Boltzmann generation for regular languages with shuffle

Alexis Darrasse, Konstantinos Panagiotou, Olivier Roussel, Michèle Soria

Boltzmann generation for regular languages with shuffle

A. Darrasse K. Panagiotou O. Roussel M. Soria

September 7, 2010

Abstract

This paper is devoted to uniform generation of words produced by regular specifications containing shuffle operators. We provide a Boltzmann sampler with complexity linear in the size of the output, where complexity is measured in the number of real-arithmetic operations and evaluations of generating functions.

1 Introduction

Enumeration and classification of structures specified by simple combinatorial grammars have been intensively studied during the last decade, via methods of analytic combinatorics [FS08], relying on algebraic and analytic properties of generating functions. For structures specifiable in terms of the basic combinatorial operators of union, product, set and cycle, issues of sampling uniformly at random are theoretically and algorithmically well understood, and fair implementations have also been designed, so that huge objects can be produced. The recursive method, based on enumeration sequences [NW75, **?**], allows for uniformly generating structures of a given exact size in subquadratic complexity [DZ99]. On the other hand, Boltzman model, relying on the evaluation of generating functions [DFLS04, FFP07] provides samplers with linear complexity, by relaxing the constraint of exact size.

The class of regular languages is classically defined as the smallest class of languages containing the finite sets and closed under the operations of union, product and sequence. Thus the classical specifications of Regular languages belongs to the Boltzman models of random generation.

The *Shuffle Product* is an important and useful operator on languages, both for its structural properties and its applications (see e.g. [FGT92, DGG$^+$06, MZ08]). The shuffle of two words α and β consists of the set of all words obtained by mixing in all possible ways letters of α and β while preserving their order inside α and β. The shuffle operation can be more generally applied to languages: the shuffle of two languages A and B is the set of all shuffles of words in A and B.

The shuffle product of regular langages is still a regular language, so that sampling is possible, via rephrasing the specification, or giving an automaton description for example. But the shuffle product *as it is defined*, cannot be handled in the classical Boltzman frame. In this paper we adress this problem and consider not only top-level shuffle product, but more generally *regular specifications containing shuffle operators*. Our contribution is the design of Boltzman generation algorithms with linear complexity for approximate size sampling.

Before presenting our approach, let us mention other existing methods for the random generation of regular languages with shuffle [DGG$^+$06]: the brute force algorithm for the shuffle of r languages, constructs the shuffle product of the corresponding finite automata, and the complexity of random generation is exponential in the number of automata; the recursive method can be adapted to generate words of shuffle products almost uniformly at random, with linear complexity.

Here we describe an alternative method relying on Boltzman sampling and generating functions. The main difficulty is to deal at the same time with both ordinary and exponential generating functions. To languages are naturally associated ordinary generating functions : $A(z) = \sum a_n z^n$, where a_n is the number of words of size n. However the shuffle product only translates nicely in terms of exponential generating functions $\widehat{A}(z) = \sum a_n z^n/n!$: the exponential generating

functions of the shuffle product of languages is the product of the exponential generating functions of its components.

Regarding Boltzman samplers, these two worlds correspond to different probabilities for random generation : in the case of an ordinary Boltzman sampler, a word γ of size n is gbiasenerated with a probability wich is proportional to an exponential of its size : $\mathbb{P}(\gamma) = \frac{x^n}{A(x)}$, whereas in the case of an exponential Boltzman sampler, the probability is proportional to an exponential of the size, divided by the factorial of the size : $\mathbb{P}(\gamma) = \frac{x^n}{n!\widehat{A}(x)}$.

The ordinary and exponential generating function of a language are related by the Laplace-Borel transform; using this transform, it is easy to introduce a density measure for biaising an exponential Boltzman sampler into an ordinary Boltzman sampler. Unfortunately the inverse transformation leads to an impossible problem of moments. We can think of using Boltzman sampling for Hadamard product [**?**], since an exponential generating function $\widehat{A}(z)$ expresses as the Hadamard product of $A(z)$ and $exp(z)$, but the complexity of generation is surlinear, due to rejection.

Our solution is to make a topdown decomposition according to the specification of the language, and construct an exponential Boltzman sampler from biased exponential Boltzman samplers for its components. The central idea in all our algorithms is that of biasing: when a Boltzman sampler for a class $\mathcal{A}$ is undoable, we construct a Boltzman sampler for a derived class $\mathcal{B}$ and then bias, with a probabilistic value, the parameter of the Boltzman sampler for $\mathcal{B}$, in order to get back to a uniform generation on $\mathcal{A}$.

Outline In Section 2 we review some well-known results about Boltzmann sampling and regular languages, and describe how this can be used to construct appropriate sampling algorithms. Section 3 introduces our main tool, which essentially says that we can sample according to the Boltzmann distribution, provided that we have an algorithm that generates words according to a biased distribution; in particular, we show how to bias an exponential Boltzman sampler for any combinatorial class, in order to obtain an ordinary Boltzman sampler. In Section 4 we present for all involved combinatorial constructions (union, product, sequence, and shuffle) appropriate biased Boltzmann samplers. Finally section 5 addresses some implementation issues and closes with a combinatorial interpretation of our algorithms.

2 Regular Languages and Boltzmann Sampling

We review a few basic results about regular languages, their specification and their generation. Regular languages can be described non-recursively in terms of three basic combinatorial operations (union, product and sequence), and we present Boltzmann samplers for these operations.

2.1 Regular languages

Let $\mathcal{A}, \mathcal{B}$ be two languages. We say that $\mathcal{C}$ is the *disjoint union* of $\mathcal{A}$ and $\mathcal{B}$, in symbols $\mathcal{C} = \mathcal{A} + \mathcal{B}$, if $\mathcal{A} \cap \mathcal{B} = \varnothing$ and $\mathcal{C} = \{w \ : \ w \in \mathcal{A} \text{ or } w \in \mathcal{B}\}$. So, $\mathcal{C}$ contains every word in $\mathcal{A}$ and $\mathcal{B}$. Similarly, we say that $\mathcal{C}$ is the product of $\mathcal{A}$ and $\mathcal{B}$, in symbols $\mathcal{C} = \mathcal{A} \times \mathcal{B}$, if $\mathcal{C} = \{w\, v \ : \ w \in \mathcal{A} \text{ and } v \in \mathcal{B}\}$. Finally, we say that $\mathcal{C}$ is a *sequence* of $\mathcal{A}$'s, if $\mathcal{A} \neq \varnothing$ and $\mathcal{C} = \{w^i : w \in \mathcal{A} \text{ and } i \in \mathbb{N}\}$. The following well-known theorem says that all regular languages can be defined non-recursively in terms of the above three operations.

