Counting and generating permutations using timed languages (long version)

Nicolas Basset

Counting and generating permutations using timed languages* **

Nicolas Basset[1,2]

[1] LIGM, University Paris-Est Marne-la-Vallée and CNRS, France.
[2] LIAFA, University Paris Diderot and CNRS, France
nbasset@liafa.univ-paris-diderot.fr

Abstract. The signature of a permutation σ is a word $\mathtt{sg}(\sigma) \subseteq \{\mathbf{a}, \mathbf{d}\}^*$ whose i^{th} letter is $\mathbf{d}$ when σ has a descent (i.e. $\sigma(i) > \sigma(i+1)$) and is $\mathbf{a}$ when σ has an ascent (i.e. $\sigma(i) < \sigma(i+1)$). Combinatorics of permutations with a prescribed signature is quite well explored. Here we state and address the two problems of counting and randomly generating in the set $\mathtt{sg}^{-1}(L)$ of permutations with signature in a given regular language $L \subseteq \{\mathbf{a}, \mathbf{d}\}^*$. First we give an algorithm that computes a closed form formula for the exponential generating function of $\mathtt{sg}^{-1}(L)$. Then we give an algorithm that generates randomly the n-length permutations of $\mathtt{sg}^{-1}(L)$ in a uniform manner, i.e. all the permutations of a given length with signature in L are equally probable to be returned. Both contributions are based on a geometric interpretation of a subclass of regular timed languages.

Generating all the permutations with a prescribed signature (described in the abstract) or simply counting them are two classical combinatorial topics (see [17] 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 [16] for a survey). Their signatures belong to the language expressed by the regular expression $(\mathbf{da})^*(\mathbf{d} + \varepsilon)$ (i.e. they satisfy $\sigma_1 > \sigma_2 < \sigma_3 > \sigma_4 ...$).

Such a definition of a class of permutations in terms of a language of signatures is in fact a novelty of the present work. To a language $L \subseteq \{\mathbf{a}, \mathbf{d}\}^*$, we associate the class $\mathtt{sg}^{-1}(L)$ of permutations whose signature is in L. Many classes of permutations can be expressed in that way, e.g. alternating permutations, those with an even number of descents.

We state and address the two problems of counting and randomly generating when the language of signatures is regular. We propose Algorithm 1 which takes as its input a regular language L and returns a closed form formula for the exponential generating function (EGF) of $\mathtt{sg}^{-1}(L)$ i.e. a formal power series $\sum a_n \frac{z^n}{n!}$ where the n^{th} coefficient a_n counts the permutations of length n with

* The support of Agence Nationale de la Recherche under the project EQINOCS (ANR-11-BS02-004) is gratefully acknowledged.

** The present paper is a long version of an article under submission to a conference.

signature in L. With such an EGF, it is easy to recover the number a_n and some estimation of the growth rate of a_n (see [9] for an overview of analytic combinatorics). The random generation is done by an algorithm described in Theorem 4. The regular language of signatures L together with n the size of permutation to generate are given in input and the output are random permutations of size n whose signatures are in L with equal probability to be returned.

Our theory is based on a geometric interpretation of regular timed languages initiated in [4]. In that paper the authors introduce the concept of volume and entropy of regular timed languages as well as recurrent equations on timed languages and their volume. With these authors we defined and characterized volume generating function of timed language in [3]. In this latter paper a link between enumerative combinatorics and regular timed languages was foreseen. Here we establish such a link. The passage from a class of permutations to a timed language is in two steps. First we associate order and chain polytopes to signatures which are particular cases of Stanley's poset polytopes [15]. Then we interpret the chain polytopes of a signature w as the set of delays which together with w forms a timed word of a well chosen timed language.

Related works. The link between geometry and permutations is not new and can be found in several articles including [8,13,17] that state integral equations similar to ours. Our use of timed languages provides a new tool necessary to catch the dynamic of the regular languages of signatures.

Particular regular languages of signatures are considered in [8] under the name of consecutive descent pattern avoidance. Numerous other works treat more general cases of (consecutive) pattern avoidance (see, e.g. the monograph [12]) 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, e.g. the permutations with an even number of descents.

The random sampler of timed words (Algorithm 2) is an adaptation to the timed case of the so-called recursive method of [14] developed by [10]. It has been improved for the particular case of generation of words in regular languages [6].

Paper structure In section 1 we expose the problem statements. In section 2 we establish the link between the classes of permutations associated with languages of signatures and timed languages of a particular form. We address the two problems in section 3, treat several examples and discuss our results and perspectives in the last section.

1 Two problem statements

All along the paper we use the two letter alphabet $\{\mathbf{a}, \mathbf{d}\}$ whose elements must be read as "ascent" and "descent". Words of $\{\mathbf{a}, \mathbf{d}\}^*$ are called signatures. For $n \in \mathbb{N}$ we denote $[n] = \{1, \ldots, n\}$ and by $\mathfrak{S}_n$ the set of permutation of $[n]$. We

also use the one line notation of permutations e.g. $\sigma = 231$ 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 $u = u_1 \cdots u_{n-1} \in \{\mathbf{a}, \mathbf{d}\}^{n-1}$ denoted by $\mathtt{sg}(\sigma)$ such that for $i \in [n]$, $\sigma_i < \sigma_{i+1}$ iff $u_i = \mathbf{a}$ (we speak of an "ascent", also known as an ascent) and $\sigma_i > \sigma_{i+1}$ iff $u_i = \mathbf{d}$ (we speak of a "descent", also known as a descent) e.g. $\mathtt{sg}(21354) = \mathtt{sg}(32451) = \mathbf{daad}$.

This notion appears in the literature under several different names and forms such as descent word, descent set, ribbon diagram, etc. The usual definition of signature of a permutation is an n-tuple of $+1$ ("ascent") and -1 ("descent"). Here we use words to express in a very convenient way constraints on permutations in terms of languages. More precisely we are interested in $\mathtt{sg}^{-1}(L) = \{\sigma \mid p(\sigma) \in L\}$: the class of permutations with a signature in $L \subseteq \{\mathbf{a}, \mathbf{d}\}^*$. Given a language L we denote by L_n the sub-language of L restricted to words of length n. The exponential generating function of $\mathtt{sg}^{-1}(L)$ is

$$EGF[\mathtt{sg}^{-1}(L)](z) = \sum_{\sigma \in \mathtt{sg}^{-1}(L)} \frac{z^{|\sigma|}}{|\sigma|!} = \sum_{n \geq 1} |\mathtt{sg}^{-1}(L_{n-1})| \frac{z^n}{n!} = \sum_{u \in L} |\mathtt{sg}^{-1}(u)| \frac{z^{|u|+1}}{(|u|+1)!}.$$

Example 1. Consider as a running example of the paper the class of "up-up-down-down" permutations with signature in the language[3] $L^{ex} = (\mathbf{aadd})^*(\mathbf{aa} + \varepsilon)$ recognized by the automaton depicted in the left of Figure 1. The theory developed in the paper permits to find the exponential generating function of $\mathtt{sg}^{-1}(L^{ex})$.

$$EGF[\mathtt{sg}^{-1}(L^{ex})](z) = \frac{\sinh(z) - \sin(z) + \sin(z)\cosh(z) + \sinh(z)\cos(z)}{1 + \cos(z)\cosh(z)}.$$

Its Taylor expansion is

$$z + \frac{z^3}{3!} + 6\frac{z^5}{5!} + 71\frac{z^7}{7!} + 1456\frac{z^9}{9!} + 45541\frac{z^{11}}{11!} + 2020656\frac{z^{13}}{13!} + \ldots.$$

For instance, there are 1456 up-up-down-down permutations of length 9.

Now we state the two problems solved in this paper.

Problem 1. Design an algorithm which takes as input a regular language $L \subseteq \{\mathbf{a}, \mathbf{d}\}^*$ and returns a closed form formula for $EGF(\mathtt{sg}^{-1}(L))$

Problem 2. Design an algorithm which takes as input a regular language $L \subseteq \{\mathbf{a}, \mathbf{d}\}^*$ and a positive integer n and returns a random permutation σ uniformly in $\mathtt{sg}^{-1}(L_{n-1})$ i.e. such that the probability for each $\sigma \in \mathtt{sg}^{-1}(L_{n-1})$ to be returned is $1/|\mathtt{sg}^{-1}(L_{n-1})|$.

[3] We confuse regular expressions with the regular languages they express.

