightarrow{(0.7,\mathfrak{d})(0.2,\mathfrak{a})(0.2,\mathfrak{a})(0.5,\mathfrak{d})} (0,0.5) \in \mathcal{L}(\mathfrak{d}\mathfrak{a}\mathfrak{a}\mathfrak{d})$ since
$(0,0) \xrightarrow{(0.7,\mathfrak{d})} (0,0.7) \in \mathcal{L}(\mathfrak{d})$; $(0,0.7) \xrightarrow{(0.2,\mathfrak{a})} (0.2,0) \in \mathcal{L}(\mathfrak{a})$;
$(0.2,0) \xrightarrow{(0.2,\mathfrak{a})} (0.4,0) \in \mathcal{L}(\mathfrak{a})$; $(0.4,0) \xrightarrow{(0.5,\mathfrak{a})} (0,0.5) \in \mathcal{L}(\mathfrak{d})$.

4.1.4 The timed semantics of a language of signatures

The *timed polytope* associated to a signature $w\in\{\mathfrak{a},\mathfrak{d}\}^*$ is

$$P_w \stackrel{\text{def}}{=} \{\mathbf{t} \mid [(0,0) \xrightarrow{(\mathbf{t},w)} \mathbf{y}] \in \mathcal{L}(w) \text{ for some } \mathbf{y}\in[0,1]^2\}.$$

For instance $(0.7,0.2,0.2,0.5,0.1)\in P_{\mathfrak{d}\mathfrak{a}\mathfrak{a}\mathfrak{d}\mathfrak{a}}$. The timed semantics of a language of signatures L' is

$$\mathbb{L}' = \{(\mathbf{t},w) \mid \mathbf{t}\in P_w \text{ and } w\in L'\} = \cup_{w\in L'} P_w\times\{w\}.$$

This language restricted to words of length n is $\mathbb{L}'_n = \cup_{w\in L'_n} P_w\times\{w\}$, its volume is $\mathtt{Vol}(\mathbb{L}'_n) = \sum_{w\in L'} \mathtt{Vol}(P_w)$.

4.1.5 The link with order and chain polytopes of signatures

We first state the link between timed polytopes and chain polytopes.

Proposition 10 *Given a word $u\in\{\mathfrak{a},\mathfrak{d}\}^*$ and $l\in\{\mathfrak{a},\mathfrak{d}\}$, the timed polytope of ul is the chain polytope of u: $P_{ul} = \mathcal{C}(u)$.*

Proof Let $w = ul$, that is, for all $i\in[n-1]$, $w_i = u_i$ and $w_n = l$. We first prove the inclusion $P_{ul}\subseteq\mathcal{C}(u)$. Let $(t_1,\ldots,t_n)\in P_w$. There exist values of clocks x_k^a for $a\in\{\mathfrak{a},\mathfrak{d}\}$ and $k\in\{0,\ldots,n\}$ such that $x_0^{\mathfrak{a}} = x_0^{\mathfrak{d}} = 0$ and $(x_{k-1}^{\mathfrak{a}}, x_{k-1}^{\mathfrak{d}}) \xrightarrow{(t_k,w_k)} (x_k^{\mathfrak{a}}, x_k^{\mathfrak{d}}) \in \mathcal{L}(w_k)$. Let $i<j\leq n$ and $a\in\{\mathfrak{a},\mathfrak{d}\}$ such that $w_i\cdots w_{j-1} = a^{j-i}$, then for $k\in\{i,\ldots,j-1\}$, $x_k^a = x_{k-1}^a + t_k$ by definition of $\mathcal{L}(a)$. Then $x_{j-1}^a = x_{i-1}^a + t_i + \ldots + t_{j-1}$. Moreover $x_{j-1}^a + t_j \leq 1$ by definition of $\mathcal{L}(w_j)$ (whatever $w_j\in\{\mathfrak{a},\mathfrak{d}\}$ is) and thus $t_i+\ldots+t_{j-1}+t_j = x_{j-1}^a - x_{i-1}^a + t_j \leq x_{j-1}^a + t_j \leq 1$ which is the wanted inequality.

We now prove the inclusion $\mathcal{C}(u)\subseteq P_{ul}$. Let $(t_1,\ldots,t_n)\in\mathcal{C}(u)$. We show inductively that for every $a\in\{\mathfrak{a},\mathfrak{d}\}$, the condition $x_{j-1}^a + t_j \leq 1$ is satisfied and

[7] A reader acquainted with timed automata would have noticed that the clock language $\mathcal{L}(\mathfrak{a})$ (resp. $\mathcal{L}(\mathfrak{d})$) corresponds to a transition of a timed automaton where the guards $x^{\mathfrak{a}}\leq 1$ and $x^{\mathfrak{d}}\leq 1$ are satisfied and where $x^{\mathfrak{d}}$ (resp. $x^{\mathfrak{a}}$) is reset.

thus that x^a_j can be defined ($x^a_j = x^a_{j-1} + t_j$ if $w_j = a$ and $x^a_j = 0$ otherwise). For this we suppose that the clock values lower than some index j are well defined and show that $x^a_{j-1} + t_j \leq 1$ for $a \in \{\mathfrak{a}, \mathfrak{d}\}$. If $a \neq u_{j-1}$ then $x^a_{j-1} = 0$ and hence $x^a_{j-1} + t_j \leq 1$. Otherwise $a = u_{j-1}$ and $x^a_{j-1} = t_i + \ldots + t_{j-1}$ where i is the index of the last reset of x^a before $j-1$, that is, the minimal index such that $u_{i+1} \ldots u_{j-1} = a^{j-i-1}$. By definition of the chain polytope this latter chain of ascents (or descents) yields the wanted inequality: $x^a_{j-1} + t_j = t_i + \ldots + t_j \leq 1$. □

Hence, Proposition 9 links the timed polytope $P_{ul} = \mathcal{C}(u)$ of a signature of length $n+1$ and the order polytope $\mathcal{O}(u)$ of a signature of length n. We correct the mismatch of length using prolongation of languages. A language L' is called a *prolongation* of a language L whenever the truncation of the last letter $w_1 \ldots w_n \mapsto w_1 \ldots w_{n-1}$ is a bijection between L' and L.

Every language has prolongations. In particular, $L' = Ll$ for $l \in \{\mathfrak{a}, \mathfrak{d}\}$ is a prolongation of a language L that is regular whenever L is regular.

Example 6 A prolongation of $L^{(\mathrm{run})}$ is $L^{(\mathrm{run})'} = (\{\mathfrak{aa}, \mathfrak{dd}\})^* \{\mathfrak{aa}, \mathfrak{dd}\}$. An automaton recognising $L^{(\mathrm{run})'} \cup \{\varepsilon\}$ is depicted in the middle of Figure 1.

Proposition 9 can be extended to language of signatures as follows.

Corollary 3 *Let $L \subseteq \{\mathfrak{a}, \mathfrak{d}\}^*$ and $\mathbb{L}'$ be the timed semantics of a prolongation of L then for all $n \in \mathbb{N}$, the following function is a volume preserving transformation between $\mathbb{L}'_n$ and $\mathcal{O}_n(L)$. Moreover it is computable in linear time.*

$$\begin{aligned} \phi : \quad \mathbb{L}'_n \; &\to \mathcal{O}_n(L) \\ (\mathbf{t}, w) &\mapsto \phi_w(\mathbf{t}) \end{aligned} \tag{43}$$

As a consequence, the Problems 2 and 3 can be solved if we know how to compute the VGF of a timed language $\mathbb{L}'$ and how to generate timed vectors uniformly in $\mathbb{L}'_n$. A characterization of the VGF of a timed language as a solution of a system of differential equations is given in [ABDP12]. Nevertheless the equations of this article are quite difficult to handle and do not give a closed form formula for the VGF. To get simpler equations than the ones obtained by using the methods of [ABDP12] we work with a novel class of timed languages involving two kinds of transitions labelled by $\mathfrak{s}$ and $\mathfrak{t}$.

4.2 The $\mathfrak{s}$-$\mathfrak{t}$ (timed) language encodings

For order polytopes, chains of ascents and chains of descents give inequalities in opposite directions ($\nu_i \leq \ldots \leq \nu_j$ and $\nu_i \geq \ldots \geq \nu_j$). By contrast, when dealing with chain polytopes, chains of ascents and chains of descents give exactly the same inequalities ($t_i + \ldots + t_j$). To retrieve the successive letters of a signature while looking at its chain polytope, one has to know the first letter and keep track of the successive changes of chains. In other words, a signature goes "straight" during chains of ascent and chains of descent and when a peak ($\mathfrak{ad}$) or a valley ($\mathfrak{da}$) happens the signature "turns".

This encoding in terms of straights and turns formally defined below, is also well suited to consider only one clock x instead of the two clocks $x^{\mathfrak{a}}$ and $x^{\mathfrak{d}}$.

4.2.1 The $\mathfrak{s}$-$\mathfrak{t}$-encodings

The $\mathfrak{s}$-$\mathfrak{t}$-encoding of type $l \in \{\mathfrak{a}, \mathfrak{d}\}$ of a word $w \in \{\mathfrak{a}, \mathfrak{d}\}^*$ is a word $w' \in \{\mathfrak{s}, \mathfrak{t}\}^*$ denoted by $\mathtt{st}_l(w)$ and defined recursively as follows: for every $i \in [n]$, $w'_i = \mathfrak{s}$ if $w_i = w_{i-1}$ and $w'_i = \mathfrak{t}$ otherwise, with the convention that $w_0 = l$. The inverse mapping of $\mathtt{st}_l$ exists and can also be defined recursively. Indeed $w = \mathtt{st}_l^{-1}(w')$ iff for every $i \in [n]$, $w_i = w_{i-1}$ if $w'_i = \mathfrak{s}$ and $w_i \neq w_{i-1}$ otherwise, with convention that $w_0 = l$. Notion of $\mathfrak{s}$-$\mathfrak{t}$-encodings can be extended naturally to languages.

Example 7 Continuing the running example, (Example 1,2,6), we give the $\mathfrak{s}$-$\mathfrak{t}$-encoding of $L^{(\mathrm{run})'}$: For the running example: $\mathtt{st}_{\mathfrak{d}}\left(L^{(\mathrm{run})'}\right) = (\{\mathfrak{s}, \mathfrak{t}\}\{\mathfrak{s}\})^*$ An automaton recognising this language is depicted in the right of Figure 1.

4.2.2 Timed semantics of the $\mathfrak{s}$-$\mathfrak{t}$-encodings

In the following, we define clock and timed languages similarly to what we have done in Sections 4.1.4 and 4.1.3. Here, we need only one clock x that remains bounded by 1. We define the clock language associated to $\mathfrak{s}$ by $\mathcal{L}(\mathfrak{s}) = \{x \xrightarrow{(t,\mathfrak{s})} x + t \mid x \in [0,1], t \in [0, 1-x]\}$ and the clock language associated to $\mathfrak{t}$ by $\mathcal{L}(\mathfrak{t}) = \{x \xrightarrow{(t,\mathfrak{t})} t \mid x \in [0,1], t \in [0, 1-x]\}$. Let $L'' \subseteq \{\mathfrak{s}, \mathfrak{t}\}^*$, we denote by $L''(x)$ the timed language starting from x: $L''(x) = \{(\mathbf{t}, w) \mid \exists y \in [0,1], x \xrightarrow{(\mathbf{t},w)} y \in \mathcal{L}(w), w \in L''\}$. The *timed semantics* of $L'' \subseteq \{\mathfrak{s}, \mathfrak{t}\}^*$ is $L''(0)$.