Theorem 2.1. *Let $\mathcal{A}$ be a regular language. Then $\mathcal{A}$ can be defined by a non-recursive specification involving the operations of disjoint union, product, and sequence. In particular, there are regular languages $\mathcal{A}_1, \ldots, \mathcal{A}_s$ such that $\mathcal{A} = \mathcal{A}_1$ and for all $2 \leq i \leq s$ the language $\mathcal{A}_i$ is either finite, or the disjoint union $\mathcal{A}_j + \mathcal{A}_{j'}$ where $j, j' > i$, or the product $\mathcal{A}_j \times \mathcal{A}_{j'}$ where $j, j' > i$, or the sequence $\mathsf{Seq}(\mathcal{A}_j)$ where $j > i$.*

In this paper, our focus is on regular languages, where in addition we allow the shuffle operation. Let Σ denote a finite alphabet. We will denote by Σ^* the set of all words (including the empty

one) containing letters only from Σ. Moreover, we will write uv for the concatenation of any two words $u, v \in \Sigma^*$. The *shuffle product* of $u, v \in \Sigma^*$ is defined as

$$u \sqcup\!\sqcup v = \{u_1 v_1 u_2 v_2 \dots u_n v_n \ : \ u = u_1 \dots u_n, v = v_1 \dots v_n, \forall 1 \le i \le n : u_i, v_i \in \Sigma^*\}$$

Analogously, the *shuffle* product of two languages $\mathcal{A}$ and $\mathcal{B}$ — in symbols $\mathcal{C} = \mathcal{A} \sqcup\!\sqcup \mathcal{B}$ — is, if the alphabets of $\mathcal{A}$ and $\mathcal{B}$ are disjoint, the language $\mathcal{A} \sqcup\!\sqcup \mathcal{B} = \{u \sqcup\!\sqcup w \ : \ u \in \mathcal{A}, w \in \mathcal{B}\}$. It is well-known that the class of regular languages is closed under the shuffle operator.

In the present work, we not only consider the shuffle as an operator *on* languages, but also as a grammar production rule to *define* languages.

2.2 Ordinary Boltzmann Samplers for Regular Languages

The (ordinary) generating function of a language $\mathcal{A}$ is the sum $A(z) = \sum_{w \in \mathcal{A}} z^{|w|} = \sum_{n \in \mathbb{N}} a_n z^n$, where $|w|$ is the length of the word, and a_n is the number of words in $\mathcal{A}$ of length n. In the case where there exists an unambiguous grammar to describe a regular language, one can automatically generate equations satisfied by generating function directly from the grammar:

$$\begin{array}{lllll} \mathcal{C} = & \mathbf{1} = \{\varepsilon\} & \Longrightarrow & C(z) = & 1 \\ \mathcal{C} = & \mathcal{Z} & \Longrightarrow & C(z) = & z \\ \mathcal{C} = & \mathcal{A} + \mathcal{B} & \Longrightarrow & C(z) = & A(z) + B(z) \\ \mathcal{C} = & \mathcal{A} \times \mathcal{B} & \Longrightarrow & C(z) = & A(z)B(z) \\ \mathcal{C} = & \mathsf{Seq}(\mathcal{A}) & \Longrightarrow & C(z) = & (1 - A(z))^{-1} \end{array}$$

Generating functions of formal languages are now a very established tool for algorithm analysis (see [FS08] for many references).

The ordinary Boltzmann distribution on $\mathcal{C}$ with parameter x is defined by

$$\mathbb{P}_x(\gamma) = \frac{x^{|\gamma|}}{C(x)} \tag{2.1}$$

if the above expression is well-defined. We say that an algorithm $\Gamma\mathcal{C}(x)$ is a *Boltzmann sampler* for $\mathcal{C}$ if the output distribution is equivalent to $\mathbb{P}_x$.

We now review very briefly how ordinary Boltzmann samplers can be constructed for regular languages, and we refer the reader to [DFLS04, FFP07] for a detailed discussion.

Let us begin with the case that $\mathcal{C}$ is the disjoint union of $\mathcal{A}$ and $\mathcal{B}$. Then, a ordinary Boltzmann sampler $\Gamma\mathcal{C}(x)$ for $\mathcal{C}$ flips a coin with an appropriate probability, and depending on the outcome it either generates a word from either $\mathcal{A}$ or $\mathcal{B}$.

Algorithm 1 $\Gamma\mathcal{C}(x)$: Boltzmann sampler $\Gamma\mathcal{C}$ for $\mathcal{C} = \mathcal{A} + \mathcal{B}$

Input: A real number x
Output: An object $\gamma \in \mathcal{C} = \mathcal{A} + \mathcal{B}$ with probability $\mathbb{P}_x(\gamma)$
Require: $0 \le x < \rho_C$
if $\mathsf{Bernoulli}\left(\frac{A(x)}{B(x)}\right)$ **then return** $\Gamma\mathcal{A}(x)$ **else return** $\Gamma\mathcal{B}(x)$ **end if**

If $\mathcal{C}$ is given by the product of $\mathcal{A}$ and $\mathcal{B}$, then a Boltzmann sampler initiates two independent calls to $\Gamma\mathcal{A}(x)$ and $\Gamma\mathcal{B}(x)$, and concatenates the resulting words.

Algorithm 2 $\Gamma\mathcal{C}(x)$: Boltzmann sampler $\Gamma\mathcal{C}$ for $\mathcal{C} = \mathcal{A} \times \mathcal{B}$

Input: A real number x
Output: An object $\gamma \in \mathcal{C} = \mathcal{A} \times \mathcal{B}$ with probability $\mathbb{P}_x(\gamma)$
Require: $0 \le x < \rho_C$
return $\Gamma\mathcal{A}(x) \cdot \Gamma\mathcal{B}(x)$

If $\mathcal{A} \neq \varnothing$, then the sequence construction can be described recursively in terms of disjoint union and product. Particularly, if $\mathbf{1}$ denotes the language that contains the empty word only, then $\mathcal{C} = \mathsf{Seq}(\mathcal{A}) = \mathbf{1} + \mathcal{A} \times \mathcal{C}$. This can be used to combine the two samplers given above in order to obtain a sampler for the sequence construction (notice that an alternative sampler for the sequence construction consists in drawing the number of copies of $\mathcal{A}$ according to a geometric law, and then draw them independently).

Algorithm 3 $\Gamma\mathcal{C}(x)$: Boltzmann sampler $\Gamma\mathcal{C}$ for $\mathcal{C} = \mathsf{Seq}(\mathcal{A})$

Input: A real number x
Output: An object $\gamma \in \mathcal{C} = \mathsf{Seq}(\mathcal{A})$ with probability $\mathbb{P}_x(\gamma)$
Require: $0 \leq x < \rho_C$
if $\mathsf{Bernoulli}\left(\frac{1}{C(x)}\right)$ **then return** ε **else return** $\Gamma\mathcal{A}(x) \cdot \Gamma\mathcal{C}(x)$ **end if**

Using the results from [DFLS04, FFP07] we immediately get the following statement.