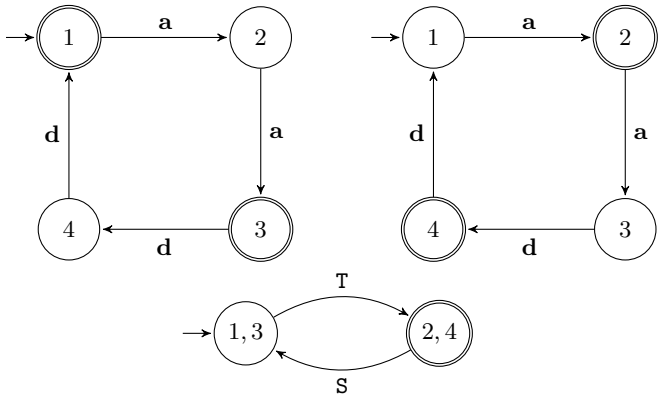

Fig. 1. From left to right: automata for L^{ex}, $L^{ex'}$ and $\mathtt{st}_{\mathbf{d}}(L^{ex'})$

2 A timed and geometric approach

In section 2.1 we introduce a sequence of sets $\mathcal{O}_n(L) \subseteq [0,1]^n$ and see how the two problems posed can be reformulated as computing the volume generating function of the sequence $(\mathcal{O}_n(L))_{n\geq 1}$ and generating points uniformly in $\mathcal{O}_n(L)$. Then we define a timed language $\mathbb{L}'$ associated to L as well as its volume (section 2.2) and describe a volume preserving transformation between $\mathcal{O}_n(L)$ and $\mathbb{L}'_n$.

2.1 Order sets of a language of signatures $(\mathcal{O}_n(L))_{n\geq 1}$

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[4] of σ and denoted by $\mathcal{O}(\sigma)$ e.g. $\boldsymbol{\nu} = (0.3, 0.2, 0.4, 0.5, 0.1)$ belongs to $\mathcal{O}(32451)$ since $\nu_5 \leq \nu_2 \leq \nu_1 \leq \nu_3 \leq \nu_4$ and $(32451)^{-1} = 52134$. The set $O(\sigma)$ 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 $O(\sigma)$ with probability $1/n!$. To retrieve σ from $\boldsymbol{\nu}$ it suffices to use a sorting algorithm. We denote by $\mathit{\Pi}(\boldsymbol{\nu})$ the permutation σ returned by the sorting algorithm on $\boldsymbol{\nu}$ i.e. such that $0 \leq \nu_{\sigma_1^{-1}} \leq \ldots \leq \nu_{\sigma_n^{-1}} \leq 1$.

The *signature* of a vector $\boldsymbol{\nu} \in [0,1]^n$ is the word $\mathtt{sg}(\boldsymbol{\nu}) = \mathtt{sg}(\mathit{\Pi}(\nu))$ i.e. such that $\nu_i < \nu_{i+1}$ iff $u_i = \mathbf{a}$ and $\nu_i > \nu_{i+1}$ iff $u_i = \mathbf{d}$. e.g. $\mathtt{sg}(0.3, 0.2, 0.4, 0.5, 0.1) = \mathbf{daad}$. The *order polytope* [15] of a signature $u \in \{\mathbf{a}, \mathbf{d}\}^{n-1}$ is the polytope $\mathcal{O}(u) = \{\boldsymbol{\nu} \in [0,1]^n \mid \mathtt{sg}(\boldsymbol{\nu}) = u\}$. 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

[4] Order simplices, order and chain polytopes of signatures defined here are particular cases of Stanley's order and chain polytopes [15].

of the order polytope $\mathcal{O}(u)$: $\mathcal{O}(u) = \bigsqcup_{\sigma \in \mathtt{sg}^{-1}(u)} \mathcal{O}(\sigma)$ (e.g. $\mathcal{O}(\mathbf{daa}) = \mathcal{O}(2134) \sqcup \mathcal{O}(3124) \sqcup \mathcal{O}(4123)$). Passing to volume we get:

$$\mathtt{Vol}(\mathcal{O}(u)) = \sum_{\sigma \in \mathtt{sg}^{-1}(u)} \mathtt{Vol}(\mathcal{O}(\sigma)) = \frac{|\mathtt{sg}^{-1}(u)|}{n!} \tag{1}$$

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

$$\mathcal{O}_n(L) = \bigsqcup_{u \in L_{n-1}} \mathcal{O}(u) = \bigsqcup_{\sigma \in \mathtt{sg}^{-1}(L_{n-1})} \mathcal{O}(\sigma) = \{\boldsymbol{\nu} \in [0,1]^n \mid \mathtt{sg}(\boldsymbol{\nu}) \in L_{n-1}\}. \tag{2}$$

For volumes we get:

$$\mathtt{Vol}(\mathcal{O}_n(L)) = \sum_{u \in L_{n-1}} \mathtt{Vol}(\mathcal{O}(u)) = \sum_{\sigma \in \mathtt{sg}^{-1}(L_{n-1})} \mathtt{Vol}(\mathcal{O}(\sigma)) = \frac{|\mathtt{sg}^{-1}(L_{n-1})|}{n!} \tag{3}$$

Reformulating the two problems with the geometric approach As a consequence of (3), Problem 1 can be reformulated as computing the *volume generating function* (VGF) of the sequence $\mathcal{O}(L) =_{\text{def}} (\mathcal{O}_n(L))_{n \geq 1}$:

$$VGF(\mathcal{O}(L))(z) =_{\text{def}} \sum_{n \geq 1} \mathtt{Vol}(\mathcal{O}_n(L)) z^n = EGF(\mathtt{sg}^{-1}(L))(z) \tag{4}$$

Problem 2 can also be treated using order polytopes $\mathcal{O}_n(L)$. Indeed it suffices to generate uniformly a vector $\nu \in \mathcal{O}_n(L)$ and then sort it to get a permutation $\sigma = \Pi(\nu)$. As the simplices $\mathcal{O}(\sigma)$ for $\sigma \in \mathtt{sg}^{-1}(L_n)$ form an almost disjoint partition of $\mathcal{O}_n(L)$ and all these simplices have the same volume $1/n!$, they are equally probable to receive the random vector ν, and thus all $\sigma \in \mathtt{sg}^{-1}(L_n)$ have the same probability to be chosen.

We have seen with (2) that permutations of a fixed length n fit well with the n^{th} order set. However, it is not clear how to fit the sequence of order sets (when n varies) with the dynamics of the language L. It is easier to handle a timed language $\mathbb{L}$ since its sequence of volumes $(\mathtt{Vol}(\mathbb{L}_n))_{n \in \mathbb{N}}$ satisfies a recursive equation (see [4]). We will find a volume preserving transformation between order sets $\mathcal{O}_n(L)$ and timed languages $(\mathbb{L}_n)_{n \in \mathbb{N}}$ and hence reduce Problem 1 to the computing of the ordinary generating function of $(\mathtt{Vol}(\mathbb{L}_n))_{n \in \mathbb{N}}$. For the second problem, by generating uniformly a timed word in $\mathbb{L}_n$ and applying the volume preserving transformation we will get a uniform random point in $\mathcal{O}_n(L)$.

2.2 Timed semantics of a language of signatures $\left(\mathbb{L}'_{\boldsymbol{n}}\right)_{\boldsymbol{n} \in \mathbb{N}}$

This section is inspired by timed automata theory and designed for non experts. We adopt a non standard[5] and self-contained approach based on the notion of clock languages introduced by [7] and used in our previous work [3].

[5] We refer the reader to [1] for a standard approach of timed automata theory.

Timed languages, their volumes and their 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 $(t_1, \ldots, t_n) \in \mathbb{R}^n$ together with a word of events $w = w_1 \cdots w_n \in \Sigma^n$. We adopt the following convention, we write $(\boldsymbol{t}, w)$ for the timed word $(t_1, w_1) \cdots (t_n, w_n)$ with $\boldsymbol{t} = (t_1, \cdots, t_n)$ and $w \in \Sigma^n$ $(n \geq 1)$. Continuing with the same convention, given a timed language $\mathbb{L}' \subseteq (\mathbb{R}^+ \times \Sigma)^*$, then the timed language restricted to words of length n, $\mathbb{L}'_n$ can be seen as a formal union of sets $\biguplus_{w \in \Sigma^n} \mathbb{L}'_w \times \{w\}$ where $\mathbb{L}'_w = \{\boldsymbol{t} \in \mathbb{R}^n \mid (\boldsymbol{t}, w) \in \mathbb{L}'\}$ 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 volume measurable. To such a $\mathbb{L}'_n$ one can associate a sequence of volumes and a VGF as follows:

$$\mathtt{Vol}(\mathbb{L}'_n) = \sum_{w \in \Sigma^n} \mathtt{Vol}(\mathbb{L}'_w);$$

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

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 $x^{\mathbf{a}}$ and $x^{\mathbf{d}}$. A clock word is a tuple whose component are a starting clock vector $(x_0^{\mathbf{a}}, x_0^{\mathbf{d}}) \in [0,1]$, a timed word $(t_1, a_1) \cdots (t_n, a_n) \in ([0,1] \times \{\mathbf{a}, \mathbf{d}\})^*$ and an ending clock vector $(x_n^{\mathbf{a}}, x_n^{\mathbf{d}}) \in [0,1]^2$, it is denoted by $(x_0^{\mathbf{a}}, x_0^{\mathbf{d}}) \xrightarrow{(t_1,a_1)\cdots(t_n,a_n)} (x_n^{\mathbf{a}}, x_n^{\mathbf{d}})$.