The $\mathfrak{s}$-$\mathfrak{t}$-encodings yield natural volume preserving transformations between timed languages:

Proposition 11 *Let $L' \subseteq \{\mathfrak{a}, \mathfrak{d}\}^*$, $l \in \{\mathfrak{a}, \mathfrak{d}\}$, $\mathbb{L}'$ be the timed semantics of L' and $\mathbb{L}''$ be the timed semantics of $st_l(L')$ then the function $(\mathbf{t}, w) \mapsto (\mathbf{t}, st_l^{-1}(w))$ is a volume preserving transformation from $\mathbb{L}''_n$ to $\mathbb{L}'_n$.*

Using notation and results of Corollary 3 and Proposition 11 we get a volume preserving transformation from $\mathbb{L}''_n$ to $\mathcal{O}_n(L)$.

Theorem 2 *The function $(\mathbf{t}, w) \mapsto \phi_{st_l^{-1}(w)}(\mathbf{t})$ is a volume preserving transformation from $\mathbb{L}''_n$ to $\mathcal{O}_n(L)$ computable in linear time. In particular*

$$\mathtt{Vol}(\mathbb{L}''_n) = \frac{\alpha_n(L)}{n!} \text{ for } n \geq 1 \text{ and } T_{\mathbb{L}''}(z) = G_L(z).$$

Thus to solve Problem 2 it suffices to characterize the VGF of a regular language $L'' \subseteq \{\mathfrak{s}, \mathfrak{t}\}^*$.

Remark 2 Prolongations of languages and their $\mathfrak{s}$-$\mathfrak{t}$-encodings do not contain the empty word ε. Adding such a word to these languages has the sole effect of adding 1 to their volume generating functions. In terms of permutations this models the addition of the 0-length permutation to a regular class[8]. This allows us to simplify the automata considered as those depicted in Figure 1 and 4 that are respectively involved in the running example and in the example of alternating permutations.

[8] The unique permutation on the empty set has no signature and thus $\mathfrak{S}_0 \not\subseteq \Lambda(L)$ for any language L of signature.

4.3 Recursive formulae for volume functions and cardinalities

We assume that for the rest of Section 4 an automaton $\mathcal{B} = (\{\mathfrak{s}, \mathfrak{t}\}, Q, q_0, F, \delta)$ is given and we denote by $\mathbb{L}''$ its timed semantics. We describe recursive equations on timed languages starting from clock vectors (as defined in Section 4.2.2), that is, timed languages of the form $[\mathcal{B}_q]_n(x)$ for $q \in Q$, $n \in \mathbb{N}$ and $x \in [0,1]$. The languages $[\mathcal{B}_q]_n(x)$ can be recursively defined as follows: $[\mathcal{B}_q]_0(x) = \varepsilon$ if $q \in F$ and $[\mathcal{B}_q]_0 = \emptyset$ otherwise;

$$[\mathcal{B}_q]_{n+1}(x) = \bigcup_{t \leq 1-x} (t, \mathfrak{s})[\mathcal{B}_{q.\mathfrak{s}}]_n(x+t) \cup \bigcup_{t \leq 1-x} (t, \mathfrak{t})[\mathcal{B}_{q.\mathfrak{t}}]_n(t). \quad (44)$$

For $q \in Q$ and $n \geq 0$, we denote by $\mathrm{vc}_{q,n}$ the function $x \mapsto \mathtt{Vol}[[\mathcal{B}_q]_n(x)]$ from $[0,1]$ to $\mathbb{R}^+$. Hence, each $\mathrm{vc}_{q,n}$ is a polynomial of a degree at most n that can be computed recursively using the recurrence formulas: $\mathrm{vc}_{q,0}(x) = 1_{q \in F}$ and

$$\mathrm{vc}_{q,n+1}(x) = \int_x^1 \mathrm{vc}_{q.\mathfrak{s},n}(y)dy + \int_0^{1-x} \mathrm{vc}_{q.\mathfrak{t},n}(y)dy. \quad (45)$$

Note that $\mathrm{vc}_{q,n}(1) = 0$ and $\mathrm{vc}_{q_0,n}(0) = \mathtt{Vol}(\mathbb{L}''_n)$ is the nth coefficient of the VGF $T_{\mathbb{L}''}$ we want to evaluate.

Now we have a new integral operator $\int_0^{1-x} dy$ that still behaves well on Bernstein polynomials due to the following remarkable property: $b_{j,n}(1-t) = b_{n-j,n}(t)$ for every $j \leq n \in \mathbb{N}$ and $t \in [0,1]$. More explicitly it holds that:

$$\int_0^{1-x} b_{j,n}(y)dy = \int_x^1 b_{j,n}(1-t)dt = \int_x^1 b_{n-j,n}(t)dt = \frac{1}{n+1}\sum_{i=0}^{n-j} b_{i,n+1}(x) \quad (46)$$

Proposition 12 *For every $q \in Q$, $n \in \mathbb{N}$, it holds that*

$$\mathrm{vc}_{q,n}(x) = \frac{1}{n!}\sum_{k=0}^{n} \beta_{n+1,k+1}([\mathcal{B}_q])b_{k,n}(x) \quad (47)$$

where the coefficients $\beta_{n,k}([\mathcal{B}_q])$ are defined recursively as follows: $\beta_{1,1}([\mathcal{B}_q]) = 1_{q \in F}$ and for $n \geq 1$, and for $k \in [n+1]$

$$\beta_{n+1,k}([\mathcal{B}_q]) = \sum_{i=k}^{n} \beta_{n,i}([\mathcal{B}_{q.\mathfrak{s}}]) + \sum_{i=1}^{n+1-k} \beta_{n,i}([\mathcal{B}_{q.\mathfrak{t}}]), \quad (48)$$

where by convention empty sums are null and hence $\beta_{n+1,n+1}([\mathcal{B}_q]) = 0$.

Proof We prove this by induction. The base case is satisfied $\mathrm{vc}_{q,1}(x) = 1_{q\in F} = \beta_{1,1}([B_q])b_{1,1}(x)$. To prove the induction step we use (45). The first integral yields:

$$\begin{aligned}\int_x^1 \mathrm{vc}_{q.\mathfrak{s},n}(y)dy &= \frac{1}{n!}\sum_{j=0}^{n}\beta_{n+1,j+1}([\mathcal{B}_{q.\mathfrak{s}}])\int_x^1 b_{j,n}(y)dy;\\ &= \frac{1}{(n+1)!}\sum_{j=0}^{n}\beta_{n+1,j+1}([\mathcal{B}_{q.\mathfrak{s}}])\sum_{k=0}^{j} b_{k,n+1}(x) \quad \text{(by (19))};\\ &= \frac{1}{(n+1)!}\sum_{k=0}^{n}\left(\sum_{j=k}^{n}\beta_{n+1,j+1}([\mathcal{B}_{q.\mathfrak{s}}])\right) b_{k,n+1}(x);\end{aligned}$$

while the second integral yields:

$$\begin{aligned}\int_0^{1-x} \mathrm{vc}_{q.\mathfrak{t},n}(y)dy &= \frac{1}{n!}\sum_{j=0}^{n}\beta_{n+1,j+1}([\mathcal{B}_{q.\mathfrak{t}}])\int_0^{1-x} b_{j,n}(y)dy;\\ &= \frac{1}{(n+1)!}\sum_{j=0}^{n}\beta_{n+1,j+1}([\mathcal{B}_{q.\mathfrak{t}}])\sum_{k=0}^{n-j} b_{k,n+1}(x) \quad \text{(by (46))};\\ &= \frac{1}{(n+1)!}\sum_{k=0}^{n}\left(\sum_{j=0}^{n-k}\beta_{n+1,j+1}([\mathcal{B}_{q.\mathfrak{t}}])\right) b_{k,n+1}(x).\end{aligned}$$

Then if we write $\mathrm{vc}_{q,n+1}(x)$ in the basis of Bernstein polynomials, the coefficient associated to $b_{k,n+1}(x)/(n+1)!$ is as expected:

$$\beta_{n+2,k+1}([\mathcal{B}_q]) = \sum_{i=k+1}^{n+1}\beta_{n+1,i}([\mathcal{B}_{q.\mathfrak{s}}]) + \sum_{i=1}^{n+1-k}\beta_{n+1,i}([\mathcal{B}_{q.\mathfrak{t}}]).$$

□

In Section 3.2.1, the coefficients $\alpha_{n,k}$ were directly defined as cardinalities of sets of permutations that are the discrete version of order polytopes. We think that a discrete version of the chain polytopes can be defined as well to give a direct interpretation of the coefficient $\beta_{n,k}$ in terms of cardinality of sets of permutations. In any case, these coefficients can be computed using local recursive rules as follows.

Proposition 13 *One can use the following recursion scheme to compute the coefficients* $\beta_{n,k}([\mathcal{B}_q])$ *and hence the volume functions* $\mathrm{vc}_{q,n}$ *in the Bernstein basis in arithmetic time and space complexity* $O(|Q|n^2)$.

$$\beta_{n+1,k}([\mathcal{B}_q]) = \beta_{n+1,k}(\mathfrak{s}[\mathcal{B}_{q.\mathfrak{s}}]) + \beta_{n+1,k}(\mathfrak{t}[\mathcal{B}_{q.\mathfrak{t}}]); \tag{49}$$

$$\beta_{n+1,k}(\mathfrak{s}[\mathcal{B}_q]) = \beta_{n+1,k+1}(\mathfrak{s}[\mathcal{B}_q]) + \beta_{n,k}([\mathcal{B}_q]); \tag{50}$$

$$\beta_{n+1,k}(\mathfrak{t}[\mathcal{B}_q]) = \beta_{n+1,k}(\mathfrak{t}[\mathcal{B}_q]) + \beta_{n,n+1-k}([\mathcal{B}_q]); \tag{51}$$

$$\beta_{n+1,n+1}(\mathfrak{s}[\mathcal{B}_q]) = 0; \quad \beta_{n+1,n+1}(\mathfrak{t}[\mathcal{B}_q]) = 0; \tag{52}$$

$$\beta_{1,1}([\mathcal{B}_q]) = 1_{q\in F}. \tag{53}$$

Proposition 14 *The coefficients of the volume function* $\mathrm{vc}_{q,n}$ *can be computed in the standard basis in arithmetic time and space complexity* $O(|Q|n^2)$*, using the following system of recursive equations on* $\mathrm{vc}_{q,n}(x)$ *and* $\tilde{\mathrm{vc}}_{q,n}(x) \stackrel{\text{def}}{=} \mathrm{vc}_{q,n}(1-x)$*:*

$$\begin{aligned}
\mathrm{vc}_{q,0}(x) &= \tilde{\mathrm{vc}}_{q,0}(x) = 1_{q\in F};\\
\mathrm{vc}_{q,n+1}(x) &= \int_x^1 \mathrm{vc}_{q.\mathtt{s},n}(y)dy + \int_x^1 \tilde{\mathrm{vc}}_{q.\mathtt{t},n}(y)dy;\\
\tilde{\mathrm{vc}}_{q,n+1}(x) &= \int_0^x \tilde{\mathrm{vc}}_{q.\mathtt{s},n}(y)dy + \int_0^x \mathrm{vc}_{q.\mathtt{t},n}(y)dy.
\end{aligned}$$

These equations are obtained from (45). We introduced the functions $\tilde{\mathrm{vc}}_{q,n}$ because developing the expression

$$\int_0^{1-x} \mathrm{vc}_{q.\mathtt{t},n}(y)dy = \sum_{k=0}^{n} \frac{1}{k+1} a_k (1-x)^{k+1}.$$

knowing only $\mathrm{vc}_{q.\mathtt{t},n}(y) = \sum_{k=0}^n a_k y^k$ needs $O(n^2)$ arithmetic operations. By contrast, when $\tilde{\mathrm{vc}}_{q.\mathtt{t},n}(y) = \sum_{k=0}^n b_k y^k$ is given, the expression

$$\int_0^{1-x} \mathrm{vc}_{q.\mathtt{t},n}(y)dy = \int_x^1 \tilde{\mathrm{vc}}_{q.\mathtt{t},n}(y)dy = \sum_{k=0}^{n} \frac{1}{k+1} b_k - \sum_{k=0}^{n} \frac{1}{k+1} b_k y^{k+1}.$$

is computed in linear time.