Theorem 2.2. *Suppose that a language $\mathcal{C}$ is given by a finite specification involving the constructions of disjoint union, product, and sequence. Then, the Boltzmann generator $\Gamma\mathcal{C}(x)$ assembled from the specification of $\mathcal{C}$ given by the procedures above has a complexity, measured in the number of real-arithmetic operations, that is linear in the size of the generated string.*

Usually in random sampling, one wants to control the size: either generate an object of exact size n uniformly at random; or allow a relative tolerance ε on the size: an object of size $N \in [n(1-\varepsilon), n(1+\varepsilon)]$ has to be generated uniformly at random in its size class. Size control is made by repeatedly using a Boltzmann sampler and rejecting until an object of satisfactory size is obtained. The original paper [DFLS04] also provides general results of complexity for controlled-size, depending on the nature of the generating functions of the combinatorial classes. In particular:

Theorem 2.3. *For regular languages, Boltzmann samplers ensures linear real-arithmetic complexity, when a non-zero tolerance on size is allowed.*

3 Exponential-like Boltzmann Samplers, *a.k.a* ElBS

In this section we propose a general tool, that allows us to sample according to the (ordinary) Boltzmann distribution (2.1) by sampling from a different distribution, which we call *exponential-like*. Informally, in the exponential model, each object of size n is assigned a weight that is proportional to $1/n!$.

3.1 Exponential-like generating fuctions

Let us begin with introducing some notation. Let $\mathcal{A}$ be a class of combinatorial objects, $\mathcal{A}_n$ the subclass of objects of size n and a_n its cardinality. We define the following exponential-like generating functions:

$$\widehat{A}(x) = \widehat{A}^{(0)}(x) = \sum_{n\geq 0} \frac{a_n}{n!} x^n \quad \text{and} \quad \widehat{A}^{(k)}(x) = \frac{\mathrm{d}}{\mathrm{d}x}\widehat{A}^{(k-1)}(x) = \sum_{n\geq k} \frac{a_n}{(n-k)!} x^{n-k}$$

Barely speaking, the generating function $\widehat{A}^{(k)}$ counts the objects in class $\mathcal{A}$ with their weight shifted by a value k. In other words, to get the actual weight of an object γ, we use a *tare weight* of k.

Note that if $A(x)$ has a non-zero radius of convergence, then the functions $\widehat{A}^{(k)}(x)$ are entire, i.e. define functions that are analytic everywhere on the complex plane. Moreover, note that $\widehat{A}(x)$ is given by the well-known Laplace transform

$$A(x) = \sum_{n\geq 0} a_n x^n = \int_0^\infty \widehat{A}(xu) e^{-u} \mathrm{d}u \tag{3.1}$$

With this notation at hand, let the *exponential-like Boltzmann distribution* with parameter x and order k on $\mathcal{A}$ be defined by

$$\mathbb{P}_{x,k}(\gamma) = \begin{cases} 0 & \text{if } |\gamma| < k \\ \frac{x^{|\gamma|-k}}{(|\gamma|-k)!\widehat{A}^{(k)}(x)} & \text{if } |\gamma| \geq k \end{cases} \tag{3.2}$$

In complete analogy to ordinary Boltzmann samplers, we say that an algorithm $\widehat{\Gamma}\mathcal{A}(x,k)$ is an (x,k)-exponential-like Boltzmann sampler, if the output distribution is equivalent to (3.2). Moreover, whenever we write $\widehat{\Gamma}\mathcal{A}(x)$ we will silently assume that $k=0$.

As an illustration, we give the (x,k)-exponential-like Boltzmann samplers for the empty word, and for an atom. Notice that, in the following sections, because of consistent choices of k, the probability of getting an error is 0, in both algorithms.

Algorithm 4 $\widehat{\Gamma}\mathbf{1}(x,k)$: Exponential-like Boltzmann sampler for $\mathbf{1}$

Input: A real number x
Output: The word ε with probability $\mathbb{P}_{x,k}(\varepsilon)$
Require: $0 \leq x$
if $k=0$ **then return** the empty word ε **else return** ***error*** **end if**

Algorithm 5 $\widehat{\Gamma}\mathcal{Z}(x,k)$: Exponential-like Boltzmann sampler for $\mathcal{Z}$

Input: A real number x
Output: A letter z with probability $\mathbb{P}_{x,k}(z)$
Require: $0 \leq x$
if $k \leq 1$ **then return** the letter z **else return** ***error*** **end if**

3.2 Transformation into OBS

In this section, we prove that, starting from a exponential-like Boltzmann sampler $\widehat{\Gamma}\mathcal{C}(x)$ for a class $\mathcal{C}$, one can easily get an ordinary Boltzmann sampler $\Gamma\mathcal{C}(x)$ for $\mathcal{C}$, by computing $\widehat{\Gamma}\mathcal{C}(ux)$, with a probabilistic value for u.

Let x be such that $C(x) < \infty$; we define $\delta_x^{\mathcal{C}}$, which is a probability distribution on $[0, +\infty)$

$$\delta_x^{\mathcal{C}}(u) = \frac{\widehat{C}(xu)}{C(x)} e^{-u} \tag{3.3}$$

Let $R_x^{\mathcal{C}}$ be a random variable drawn according to the distribution defined by $\delta_x^{\mathcal{C}}$. Then the following simple algorithm shows how sampling according to the (ordinary) Boltzmann distribution can be reduced to sampling from the corresponding exponential-like distribution.

Algorithm 6 $\Gamma\mathcal{C}(x)$: Boltzmann sampler for $\Gamma\mathcal{C}$ being given $\widehat{\Gamma}\mathcal{C}$

Input: A real number x
Output: An object $\gamma \in \mathcal{C}$ with probability $\mathbb{P}_x(\gamma)$
Require: $0 \leq x < \rho_C$
1. $R_x^{\mathcal{C}} \leftarrow \mathsf{draw}\left(\delta_x^{\mathcal{C}}\right)$
2. **return** $\widehat{\Gamma}\mathcal{C}(xR_x^{\mathcal{C}})$

Proposition 3.1. *The algorithm $\Gamma\mathcal{C}(x)$ is an ordinary Boltzmann sampler for $\mathcal{C}$.*

Proof. Let $\gamma \in \mathcal{C}$ be such that $|\gamma| = n$. Then,

$$\mathbb{P}\left(\Gamma\mathcal{C}(x) = \gamma\right) = \mathbb{E}\left(\mathbb{P}\left(\widehat{\Gamma}\mathcal{C}(xR_x^{\mathcal{C}}) = \gamma\right)\right) = \int_0^\infty \mathbb{P}\left(\widehat{\Gamma}\mathcal{C}(xu) = \gamma\right) \delta_x^{\mathcal{C}}(u)\mathrm{d}u$$

Note that $\mathbb{P}\left(\widehat{\Gamma}\mathcal{C}(xu) = \gamma\right) = \frac{(xu)^n}{n!\widehat{C}(xu)}$, so that the above expression is equal to $\frac{x^n}{C(x)}$. □

Remark. Notice that the inverse transformation is impossible, since there is no probability measure with moments $1/n!$

4 ElBS for regular languages with shuffle

In this section we construct exponential-like Boltzmann samplers for regular languages with shuffle. In particular, we show that, assuming we are given the two ElBS $\widehat{\Gamma}\mathcal{A}$ and $\widehat{\Gamma}\mathcal{B}$, then we can deduce the ElBS $\widehat{\Gamma}(\mathcal{A}+\mathcal{B})$, $\widehat{\Gamma}(\mathcal{A}\times\mathcal{B})$ and $\widehat{\Gamma}(\mathsf{Seq}(\mathcal{A}))$.