Two clock words $\boldsymbol{x}_0 \xrightarrow{\mathbf{w}} \boldsymbol{x}_1$ and $\boldsymbol{x}_2 \xrightarrow{\mathbf{w}'} \boldsymbol{x}_3$ are said to be compatible if $\boldsymbol{x}_2 = \boldsymbol{x}_1$, in this case their product is $(\boldsymbol{x}_0 \xrightarrow{\mathbf{w}} \boldsymbol{x}_1) \cdot (\boldsymbol{x}_2 \xrightarrow{\mathbf{w}'} \boldsymbol{x}_3) = \boldsymbol{x}_0 \xrightarrow{\mathbf{w}\mathbf{w}'} \boldsymbol{x}_3$. A *clock language* is a set of clock words. The product of two clock languages $\mathcal{L}$ and $\mathcal{L}'$ is

$$\mathcal{L} \cdot \mathcal{L}' = \{c \cdot c' \mid c \in \mathcal{L},\ c' \in \mathcal{L}',\ c \text{ and } c' \text{ compatible}\}. \tag{5}$$

The clock languages[6] $\mathcal{L}(\mathbf{a})$ (resp. $\mathcal{L}(\mathbf{d})$) associated to an ascent (resp. a descent) is the set of clock words of the form $(x^{\mathbf{a}}, x^{\mathbf{d}}) \xrightarrow{(t,\mathbf{a})} (x^{\mathbf{a}} + t, 0)$ (resp. $(x^{\mathbf{a}}, x^{\mathbf{d}}) \xrightarrow{(t,\mathbf{d})} (0, x^{\mathbf{d}} + t)$) and such that $x^{\mathbf{a}} + t \in [0,1], x^{\mathbf{d}} + t \in [0,1]$ (and by definition of clocks and delays $x^{\mathbf{a}} \geq 0, x^{\mathbf{d}} \geq 0, t \geq 0$). These definitions extend inductively to all signatures $\mathcal{L}(u_1 \cdots u_n) = \mathcal{L}(u_1) \cdots \mathcal{L}(u_n)$ (using the product of clock languages as defined in (5)).

Example 2. $(0,0) \xrightarrow{(0.7,\mathbf{d})(0.2,\mathbf{a})(0.2,\mathbf{a})(0.5,\mathbf{d})} (0, 0.5) \in \mathcal{L}(\mathbf{daad})$ since

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

$(0,0) \xrightarrow{(0.7,\mathbf{d})} (0,0.7) \in \mathcal{L}(\mathbf{d})$; $\quad (0,0.7) \xrightarrow{(0.2,\mathbf{a})} (0.2,0) \in \mathcal{L}(\mathbf{a})$;
$(0.2,0) \xrightarrow{(0.2,\mathbf{a})} (0.4,0) \in \mathcal{L}(\mathbf{a})$; $\quad (0.4,0) \xrightarrow{(0.5,\mathbf{a})} (0,0.5) \in \mathcal{L}(\mathbf{d})$.

The timed semantics of a language of signatures. The *timed polytope* associated to a signature $w \in \{\mathbf{a},\mathbf{d}\}^*$ is $P_w =_{\text{def}} \{\boldsymbol{t} \mid (0,0) \xrightarrow{(\boldsymbol{t},w)} \boldsymbol{y} \in \mathcal{L}(w)$ for some $\boldsymbol{y} \in [0,1]^2\}$ e.g. $(0.7,0.2,0.2,0.5,0.1) \in P_{\mathbf{daada}}$. The definition of such a timed polytope will be clarified in Proposition 1 and its following example. The timed semantics of a language of signatures L' is

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

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

The *chain polytope* [15] of a signature u is the set $\mathcal{C}(u)$ of vectors $\boldsymbol{t} \in [0,1]^n$ such that for all $i < j \leq n$ and $l \in \{\mathbf{a},\mathbf{d}\}$, $w_i \cdots w_{j-1} = l^{j-i} \Rightarrow t_i + \ldots + t_j \leq 1$.

Proposition 1. *Given a word $u \in \{\mathbf{a},\mathbf{d}\}^*$ and $l \in \{\mathbf{a},\mathbf{d}\}$, the timed polytope of ul is the chain polytope of u: $P_{ul} = \mathcal{C}(u)$.*

Proof. Let $w = ul$ i.e. for all $i \in [n-1]$ $w_i = u_i$ and $w_n = l$. $P_{ul} \subseteq \mathcal{C}(u))$ Let $(t_1,\ldots,t_n) \in P_w$ i.e. there exist value of clocks x^a_k ($a \in \{\mathbf{a},\mathbf{d}\}, k \in [n]$) such that $x^{\mathbf{a}}_0 = x^{\mathbf{d}}_0 = 0$ and $(x^{\mathbf{a}}_{k-1}, x^{\mathbf{d}}_{k-1}) \xrightarrow{(t_k,w_k)} (x^{\mathbf{a}}_k, x^{\mathbf{d}}_k) \in \mathcal{L}(w_k)$. Let $i < j \leq n$ and $a \in \{\mathbf{a},\mathbf{d}\}$ such that $w_i \cdots w_{j-1} = a^{j-i}$, then for $k \in \{i,\ldots,j-1\}$, $x^a_k = x^a_{k-1} + t_k$ by definition of $\mathcal{L}(a)$. Then $x^a_{j-1} = x^a_{i-1} + t_i + \ldots + t_{j-1}$. Moreover $x^a_{j-1} + t_j \leq 1$ by definition of $\mathcal{L}(w_j)$ and thus $t_i + \ldots + t_{j-1} + t_j \leq x^a_{i-1} + t_j \leq 1$ which is the wanted inequality.
$\mathcal{C}(u) \subseteq P_{ul}$) Let $(t_1,\ldots,t_n) \in \mathcal{C}(u)$. We show inductively that for every $a \in \{\mathbf{a},\mathbf{d}\}$, the condition $x^a_{j-1} + t_j \leq 1$ is satisfied and 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 clock values $x^a_0,\ldots,x^a_{j-1}$ are well defined. Let $lr(x^a,j)$ be the maximal index before transition j such that $w_{lr(x^a,j)} \neq a$. Necessarily $w_{lr(x^a,j)+1} \ldots w_j = a^{j-i}$ and thus $t_{lr(x^j)+1} + \ldots + t_j \leq 1$ by definition of $\mathcal{C}(u)$. This latter sum is equal to $x^a_{j-1} + t_j \leq 1$ and thus the condition on x^a imposed by $\mathcal{L}(u_j)$ is satisfied.

Example 3. A vector $(t_1,t_2,t_3,t_4,t_5) \in [0,1]^5$ belongs to $P_{\mathbf{daada}} = \mathcal{C}(\mathbf{daad})$ 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 $(1 - t_1, t_2, t_2 + t_3, 1 - t_4, t_5) \in \mathcal{O}(\mathbf{daad})$. One can check this fact on examples given before: $(0.7,0.2,0.2,0.5,0.1) \in P_{\mathbf{daada}}$ corresponds to the vector $(0.3,0.2,0.4,0.5,0.1) \in \mathcal{O}(\mathbf{daad})$.

The purpose of the following section is to give the general formula for the correspondence between timed polytopes and order polytopes we have foreseen in the previous example.

2.3 Volume preserving transformation between $\mathbb{L}'_n$ and $\mathcal{O}_n(L)$.

Let n be a positive integer. We define for $w = ul$ with $u \in \{\mathbf{a}, \mathbf{d}\}^{n-1}$ and $l \in \{\mathbf{a}, \mathbf{d}\}$ a volume preserving function $(t_1, \cdots, t_n) \mapsto (\nu_1, \cdots, \nu_n)$ from the chain polytope $\mathcal{C}(u) = P_{ul}$ to the order polytope $\mathcal{O}(u)$. This is a simple case of Theorem 2.1 of [11].