The following proposition states remarkable properties on the shape of the volume functions that will be extended to generating functions in Section 4.4.

Proposition 15 *For every* $q \in Q$*,* $n \geq 0$*, the function* $\mathrm{vc}_{q,n}$ *is non-increasing on* $[0,1]$*, and satisfies for every* $x \in [0,1]$*:*

$$(1-x)^n \mathrm{vc}_{q,n}(0) \leq \mathrm{vc}_{q,n}(x) \leq \mathrm{vc}_{q,n}(0) = \beta_{n+1,1}([\mathcal{B}_q]).$$

Proof The fact that the function is non-increasing is straightforward using (45) and the positivity of the volume function on $[0,1]$.

For the first inequality we use the decomposition of $\mathrm{vc}_{q,n}$ as a sum of Bernstein polynomials (47):

$$\mathrm{vc}_{q,n}(x) = \frac{1}{n!} \sum_{k=0}^{n} \beta_{n+1,k+1}([\mathcal{B}_q]) b_{k,n}(x).$$

All the terms of this sum are non-negative for $x \in [0,1]$ and for $x = 0$, $b_{k,n}(0)$ is not null only for $k = 0$ (and in that case equal to 1). Hence $\mathrm{vc}_{q,n}(0) = \beta_{n+1,1}([\mathcal{B}_q])$ and $\mathrm{vc}_{q,n}(0)(1-x)^n = \beta_{n+1,1}([\mathcal{B}_q]) b_{0,n}(x) \leq \mathrm{vc}_{q,n}(x)$. □

4.4 Generating functions

4.4.1 Characterisation of the generating functions

The timed language $[\mathcal{B}_q](x)$ starting from states $q \in Q$ and clock $x \in [0,1]$ satisfies the following system of language equations:

$$[\mathcal{B}_q](x) = \bigcup_{t \le 1-x} (t, \mathfrak{s})[\mathcal{B}_{q.\mathfrak{s}}](x+t) \cup \bigcup_{t \le 1-x} (t, \mathfrak{t})[\mathcal{B}_{q.\mathfrak{t}}](t) \cup (\varepsilon \text{ if } q \in F). \tag{54}$$

We denote by $\mathrm{VC}_q(x,z)$ and $T_q(z)$ the volume generating function of $[\mathcal{B}_q](x)$ and $[\mathcal{B}_q]$ respectively. We are interested in $T_{\mathbb{L}''}(z) = T_{q_0}(z) = \mathrm{VC}_{q_0}(0,z)$.

We recall that $\texttt{Rconv}(\mathrm{VC}_q) = \inf_{x\in[0,1]} \texttt{Rconv}(z \mapsto \mathrm{VC}_q(x,z))$ and $\texttt{Rconv}(\mathbf{VC}) = \min_{q\in Q} \texttt{Rconv}(\mathrm{VC}_q)$. The link between the different convergence radii is done in Corollary 5 below.

As in [ABDP12], we translate systems of equations on timed languages (54) into systems of equations on generating functions (55):

Theorem 3 *For $z < \texttt{Rconv}(\mathbf{VC})$, $x \mapsto \mathbf{VC}(x,z) = (\mathrm{VC}_q(x,z))_{q\in Q}$ is the unique solution of the following system of integral equations:*

$$\mathrm{VC}_q(x,z) = z\int_x^1 \mathrm{VC}_{q.\mathfrak{s}}(y,z)dy + z\int_0^{1-x} \mathrm{VC}_{q.\mathfrak{t}}(y,z)dy + 1_{q\in F}. \tag{55}$$

One can equivalently state this theorem in terms of a system of differential equations as follows.

Corollary 4 *For $z < \texttt{Rconv}(\mathbf{VC})$, $x \mapsto \mathbf{VC}(x,z) = (\mathrm{VC}_q(x,z))_{q\in Q}$ is the unique solution of the following system of differential equations*

$$\frac{\partial}{\partial x}\mathrm{VC}_q(x,z) = -z\mathrm{VC}_{q.\mathfrak{s}}(x,z) - z\mathrm{VC}_{q.\mathfrak{t}}(1-x,z); \tag{56}$$

with boundary conditions

$$\mathrm{VC}_q(1,z) = 1_{q\in F}. \tag{57}$$

In matrix notation (55), (56) and (57) give:

$$\mathbf{VC}(x,z) = zM_{\mathfrak{s}}\int_x^1 \mathbf{VC}(y,z)dy + zM_{\mathfrak{t}}\int_0^{1-x} \mathbf{VC}(y,z)dy + \mathbf{F}; \tag{58}$$

$$\frac{\partial}{\partial x}\mathbf{VC}(x,z) = -zM_{\mathfrak{s}}\mathbf{VC}(x,z) - zM_{\mathfrak{t}}\mathbf{VC}(1-x,z); \tag{59}$$

$$\mathbf{VC}(1,z) = \mathbf{F}. \tag{60}$$

The following theorem gives the form of the solution in terms of the exponential $E(z)$ of a matrix M defined as follows:

$$M \stackrel{\text{def}}{=} \begin{pmatrix} -M_{\mathfrak{s}} & -M_{\mathfrak{t}} \\ M_{\mathfrak{t}} & M_{\mathfrak{s}} \end{pmatrix} \text{ and } E(z) \stackrel{\text{def}}{=} \begin{pmatrix} E_1(z) & E_2(z) \\ E_3(z) & E_4(z) \end{pmatrix} \stackrel{\text{def}}{=} \exp(zM).$$

Theorem 4 *In a neighbourhood of 0, the matrix $E_1(z)$ and $I_{|Q|} - E_3(z)$ are invertible and the following two characterizations of $\mathbf{T}(z)$ hold:*

$$\mathbf{T}(z) = [E_1(z)]^{-1}[I_{|Q|} - E_2(z)]\mathbf{F}; \tag{61}$$

$$\mathbf{T}(z) = [I_{|Q|} - E_3(z)]^{-1}E_4(z)\mathbf{F}. \tag{62}$$

Proof The equation (59) is equivalent to the following linear homogeneous system of ordinary differential equations with constant coefficients:

$$\frac{\partial}{\partial x}\begin{pmatrix}\mathbf{VC}(x,z)\\ \mathbf{VC}(1-x,z)\end{pmatrix} = zM\begin{pmatrix}\mathbf{VC}(x,z)\\ \mathbf{VC}(1-x,z)\end{pmatrix}. \tag{63}$$

Its solution is of the form

$$\begin{pmatrix}\mathbf{VC}(x,z)\\ \mathbf{VC}(1-x,z)\end{pmatrix} = E(xz)\begin{pmatrix}\mathbf{VC}(0,z)\\ \mathbf{VC}(1,z)\end{pmatrix}. \tag{64}$$

Taking $x = 1$ in (64) and using the boundary condition (60) we obtain:

$$\begin{pmatrix}\mathbf{F}\\ \mathbf{T}(z)\end{pmatrix} = E(z)\begin{pmatrix}\mathbf{T}(z)\\ \mathbf{F}\end{pmatrix}. \tag{65}$$

Hence,

$$\mathbf{F} = E_1(z)\mathbf{T}(z) + E_2(z)\mathbf{F}; \quad \mathbf{T}(z) = E_3(z)\mathbf{T}(z) + E_4(z)\mathbf{F}. \tag{66}$$

In particular when $z = 0$, it holds that $E_1(0) = I_{|Q|} - E_3(0) = I_{|Q|}$ and thus the two continuous functions $z \mapsto \det E_1(z)$ and $z \mapsto \det(I_{|Q|} - E_3(z))$ are positive in a neighbourhood of 0. We deduce that the inverses of the matrices $E_1(z)$ and $I - E_3(z)$ are well defined in a neighbourhood of 0 and thus both equations of (66) permit to express $\mathbf{T}(z)$ with respect to $\mathbf{F}$ to get respectively (61) and (62). □

As a corollary of the proof of Theorem 4 one can obtain the vector of generating functions $\begin{pmatrix}\mathbf{VC}(x,z)\\ \mathbf{VC}(1-x,z)\end{pmatrix}$ as follows

$$\begin{pmatrix}\mathbf{VC}(x,z)\\ \mathbf{VC}(1-x,z)\end{pmatrix} = E(xz)\begin{pmatrix}\mathbf{T}(z)\\ \mathbf{F}\end{pmatrix}. \tag{67}$$

4.4.2 Closed form formula and Taylor expansion of the generating function

The characterisation of Theorem 4 allows us to derive an algorithm (Algorithm 1) that determines a closed form formula for the generating function $G_L(z) = T_{q_0}(z)$. For this we need to compute symbolically the matrix $E(z) = \exp(zM)$.

Proposition 16 (Direct consequence of [Cai94]) *Given a square matrix M with rational coefficients and eigenvalues $\lambda_1, \ldots, \lambda_r$, the coefficients of $E(z) = \exp(zM)$ are polynomials in $\lambda_1, \ldots, \lambda_r, e^{\lambda_1 z}, \ldots, e^{\lambda_r z}, z$ that can be computed in polynomial time.*

Proof This proposition is based on the two following assertions. The former (A1) is proved in [Cai94]. The latter (A2) is folklore and can be found for instance in [OPW15].

- A1 One can compute a Jordan form, $M = PJP^{-1}$ in polynomial time, the coefficients of P and P^{-1} are polynomials in $\lambda_1, \ldots, \lambda_r$ with rational coefficients and a polynomial number of coefficients.
- A2 It holds that $\exp(zM) = P\exp(zJ)P^{-1}$ with $\exp(zJ)$ having coefficients of the form $e^{\lambda z}p(z)$ with λ an eigenvalue of M and $p(z)$ a polynomial in z of degree strictly smaller than the multiplicity of λ.

Algorithm 1 Computation of a closed form formula for the EGF $G_L(z)$

1: Compute an automaton $\mathcal{B} = (\{\mathfrak{s}, \mathfrak{t}\}, Q, q_0, F, \delta)$ that recognises an $\mathfrak{s}$-$\mathfrak{t}$-encoding of an extension of L and compute its adjacency matrices $M_\mathfrak{s}$ and $M_\mathfrak{t}$;

2: Compute $\begin{pmatrix} E_1(z) & E_2(z) \\ E_3(z) & E_4(z) \end{pmatrix} \stackrel{\text{def}}{=} \exp\left[z \begin{pmatrix} -M_\mathfrak{s} & -M_\mathfrak{t} \\ M_\mathfrak{t} & M_\mathfrak{s} \end{pmatrix}\right]$;

3: Solve the system of linear equations $E_1(z)\mathbf{T}(z) = [I_{|Q|} - E_2(z)]\mathbf{F}$;

4: **return** $G_L(z) = T_{q_0}(z)$ the component of $\mathbf{T}(z)$ corresponding to the initial state of $\mathcal{B}$.