4.1 Shuffle

We first consider the shuffle construction, and construct an exponential-like Boltzmann sampler for the shuffle of two languages.

From the definition of the shuffle operation on languages recalled in section2, it is easy to show (see e.g.[FGT92]) that if $\mathcal{C} = \mathcal{A} \sqcup\!\sqcup \mathcal{B}$, where the alphabets of $\mathcal{A}$ and $\mathcal{B}$ are disjoint. Then

$$\widehat{C}(x) = \widehat{A}(x)\widehat{B}(x). \tag{4.1}$$

In the following, we shall need an algorithm for generating a random shuffle of two given words:

Algorithm 7 shuffle(u, v): Shuffle algorithm for two given words

Input: Two words u and v
Output: A word drawn uniformly from $u \sqcup\!\sqcup v$
Require: The alphabets of u and v are disjoints
1. Choose uniformly at random $|u|$ out of $|u| + |v|$ positions
2. **return** the word w corresponding to the shuffle of u and v according to these positions

Clearly, if we assume that u and v contain distinct letters, shuffle(u, v) generates all $\binom{|u|+|v|}{|u|}$ strings in $u \sqcup\!\sqcup v$ with the same probability.

In order to sample from the shuffle of two languages we need to express the involed distribution, obtained by differentiating k times equation (4.1). Let $D^{\mathcal{C}}_{k,x}$ be a random variable with distribution

$$\mathbb{P}\left(D^{\mathcal{C}}_{k,x} = \ell\right) = \frac{1}{\widehat{C}^{(k)}(x)}\binom{k}{\ell}\widehat{A}^{(\ell)}(x)\widehat{B}^{(k-\ell)}(x) \quad \text{where} \quad \ell \in \{0, \dots, k\}$$

With this notation at hand, an (x, k)-exponential-like Boltzmann sampler for the shuffle of two languages is given by the following algorithm.

Algorithm 8 $\widehat{\Gamma}\mathcal{C}(x, k)$: Exponential-like Bolzmann sampler for $\mathcal{C} = \mathcal{A} \sqcup\!\sqcup \mathcal{B}$

Input: A real x and an integer k
Output: A word γ from $\mathcal{C} = \mathcal{A} \sqcup\!\sqcup \mathcal{B}$ with probability $\mathbb{P}_{x,k}(\gamma)$
Require: $0 \le x < \rho_{\mathcal{C}}$ and $k \in \mathbb{N}$
1. $\ell \leftarrow \mathsf{draw}\left(D^{\mathcal{C}}_{k,x}\right)$
2. $\gamma_{\mathcal{A}} \leftarrow \widehat{\Gamma}\mathcal{A}(x, \ell)$
3. $\gamma_{\mathcal{B}} \leftarrow \widehat{\Gamma}\mathcal{B}(x, k-\ell)$
4. $n_{\mathcal{A}} \leftarrow |\gamma_{\mathcal{A}}|$ and $n_{\mathcal{B}} \leftarrow |\gamma_{\mathcal{B}}|$
5. Let $b_{\mathcal{A}}e_{\mathcal{A}} = \gamma_{\mathcal{A}}$ where $|b_{\mathcal{A}}| = \ell$
6. Let $b_{\mathcal{B}}e_{\mathcal{B}} = \gamma_{\mathcal{B}}$ where $|b_{\mathcal{B}}| = k - \ell$
7. $b \leftarrow$ shuffle$(b_{\mathcal{A}}, b_{\mathcal{B}})$
8. $e \leftarrow$ shuffle$(e_{\mathcal{A}}, e_{\mathcal{B}})$
9. **return** the word be

Proposition 4.1. *The algorithm $\widehat{\Gamma}\mathcal{C}(x,k)$ is an (x,k)-exponential-like Boltzmann sampler for $\mathcal{C}$, where $\mathcal{C} = \mathcal{A} \sqcup\!\sqcup \mathcal{B}$, and the alphabets of $\mathcal{A}$ and $\mathcal{B}$ are disjoint.*

Proof. Let $\gamma \in \mathcal{C} \setminus (\mathcal{C}_0 \cup \cdots \cup \mathcal{C}_{k-1})$, and let us write $\gamma = b\,e$, where $|b| = k$. The definition of the shuffle product implies that there are unique $\gamma_{\mathcal{A}} \in \mathcal{A}$ and $\gamma_{\mathcal{B}} \in \mathcal{B}$ such that γ is obtained from $\gamma_{\mathcal{A}}$ and $\gamma_{\mathcal{B}}$ by shuffling, i.e. by selecting $n_{\mathcal{A}}$ specific positions out of $n = n_{\mathcal{A}} + n_{\mathcal{B}}$ positions, where we place the letters of $\gamma_{\mathcal{A}}$. In particular, there is a unique $0 \le \ell \le k$ such that b contains the first ℓ letters of $\gamma_{\mathcal{A}}$, and the remaining letters are the first $k-\ell$ letters from $\gamma_{\mathcal{B}}$.

The above considerations imply that $\widehat{\Gamma}\mathcal{C}(x,k)$ outputs γ if and only if *i)* the random variable $D^{\mathcal{C}}_{k,x}$ evaluates to ℓ, *ii)* $\widehat{\Gamma}\mathcal{A}(x,\ell)$ generates $\gamma_{\mathcal{A}}$, *iii)* $\widehat{\Gamma}\mathcal{B}(x,k-\ell)$ generates $\gamma_{\mathcal{B}}$, *iv)* the shuffle of $b_{\mathcal{A}}, b_{\mathcal{B}}$ is b and finally, *v)* the shuffle of $e_{\mathcal{A}}, e_{\mathcal{B}}$ is e. Since *i)* – *v)* are all independent events,

$$\mathbb{P}\left(\widehat{\Gamma}\mathcal{C}(x,k) = \gamma\right) = \mathbb{P}\left(D^{\mathcal{C}}_{k,x} = \ell\right)\mathbb{P}\left(\widehat{\Gamma}\mathcal{A}(x,\ell) = \gamma_{\mathcal{A}}\right)\mathbb{P}\left(\widehat{\Gamma}\mathcal{B}(x,k-\ell) = \gamma_{\mathcal{B}}\right)\cdot\frac{1}{\binom{k}{\ell}}\frac{1}{\binom{n-k}{n_{\mathcal{A}}-\ell}}$$