Let $w \in \{\mathbf{a}, \mathbf{d}\}^n$ and $n = |w|$. Let $j \in [n]$ and i be the index such that $w_i \cdots w_{j-1}$ is a maximal ascending or descending block i.e. i is minimal such that $w_i \cdots w_{j-1} = l^{j-i}$ with $l \in \{\mathbf{a}, \mathbf{d}\}^*$. If $w_j = \mathbf{d}$ we define $\nu_j = 1 - \sum_{k=i}^{j} t_k$ and $\nu_j = \sum_{k=i}^{j} t_k$ otherwise.

Proposition 2. *The mapping $\phi_{ul} : (t_1, \cdots, t_n) \mapsto (\nu_1, \cdots, \nu_n)$ is a volume preserving transformation from $\mathcal{C}(u) = P_{ul}$ to $\mathcal{O}(u)$. It can be computed in linear time using the following recursive characterization:*

$$\left|\begin{array}{ll} \nu_1 = t_1 & \text{if } w_1 = \mathbf{a} \\ \nu_1 = 1 - t_1 & \text{if } w_1 = \mathbf{d} \end{array}\right. \text{ and for } i \geq 2\text{: } \left|\begin{array}{ll} \nu_i = \nu_{i-1} + t_i & \text{if } w_{i-1}w_i = \mathbf{aa}; \\ \nu_i = t_i & \text{if } w_{i-1}w_i = \mathbf{da}; \\ \nu_i = 1 - t_i & \text{if } w_{i-1}w_i = \mathbf{ad}; \\ \nu_i = \nu_{i-1} - t_i & \text{if } w_{i-1}w_i = \mathbf{dd}. \end{array}\right.$$

Proof. The function ϕ_{ul} is a volume preserving transformation since it is a linear function given by a unimodular (i.e. an integer matrix having determinant $+1$ or -1) matrix. Indeed $\phi_{ul}(\boldsymbol{t}) = \boldsymbol{\nu}$ iff $\boldsymbol{\nu}^\top = M_{ul}\boldsymbol{t}^\top + \boldsymbol{b}$ with for all $j \in [n]$: if $w_j = \mathbf{a}$ (resp. $w_j = \mathbf{d}$) then the j^{th} row of the matrix M_{ul} has only 1s (resp. -1s) between coordinates i and j included and the j^{th} row of $\boldsymbol{b}$ is 0 (resp. -1). One can see that M_{ul} is upper triangular and has only 1 and -1 on its diagonal and thus is unimodular. Now it remains to prove that $\boldsymbol{\nu}$ $(= \phi_{ul}(\boldsymbol{t}))$ belongs to $\mathcal{O}(u)$ for $\boldsymbol{t} \in \mathcal{C}(u)$. For this we show that the two conditions (C-1) and (C-2) below are equivalent; the former is the definition of $(t_1, \cdots, t_n) \in \mathcal{C}(u)$ while the latter is equivalent to $\nu_1, \ldots, \nu_n \in \mathcal{O}(u)$:

(C-1) for all $i < j \leq n$ and $l \in \{\mathbf{a}, \mathbf{d}\}$, $u_i \cdots u_{j-1} = l^{j-i} \Rightarrow t_i + \ldots + t_j \leq 1$;
(C-2) for all $i < j \leq n$, $u_i \cdots u_{j-1} = \mathbf{a}^{j-i} \Rightarrow \nu_i \leq \ldots \leq \nu_j \leq 1$ and $u_i \cdots u_{j-1} = \mathbf{d}^{j-i} \Rightarrow \nu_j \leq \ldots \leq \nu_i \leq 1$.

Let $i < j \leq n$ and $u_i \cdots u_{j-1} = \mathbf{a}^{j-i}$ then the following chain of inequalities $[0 \leq \nu_i = t_i \leq \ldots \leq \nu_{j-1} = t_i + \ldots + t_{j-1} \leq \nu_j = (1 - t_j \text{ or } t_i + \ldots + t_j) \leq 1]$ is equivalent to $t_i + \ldots + t_j \leq 1$. The case of descents can be proved in a similar way by applying $x \mapsto 1 - x$ to the preceding inequalities.

Proposition 2 links the timed polytope of a signature of length $n+1$ and the order polytopes of a signature of length n. We correct this mismatch of length using prolongation of languages. We say that a language L' is 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 L has prolongations e.g. $L' = Ll$ for $l \in \{\mathbf{a}, \mathbf{d}\}$ are prolongations of L. A prolongation of L^{ex} is $L^{ex'} = (\mathbf{aadd})^*(\mathbf{aad}+\mathbf{a})$ recognized by the automaton depicted in the middle of Figure 1.

Now we can extend Proposition 2 to a language of signatures.

Theorem 1. *Let $L \subseteq \{\mathbf{a}, \mathbf{d}\}^*$ 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.*

$$\begin{array}{rl} \phi : & \mathbb{L}'_n \to \mathcal{O}_n(L) \\ & (\boldsymbol{t}, w) \mapsto \phi_w(\boldsymbol{t}) \end{array} \tag{6}$$

As a consequence, the two problems can be solved if we know how to compute the VGF of a timed language $\mathbb{L}'$ and how to generate timed vector uniformly in $\mathbb{L}'_n$. A characterization of the VGF of a timed language as a solution of a system of differential equations is done in our previous work [3]. Nevertheless the equations of this article were quite uneasy to handle and did not give a closed form formula for the VGF. To get more precise and simple equations than in [3] we work with a novel class of timed languages involving two kinds of transitions $\mathtt{S}$ and $\mathtt{T}$.

2.4 The S-T (timed) language encoding.

The S-T-encoding We consider the finite alphabet $\{\mathtt{S}, \mathtt{T}\}$ whose elements must be respectively read as *straight* and *turn*. The S-T-encoding of type $l \in \{\mathbf{a}, \mathbf{d}\}$ of a word $w \in \{\mathbf{a}, \mathbf{d}\}^*$ is a word $w' \in \{\mathtt{S}, \mathtt{T}\}^*$ denoted by $\mathtt{st}_l(w)$ and defined recursively as follows: for every $i \in [n]$, $w'_i = \mathtt{S}$ if $w_i = w_{i-1}$ and $w'_i = \mathtt{T}$ otherwise, with the convention that $w_0 = l$. The mapping $\mathtt{st}_l$ is invertible: $w = \mathtt{st}_l^{-1}(w')$ is defined recursively as follows: for every $i \in [n]$, $w_i = w_{i-1}$ if $w'_i = \mathtt{S}$ and $w_i \neq w_{i-1}$ otherwise, with convention that $w_0 = l$. Notion of S-T-encoding can be extended naturally to languages e.g. for the running example: $\mathtt{st}_{\mathbf{d}}(L^{ex'}) = (\mathtt{TS})^*\mathtt{T}$. We call an S-T-automaton, a deterministic finite state automaton with transition alphabet $\{\mathtt{S}, \mathtt{T}\}$ (see Figure 1 for an S-T-automaton recognizing $\mathtt{st}_{\mathbf{d}}(L^{ex'})$).

Timed semantics and S-T-encoding In the following we define clock and timed languages similarly to what we have done in section 2.2. Here we need only one clock x that remains bounded by 1. We define the clock languages associated to $\mathtt{S}$ by $\mathcal{L}(\mathtt{S}) = \{x \xrightarrow{(t,\mathtt{S})} x + t \mid x \in [0,1], t \in [0, 1-x]\}$ and the clock language associated to $\mathtt{T}$ by $\mathcal{L}(\mathtt{T}) = \{x \xrightarrow{(t,\mathtt{T})} t \mid x \in [0,1], t \in [0, 1-x]\}$. Let $L'' \subseteq \{\mathtt{S}, \mathtt{T}\}^*$ we denote by $L''(x)$ the timed language starting from x: $L''(x) = \{(\boldsymbol{t}, w) \mid \exists y \in [0,1], x \xrightarrow{(\boldsymbol{t},w)} y \in \mathcal{L}(w), w \in L''\}$. The *timed semantics* of $L'' \subseteq \{\mathtt{S}, \mathtt{T}\}^*$ is $L''(0)$.

The S-T-encodings yields a natural volume preserving transformation between timed languages:

Proposition 3. *Let $L' \subseteq \{\mathbf{a}, \mathbf{d}\}^*$, $l \in \{\mathbf{a}, \mathbf{d}\}$, $\mathbb{L}'$ be the timed semantics of L' and $\mathbb{L}''$ be the timed semantics of $\mathtt{st}_l(L')$ then the function $(\boldsymbol{t}, w) \mapsto (\boldsymbol{t}, \mathtt{st}_l^{-1}(w))$ is a volume preserving transformation from $\mathbb{L}''_n$ to $\mathbb{L}'_n$.*

Using notation and results of Theorem 1 and Proposition 3 we get a volume preserving transformation from $\mathbb{L}''_n$ to $\mathcal{O}_n(L)$.