Hence the coefficients of the matrix $\exp(zM) = P\exp(zJ)P^{-1}$ are polynomials in $\lambda_1, \ldots, \lambda_r, e^{\lambda_1 z}, \ldots, e^{\lambda_r z}, z$ that can be computed in polynomial time. □

As a consequence of this proposition and of Theorem 4, we have the following:

Theorem 5 (Closed form formula for the generating function) *Let $\lambda_1, \ldots, \lambda_r$ be the eigenvalues of M. A closed form formula for the generating function $G_L(z) = T_{q_0}(z)$ as a rational function with rational coefficients over variables λ_1, …, λ_r, $e^{\lambda_1 z}$, …, $e^{\lambda_r z}$, z can be computed in* **EXPTIME**$(|Q|)$ *using Algorithm 1.*

Some comments about Algorithm 1. In line 1, several choices are left to the user: the prolongation L' of the language L, the type of the $\mathfrak{s}$-$\mathfrak{t}$-encoding and the automaton that realizes the $\mathfrak{s}$-$\mathfrak{t}$-encoding. These choices should be made in such a way that the output automaton has a small number of states or more generally such that the matrices $M_\mathfrak{s}$ and $M_\mathfrak{t}$ are the simplest possible. Algorithm 1 can also be used as a numerical method given an input complex number $z < \mathtt{Rconv}(\mathbf{T})$. Further properties of $E(z)$ are given in Section 4.4.4.

Proof (of Theorem 5) Solving the system $E_1(z)\mathbf{T}(z) = (I_{|Q|} - E_2(z))\mathbf{F}$ involves doing a polynomial number of products and inversions of rational functions in variables λ_1, …, λ_r, $e^{\lambda_1 z}$, …, $e^{\lambda_r z}$, z. This yields rational functions with at most an exponential number of coefficients. This gives the **EXPTIME**$(|Q|)$ upper-bound. □

For the lower-bound on the complexity, further studies are required that go beyond the scope of the present paper. We do not know if a closed form formula for $G_L(z)$ that needs only a polynomial number of symbols exists in general.

Taylor expansion computation. The implicit characterisation of the generating function stated in Theorem 4 can be used to derive a novel algorithm to solve Problem 1 in subquadratic time with respect to $n \in \mathbb{N}$.

Theorem 6 *Given a regular language L and an automaton $\mathcal{B} = (\{\mathfrak{s}, \mathfrak{t}\}, Q, q_0, F, \delta)$ that recognises an $\mathfrak{s}$-$\mathfrak{t}$-encoding of an extension of L. For every $n \in \mathbb{N}$, the Taylor expansion $\sum_{i=1}^{n} \alpha_i(L)z^i/i!$ at order n of $G_L(z) = T_{q_0}(z)$ can be computed in arithmetic time complexity $O(n\log(n)|Q|^3)$ using Algorithm 2.*

Proof First we recall that product of nth degree (truncated) polynomials can be done in $O(n \log n)$ using Fast Fourier Transform (see for instance [CLRS09]). Inverting truncated polynomials requires also $O(n \log n)$ operations [Kal93]. Line 3 needs n multiplications of $2m \times 2m$ matrices and hence is achieved in $O(n|Q|^3)$.

Algorithm 2 Computation of a Taylor expansion at order n for the EGF $G_L(z)$

1: Compute an automaton $\mathcal{B} = (\{\mathfrak{s},\mathfrak{t}\}, Q, q_0, F, \delta)$ that recognises an $\mathfrak{s}$-$\mathfrak{t}$-encoding of an extension of L and compute its adjacency matrices $M_\mathfrak{s}$ and $M_\mathfrak{t}$;
2: Define $M = \begin{pmatrix} -M_\mathfrak{s} & -M_\mathfrak{t} \\ M_\mathfrak{t} & M_\mathfrak{s} \end{pmatrix}$;
3: Compute M^i for $i \leq n$;
4: Compute $\begin{pmatrix} E_1^{(n)}(z) & E_2^{(n)}(z) \\ E_3^{(n)}(z) & E_4^{(n)}(z) \end{pmatrix} \stackrel{\text{def}}{=} \sum_{i=1}^{n} (z^i/i!) M^i$;
5: Solve the linear system of equations $E_1^{(n)}(z)\mathbf{T}^{(n)}(z) = [I_{|Q|} - E_2^{(n)}(z)]\mathbf{F}$ with operations done on polynomials truncated to the nth order.
6: **return** the polynomial $T_{q_0}^{(n)}(z)$ corresponding to the initial state of $\mathcal{B}$.

Line 4 consists in a summation of matrices computed just above, it has complexity $O(n|Q|^2)$. In Line 5 the inversion of the matrix can be done using Gaussian elimination, requiring $O(|Q|^3)$ operations on truncated polynomials each one being achieved in $O(n \log n)$ arithmetic operations. At the end, the arithmetic time complexity is $O(n \log(n)|Q|^3)$.

4.4.3 Properties of the generating functions and convergence radii

The following proposition gives some properties on the shape of the generating functions:

Proposition 17 *For every $z < \texttt{Rconv}(\mathrm{VC}_q)$, the function $x \mapsto \mathrm{VC}_q(x, z)$ is non-increasing in $[0, 1]$ and decreasing in $(0, 1)$ if non constant and satisfy for every $x \in [0, 1]$:*

$$\mathrm{VC}_q(0, (1-x)z) \leq \mathrm{VC}_q(x, z) \leq \mathrm{VC}_q(0, z); \tag{68}$$

moreover for every $l \in \{\mathfrak{s},\mathfrak{t}\}$ it also holds that

$$\mathrm{VC}_q(0, z) \geq z\mathrm{VC}_{q.l}(0, z). \tag{69}$$

Proof The fact that the function $x \mapsto \mathrm{VC}_q(x, z)$ is non-increasing comes from Proposition 15. The sequence of inequalities (68) is obtained by taking $\sum .z^n$ in the inequalities of Proposition 15. The last inequality (69) relies on the fixed point equation (55) and non-negativity of the generating functions:

$$\mathrm{VC}_q(0, z) \geq z \int_0^1 \mathrm{VC}_{q.l}(y, z)dy \geq z\mathrm{VC}_{q.l}(0, z).$$

□

As a corollary we can compare convergence radii as follows.

Corollary 5 *For every $q \in Q$, $l \in \Sigma$ it holds that $\texttt{Rconv}(\mathrm{VC}_q) = \texttt{Rconv}(z \mapsto \mathrm{VC}_q(0, z))$ and $\texttt{Rconv}(\mathrm{VC}_q) \leq \texttt{Rconv}(\mathrm{VC}_{q.l})$. In particular, since the automaton is accessible it also holds that:*

$$\texttt{Rconv}(\mathbf{T}) = \texttt{Rconv}(z \mapsto \mathrm{VC}_{q_0}(0, z)) = \texttt{Rconv}(T_{\mathbb{L}''}).$$

4.4.4 Further properties of the matrix exponentiation

To evaluate the generating function in a given point $(x, z) \in [0,1] \times [0, \mathtt{Rconv}(\mathbf{T}))$, one can compute numerically $E(z) = \exp(zM)$ and $E(xz) = \exp(xzM)$, and solve systems (65) and (67). There are numerous ways of computing the matrix exponential (see [MVL03] for nineteen of them). To choose the right algorithm one has to take into account properties of M such as its sparsity. Each row of the matrix M has at most two non-null entries that are either 1 or -1.

The purpose of this section is to highlight some remarkable properties of M that could be helpful for its exponentiation (either numerical or symbolical). In particular we describe some properties of the spectrum of M (the set of eigenvalues of M) denoted by $\mathrm{Sp}(M)$.

We define the matrix $S = \begin{pmatrix} 0 & I_{|Q|} \\ I_{|Q|} & 0 \end{pmatrix}$. This matrix permutes the $|Q|$ first lines with the $|Q|$ last lines of any matrix it multiplies from the left, and permutes the $|Q|$ first columns with the $|Q|$ last columns of any matrix it multiplies from the right. The matrix S is an involutory matrix (that is, equal to its own inverse). Note that M enjoys a central antisymmetry in the following sense: $SMS = -M$.

The following results hold for every matrix M such that $SMS = -M$, that is, of the form $M = \begin{pmatrix} -A & -B \\ B & A \end{pmatrix}$ for some matrices A and B.

One can first remark that M admits the following block anti-diagonalisation:

$$P \begin{pmatrix} -A & -B \\ B & A \end{pmatrix} P^\top = \begin{pmatrix} 0 & A-B \\ A+B & 0 \end{pmatrix} \tag{70}$$

with P the orthonormal matrix defined by block as follows $P = \frac{1}{\sqrt{2}} \begin{pmatrix} I & I \\ -I & I \end{pmatrix}$.

We let $C = A - B$ and $D = A + B$ and remark that

$$\begin{pmatrix} 0 & C \\ D & 0 \end{pmatrix}^{2n} = \begin{pmatrix} (CD)^n & \\ 0 & (DC)^n \end{pmatrix} \text{ and } \begin{pmatrix} 0 & C \\ D & 0 \end{pmatrix}^{2n+1} = \begin{pmatrix} 0 & (CD)^n C \\ D(CD)^n & 0 \end{pmatrix}.$$

We denote

$$F_1(z) = \sum_{n=0}^{+\infty} (CD)^n \frac{z^{2n}}{2n!};\ F_2(z) = \sum_{n=0}^{+\infty} (DC)^n \frac{z^{2n}}{2n!};\ F_3(z) = \sum_{n=0}^{+\infty} (CD)^n \frac{z^{2n+1}}{(2n+1)!},$$

and then it holds that:

$$E(z) = P^\top \begin{pmatrix} F_1(z) & F_3(z)C \\ DF_3(z) & F_2(z) \end{pmatrix} P \tag{71}$$

In the next proposition, for a complex number λ, we denote by $E_\lambda(M)$ the vector space $\{\mathbf{v} \in \mathbb{C}^{2|Q|} \mid M\mathbf{v} = \lambda\mathbf{v}\}$. If $E_\lambda(M) \neq \{0\}$, this set is known as the eigenspace of M for the eigenvalue λ.