As $\widehat{\Gamma}\mathcal{A}(x,\ell)$ and $\widehat{\Gamma}\mathcal{B}(x,k-\ell)$ are exponential-like Boltzmann samplers, we infer that the above probability is equal to

$$\frac{\binom{k}{\ell}\widehat{A}^{(\ell)}(x)\widehat{B}^{(k-\ell)}(x)}{\widehat{C}^{(k)}(x)}\frac{x^{n_{\mathcal{A}}-\ell}}{(n_{\mathcal{A}}-\ell)!\widehat{A}^{(\ell)}(x)}\frac{x^{n_{\mathcal{B}}-(k-\ell)}}{(n_{\mathcal{B}}-(k-\ell))!\widehat{B}^{(k-\ell)}(x)}\cdot\frac{1}{\binom{k}{\ell}}\frac{1}{\binom{n-k}{n_{\mathcal{A}}-\ell}}$$

This readily simplifies to $\frac{x^{n-k}}{(n-k)!\widehat{C}^{(k)}(x)}$ and the proof is completed. □

4.2 Disjoint Union

We construct an exponential-like Boltzmann sampler for a language $\mathcal{C}$ defined as the disjoint union of two other languages, i.e. $\mathcal{C} = \mathcal{A} + \mathcal{B}$. . The exponential-like generating function satisfies $\widehat{C}(z) = \widehat{A}(z) + \widehat{B}(z)$, and the sampler works very similar to the ordinary case.

Algorithm 9 $\widehat{\Gamma}\mathcal{C}(x,k)$: Exponential-like Bolzmann sampler for $\mathcal{C} = \mathcal{A} + \mathcal{B}$

Input: A real x and an integer k
Output: A word γ from $\mathcal{C} = \mathcal{A} + \mathcal{B}$ with probability $\mathbb{P}_{x,k}(\gamma)$
Require: $0 \le x < \rho_{\mathcal{C}}$ and $k \in \mathbb{N}$
if Bernoulli $\left(\frac{\widehat{A}^{(k)}(x)}{\widehat{C}^{(k)}(x)}\right)$ **then return** $\widehat{\Gamma}\mathcal{A}(x,k)$ **else return** $\widehat{\Gamma}\mathcal{B}(x,k)$ **end if**

Proposition 4.2. *The algorithm $\widehat{\Gamma}\mathcal{C}(x,k)$ is an (x,k)-exponential-like Boltzmann sampler for $\mathcal{C}$, where $\mathcal{C} = \mathcal{A} + \mathcal{B}$, and $\mathcal{A} \cap \mathcal{B} = \varnothing$.*

Proof. $\widehat{\Gamma}\mathcal{C}(x,k)$ outputs $\gamma \in \mathcal{A}$ iff the Bernoulli choice succeeds, and $\widehat{\Gamma}\mathcal{A}(x,k)$ returns γ. So,

$$\mathbb{P}\left(\widehat{\Gamma}\mathcal{C}(x,k) = \gamma\right) = \frac{\widehat{A}^{(k)}(x)}{\widehat{C}^{(k)}(x)}\cdot\frac{x^{|\gamma|-k}}{(|\gamma|-k)!\widehat{A}^{(k)}(x)} = \frac{x^{|\gamma|-k}}{(|\gamma|-k)!C^{(k)}(x)}$$

as claimed. The proof for the case $\gamma \in \mathcal{B}$ is completely analogous. □

4.3 Product

Let a language $\mathcal{C}$ be product of two other languages, i.e. $\mathcal{C} = \mathcal{A} \times \mathcal{B}$. The description of an exponential-like Boltzmann sampler for this construction relies on expressing the exponential-like generating functions, and the involved probabilities. We first state a standard in the context of Laplace transforms, which will be repeatedly used in this section.

Lemma 4.3. *Let $p, q \in \mathbb{N}$, $x \ge 0$. Then* $\displaystyle\int_0^x (x-t)^p t^q \,\mathrm{d}t = \frac{p!q!}{(p+q+1)!}x^{p+q+1}$.

Lemma 4.4. *Suppose that* $\mathcal{C} = \mathcal{A} \times \mathcal{B}$. *Then* $\widehat{C}(x) = |\mathcal{A}_0|\,\widehat{B}(x) + \displaystyle\int_0^x \widehat{A}'(x-t)\widehat{B}(t)\,\mathrm{d}t$, *and more generally* $\widehat{C}^{(k)}(x) = \displaystyle\sum_{\ell=0}^{k} |\mathcal{A}_\ell|\widehat{B}^{(k-\ell)}(x) + \int_0^x \widehat{A}^{(k+1)}(x-t)\widehat{B}(t)\,\mathrm{d}t$.

Proof. First, admit the first of these two equations, and argue that

$$\widehat{C}(x) = \frac{\partial}{\partial x}\int_0^x \widehat{A}(x-t)\widehat{B}(t)\,\mathrm{d}t = |\mathcal{A}_0|\widehat{B}^{(0)}(x) + \int_0^x \widehat{A}^{(1)}(x-t)\widehat{B}(t)\,\mathrm{d}t$$

By differentiating both sides of the above equation k times with respect to x yields immediately the second equation of the lemma.

Let us now prove the first equation. First of all, note that the definition of $\mathcal{C}$ implies that

$$\widehat{C}(x) = \sum_{n\geq 0} \frac{c_n}{n!}x^n, \text{ where } c_n = [x^n]C(x) = \sum_{s=0}^{n} a_s b_{n-s} \tag{4.2}$$

Moreover,

$$\int_0^x \widehat{A}(x-t)\widehat{B}(t)\,\mathrm{d}t = \int_0^x \sum_{n\geq 0}\sum_{s=0}^{n} \frac{a_s}{s!}\frac{b_{n-s}}{(n-s)!}(x-t)^s t^{n-s}\,\mathrm{d}t = \sum_{n\geq 0}\sum_{s=0}^{n}\frac{a_s}{s!}\frac{b_{n-s}}{(n-s)!}\int_0^x (x-t)^s t^{n-s}\,\mathrm{d}t$$

By applying Proposition 4.3 to the last integral, where we set $p = s$ and $q = n - s$, we infer that

$$\int_0^x \widehat{A}(x-t)\widehat{B}(t)\,\mathrm{d}t = \sum_{n\geq 0}\left(\sum_{s=0}^{n}\frac{a_s b_{n-s}}{(n+1)!}\right)x^{n+1}$$

The proof completes by differentiating and comparing this expression with (4.2). □

Before we construct an exponential-like Boltzmann sampler for the product of two languages we need to express the involved distributions. Let $P_{k,x}^{\mathcal{C}}$ be a discrete random variable drawn according to the distribution

$$\mathbb{P}\left(P_{k,x}^{\mathcal{C}} = \ell\right) = \frac{1}{\widehat{C}^{(k)}(x)} \cdot \begin{cases} |\mathcal{A}_\ell|\widehat{B}^{(k-\ell)}(x) & \text{if } \ell \in \{0,\dots,k\} \\ \int_0^x \widehat{A}^{(k+1)}(x-t)\widehat{B}(t)\,\mathrm{d}t & \text{if } \ell = k+1 \end{cases}$$

Moreover, let $B_{k,x}^{\mathcal{C}}$ be a continuous random variable whose probability density function is

$$t \mapsto \frac{\widehat{A}^{(k+1)}(x-t)\widehat{B}(t)}{\int_0^x \widehat{A}^{(k+1)}(x-u)\widehat{B}(u)\,\mathrm{d}u} \quad \text{where } 0 \leq t \leq x \tag{4.3}$$

With the above notation at hand, an exponential-like Boltzmann sampler for $\mathcal{C}$ works as follows.