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

$$\texttt{Vol}(\mathbb{L}''_n) = \frac{|\texttt{sg}^{-1}(L_{n-1})|}{n!} \text{ for } n \geq 1, \text{ and } VGF(\mathbb{L}'')(z) = EGF(\texttt{sg}^{-1}(L))(z)$$

Thus to solve Problem 1 it suffices to characterize the VGF of an S-T-automaton.

3 Solving the two problems

3.1 Characterization of the VGF of an S-T-automaton.

In this section we characterize precisely the VGF of the timed language recognized by an S-T-automaton. This solves Problem 1.

We have defined just above timed language $L''(x)$ parametrized by an initial clock vector x. Given an S-T-automaton, we can also consider the intial state as a parameter and write Kleene like systems of equations on parametric language $L_p(x)$ (similarly to [3]). More precisely, let $\mathcal{A} = \{\{\texttt{S}, \texttt{T}\}, Q, i, F, \delta\}$ be S-T-automaton. To every state $p \in Q$ we denote by $L_p \subseteq \{\texttt{S}, \texttt{T}\}^*$ the language starting from p i.e. recognized by $\mathcal{A}_p =_{\text{def}} \{\{\texttt{S}, \texttt{T}\}, Q, p, F, \delta\}$. We adopt the convention that $L_{\delta(p,l)}$ is empty when $\delta(p, l)$ is undefined and the corresponding generating function is null. Then for every $p \in Q$, we have a parametric language equation:

$$L_p(x) = \left[\cup_{t \leq 1-x}(t, \texttt{S})L_{\delta(p,\texttt{S})}(x+t)\right] \cup \left[\cup_{t \leq 1-x}(t, \texttt{T})L_{\delta(p,\texttt{T})}(t)\right] \cup (\epsilon \text{ if } p \in F) \quad (7)$$

Passing to volume generating functions $f_p(x, z) =_{\text{def}} VGF(L_p(x))(z)$ (as in [3]) we get:

$$f_p(x, z) = z \int_x^1 f_{\delta(p,\texttt{S})}(s, z)ds + z \int_0^{1-x} f_{\delta(p,\texttt{T})}(t, z)dt + (1 \text{ if } p \in F) \quad (8)$$

In matrix notation:

$$\boldsymbol{f}(x, z) = zM_{\texttt{S}} \int_x^1 \boldsymbol{f}(s, z)ds + zM_{\texttt{T}} \int_0^{1-x} \boldsymbol{f}(t, z)dt + \boldsymbol{F} \quad (9)$$

where $\boldsymbol{f}(x, z)$, $\int_x^1 \boldsymbol{f}(s, z)ds$ and $\int_0^{1-x} \boldsymbol{f}(t, z)dt$ are the column vectors whose coordinates are respectively the $f_p(x, z)$, $\int_x^1 \boldsymbol{f}_p(s, z)ds$ and $\int_0^{1-x} \boldsymbol{f}_p(t, z)dt$ for $p \in Q$. The p^{th} coordinate of the column vector $\boldsymbol{F}$ is 1 if $p \in F$ and 0 otherwise. The $Q \times Q$-matrices $M_{\texttt{S}}$ and $M_{\texttt{T}}$ are the adjacency matrices corresponding to letter S and T i.e. for $l \in \{\texttt{S}, \texttt{T}\}$, $M_l(p, q) = 1$ if $\delta(p, l) = q$, 0 otherwise.

The equation (9) is equivalent to the differential equation:

$$\frac{\partial}{\partial x}\boldsymbol{f}(x, z) = -zM_{\texttt{S}}\boldsymbol{f}(x, z) - zM_{\texttt{T}}\boldsymbol{f}(1-x, z) \quad (10)$$

with boundary condition

$$\boldsymbol{f}(1, z) = \boldsymbol{F}. \quad (11)$$

Algorithm 1 Computation of the generating function

1: Compute an S-T-automaton $\mathcal{A}$ for an extension of L and its corresponding adjacency matrices $M_{\mathtt{T}}$ and $M_{\mathtt{S}}$;
2: Compute $\begin{pmatrix} A_1(z) & A_2(z) \\ A_3(z) & A_4(z) \end{pmatrix} =_{\text{def}} \exp\left[z \begin{pmatrix} -M_{\mathtt{S}} & -M_{\mathtt{T}} \\ M_{\mathtt{T}} & M_{\mathtt{S}} \end{pmatrix}\right]$;
3: Compute $\boldsymbol{f}(0,z) = [A_1(z)]^{-1}[I - A_2(z)]\boldsymbol{F}$ (or $\boldsymbol{f}(0,z) = [I - A_3(z)]^{-1}A_4(z)\boldsymbol{F}$);
4: **return** the component of $\boldsymbol{f}(0,z)$ corresponding to the initial state of $\mathcal{A}$.

The equation (10) is equivalent to the following linear homogeneous system of ordinary differential equations with constant coefficients:

$$\frac{\partial}{\partial x}\begin{pmatrix} \boldsymbol{f}(x,z) \\ \boldsymbol{f}(1-x,z) \end{pmatrix} = z \begin{pmatrix} -M_{\mathtt{S}} & -M_{\mathtt{T}} \\ M_{\mathtt{T}} & M_{\mathtt{S}} \end{pmatrix} \begin{pmatrix} \boldsymbol{f}(x,z) \\ \boldsymbol{f}(1-x,z) \end{pmatrix}. \tag{12}$$

whose solution is of the form

$$\begin{pmatrix} \boldsymbol{f}(x,z) \\ \boldsymbol{f}(1-x,z) \end{pmatrix} = \exp\left[xz \begin{pmatrix} -M_{\mathtt{S}} & -M_{\mathtt{T}} \\ M_{\mathtt{T}} & M_{\mathtt{S}} \end{pmatrix}\right] \begin{pmatrix} \boldsymbol{f}(0,z) \\ \boldsymbol{f}(1,z) \end{pmatrix} \tag{13}$$

Taking $x = 1$ in (13) and using the boundary condition (11) we obtain:

$$\begin{aligned} \boldsymbol{F} &= A_1(z)\boldsymbol{f}(0,z) + A_2(z)\boldsymbol{F} \\ \boldsymbol{f}(0,z) &= A_3(z)\boldsymbol{f}(0,z) + A_4(z)\boldsymbol{F} \end{aligned} \tag{14}$$

where $\begin{pmatrix} A_1(z) & A_2(z) \\ A_3(z) & A_4(z) \end{pmatrix} = \exp\left[z \begin{pmatrix} -M_{\mathtt{S}} & -M_{\mathtt{T}} \\ M_{\mathtt{T}} & M_{\mathtt{S}} \end{pmatrix}\right]$. In particular when $z = 0$, $A_1(0) = I - A_3(0) = I$ and thus the two continuous functions $z \mapsto \det A_1(z)$ and $z \mapsto \det(I - A_3(z))$ are positive in a neighbourhood of 0. We deduce that the inverses of the matrices $A_1(z)$ and $I - A_3(z)$ are well defined in a neighbourhood of 0 and thus both rows of (14) permit to express $\boldsymbol{f}(0,z)$ with respect to $\boldsymbol{F}$:

$$\begin{aligned} \boldsymbol{f}(0,z) &= [A_1(z)]^{-1}[I - A_2(z)]\boldsymbol{F} \\ \boldsymbol{f}(0,z) &= [I - A_3(z)]^{-1}A_4(z)\boldsymbol{F} \end{aligned} \tag{15}$$

Finally the coordinate of the column vector $\boldsymbol{f}(0,z)$ associated to the initial state gives the expected VGF. To sum up we have:

Theorem 3. *Given a regular language $L \subseteq \{\mathbf{a}, \mathbf{d}\}^*$, one can compute the exponential generating function $EGF(\mathtt{sg}^{-1}(L))(z)$ using Algorithm 1.*

Some comments about the algorithm. In line 1, several choices are left to the user: the prolongation L' of the language L, the type of the S-T-encoding and the automaton that realizes the S-T-encoding. These choices should be made such that the output automaton has a minimal number of states or more generally such that the matrices $M_{\mathtt{T}}$ and $M_{\mathtt{S}}$ are the simplest possibles. Exponentiation of matrices is implemented in most of computer algebra systems.

3.2 An algorithm for Problem 2

Now we can solve Problem 2 using a uniform sampler of timed words (Algorithm 2), the volume preserving transformation of Theorem 2 and a sorting algorithm.