Proposition 18 *For every non-null complex number μ, and for both $\lambda \in \mathbb{C}$ such that $\lambda^2 = \mu$ the following holds: $\mu \in \mathrm{Sp}\,[(A-B)(A+B)]$ iff $\lambda \in \mathrm{Sp}(M)$. More precisely, if $\mu \neq 0$ and λ is such that $\lambda^2 = \mu$ then $\mathbf{v} \mapsto \begin{pmatrix} \lambda I_{|Q|} - A - B \\ \lambda I_{|Q|} + A + B \end{pmatrix} \mathbf{v}$ is an isomorphism between $E_\mu((A-B)(A+B))$ and $E_\lambda(M)$.*

Proof Note that $\begin{pmatrix} \lambda I_{|Q|} - A - B \\ \lambda I_{|Q|} + A + B \end{pmatrix} \mathbf{v} = \lambda\sqrt{2}P^T \begin{pmatrix} \mathbf{v} \\ \frac{1}{\lambda}D\mathbf{v} \end{pmatrix}$ and hence, it suffices to show that $\mathbf{v} \mapsto \begin{pmatrix} \mathbf{v} \\ \frac{1}{\lambda}D\mathbf{v} \end{pmatrix}$ is an isomorphism between $E_{\lambda^2}(CD)$ and $E_\lambda(PMP^\top) = E_\lambda\left[\begin{pmatrix} 0 & C \\ D & 0 \end{pmatrix}\right]$. This function is clearly linear. Take $\mathbf{v} \in E_{\lambda^2}(CD)$, then

$$\begin{pmatrix} 0 & C \\ D & 0 \end{pmatrix}\begin{pmatrix} \mathbf{v} \\ \frac{1}{\lambda}D\mathbf{v} \end{pmatrix} = \begin{pmatrix} C\frac{1}{\lambda}D\mathbf{v} \\ D\mathbf{v} \end{pmatrix} = \lambda\begin{pmatrix} \mathbf{v} \\ \frac{1}{\lambda}D\mathbf{v} \end{pmatrix}.$$

Take any $\begin{pmatrix} \mathbf{v} \\ \mathbf{y} \end{pmatrix} \in E_\lambda\left[\begin{pmatrix} 0 & C \\ D & 0 \end{pmatrix}\right]$. It exactly means that $C\mathbf{v} = \lambda\mathbf{v}$ and $D\mathbf{v} = \lambda\mathbf{y}$. Thus $\begin{pmatrix} \mathbf{v} \\ \mathbf{y} \end{pmatrix} = \lambda\begin{pmatrix} \mathbf{v} \\ \frac{1}{\lambda}D\mathbf{v} \end{pmatrix}$ is of the required form and $CD\mathbf{v} = \lambda C\mathbf{y} = \lambda^2\mathbf{v}$. □

The proposition above implies that if λ is an eigenvalue of M, then so is $-\lambda$. We have not yet compared the multiplicities of such mutually opposite eigenvalues. The multiplicities are the same as a corollary of the following remarkable identity.

Proposition 19 *The following identity holds $E(-z) = SE(z)S$.*

Proof Indeed $E(-z) = \exp(-zM) = \exp(zSMS) = S\exp(zM)S = SE(z)S$. □

We can deduce from this proposition that $SE(z)$ and $E(z)S$ are involutory matrices $(SE(z))^2 = (E(z)S)^2 = I_{2|Q|}$. We have also that $E(z)$ is of the following form: $E(z) = \begin{pmatrix} E_1(z) & E_2(z) \\ E_2(-z) & E_1(-z) \end{pmatrix}$. The identity $E(z)E(-z) = I_{2|Q|}$ yields

$$\begin{array}{ll} E_1(z)E_2(-z) \;\;= I_{|Q|} - E_2(z)^2 \;\; ; & E_1(z)E_1(-z) = -E_2(z)E_1(z); \\ E_1(-z)E_2(-z) = -E_2(-z)E_2(z) \; ; & E_2(-z)E_1(z) = I_{|Q|} - E_1(-z)^2; \end{array}$$

that can be used for instance to show that equations (61) and (62) are equivalent.

Corollary 6 *The spectrum of M is symmetric with respect to the origin, that is, if λ is an eigenvalue of M with a multiplicity m_λ then so is $-\lambda$.*

Proof The characteristic polynomial of M is $\chi_M(X) = \prod_{\lambda\in \mathrm{Sp}(M)}(\lambda - X)^{m_\lambda}$, that of $E(z) = \exp(zM)$ is $\chi_{E(z)}(X) = \prod_{\lambda\in \mathrm{Sp}(M)}(e^{z\lambda} - X)^{m_\lambda}$ and that of $E(-z) = \exp(zM)$ is $\chi_{E(-z)}(X) = \prod_{\lambda\in \mathrm{Sp}(M)}(e^{-z\lambda} - X)^{m_\lambda}$. By virtue of Proposition 19, $E(-z)$ and $E(z)$ are conjugate and hence have the same characteristic polynomial. Necessarily for every $\lambda \in \mathrm{Sp}(M)$, it holds that $-\lambda \in \mathrm{Sp}(M)$ and $m_\lambda = m_{-\lambda}$.

4.4.5 Examples

Alternating permutations. The class of alternating permutations is $\texttt{Alt} = \mathfrak{S}_0 \cup \Lambda[\{\mathfrak{d}\mathfrak{a}\}^*\{\varepsilon, \mathfrak{d}\}]$. It is well known since the 19th century and the work of Désiré André that the exponential generating function of this class of permutations is

$$\tan(z) + \sec(z) \qquad (\text{where } \sec(z) = 1/\cos(z)).$$

Several different proofs of this results can be found in [Sta10]. Here we give a novel proof illustrating our method.

A prolongation of $\{\mathfrak{d}\mathfrak{a}\}^*\{\varepsilon, \mathfrak{d}\}$ is $\{\mathfrak{d}\mathfrak{a}\}^*\{\mathfrak{d}, \mathfrak{d}\mathfrak{a}\}$. We add ε to the language (to add 1 to its VGF that counts the 0-length permutation as described in remark 2) and obtain $\{\mathfrak{d}\mathfrak{a}\}^*\{\varepsilon, \mathfrak{d}\}$.

The $\mathfrak{s}$-$\mathfrak{t}$ encoding of type $\mathfrak{a}$ of $\{\mathfrak{d}\mathfrak{a}\}^*\{\varepsilon, \mathfrak{d}\}$ is just $\{\mathfrak{t}\}^*$ which is recognized by the one loop automaton depicted in the right of Figure 4. Thus $M_{\mathfrak{s}} = (0)$, $M_{\mathfrak{t}} = (1)$ and we must compute $\exp(zM) = \sum_{n\in\mathbb{N}} z^n M^n/n!$ with $M = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$.

This is an easy particular case of exponentiation done in Section 4.4.4 with $C = (-1)$ and $D = (1)$. It gives $\exp(zM) = \begin{pmatrix} \cos(z) & -\sin(z) \\ \sin(z) & \cos(z) \end{pmatrix}$.

By definition $E_1(z) = \cos(z)$, $E_2(z) = -\sin(z)$. We can conclude that the desired generating function is:

$$E_1(z)^{-1}(1 - E_2(z)) = \frac{1}{\cos(z)} + \tan(z).$$

One could alternatively use (62) to get the same result:

$$[I_{|Q|} - E_3(z)]^{-1} E_4(z) = \frac{\cos(z)}{1 - \sin(z)} = \frac{\cos(z)\,[1 + \sin(z)]}{1 - \sin^2(z)} = \frac{1 + \sin(z)}{\cos(z)}. \tag{72}$$

Moreover the generating function $\mathrm{VC}(x, z)$ can be obtained using (67):

$$\begin{pmatrix} \mathrm{VC}(x, z) \\ \mathrm{VC}(1 - x, z) \end{pmatrix} = \begin{pmatrix} \cos(xz) & -\sin(xz) \\ \sin(xz) & \cos(xz) \end{pmatrix} \begin{pmatrix} \frac{1}{\cos(z)} + \tan(z) \\ 1 \end{pmatrix}$$

Hence $\mathrm{VC}(x, z) = [\cos(xz) + \sin(z)\cos(xz) - \sin(xz)\cos(z)] / \cos(z)$ which after simplification gives:

$$\mathrm{VC}(x, z) = \frac{\cos(xz) + \sin((1 - x)z)}{\cos(z)} \tag{73}$$

Running example. Now we apply our method to the running example $L^{(\mathrm{run})} = (\{\mathfrak{a}\mathfrak{a}, \mathfrak{d}\mathfrak{d}\})^*\{\mathfrak{a}, \mathfrak{d}\}$. We have already described one of its prolongations and an $\mathfrak{s}$-$\mathfrak{t}$-encoding of this prolongation. Automata for these languages with the empty word added are depicted in Figure 1.

The matrix $M_{\mathfrak{s}}$, $M_{\mathfrak{t}}$ and M are

$$M_{\mathfrak{s}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}; M_{\mathfrak{t}} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \text{ and } M = \begin{pmatrix} 0 & -1 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

We use the method described in Section 4.4.4, and define matrices

$$C = M_{\mathfrak{s}} - M_{\mathfrak{t}} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \text{ and } D = M_{\mathfrak{s}} + M_{\mathfrak{t}} = \begin{pmatrix} 0 & 2 \\ 1 & 0 \end{pmatrix}.$$

We compute

$$CD = \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix} \text{ and } DC = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix}.$$

There are now two options. Either we reduce the matrix M into a triangular or diagonal form using Proposition 18, or we compute the exponential of the matrix by using (71). We choose the second option and compute

$$F_1(z) = \begin{pmatrix} 1 & 0 \\ 0 & \cosh(\sqrt{2}z) \end{pmatrix}; \quad F_2(z) = \begin{pmatrix} \cosh(\sqrt{2}z) & 0 \\ 0 & 1 \end{pmatrix};$$

$$F_3(z)C = \begin{pmatrix} 0 & 0 \\ \frac{\sqrt{2}}{2}\sinh(\sqrt{2}z) & 0 \end{pmatrix} \text{ and } DF_4(z) = \begin{pmatrix} 0 & \sinh(\sqrt{2}z) \\ 1 & 0 \end{pmatrix}.$$

We deduce that,

$$E(z) = P^{\top} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cosh(\sqrt{2}z) & \frac{\sqrt{2}}{2}\sinh(\sqrt{2}z) & 0 \\ 0 & \sinh(\sqrt{2}z) & \cosh(\sqrt{2}z) & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} P$$

which yields after straightforward computations: $E(z) = \begin{pmatrix} E_1(z) & E_2(z) \\ E_2(-z) & E_1(-z) \end{pmatrix}$ with

$$E_1(z) = \begin{pmatrix} \frac{1}{2}\cosh\left(\sqrt{2}z\right) + \frac{1}{2} & -\frac{1}{2}\sinh\left(\sqrt{2}z\right) \\ -\frac{1}{4}\sqrt{2}\sinh\left(\sqrt{2}z\right) - \frac{1}{2} & \frac{1}{2}\cosh\left(\sqrt{2}z\right) + \frac{1}{2} \end{pmatrix}$$

and

$$E_2(z) = \begin{pmatrix} -\frac{1}{2}\cosh\left(\sqrt{2}z\right) + \frac{1}{2} & -\frac{1}{2}\sinh\left(\sqrt{2}z\right) \\ \frac{1}{4}\sqrt{2}\sinh\left(\sqrt{2}z\right) - \frac{1}{2} & \frac{1}{2}\cosh\left(\sqrt{2}z\right) - \frac{1}{2} \end{pmatrix}.$$

The vector of VGF is given by $\mathbf{T}(z) = [E_1(z)]^{-1}[I - E_2(z)]\mathbf{F}$. After some elementary computations[9], we obtain:

$$\mathbf{T}(z) = \begin{pmatrix} \frac{2-\sqrt{2}z+(2+\sqrt{2}z)e^{\sqrt{2}z}}{2+\sqrt{2}z-(2-\sqrt{2}z)e^{\sqrt{2}z}} \\ \frac{2\,z\left(e^{\sqrt{2}z}+1\right)}{2+\sqrt{2}z-(2-\sqrt{2}z)e^{\sqrt{2}z}} \end{pmatrix}.$$

To get $T_{L^{(\mathrm{run})}}$, we subtract 1 from the first coordinate of $\mathbf{T}(z)$ and get the answer announced in (4):

$$T_{L^{(\mathrm{run})}} = \frac{2\sqrt{2}z(e^{\sqrt{2}z} - 1)}{2 + \sqrt{2}z + (2 - \sqrt{2}z)e^{\sqrt{2}z}}.$$

By factorising the numerator and denominator by $4e^{\frac{\sqrt{2}}{2}z}$ we get the other result announced in (5):

$$T_{L^{(\mathrm{run})}}(z) = \frac{\sqrt{2}z\sinh(\frac{\sqrt{2}}{2}z)}{\cosh(\frac{\sqrt{2}}{2}z) - \frac{\sqrt{2}}{2}z\sinh(\frac{\sqrt{2}}{2}z)} = 2\frac{\frac{\sqrt{2}}{2}z\tanh(\frac{\sqrt{2}}{2}z)}{1 - \frac{\sqrt{2}}{2}z\tanh(\frac{\sqrt{2}}{2}z)} = 2f\left(\frac{\sqrt{2}}{2}z\right)$$

with $f(X) = \frac{1}{1-X\tanh(X)} - 1$.