Algorithm 10 $\widehat{\Gamma}\mathcal{C}(x,k)$: Exponential-like Bolzmann sampler for $\mathcal{C} = \mathcal{A} \times \mathcal{B}$

Input: A real x and an integer k
Output: A word γ from $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ with probability $\mathbb{P}_{x,k}(\gamma)$
Require: $0 \leq x < \rho_{\mathcal{C}}$ and $k \in \mathbb{N}$

1. $\ell \leftarrow \mathsf{draw}\left(P_{k,x}^{\mathcal{C}}\right)$
2. **if** $0 \leq \ell \leq k$ **then**
3. Draw $\gamma_{\mathcal{A}} \in \mathcal{A}_\ell$ uniformly
4. $\gamma_{\mathcal{B}} \leftarrow \Gamma\mathcal{B}(x, k-\ell)$
5. **else**
6. $t \leftarrow \mathsf{draw}\left(B_{k,x}^{\mathcal{C}}\right)$
7. $\gamma_{\mathcal{A}} \leftarrow \widehat{\Gamma}\mathcal{A}(x-t, k+1)$
8. $\gamma_{\mathcal{B}} \leftarrow \widehat{\Gamma}\mathcal{B}(t, 0)$
9. **end if**
10. **return** the word $\gamma_{\mathcal{A}}\gamma_{\mathcal{B}}$

Proposition 4.5. *The algorithm $\widehat{\Gamma}\mathcal{C}(x,k)$ is an (x,k)-exponential-like Boltzmann sampler for $\mathcal{C}$, where $\mathcal{C} = \mathcal{A} \times \mathcal{B}$.*

Proof. Let $\gamma \in \mathcal{C}$. Then there are unique $\gamma_{\mathcal{A}} \in \mathcal{A}$ and $\gamma_{\mathcal{B}} \in \mathcal{B}$ such that $\gamma = \gamma_{\mathcal{A}}\,\gamma_{\mathcal{B}}$. We shall begin with the case $\ell = |\gamma_{\mathcal{A}}| \leq k$. The definition of $\widehat{\Gamma}\mathcal{C}(x,k)$ implies that, with probability 1, it returns γ if and only if: (i) $P^{\mathcal{C}}_{k,x} = \ell$, (ii) a uniform random string from $\mathcal{A}_\ell$ is $\gamma_{\mathcal{A}}$, and (iii) $\widehat{\Gamma}\mathcal{B}(x, k-\ell)$ generates $\gamma_{\mathcal{B}}$. So that

$$\mathbb{P}\left(\widehat{\Gamma}\mathcal{C}(x,k) = \gamma\right) = \frac{|\mathcal{A}_\ell|\widehat{B}^{(k-\ell)}(x)}{\widehat{C}^{(k)}(x)} \cdot \frac{1}{|\mathcal{A}_\ell|} \cdot \mathbb{P}\left(\widehat{\Gamma}B(x,k-\ell) = \gamma_{\mathcal{B}}\right)$$

Since $\widehat{\Gamma}B$ is an exponential-like Boltzmann sampler we readily infer that the above probability is

$$\frac{\widehat{B}^{(k-\ell)}(x)}{\widehat{C}^{(k)}(x)} \cdot \frac{x^{|\gamma_{\mathcal{B}}|-(k-\ell)}}{(|\gamma_{\mathcal{B}}|-(k-\ell))!\widehat{B}^{(k-\ell)}(x)}$$

Note that $|\gamma_{\mathcal{B}}|-(k-\ell) = (|\gamma|-|\gamma_{\mathcal{A}}|)-(k-\ell) = |\gamma|-k$. By plugging this into the above expression we see that $\widehat{\Gamma}\mathcal{C}(x,k)$ is indeed a Boltzmann sampler whenever $|\gamma_{\mathcal{A}}| \leq k$.

Let us now treat the case $|\gamma_{\mathcal{A}}| > k$. There, $\widehat{\Gamma}\mathcal{C}(x,k)$ generates γ if and only if: (i) $P^{\mathcal{C}}_{k,x} = k+1$, (ii) $\widehat{\Gamma}\mathcal{A}(x - B^{\mathcal{C}}_{k,x}, k+1) = \gamma_{\mathcal{A}}$, where $B^{\mathcal{C}}_{k,x}$ is distributed as in (4.3), and (iii) $\widehat{\Gamma}\mathcal{B}(B^{\mathcal{C}}_{k,x}, 0)$ generates $\gamma_{\mathcal{B}}$. Let us abbreviate $N = \int_0^x \widehat{A}^{(k+1)}(x-t)\widehat{B}(t)\,\mathrm{d}t$ and $B = B^{\mathcal{C}}_{k,x}$. We deduce then that

$$\mathbb{P}\left(\widehat{\Gamma}\mathcal{C}(x,k) = \gamma\right) = \frac{N}{C^{(k)}(x)} \cdot \mathbb{E}\left(\mathbb{P}\left(\widehat{\Gamma}\mathcal{A}(x-B,k+1) = \gamma_{\mathcal{A}}\right)\mathbb{P}\left(\widehat{\Gamma}\mathcal{B}(B) = \gamma_{\mathcal{B}}\right)\right) \tag{4.4}$$

where the expectation is taken over the random choice of B. Since $\widehat{\Gamma}\mathcal{A}$ and $\widehat{\Gamma}\mathcal{B}$ are exponential-like Boltzmann samplers we infer that for any $0 \leq t \leq x$

$$\mathbb{P}\left(\widehat{\Gamma}\mathcal{A}(x-t,k+1) = \gamma_{\mathcal{A}}\right)\mathbb{P}\left(\widehat{\Gamma}\mathcal{B}(t) = \gamma_{\mathcal{B}}\right) = \frac{(x-t)^{|\gamma_{\mathcal{A}}|-(k+1)}t^{|\gamma_{\mathcal{B}}|}}{(|\gamma_{\mathcal{A}}|-(k+1))!(|\gamma_{\mathcal{B}}|)!}\frac{1}{\widehat{A}^{(k+1)}(x-t)\widehat{B}(t)}$$