Theorem 4. *Let $L \subseteq \{\mathbf{a}, \mathbf{d}\}^*$ and $\mathbb{L}''$ be the timed semantics of a* S-T*-encoding of type l (for some $l \in \{\mathbf{a}, \mathbf{d}\}$) of a prolongation of L. The following algorithm permits to achieve a uniform sampling of permutation in $\mathtt{sg}^{-1}(L_{n-1})$. i.e. For $\sigma \in \mathtt{sg}_n^{-1}(L)$, the probability that the permutation σ is returned is $1/|\mathtt{sg}^{-1}(L_{n-1})|$.*

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

Proof. For all $\sigma \in \mathtt{sg}_n^{-1}(L)$, the probability $p(\sigma)$ that the output is σ is the probability to choose a timed word $(\boldsymbol{t}, w)$ such that $\Pi[\phi_{\mathtt{st}^{-1}(w)}(\boldsymbol{t})] = \sigma$. Since the timed words are uniformly sampled this probability is equal to $\mathtt{Vol}(\{(\boldsymbol{t}, w) \mid \Pi[\phi_w(\boldsymbol{t})] = \sigma\})/\mathtt{Vol}(\mathbb{L}''_n)$ which is equal to $\mathtt{Vol}(\{\boldsymbol{\nu} \mid \Pi(\boldsymbol{\nu}) = \sigma\})/\mathtt{Vol}(\mathbb{L}''_n)$ since the mapping $(\boldsymbol{t}, w) \mapsto \phi_{\mathtt{st}^{-1}(w)}(\boldsymbol{t})$ is a volume preserving transformation. The numerator is the volume of the order simplex associated to σ which is $\mathtt{Vol}(\mathcal{O}(\sigma)) = 1/n!$; the denominator $\mathtt{Vol}(\mathbb{L}''_n)$ is $|\mathtt{sg}^{-1}(L_{n-1})|/n!$ by virtue of Theorem 2. We get the expected result $p(\sigma) = (1/n!)/(|\mathtt{sg}^{-1}(L_{n-1})|/n!) = 1/|\mathtt{sg}^{-1}(L_{n-1})|$.

Uniform sampling of timed words. Recursive formulae (16) and (17) below are freely inspired by those of [4] and of [3]. They are the key tools to design a uniform sampler of timed word. This algorithm is a lifting from the discrete case of the so-called recursive method (see [6,10]). For all $q \in Q$, $n \in \mathbb{N}$ and $x \in [0, 1]$ we denote by $L_{q,n}(x)$ the language $L_q(x)$ restricted to n-length timed words. The languages $L_{q,n}(x)$ can be recursively defined as follows: $L_{q,0}(x) = \epsilon$ if $q \in F$ and $L_{q,0} = \emptyset$ otherwise;

$$L_{q,n+1}(x) = \left[\cup_{t \leq 1-x}(t, \mathtt{S})L_{\delta(q,\mathtt{S}),n}(x+t)\right] \cup \left[\cup_{t \leq 1-x}(t, \mathtt{T})L_{\delta(q,\mathtt{T}),n}(t)\right]. \quad (16)$$

For $q \in Q$ and $n \geq 0$, we denote by $v_{q,n}$ the function $x \mapsto \mathtt{Vol}[L_{q,n}(x)]$ from $[0, 1]$ to $\mathbb{R}^+$. Each function $v_{q,n}$ is a polynomial of a degree less or equal to n that can be computed recursively using the recurrent formula: $v_{q,0}(x) = 1_{q \in F}$ and

$$v_{q,n+1}(x) = \int_x^1 v_{\delta(q,\mathtt{S}),n}(y)dy + \int_0^{1-x} v_{\delta(q,\mathtt{T}),n}(y)dy. \quad (17)$$

The polynomials $v_{q,n}(x)$ play a key role for the uniform sampler, they permit also to retrieve directly the terms of the wanted VGF: $\mathtt{Vol}(\mathbb{L}''_n) = v_{q_0,n}(0)$ where q_0 is the initial state of the S-T automaton.

Theorem 5. *Algorithm 2 is a uniform sampler of timed words of $\mathbb{L}''_n$ i.e. for every volume measurable subset $A \subseteq \mathbb{L}''_n$, the probability that the returned timed word belongs to A is $\mathtt{Vol}(A)/\mathtt{Vol}(\mathbb{L}''_n)$.*

Algorithm 2 Recursive uniform sampler of timed words

$x_0 \leftarrow 0$; $q_0 \leftarrow$ initial state;
for $k = 1$ to n **do**
  Compute $m_k = v_{q_{k-1},n-(k-1)}(x_{k-1})$ and $p_{\mathtt{S}} = \int_{x_{k-1}}^{1} v_{\delta(q_{k-1},\mathtt{S}),n-k}(y)dy/m_k$;
  $b \leftarrow \mathtt{BERNOULLI}(p_{\mathtt{S}})$; (return 1 with probability $p_{\mathtt{S}}$ and 0 otherwise)
  if $b = 1$ **then**
    $w_k \leftarrow \mathtt{S}$; $q_k \leftarrow \delta(q_{k-1}, \mathtt{S})$;
    $r \leftarrow \mathtt{RAND}([0,1])$; (return a number uniformly sampled in $[0,1]$)
    $t_k \leftarrow$ the unique solution in $[0, 1-x_{k-1}]$ of $\frac{1}{m_k p_{\mathtt{S}}} \int_{x_{k-1}}^{x_{k-1}+t_k} v_{q_k,n-k}(y)dy - r = 0$;
    $x_k \leftarrow x_{k-1} + t_k$;
  else
    $w_k \leftarrow \mathtt{T}$; $q_k \leftarrow \delta(q_{k-1}, \mathtt{T})$;
    $r \leftarrow \mathtt{RAND}([0,1])$; (return a number uniformly sampled in $[0,1]$)
    $t_k \leftarrow$ the unique solution in $[0, 1-x_{k-1}]$ of $\frac{1}{m_k(1-p_{\mathtt{S}})} \int_{0}^{t_k} v_{q_k,n-k}(y)dy - r = 0$;
    $x_k \leftarrow t_k$;
  end if
end for
return $(t_1, w_1)(t_2, w_2)\dots(t_n, w_n)$

Proof. 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 that $w_1 \cdots w_n \in L''$.

We denote by $p[(t_1, w_1) \cdots (t_n, w_n)]$ the density of probability of the timed word $(t_1, w_1) \cdots (t_n, w_n) \in \mathbb{L}''$ to be returned. The algorithm is a uniform sampler if it assign the same density of probability to every timed word of $\mathbb{L}''$ i.e. $p[(t_1, w_1) \cdots (t_n, w_n)] = 1/\mathtt{Vol}(\mathbb{L}'')$.

During the k^{th} loop, w_k and t_k are chosen, knowing q_{k-1}, x_{k-1} and the index k, according to a density of probability (implicitly defined by the algorithm) denoted by $p_k[(t_k, w_k) \mid q_{k-1}, x_{k-1}]$. The new general state (q_k, x_k) is (deterministically) defined using q_{k-1}, x_{k-1}, t_k, w_k. The following chain rule is satisfied

$$p[(t_1, w_1) \cdots (t_n, w_n)] = \prod_{k=1}^{n} p_k[(t_k, w_k) \mid q_{k-1}, x_{k-1}] \tag{18}$$

No it suffices to plug (19) proven in Lemma 1 just below in (18) to get the expected result:

$$p[(t_1, w_1) \cdots (t_n, w_n)] = \frac{\prod_{k=1}^{n} m_{k+1}}{\prod_{k=1}^{n} m_k} = \frac{m_{n+1}}{m_1} = \frac{v_{q_n,0}(x_n)}{v_{q_0,n}(0)} = \frac{1}{\mathtt{Vol}(\mathbb{L}''_n)}.$$

Lemma 1. *In Algorithm 2 during the k^{th} loop* **for** *the timed transition (t_k, w_k) is chosen knowing the current state (q_{k-1}, x_{k-1}) according to the following probability distribution function (variables of the following equation such as m_k are defined in the algorithm):*

$$p_k[(t_k, w_k) \mid q_{k-1}, x_{k-1}] = \frac{m_{k+1}}{m_k} = \frac{v_{q_k,n-k}(x_k)}{v_{q_{k-1},n-(k-1)}(x_{k-1})}. \tag{19}$$

Proof. The choice of (t_k, w_k) is done in two steps: first w_k is chosen (and thus $q_k = \delta(q_{k-1}, w_k)$) and then t_k. We write this

$$p_k[(t_k, w_k) \mid q_{k-1}, x_{k-1}] = p_k[w_k \mid q_{k-1}, x_{k-1}] p_k[t_k \mid q_k, x_{k-1}] \tag{20}$$

Remark that $b = 1$ iff $w_k = \mathtt{S}$ and thus $p_k[\mathtt{S} \mid q_{k-1}, x_{k-1}] = p_{\mathtt{S}}$ (the probability that 1 is returned in line 4) and $p_k[\mathtt{T} \mid q_{k-1}, x_{k-1}] = 1 - p_{\mathtt{S}}$ otherwise.