We can find numerically the convergence radius of f. We set $X = z/\sqrt{2}$, find the smallest root of $1 - X\tanh(X)$ and multiply it by $\sqrt{2}$. We obtain that $\mathtt{Rconv}(T_{L^{(\mathrm{run})}}) \approx 1.6966$.

[9] We used the computer algebra software Sage [S+15] and the simplification method of Sympy [Sym14].

5 Uniform random sampling

In this Section we propose three methods for the problem of random sampling of permutations in a regular class. The first algorithm is based on a discrete recursive method for sampling (Section 5.1). We state in Theorem 8, that sampling permutations in a regular class reduces to sampling timed words in the corresponding language of signatures. We then present two methods for generating timed words uniformly at random: a continuous recursive method in Section 5.2, and an extension of the Boltzmann sampling method to timed languages in Section 5.3.

5.1 A discrete recursive method

In this section as in Section 3.2 and 3.3 we consider an arbitrary regular language L recognised by an automaton $\mathcal{A} = (\{\mathfrak{a}, \mathfrak{d}\}, Q, q_0, F, \delta)$.

Building a sampling algorithm based on the recursive method is done in three main steps: (i) find a recursive characterisation of the class of objects to sample; (ii) write corresponding recursive equations on cardinalities; and (iii) turn them into discrete probability distribution. For instance, the set equation

$$\Lambda_n([\mathcal{A}_q]) = \bigcup_{k=1}^{n} \Lambda_{n,k}([\mathcal{A}_q])$$

gives the cardinality equation (21) recalled here:

$$\alpha_n([\mathcal{A}_q]) = \sum_{k=1}^{n} \alpha_{n,k}([\mathcal{A}_q]).$$

Dividing both sides by $\alpha_n([\mathcal{A}_q])$ gives the equation of a discrete probability distribution:

$$1 = \sum_{k=1}^{n} \frac{\alpha_{n,k}([\mathcal{A}_q])}{\alpha_n([\mathcal{A}_q])}.$$

Hence the initial distribution for choosing the first element k of a random permutation of $\Lambda_n([\mathcal{A}_q])$ is given by:

$$[\texttt{weight-init}]_k = \alpha_{n,k}([\mathcal{A}_{q_0}])/\alpha_n([\mathcal{A}_{q_0}])$$

The other distributions needed in the algorithm are:

$$[\texttt{weight}_{q,m,k}]_i = \begin{cases} \alpha_{m,i}([\mathcal{A}_{q.\mathfrak{a}}])/\alpha_{m+1,k}([\mathcal{A}_q]) \text{ if } k \leq i \leq m \\ \alpha_{m,i}([\mathcal{A}_{q.\mathfrak{d}}])/\alpha_{m+1,k}([\mathcal{A}_q]) \text{ if } 1 \leq i \leq k-1 \end{cases}$$

In the following algorithm, the function $\texttt{Pop}(E, k)$ returns the kth element of the set E with side effect of removing it from E. Using a standard data structure (like a self balancing binary search tree[10]) this operation can be done in time $O(\log n)$ with a creation of the structure in time $O(n)$ where n is the initial number of elements in E.

[10] In fact, simple binary search tree suffices starting with a tree of height $O(\log n)$. Indeed deletion can be achieved at a cost of the height of the tree and without increasing it so that it remains bounded by $O(\log n)$.

Algorithm 3 A discrete recursive method for uniform random generation of a permutation of fixed length n within a regular class of permutation $\Lambda(L)$.

Require: The weight vectors $\texttt{weight}_{q,m.k}$ for $q \in Q$ and $k \leq m \leq n$ are precomputed.

```
q ← q0; E ← [n];
Pick randomly k according to weight vector weight-initial;
σ(1) ← Pop(E, k);
for j from 2 to n do
    Pick randomly i according to weight vector weight_{q,n+1−j,k};
    σ(j) ← Pop(E, i);
    if i ≥ k then
        q ← q.𝔞;
    else
        q ← q.𝔡;
    end if
    k ← i;
end for
```

Theorem 7 *Permutations generated by Algorithm 3 are uniformly distributed among the permutations of size n with a signature in L. Time and space complexity are discussed below.*

There can be a trade-off between the complexity of the algorithm and that of its pre-computation. All the complexity results given below are in terms of arithmetic operations.

- Computing and storing in a table all the $[\texttt{weight}_{q,m,k}]_i$ have a complexity $O(n^3|Q|)$. After that pre-computation, each generation is in time $O(n \log n)$.
- Alternatively one can compute and store only the coefficients $\alpha_{m,k}([\mathcal{A}_q])$ with a complexity $O(n^2|Q|)$. Then during the generation there are n distributions $\texttt{weight}_{q,m,k}$ to compute each one at a cost of $O(n)$ operations; the complexity of generating a permutation becomes $O(n^2)$.
- A third option is to change the sampling algorithm to mimic the locality of (24), (25), (26). Now the generation is made with a lot of small steps whose number varies randomly. For instance the sequence of choices

$$\alpha_{5,4}([\mathcal{A}_q]) \to \alpha_{5,4}(\mathfrak{d}[\mathcal{A}_q]) \to \alpha_{5,3}(\mathfrak{d}[\mathcal{A}_q]) \to \alpha_{5,2}(\mathfrak{d}[\mathcal{A}_q]) \to \alpha_{4,1}([\mathcal{A}_q])$$

  done with probability

$$\frac{\alpha_{5,4}(\mathfrak{d}[\mathcal{A}_q])}{\alpha_{5,4}([\mathcal{A}_q])} \frac{\alpha_{5,3}(\mathfrak{d}[\mathcal{A}_q])}{\alpha_{5,4}(\mathfrak{d}[\mathcal{A}_q])} \frac{\alpha_{5,2}(\mathfrak{d}[\mathcal{A}_q])}{\alpha_{5,3}(\mathfrak{d}[\mathcal{A}_q])} \frac{\alpha_{5,1}([\mathcal{A}_q])}{\alpha_{5,2}(\mathfrak{d}[\mathcal{A}_q])} = \frac{\alpha_{4,1}([\mathcal{A}_q])}{\alpha_{5,4}([\mathcal{A}_q])}$$

  would correspond to only one descent from 4 to 1. The complexity of the generation of a permutation can still be upper-bounded in the worst case by $O(n^2)$.
- If we want to save memory, we can compute the coefficients needed when required during the generation. The overall space complexity is $O(n|Q|)$ and the time complexity of each generation becomes $O(n^3|Q|)$.

Algorithm 4 Recursive uniform sampler of timed words

Require: The volume functions $\mathrm{vc}_{q,m}$ for $q \in Q$ and $m \leq n$ are precomputed.

$x_0 \leftarrow 0$; $q_0 \leftarrow$ initial state;
for $k = 1$ to n **do**
  Define $p_{\mathfrak{s}} = \int_{x_{k-1}}^{1} \mathrm{vc}_{q_{k-1}.\mathfrak{s},n-k}(y)dy / \mathrm{vc}_{q_{k-1},n-(k-1)}(x_{k-1})$;
  Pick randomly between $\mathfrak{s}$ and $\mathfrak{t}$ with probability $p_{\mathfrak{s}}, 1 - p_{\mathfrak{s}}$;
  if $\mathfrak{s}$ has been chosen **then**
    $w_k \leftarrow \mathfrak{s}$; $q_k \leftarrow q_{k-1}.\mathfrak{s}$;
    $x_k \leftarrow$ a random number in $[x_{k-1}, 1]$ picked according to the probability density function

$$\mathrm{vc}_{q_k,n-k}(y) / \int_{x_{k-1}}^{1} \mathrm{vc}_{q_k,n-k}(y)dy;$$

    $t_k \leftarrow x_k - x_{k-1}$
  else
    $w_k \leftarrow \mathfrak{t}$; $q_k \leftarrow q_{k-1}.\mathfrak{t}$;
    $x_k \leftarrow$ a random number in $[0, 1 - x_{k-1}]$ picked according to the probability density function

$$\mathrm{vc}_{q_k,n-k}(y) / \int_{0}^{1-x_{k-1}} \mathrm{vc}_{q_k,n-k}(y)dy;$$

    $t_k \leftarrow x_k$
  end if
end for
return $(t_1, w_1)(t_2, w_2) \ldots (t_n, w_n)$

5.2 A continuous recursive method

In the previous section we have seen how to turn recursive equations on discrete sets of permutations and their corresponding equations on cardinalities into a random sampler.

One can alternatively turn a system of equations on timed languages and its corresponding system of equations on volume functions into a random sampler of timed words (Algorithm 4). Then using the volume preserving transformation of Theorem 2 and a sorting algorithm one can generate permutations as wanted.

Theorem 8 *Let $L \subseteq \{\mathfrak{a}, \mathfrak{d}\}^*$ and $\mathbb{L}''$ be the timed semantics of an $\mathfrak{s}$-$\mathfrak{t}$-encoding of type l (for some $l \in \{\mathfrak{a}, \mathfrak{d}\}$) of a prolongation of L. The following algorithm achieves a uniform sampling of permutations in $\Lambda_n(L)$.*

1. *Choose uniformly an n-length timed word $(\mathbf{t}, w) \in \mathbb{L}''_n$ using Algorithm 4;*
2. *Return $\Pi(\phi_{\mathtt{st}_l^{-1}(w)}(\mathbf{t}))$.*

Theorem 9 *Algorithm 4 is a uniform sampler of timed words of $\mathbb{L}''_n$, that is for every volume measurable subset $A \subseteq \mathbb{L}''_n$, the probability that the returned timed word belongs to A is $\mathtt{Vol}(A)/\mathtt{Vol}(\mathbb{L}''_n)$.*

Some comments about Algorithm 4. Picking a random real number according to a probability density function (PDF) p can be done using the so-called inverse transform sampling. To sample a random variable according to a PDF $p : [0, 1] \to \mathbb{R}^+$ it suffices to uniformly sample a random number in $[0, 1]$ and define t such that $\int_0^t p(y)dy = r$. This equation can be solved numerically and efficiently with a controlled error using a numerical scheme such as the Newton's method. The latter integral is known as the cumulative density function (CDF) associated to p. The CDF used in this algorithm are polynomials that can be pre-computed in the same time as the volume functions. An implementation of Algorithm 4 as well as that described in Theorem 8 is available on-line http://www.liafa.univ-paris-diderot.fr/~nbasset/sage/sage.htm.