Using (4.3) we thus may infer that the expectation in (4.4) equals

$$\frac{1}{N}\frac{1}{(|\gamma_{\mathcal{A}}|-(k+1))!|\gamma_{\mathcal{B}}|!}\int_0^x (x-t)^{|\gamma_{\mathcal{A}}|-(k+1)}t^{|\gamma_{\mathcal{B}}|}\,\mathrm{d}t$$

By applying Proposition 4.3 with $p = |\gamma_{\mathcal{A}}|-(k+1)$ and $q = |\gamma_{\mathcal{B}}|$ we infer that the integral is equal to $x^{p+q+1}/(p+q+1)!$, and the fact that $p+q+1 = |\gamma_{\mathcal{A}}| + |\gamma_{\mathcal{B}}| - k = |\gamma| - k$ implies with (4.4) that $\mathbb{P}\left(\widehat{\Gamma}\mathcal{C}(x,k) = \gamma\right) = \dfrac{x^{|\gamma|-k}}{(|\gamma|-k)!\widehat{C}^{(k)}(x)}$, as claimed. □

4.4 Sequence

Suppose that a language is given by the sequence construction, i.e. $\mathcal{C} = \mathbf{1} + \mathcal{A} \times \mathcal{C}$, where we assume that $\mathcal{A}_0 = \varnothing$. An exponential-like Boltzmann sampler for $\mathcal{C}$ can be easily composed from the results in the previous sections.

By applying Lemma 4.4 and using the fact $\mathcal{A}_0 = \varnothing$ we infer that $\widehat{C}^{(k)}(x)$ satisfies the relation

$$\widehat{C}^{(k)}(x) = \frac{\partial^k}{\partial x^k}1 + \sum_{\ell=1}^{k}|\mathcal{A}_\ell|\widehat{C}^{(k-\ell)}(x) + \int_0^x \widehat{A}^{(k+1)}(x-t)\widehat{C}(t)\,\mathrm{d}t$$

Let $S^{\mathcal{C}}_{k,x}$ be a random variable with distribution

$$\mathbb{P}\left(S^{\mathcal{C}}_{k,x} = \ell\right) = \frac{1}{\widehat{C}^{(k)}(x)} \cdot \begin{cases} \frac{\partial^k}{\partial x^k}1 & \text{if } \ell = 0 \\ |\mathcal{A}_\ell|\widehat{C}^{(k-\ell)}(x) & \text{if } \ell \in \{1,\ldots,k\} \\ \int_0^x \widehat{A}^{(k+1)}(x-t)\widehat{C}(t)\,\mathrm{d}t & \text{if } \ell = k+1 \end{cases}$$

Moreover, let $B^{\mathcal{C}}_{k,x}$ be a random variable as in (4.3), where the function $\widehat{B}$ is replaced by $\widehat{C}$. Then an exponential-like Boltzmann sampler for the sequence construction is given by:

Algorithm 11 $\widehat{\Gamma}\mathcal{C}(x,k)$: Exponential-like Bolzmann sampler for $\mathcal{C} = \mathsf{Seq}(\mathcal{A})$

Input: A real x and an integer k
Output: A word γ from $\mathcal{C} = \mathsf{Seq}(\mathcal{A})$ with probability $\mathbb{P}_{x,k}(\gamma)$
Require: $0 \leq x < \rho_{\mathcal{C}}$ and $k \in \mathbb{N}$

1. $\ell \leftarrow \mathsf{draw}\left(S^{\mathcal{C}}_{k,x}\right)$
2. **if** $\ell = 0$ **then**
3. **return** ε
4. **else if** $1 \leq \ell \leq k$ **then**
5. Draw $\gamma_{\mathcal{A}} \in \mathcal{A}_\ell$ uniformly
6. $\gamma_{\mathcal{C}} \leftarrow \widehat{\Gamma}\mathcal{C}(x, k-\ell)$
7. **else**
8. $t \leftarrow \mathsf{draw}\left(B^{\mathcal{C}}_{k,x}\right)$
9. $\gamma_{\mathcal{A}} \leftarrow \widehat{\Gamma}\mathcal{A}(x-t, k+1)$
10. $\gamma_{\mathcal{C}} \leftarrow \widehat{\Gamma}\mathcal{C}(t, 0)$
11. **end if**
12. **return** the word $\gamma_{\mathcal{A}}\gamma_{\mathcal{C}}$

Proposition 4.6. *The algorithm $\widehat{\Gamma}\mathcal{C}(x,k)$ is an (x,k)-exponential-like Boltzmann sampler for $\mathcal{C}$, where $\mathcal{C}$ is given by $\mathcal{C} = \mathsf{Seq}(\mathcal{A})$.*

4.5 Boltzmann Sampler for Regular Languages with Shuffle

Putting everything together, we can now state our main result.

Theorem 4.7. *Suppose that $\mathcal{C}$ is a regular language with shuffle. Then, there is a Boltzmann generator $\Gamma\mathcal{C}(x)$ with a complexity measured in the number of real-arithmetic operations and evaluations of generating functions that is linear in the size of the generated string. Moreover, $\Gamma\mathcal{C}(x)$ can be implemented in terms of $O(s^2)$ samplers, where s is the number of languages appearing in the specification of $\mathcal{C}$.*

Proof. By applying Lemma 3.1 we infer that it is enough to show the conclusion for a sampler $\widehat{\Gamma}\mathcal{C}(x,k)$, where $x \in (0,\infty)$ and $k \in \mathbb{N}$, with output distribution as in (3.2).

Then the proof is essentially the same as in [DFLS04]: we have a unique parse tree for each word, linear in its size, and on each vertex we make one choice.

To complete the proof we show that $\Gamma\mathcal{C}(x)$ can be implemented in terms of $O(s^2)$ samplers, where s is the number of languages appearing in the specification of $\mathcal{C}$. Note that it is enough to show the same statement for $\widehat{\Gamma}\mathcal{C}(x)$. Let $\mathcal{C}_1, \ldots, \mathcal{C}_s$, where $\mathcal{C}_1 = \mathcal{C}$, be the languages appearing in the specification of $\mathcal{C}$, and suppose that for every $i \geq 2$ the language C_i is either finite or it depends only on C_{i+j}, where $j > 0$. Then, a closer inspection of the samplers of sections 4 reveals that a (x,k)-exponential-like Boltzmann sampler for $\mathcal{C}_i$ makes use only of (x',ℓ)-exponential-like samplers for $\mathcal{C}_{i+1}, \ldots, \mathcal{C}_s$, where $x' \in \mathbb{R}$, and $\ell \leq k+1$. So, the largest q for which an (x,q)-exponential-like sampler will be used is s, and this completes the proof of the theorem.

□

Finally the complexity for controlling the size is also the same as in the classical case [DFLS04]:

Corollary 4.8. *The approximate sampling of regular with shuffle languages is linear in the size of the generated output.*

5 Concluding remarks and work in progress

This paper presents a linear time method for the random generation of regular specifications including shuffle. The main idea is to modify the Boltzmann parameter according to a well-chosen density, in order to modify the size distribution of the output (without any bias on a given size, so that uniformity is preserved). Thus transforming a Boltzman generator for combinatorial structures with a given distribution of sizes, into a Boltzman generator for the same combinatorial objects, with a different distribution of sizes. This idea can be adapted to other contexts, such as differential specifications [**?**].