In both cases ($b = 0$ or 1) the delay t_k is sampled using the so-called inverse transform sampling. This method states that to sample a random variable according to a probability density function (PDF) $p(t)$ (here $p(t) = p_k[t \mid q_k, x_{k-1}]$) it suffices to uniformly sample a random number in $[0, 1]$ and define t such that $\int_0^t p(t')dt' = r$. The latter integral is known as the cumulative density function[7] (CDF) associated to p.

- When $b = 1$ (and thus $w_k = \mathtt{S}$), the CDF used in the algorithm is

$$t \mapsto \frac{1}{m_k p_{\mathtt{S}}} \int_0^t v_{q_k, n-k}(x_{k-1} + t')dt'.$$

  Its corresponding PDF is

$$p_k[t_k \mid q_k, x_{k-1}] = \frac{1}{m_k p_{\mathtt{S}}} v_{q_k, n-k}(x_{k-1} + t_k) = \frac{m_{k+1}}{m_k p_{\mathtt{S}}}.$$

  Plugging this in (20) we get the expected result (19).
- When $b = 0$ (and thus $w_k = \mathtt{T}$), a similar reasoning permits to prove (19) which is then true in both cases.

Remark 1. One can remark that the probability depends on the index k of the loop which is different from stochastic processes of our previous work [5].

Some comments about the algorithm. Algorithm 2 requires a precomputing of all functions $v_{q,k}$ for $q \in Q$ and $k \leq n$ done by Algorithm 3 below (see also Proposition 4 for the complexity). The expressions in lines 8 and 13 are polynomials increasing in $[x, 1]$ (the derivative is the integrand which is positive on $(x, 1)$). Finding the root of such a polynomial can be done numerically and efficiently with a controlled error using a numerical scheme such as the Newton's method.

Proposition 4. *Algorithm 3 has space and time complexity $O(|Q|n^2)$. Its bit space complexity is $O(|Q|n^3)$.*

Proof. The polynomial $v_{q,m}$ is of degree m, it has $O(m)$ coefficients. Therefore the time and space complexity are $O(\sum_{m=1}^{n} |Q|m) = O(|Q|n^2)$.

Magnitudes of coefficients of $v_{q,m}$ behave like $2^{m\mathcal{H}}$ where $\mathcal{H}$ is the entropy of the timed language (see [4]) and thus one needs $O(m)$ bits to store them. This explains why an extra factor n appears when dealing with bit space complexity.

[7] Its inverse (t function of r) is known as the quantile function.

Algorithm 3 Preprocessing for Algorithm 2

```
for p ∈ Q do
    define v_{p,0}(x) = 1_{p∈F}.
    for k = 1 to n do
        compute v_{p,k}(x) using (17).
    end for
end for
```

4 Examples

In section 4.1 we show how Algorithm 1 applies to the classical example of alternating permutations. In section 4.2 we apply this algorithm to what we call up-up-down-down permutations. In section 4.3 we treat the running example given in section 1.

4.1 The alternating permutations

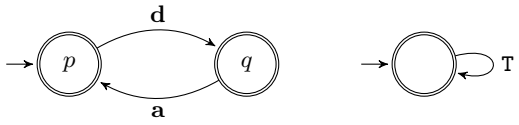

Fig. 2. An automaton for $(\mathbf{da})^*(\varepsilon + \mathbf{d})$ and its S-T encoding of type $\mathbf{d}$

The class of alternating permutation is[8] $\mathtt{Alt} = \mathfrak{S}_0 \cup \mathtt{sg}^{-1}[(\mathbf{da})^*(\varepsilon + \mathbf{d})]$. It is well known since the 19^{th} century and the work of Désiré André that

$$EGF(\mathtt{Alt})(z) = \tan(z) + \sec(z) \quad (\text{where } \sec(z) = 1/\cos(z)).$$

Several different proofs of this results can be found in [16]. Here we give a novel proof based on the application of Algorithm 1 on $(\mathbf{da})^*(\varepsilon + \mathbf{d})$.

A prolongation of $(\mathbf{da})^*(\varepsilon + \mathbf{d})$ is $(\mathbf{da})^*(\mathbf{d} + \mathbf{da})$. We add ε to the language to add 1 to its VGF, indeed

$$EGF(\mathtt{Alt})(z) = 1 + VGF[(\mathbf{da})^*(\mathbf{d} + \mathbf{da})](z) = VGF[(\mathbf{da})^*(\varepsilon + \mathbf{d})](z)$$

The S-T encoding of type $\mathbf{a}$ of $(\mathbf{da})^*(\varepsilon+\mathbf{d})$ is just $\mathtt{S}^*$ which is recognized by the one loop automaton depicted in the right of Figure 2. Thus $M_{\mathtt{S}} = (1)$, $M_{\mathtt{T}} = (0)$ and we must compute $\exp(zM) = \sum_{n\in\mathbb{N}} z^n M^n/n!$ with $M = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$.

[8] The unique permutation on the empty set has no signature and thus $\mathfrak{S}_0 \not\subseteq \mathtt{sg}^{-1}(L)$ for any language L of signature.

Computation of $\exp(zM)$ is easy since M is unipotent and thus its sequence of power M^k is periodic: $M^0 = I_2$, $M^1 = M$, $M^2 = -I_2$, $M^3 = -M$, $M^3 = I_2$, $M^4 = M$, ...

Then for all $k \geq 0$:

$$M^{2k} = \begin{pmatrix} (-1)^k & 0 \\ 0 & (-1)^k \end{pmatrix}; \quad M^{2k+1} = \begin{pmatrix} 0 & (-1)^{2k} \\ (-1)^{2k+1} & 0 \end{pmatrix}$$

Hence $\exp(zM) = \sum_{n\in\mathbb{N}} z^n M^n/n! = \begin{pmatrix} \cos(z) & -\sin(z) \\ \sin(z) & \cos(z) \end{pmatrix}$.

By definition $A_1(z) = \cos(z)$, $A_2(z) = -\sin(z)$. We can conclude:

$$EGF(\texttt{Alt})(z) = A_1(z)^{-1}(1 - A_2(z)) = \frac{1}{\cos(z)} + \tan(z).$$

4.2 The up-up-down-down permutations

Here we compute the exponential generating function of the class of up-up-down-down permutations given as running example of the paper. Recall that the corresponding regular language is $L^{ex} = (\mathbf{aadd})^*(\mathbf{aa} + \varepsilon)$, one of its extension is $L^{ex'} = (\mathbf{aadd})^*(\mathbf{aad} + \mathbf{a})$ and the S-T-encoding of type $\mathbf{d}$ of this latter language is $\mathtt{st}_{\mathbf{d}}(L') = (\mathtt{TS})^*\mathtt{T}$. These languages are recognized by automata depicted in Figure 1. The adjacency matrices of the third automaton are $M_{\mathtt{S}} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$, $M_{\mathtt{T}} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ and the row vector of final state is $\boldsymbol{F} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Let $M = \begin{pmatrix} -M_{\mathtt{S}} & -M_{\mathtt{T}} \\ M_{\mathtt{T}} & M_{\mathtt{S}} \end{pmatrix}$. Again the computation of $\exp(zM)$ is easy since M is unipotent[9]:

$$M = \begin{pmatrix} 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}; M^2 = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}; M^3 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \end{pmatrix}$$

and $M^4 = I_4$.

Hence if we denote by $f_i(z) = \sum_{n=0}^{+\infty} z^{4n+i}/(4n+i)!$ for $i \in \{0, 1, 2, 3\}$ we have:

$$\exp zM = f_0(z)I + f_1(z)M + f_2(z)M^2 + f_3(z)M^3 \text{ and}$$
$$A_1(z) = \begin{pmatrix} f_0(z) & -f_3(z) \\ -f_1(z) & f_0(z) \end{pmatrix}; A_2(z) = \begin{pmatrix} f_2(z) & -f_1(z) \\ f_3(z) & f_2(z) \end{pmatrix}.$$

The function f_i can be expressed with trigonometric and hyperbolic functions:

$$f_0(z) = [\cosh(z) + \cos(z)]/2;$$
$$f_1(z) = [\sinh(z) + \sin(z)]/2;$$
$$f_2(z) = [\cosh(z) - \cos(z)]/2;$$
$$f_3(z) = [\sinh(z) - \sin(z)]/2.$$

[9] This is in fact the case for all cyclic automata.