Proof (of Theorem 9) One can first check that for all $k \in [n]$, $[(q_{k-1}, x_{k-1}) \xrightarrow{(t_k,w_k)} (q_k, x_k)] \in \mathcal{L}(w_k)$ and hence that $w_1 \cdots w_n \in L''$.

We now show that during the kth loop (t_k, w_k) is chosen with density of probability $\frac{v_{q_k,n-k}(x_k)}{v_{q_{k-1},n-(k-1)}(x_{k-1})}$. Indeed, this implies that the density of probability to choose $(t_1, w_1)\cdots(t_n, w_n) \in \mathbb{L}''$ is $\prod_{k=1}^{n} \frac{\mathrm{vc}_{q_k,n-k}(x_k)}{\mathrm{vc}_{q_{k-1},n-(k-1)}(x_{k-1})} = \frac{\mathrm{vc}_{q_n,0}(x_n)}{\mathrm{vc}_{q_0,n}(0)} = \frac{1}{\mathtt{Vol}(\mathbb{L}''_n)}$ which means that the sampling is uniform.

During the kth loop w_k is set to $\mathfrak{s}$ with probability $p_{\mathfrak{s}} = \frac{\int_{x_{k-1}}^{1} \mathrm{vc}_{q_{k-1}\cdot\mathfrak{s},n-k}(y)dy}{\mathrm{vc}_{q_{k-1},n-(k-1)}(x_{k-1})}$, after that, t_k is fixed when $x_k = x_{k-1} + t_k$ is chosen with density of probability $\mathrm{vc}_{q_k,n-k}(x_k)/\int_{x_{k-1}}^{1} \mathrm{vc}_{q_k,n-k}(x_k)dy$. Hence (t_k, w_k) is chosen with the expected density of probability

$$\frac{\int_{x_{k-1}}^{1} \mathrm{vc}_{q_{k-1}.\mathfrak{s},n-k}(y)dy}{\mathrm{vc}_{q_{k-1},n-(k-1)}(x_{k-1})} \frac{\mathrm{vc}_{q_k,n-k}(x_k)}{\int_{x_{k-1}}^{1} \mathrm{vc}_{q_k,n-k}(y)dy} = \frac{\mathrm{vc}_{q_k,n-k}(x_k)}{\mathrm{vc}_{q_{k-1},n-(k-1)}(x_{k-1})}.$$

The case where $w_k = \mathfrak{t}$ can be proved in a similar manner using the fact that the probability that w_k is a turn is $1 - p_{\mathfrak{s}} = \frac{\int_{0}^{1-x_{k-1}} \mathrm{vc}_{q_{k-1}\cdot\mathfrak{t},n-k}(y)dy}{\mathrm{vc}_{q_{k-1},n-(k-1)}(x_{k-1})}$. □

5.3 Boltzmann sampling

A drawback of the recursive method for sampling described in the two previous sections is the pre-computations that take at least a quadratic time. When we only require approximate size of the objects to sample and when the generating function is available we can use the Boltzmann sampling method [DFLS04]. This latter method usually generates objects of a far larger size than the recursive method.

The Boltzmann sampling method has been created to generate discrete objects in combinatorial classes, like trees or words. Here, we extend this method to timed languages, the generated objects being timed words. Hence, we consider probability density function rather than discrete probability distribution.

We call *probability density function* (PDF) on a timed language $\mathbb{L}$, every non-negative function $p : \mathbb{L} \to \mathbb{R}$ such that

$$\sum_{w\in\Sigma^*} \int_{\mathbf{t}\in\mathbb{L}_w} p(\mathbf{t}, w) = 1,$$

and such that for every w, the function $\mathbf{t} \mapsto p(\mathbf{t}, w)$ is Lebesgue measurable.

The *Boltzmann model* of parameter z for $\mathbb{L}$ is a PDF on $\mathbb{L}$ denoted by $p_{\mathbb{L},z}$ and defined by $p_{\mathbb{L},z}(\mathbf{t}, w) = \frac{z^{|w|}}{T_{\mathbb{L}}(z)}$ for all $(\mathbf{t}, w) \in \mathbb{L}$. We denote by $\mathbb{P}_{\mathbb{L},z}$ the corresponding probability measure.

The length of random elements distributed according to the Boltzmann model is a random variable that we denote by N (as in [DFLS04]). It is distributed as follows:

$$\mathbb{P}_{\mathbb{L},z}(N=n) = \sum_{w\in L_n} \int_{\mathbf{t}\in\mathbb{L}_w} p_{\mathbb{L},z}(\mathbf{t},w) = \mathtt{Vol}(\mathbb{L}_n)\frac{z^n}{T_{\mathbb{L}}(z)}. \quad (74)$$

This is Equation (2.3) of [DFLS04] where we replace cardinality coefficient C_n by volume coefficient $\mathtt{Vol}(\mathbb{L}_n)$. This allows us to use results from classical Boltzmann sampling. In particular, the expected length of a timed word distributed according to $p_{\mathbb{L},z}$ is $\frac{z}{T_{\mathbb{L}}(z)}\frac{\partial T_{\mathbb{L}}(z)}{\partial z}$.

We call *Boltzmann sampler* of parameter z for a timed language $\mathbb{L}$, an algorithm that generates timed words according to the corresponding Boltzmann model $p_{\mathbb{L},z}$.

Remark 3 As in the classical setting of Boltzmann sampling [DFLS04], a Boltzmann sampler of a given parameter generates timed words of various length. Nevertheless all the timed words of a given length n have all the same density of probability to be chosen $\frac{z^n}{T_{\mathbb{L}}(z)}$. Hence if we want to randomly generate in a uniform manner timed words of a target length n, we just have to draw timed words with a Boltzmann sampler until one has the target length. To minimise the expected time complexity we have to choose the parameter z such that the expected length $\frac{z}{T_{\mathbb{L}}(z)}\frac{\partial T_{\mathbb{L}}(z)}{\partial z}$ equals the target length n. The number of samples one has to draw before obtaining one of a target length is already thoroughly studied in [DFLS04]. This number is at most linear in the target length.

The next theorem and Algorithm 5 deal with Boltzmann samplers for the timed semantics of automata $\mathcal{B} = (\{\mathfrak{s},\mathfrak{t}\}, Q, q_0, F, \delta)$ that recognise languages of words of $\{\mathfrak{s},\mathfrak{t}\}^*$ as described at the end of Section 3. In particular, we keep the same notations, for instance denoting by $[\mathcal{B}_q](x)$ the timed language starting from a state q and a clock x. The Boltzmann sampler described here can be adapted to fit the order sets considered in the first approach (Section 3).

Theorem 10 *Algorithm 5 describes* $\mathtt{Boltz}_z(q,x)$*, a Boltzmann sampler of parameter* z *for* $[\mathcal{B}_q](x)$*. In particular,* $\mathtt{Boltz}_z(q_0,0)$ *is a Boltzmann sampler of parameter* z *for* $\mathbb{L}'' = [\mathcal{B}_{q_0}](x)$*. It uses a linear number (in the length of the output word) of random picks according to single dimensional probability density functions.*

Proof It is easy to see that the timed words returned are in the required language. To complete the proof, it suffices to show the following statement by induction on the natural number n: "if a timed word of length n is returned, then it has a density of probability $z^n/\mathrm{VC}_q(x,z)$ to be returned". The base case is satisfied as ε is returned with probability $1/\mathrm{VC}_q(x,z)$ iff $\varepsilon \in [\mathcal{B}_q](x)$.

We assume the property to hold for a length $n \in \mathbb{N}$. We consider a timed word of length $n+1$ returned by $\mathtt{Boltz}_z(q,x)$. Then it is either returned in Line 7 or Line 10. We consider only the former case, as the other case is treated in a similar way. The returned timed word is of the form $(y-x,\mathfrak{s})\omega$, with ω a timed word returned by $\mathtt{Boltz}_z(q.\mathfrak{s},y)$. By induction hypothesis, ω was chosen with density of probability $\frac{z^n}{\mathrm{VC}_{q.\mathfrak{s}}(y,z)}$. Moreover the letter $\mathfrak{s}$ was chosen with probability $\frac{z\int_x^1 \mathrm{VC}_{q.\mathfrak{s}}(y,z)dy}{\mathrm{VC}_q(x,z)}$ in Line 1 and then y was chosen with probability $\frac{\mathrm{VC}_{q.\mathfrak{s}}(y,z)dy}{\int_x^1 \mathrm{VC}_{q.\mathfrak{s}}(y,z)dy}$ in Line 6. The density of probability to choose $(y-x,\mathfrak{s})\omega$ is hence as expected:

$$\frac{z^n}{\mathrm{VC}_{q.\mathfrak{s}}(y,z)}\,\frac{\mathrm{VC}_{q.\mathfrak{s}}(y,z)}{\int_x^1 \mathrm{VC}_{q.\mathfrak{s}}(y,z)dy}\,\frac{z\int_x^1 \mathrm{VC}_{q.\mathfrak{s}}(y,z)dy}{\mathrm{VC}_q(x,z)} = \frac{z^{n+1}}{\mathrm{VC}_q(x,z)}.$$

Algorithm 5 The Boltzmann sampler $\mathtt{Boltz}_z(q,x)$.

Pick randomly ε, $\mathfrak{s}$ or $\mathfrak{t}$ with weights $\frac{1_{q\in F}}{\mathrm{VC}_q(x,z)}$, $\frac{z\int_x^1 \mathrm{VC}_{q.\mathfrak{s}}(y,z)dy}{\mathrm{VC}_q(x,z)}$ and $\frac{z\int_0^{1-x} \mathrm{VC}_{q.\mathfrak{t}}(s,z)dt}{\mathrm{VC}_q(x,z)}$;
if ε has been chosen **then**
  return ε
end if
if $\mathfrak{s}$ has been chosen **then**
  pick $y \in [x,1]$ with density of probability $y \mapsto \mathrm{VC}_{q.\mathfrak{s}}(y,z)dy / \int_x^1 \mathrm{VC}_{q.\mathfrak{s}}(y,z)dy$;
  return $(y-x,\mathfrak{s})\mathtt{Boltz}_z(q.\mathfrak{s},y)$
else
  pick $y \in [0,1-x]$ with density of probability $y \mapsto \mathrm{VC}_{q.\mathfrak{t}}(y,z)dy / \int_0^{1-x} \mathrm{VC}_{q.\mathfrak{t}}(y,z)dy$;
  return $(y,\mathfrak{t})\mathtt{Boltz}_z(q.\mathfrak{t},y)$
end if

Some comments about Algorithm 5. The inverse function of a CDF is known as a quantile function. This is the function evaluated during the inverse sampling method mentioned in the paragraph that comments Algorithm 4. When the parameter z is fixed, there are only $O(|Q|)$ such functions to evaluate. Hence it could be worth pre-computing numerically and tabulating the quantile functions to speed up the running time of the Boltzmann sampler.

5.3.1 Experiments

We have implemented Algorithm 5 with Sage. The code as well as experiments are available on-line: `http://www.liafa.univ-paris-diderot.fr/~nbasset/sage/sage.htm`.