It holds that

$$[I_2 - A_2(z)]\boldsymbol{F} = \begin{pmatrix} f_1(z) \\ 1 - f_2(z) \end{pmatrix}$$

and thus

$$\begin{pmatrix} f_p(z) \\ f_q(z) \end{pmatrix} = \begin{pmatrix} f_0(z) & -f_3(z) \\ -f_1(z) & f_0(z) \end{pmatrix}^{-1} \begin{pmatrix} f_1(z) \\ 1 - f_2(z) \end{pmatrix}.$$

Using Cramer formula we get $f_p(z) = [f_1(z)f_0(z) + f_3(z)(1 - f_2(z))]/[f_0^2(z) + f_1(z)f_3(z)]$. After straightforward simplifications we obtain the wanted result:

$$f(z) = f_p(z) = \frac{\sinh(z) - \sin(z) + \sin(z)\cosh(z) + \sinh(z)\cos(z)}{1 + \cos(z)\cosh(z)}.$$

4.3 Permutations without two consecutive descents

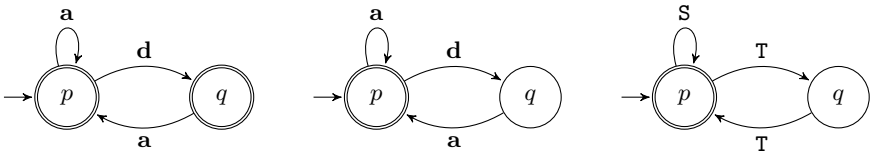

Fig. 3. From left to right automata for L^{ex_3}, $L^{ex_3'} = \{\varepsilon\} \cup L^{ex_3}.\{\mathbf{a}\}$ and $\mathtt{st}_{\mathbf{a}}(L^{ex_3'})$

Consider the class C^{ex_3} of permutations without two consecutive descents. This class has already been studied and its EGF computed. References and many details can be found in the On-Line Encyclopedia of Integer Sequences (OEIS), sequence A049774. In particular the following EGF is given:

$$EGF(C^{ex_3})(z) = \frac{\sqrt{3}e^{z/2}}{\sqrt{3}\cos\left(\frac{\sqrt{3}}{2}z\right) - \sin\left(\frac{\sqrt{3}}{2}z\right)}.$$

We give an alternative proof of this result based on the method developed in this paper. The class C^{ex_3} can be described in terms of regular languages:

$$C^{ex_3} = \mathfrak{S}_0 \cup \mathtt{sg}^{-1}[(\mathbf{a} + \mathbf{da})^*(\varepsilon + \mathbf{d})].$$

A prolongation of $(\mathbf{a}+\mathbf{da})^*(\varepsilon+\mathbf{d})$ is $(\mathbf{a}+\mathbf{da})^*\mathbf{a}$. As for alternating permutations we add the word ε to this language to add 1 to the final generating function, thus we get the language $(\mathbf{a}+\mathbf{da})^*$ recognized by the automaton depicted in the middle of Figure 3. Its S-T encoding of type $\mathbf{a}$ is $(\mathtt{S} + \mathtt{TT})^*$ which is recognized by the automaton depicted in the right of Figure 3. Its adjacency matrices are $M_{\mathtt{S}} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$, $M_{\mathtt{T}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and the row vector of final state is $\boldsymbol{F} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Let

$M = \begin{pmatrix} -M_{\mathtt{S}} & -M_{\mathtt{T}} \\ M_{\mathtt{T}} & M_{\mathtt{S}} \end{pmatrix}$. We will solve directly the differential equation (10) with boundary condition (11), i.e. the system

$$\frac{\partial f_p}{\partial x}(x,z) = -zf_p(x,z)dy - zf_q(1-x,z)dy; \tag{21}$$

$$\frac{\partial f_q}{\partial x}(x,z) = -zf_p(1-x,z). \tag{22}$$

with boundary conditions $f_p(1,z) = 1; f_q(1,z) = 0$ Equation (21) taken at $x = 1$ ensures that $\frac{\partial f_p}{\partial x}(1,z) = -zf_p(0,z) - zf_q(1,z) = -zf_p(0,z)$. Thus we have the boundary conditions

$$f_p(1,z) = 1; \tag{23}$$

$$\frac{\partial f_p}{\partial x}(1,z) = -zf_p(0,z). \tag{24}$$

Differentiating (21) and replacing $\frac{\partial f_q}{\partial x}(1-x,z)$ using (22) we get:

$$\frac{\partial^2 f_p}{\partial x^2}(x,z) = -z\frac{\partial f_p}{\partial x} - z^2 f_p(x,z); \tag{25}$$

Solutions are of the form: $f_p(x,z) = e^{-zx/2}\left[a(z)\cos\left(\frac{\sqrt{3}}{2}zx\right) + b(z)\sin\left(\frac{\sqrt{3}}{2}zx\right)\right]$ with $a(z)$ and $b(z)$ to be determined using boundary conditions (23) and (24) i.e. $a(z)$ and $b(z)$ should satisfy:

$$\begin{array}{rrr} \cos\left(\frac{\sqrt{3}}{2}z\right) a(z) + & \sin\left(\frac{\sqrt{3}}{2}z\right) b(z) = e^{z/2}; \\ a(z) + & \sqrt{3} \quad b(z) = 0. \end{array}$$

Solving this system we obtained the expected EGF:

$$EGF(C^{ex_3})(z) = f_p(0,z) = a(z) = \frac{\sqrt{3}e^{z/2}}{\sqrt{3}\cos\left(\frac{\sqrt{3}}{2}z\right) - \sin\left(\frac{\sqrt{3}}{2}z\right)}.$$

5 Conclusion and perspectives

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 [1]). 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 of signatures. For this we should use as in [3] and in [2] (see also the kernels of [5]), volume of languages parametrized both by a starting and an ending state.

Volumes and languages parametrized both by a starting and an ending states would also be useful to gain a linear factor for the time and space complexity

of the preprocessing stage (Algorithm 3). Indeed such parametrized volume are needed to adapt the divide and conquer algorithm of [6].

Our work can also benefit to timed automata research. Indeed, we have proposed a uniform sampler 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 [4]. A uniform sampler of timed word would be useful to solve the proportional model checking problem motivated in the introduction of [5]

A toy implementation of the algorithms is available on-line:
`http://www.liafa.univ-paris-diderot.fr/~nbasset/sage/sage.htm`.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
2. E. Asarin, N. Basset, and A. Degorre. Spectral gap in timed automata. In V. A. Braberman and L. Fribourg, editors, *FORMATS*, volume 8053 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2013.
3. E. Asarin, N. Basset, A. Degorre, and D. Perrin. Generating functions of timed languages. In B. Rovan, V. Sassone, and P. Widmayer, editors, *MFCS*, volume 7464 of *Lecture Notes in Computer Science*, pages 124–135. Springer, 2012.
4. E. Asarin and A. Degorre. Volume and entropy of regular timed languages: Analytic approach. In J. Ouaknine and F. W. Vaandrager, editors, *FORMATS*, volume 5813 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2009.
5. N. Basset. A maximal entropy stochastic process for a timed automaton. In F. V. Fomin, R. Freivalds, M. Z. Kwiatkowska, and D. Peleg, editors, *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 61–73. Springer, 2013.
6. O. Bernardi and O. Giménez. A linear algorithm for the random sampling from regular languages. *Algorithmica*, 62(1-2):130–145, 2012.
7. P. Bouyer and A. Petit. A Kleene/Büchi-like theorem for clock languages. *Journal of Automata, Languages and Combinatorics*, 7(2):167–186, 2002.
8. R. Ehrenborg and J. Jung. Descent pattern avoidance. *Advances in Applied Mathematics*, 2012.
9. P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Camb. Univ. press, 2009.
10. P. Flajolet, P. Zimmerman, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1):1–35, 1994.
11. T. Hibi and N. Li. Unimodular equivalence of order and chain polytopes. *arXiv preprint arXiv:1208.4029*, 2012.
12. S. Kitaev. *Patterns in permutations and words*. Springer, 2011.
13. P. Marchal. Generating random alternating permutations in time n log n. 2012.
14. A. Nijenhuis and H. S. Wilf. Combinatorial algorithms for computers and calculators. *Computer Science and Applied Mathematics, New York: Academic Press, 1978, 2nd ed.*, 1, 1978.
15. R. P. Stanley. Two poset polytopes. *Discrete & Computational Geometry*, 1(1):9–23, 1986.
16. R. P. Stanley. A survey of alternating permutations. In *Combinatorics and graphs*, volume 531 of *Contemp. Math.*, pages 165–196. Amer. Math. Soc., Providence, RI, 2010.

17. G. G. Szpiro. The number of permutations with a given signature, and the expectations of their elements. *Discrete Mathematics*, 226(1):423–430, 2001.