The generating functions $V_q(x,z)$ and $V_q(z)$ are encoded as symbolic expressions in variables z and x. We use the inverse sampling method as described above after having precomputed the CDF, and we use the function findroot of Sage to find the root X of equation of the form

$$\int_x^X \mathrm{VC}_{q.\mathfrak{s}}(y,z)dy / \int_x^1 \mathrm{VC}_{q.\mathfrak{s}}(y,z)dy - r = 0$$

and

$$\int_0^X \mathrm{VC}_{q.\mathfrak{t}}(y,z)dy / \int_0^{1-x} \mathrm{VC}_{q.\mathfrak{t}}(y,z)dy - r = 0$$

where r is a number uniformly drawn at random in $[0,1]$.

We illustrate our implementation on the running example. As stated at the end of Section 4.4.5, the convergence radius of $\mathbf{T}(z)$ is approximately 1.6966. Fixing

[12] Omitted details of the experiments show that despite its theoretical complexity $\Theta(n\log n)$, the execution time of the sort is negligible compared to the execution time of the sampling.

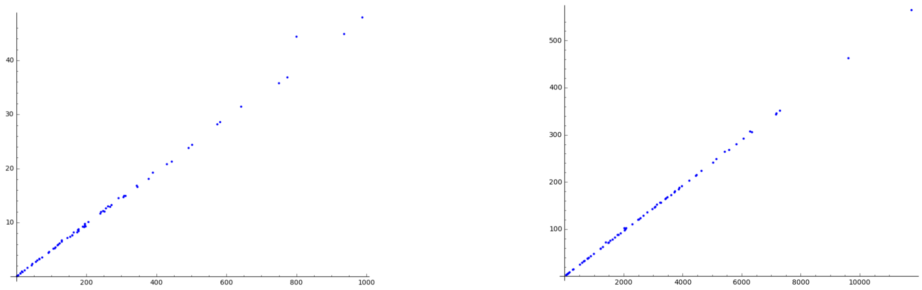

Fig. 5 Distribution of lengths (and running time), for generating permutations in the regular class $\Lambda\left[(\{\mathfrak{a}\mathfrak{a},\mathfrak{d}\mathfrak{d}\})^*\{\mathfrak{a},\mathfrak{d}\}\right]$ using Boltzmann sampling of timed words (Algorithm 5) with parameter $z_1 = 1.69$ (Left) and $z_2 = 1.696$ (Right). Each point is of the form (n,t) with n the length of the permutation generated, and t the time (in seconds) that took the generation of the timed word as well as the computation of the corresponding permutation (described in Theorem 8). One can see that with parameter z_2 the Boltzmann sampler generates permutations of an approximately 10 times larger length than with parameter z_2. This agrees with the fact that the theoretical expected length 2819 for parameter z_2 is approximately 10 times larger than the theoretical expected length 257 for parameter z_1. The experimental expected length for these sets of 71 samples are 245 for parameter z_1 and 2936 for parameter z_2. The execution time is visibly linear[12]with respect to the length of permutation generated for both parameters. The slopes in both figures are approximately equal to 20 meaning that when the algorithm returns a permutation in t seconds, its length is approximately $20t$.

the parameter $z_1 \stackrel{\text{def}}{=} 1.69$, gives an expected length of timed word $\frac{z_1}{T_{\mathbb{L}}(z_1)}\frac{\partial T_{\mathbb{L}}(z_1)}{\partial z_1} \approx 257$. Fixing the parameter $z_2 \stackrel{\text{def}}{=} 1.696$, gives an expected length of timed word $\frac{z_2}{T_{\mathbb{L}}(z_2)}\frac{\partial T_{\mathbb{L}}(z_2)}{\partial z_2} \approx 2819$. We have run 71 times Algorithm 5 for each of the parameters above. The running time and distribution of lengths are shown in Figure 5.

6 Discussion, perspectives and further related works

We have stated and solved the problems of counting and uniform sampling of permutations with signature in a given regular language of signatures. The timed semantics of such a language is a particular case of regular timed languages (i.e. recognized by timed automata [AD94]). However, with the approach used, timed languages can be defined from any kind of languages of signatures. A challenging task for us is to treat the case of context free languages. For this we should use, as in [ABDP12], volume of languages parametrized both by starting and ending states. This latter form of volume functions would also be needed to extend the divide and conquer random sampling algorithm of [BG12] to our continuous setting.

Our work can also benefit timed automata research. Indeed, we have proposed uniform samplers for a particular class of timed languages. An ongoing work is to adapt this algorithm to all deterministic timed automata with bounded clocks using recursive equations of [ABD15].

A well known fact is that among all the signatures u of a given length n the one that maximises α_u (the number of permutations with signature u) are the alternating permutations $\mathfrak{a}\mathfrak{d}\mathfrak{a}$... and $\mathfrak{d}\mathfrak{a}\mathfrak{d}$.... This corresponds to the words $\mathfrak{t}^n$ (and $\mathfrak{s}\mathfrak{t}^{n-1}$ depending on the type of the $\mathfrak{s}$-$\mathfrak{t}$-encoding). Similar questions can be asked for regular classes of permutations. For instance, we could study the expected number of turns in signatures of permutations generated at random in a given regular class. We think that we could answer this type of question by distinguishing between $z_{\mathfrak{s}}$ and $z_{\mathfrak{t}}$ in Equation (58). The new equation would be

$$\mathbf{VC}(x, z_{\mathfrak{s}}, z_{\mathfrak{t}}) = z_{\mathfrak{s}} M_{\mathfrak{s}} \int_x^1 \mathbf{VC}(y, z_{\mathfrak{s}}, z_{\mathfrak{t}}) dy + z_{\mathfrak{t}} M_{\mathfrak{t}} \int_0^{1-x} \mathbf{VC}(y, z_{\mathfrak{s}}, z_{\mathfrak{t}}) dy + \mathbf{F}. \quad (75)$$

With such equations, we could adapt the Boltzmann sampling framework for multi-dimensional generating function, proposed in [BP10]. In contrast to ours, this work does not consider uncountable unions parametrised by an auxiliary variable (called x here).

We used Boltzmann sampling on examples for which we have computed a closed form formula for the vector of generating functions. We want to study the problem of evaluating numerically the generating functions (and the quantile functions). For this we would like to adapt methods of [PSS12].

Acknowledgment

We thank Jean-Marc Luck and Philippe Marchal for fruitful discussions during the event ALEA 2014. In particular, we acknowledge Philippe Marchal for sharing the remark that, for the classical case of single signature, discrete recursive methods for sampling can be derived from discrete recursive equations on cardinalities.

We acknowledge James Worrell and Joël Ouaknine for sharing their knowledge on exponential polynomials and algebraic number theory needed in Section 4.4.2.

Last but not least, we acknowledge Eugène Asarin for his very helpful comments on preliminary versions of this work.

References

ABD15. Eugene Asarin, Nicolas Basset, and Aldric Degorre. Entropy of regular timed languages. *Information and Computation*, 241:142–176, 2015.

ABDP12. Eugene Asarin, Nicolas Basset, Aldric Degorre, and Dominique Perrin. Generating functions of timed languages. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 124–135. Springer, 2012.

AD94. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

Bas13. Nicolas Basset. *Volumetry of timed languages and applications.* PhD thesis, Université Paris-Est, 2013.

Bas14. Nicolas Basset. Counting and generating permutations using timed languages. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, volume 8392 of *Lecture Notes in Computer Science*, pages 502–513. Springer, 2014.

BG12. Olivier Bernardi and Omer Giménez. A linear algorithm for the random sampling from regular languages. *Algorithmica*, 62(1-2):130–145, 2012.

BP02. Patricia Bouyer and Antoine Petit. A Kleene/Büchi-like theorem for clock languages. *Journal of Automata, Languages and Combinatorics*, 7(2):167–186, 2002.

BP10. Olivier Bodini and Yann Ponty. Multi-dimensional Boltzmann sampling of languages. *DMTCS Proceedings*, (01):49–64, 2010.

Cai94. Jin-yi Cai. Computing Jordan normal forms exactly for commuting matrices in polynomial time. *International Journal of Foundations of Computer Science*, 5(3/4):293–302, 1994.

CLRS09. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

DB70. NG De Bruijn. Permutations with given ups and downs. *Nieuw Archief voor Wiskunde*, 18(3):61–65, 1970.

DFLS04. Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4-5):577–625, 2004.

DZ99. Alain Denise and Paul Zimmermann. Uniform random generation of decomposable structures using floating-point arithmetic. *Theoretical Computer Science*, 218(2):233–248, 1999.

EJ12. Richard Ehrenborg and Jiyoon Jung. Descent pattern avoidance. *Advances in Applied Mathematics*, 49(3-5):375–390, September 2012.

EKP11. Richard Ehrenborg, Sergey Kitaev, and Peter Perry. A spectral approach to consecutive pattern-avoiding permutations. *Journal of Combinatorics*, 2(3), 2011.

EN03. Sergi Elizalde and Marc Noy. Consecutive patterns in permutations. *Advances in Applied Mathematics*, 30(1):110–125, 2003.

FS09. Philippe Flajolet and Robert Sedgewick. *Analytic combinatorics*. Cambridge University Press, 2009.

FZVC94. Philippe Flajolet, Paul Zimmerman, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1):1–35, 1994.

HL12. T. Hibi and N. Li. Unimodular equivalence of order and chain polytopes. *arXiv preprint arXiv:1208.4029*, 2012.

Kal93. K. Kalorkoti. Inverting polynomials and formal power series. *SIAM Journal on Computing*, 22(3):552–559, 1993.

Kit11. Sergey Kitaev. *Patterns in permutations and words*. Springer, 2011.

Lot05. M. Lothaire. *Applied Combinatorics on Words (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, New York, NY, USA, 2005.

Luc14. Jean-Marc Luck. On the frequencies of patterns of rises and falls. *Physica A: Statistical Mechanics and its Applications*, 407:252–275, 2014.

Mar14. Philippe Marchal. Permutations with a prescribed descent set. *preprint hal-00944244, `https://hal.archives-ouvertes.fr/hal-00944244`*, February 2014.

MVL03. Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49, 2003.

NW78. Albert Nijenhuis and Herbert S Wilf. Combinatorial algorithms for computers and calculators. *Computer Science and Applied Mathematics, New York: Academic Press, 1978, 2nd ed.*, 1, 1978.

ODG13. Johan Oudinet, Alain Denise, and Marie-Claude Gaudel. A new dichotomic algorithm for the uniform random generation of words in regular languages. *Theoretical Computer Science*, 502:165–176, 2013.

OPW15. Joël Ouaknine, João Sousa Pinto, and James Worrell. The polyhedral escape problem is decidable. *arXiv preprint arXiv:1507.03166*, 2015.

PSS12. Carine Pivoteau, Bruno Salvy, and Michele Soria. Algorithms for combinatorial structures: Well-founded systems and newton iterations. *Journal of Combinatorial Theory, Series A*, 119(8):1711–1773, 2012.

S^+15. W. A. Stein et al. *Sage Mathematics Software*. The Sage Development Team, 2015. `http://www.sagemath.org`.

Sta86. Richard P. Stanley. Two poset polytopes. *Discrete & Computational Geometry*, 1(1):9–23, 1986.

Sta10. Richard P. Stanley. A survey of alternating permutations. In *Combinatorics and graphs*, volume 531 of *Contemporary Mathematics*, pages 165–196. American Mathematical Society, Providence, RI, 2010.

Sym14. SymPy Development Team. *SymPy: Python library for symbolic mathematics*, 2014.

Vie79. G Viennot. Permutations ayant une forme donnée. *Discrete Mathematics*, 26(3):279–284, 1979.