We finally discuss two topics that constitute a completion and an extension of this work, concerning the implementation issues and the combinatorial interpretation.

5.1 Experimentation and algorithmic issues

The implementation of the previous algorithms is quite straightforward. We managed to write a fully functionnal prototype using Maple. The major problem which needed to be addressed is a way to draw the various random variables according to nonstandard density functions. Fortunately, as our random variables are either discrete and with finite support, or with a non-implicit probability function, Maple is able to draw automatically according to these given distributions.

Hence, we are able with our prototype to ask Maple to draw according to a given specification. Consider for example the following extract from a worksheet :

```
> S:=Shuffle(Seq(Union(a, b)), Prod(c, d)):
> OGF(S)(z) assuming (z<1/2);
```

$$\frac{z^2}{(1-2z)(1-4z+4z^2)}$$

```
> Gamma(S)(0.4);
```

$$abaabaacbaabdbaa$$

The main problem here is that this prototype is quite slow. Indeed, Maple has to carry all the computations (including a massive utilisation of differentiations, convolutions and integrations) on a symbolic level back and forth the worlds of ordinary and exponential generating functions. However, since we are dealing with regular languages only here, all the ordinary generating functions are rational functions, even if their exponential counterparts are much more complex.

So, an idea to improve the speed could be to avoid, during the calculations, the shift from ordinary world to exponential, then from exponential to ordinary. Precisely, one could assume that the input generating functions are given in partial fraction decomposition, and deduce the output in the form of a rational fraction. For example, suppose that $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ and

$$A(x) = \frac{N_{\mathcal{A}}(x)}{D_{\mathcal{A}}(x)} \quad ; \quad B(x) = \frac{N_{\mathcal{B}}(x)}{D_{\mathcal{B}}(x)} \quad ; \quad C(x) = \frac{N_{\mathcal{C}}(x)}{D_{\mathcal{C}}(x)}$$

with $N_{\mathcal{X}}, D_{\mathcal{X}} \in \mathbb{R}[X]$ for any $\mathcal{X} \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$. Then, we would need a simple algorithm for deducing the actual values of $N_{\mathcal{C}}$ and $D_{\mathcal{C}}$ from those of $N_{\mathcal{A}}$, $D_{\mathcal{A}}$, $N_{\mathcal{B}}$ and $D_{\mathcal{B}}$. We do have an answer when all the poles of A and B are simples, and there the development of this idea is a work in progress.

5.2 Combinatorial interpretation

In this section, we are going to show how to interpret combinatorialy the construction of the algorithms presented in section 4. To do so, we will first reinterpret the exponential-like Boltzmann samplers of section 3.

One can interpret differently the generating function $\widehat{A}$. Rather than an alternative exponential generating function for class $\mathcal{A}$, we can see it as the usual exponential generating function of a well-defined labelled class built upon $\mathcal{A}$. Indeed we define here the *canonicaly labelled class* $\widehat{\mathcal{A}}$ as

the class of words in $\mathcal{A}$ with the following unique labelling: the i-th letter receives the label i. Then, we have that $\mathrm{ElGF}(\mathcal{A}) = \mathrm{EGF}(\widehat{\mathcal{A}})$.

For example, if $\mathcal{A} = \{ab, bb\}$ then $\widehat{\mathcal{A}} = \{a_1b_2, b_1b_2\}$ and we have $\mathrm{OGF}(\mathcal{A})(z) = 2z^2$ while $\mathrm{ElGF}(\mathcal{A})(z) = 2\frac{z^2}{2!} = z^2 = \mathrm{EGF}(\widehat{\mathcal{A}})(z)$.

Furthermore, the derivation operation on exponential-like generating functions can be seen combinatorily as the derivation on species (see [BLL98]). More simply, in our context, we can view a word in $\widehat{\mathcal{A}^{(k)}}$ as a word in $\mathcal{A}$ with an increasing labelling starting from the $(k+1)$-th position. With our previous example, we have $\widehat{\mathcal{A}^{(1)}} = \{ab_1, bb_1\}$ and $\widehat{\mathcal{A}^{(2)}} = \{ab, bb\}$. Their EGF is $A^{(1)}(z) = 2z$ and $A^{(2)}(z) = 2$. Note that $\mathcal{A} = \widehat{\mathcal{A}^{(2)}}$, but $A(z) \neq A^{(2)}(z)$. As usual with labelled classes, the actual size of an object is nothing but the number of labels it holds.

One should notice that $\widehat{\Gamma}\mathcal{A} = \Gamma\widehat{\mathcal{A}}$, the latest Γ being the usual exponential Boltzmann sampler. The advantage of going through this whole new world is that the shuffle product of unlabelled classes is transformed into the usual labelled product on the corresponding canonicaly labelled classes. In other words, we have $\widehat{\mathcal{A} \sqcup\!\sqcup \mathcal{B}} = \widehat{\mathcal{A}} \star \widehat{\mathcal{B}}$.

For example, if $\mathcal{A} = \{ab\}$ and $\mathcal{C} = \{c\}$, then $\widehat{\mathcal{A}} = \{a_1b_2\}$ and $\widehat{\mathcal{B}} = \{c_1\}$. Next, we have $\widehat{\mathcal{A}} \star \widehat{\mathcal{B}} = \{(a_1b_2, c_3), (a_1b_3, c_2), (a_2b_3, c_1)\} = \{a_1b_2c_3, a_1c_2b_3, c_1a_2b_3\} = \widehat{\mathcal{A} \sqcup\!\sqcup \mathcal{B}}$.

The issue rising here is that we can't translate the specification for $\mathcal{A}$ to a specification for $\widehat{\mathcal{A}}$ involving only operators handled by the Boltzmann samplers framework. Indeed, if $\mathcal{C} = \mathcal{A} \times \mathcal{B}$, then the corresponding canonicaly labelled class $\widehat{\mathcal{C}}$ can only be expressed as $\widehat{\mathcal{C}} = \widehat{\mathcal{A}} \odot \widehat{\mathcal{B}}$ where $\odot$ is the *ordinal product* on labelled structures — more precisely on linear species as explained in [BLL98] — defined comprehensively as follows: if $\widehat{\mathcal{A}} = \{a_1b_2, b_1b_2a_3\}$ and $\widehat{\mathcal{B}} = \{c_1d_2\}$, then $\widehat{\mathcal{A}} \odot \widehat{\mathcal{B}} = \{a_1b_2c_3d_4, b_1b_2a_3c_4d_5\}$: we concatenate the words of the two classes, and shift the labels of the second part.

In order to get the generating function of the ordinal product of $\widehat{\mathcal{A}}$ and $\widehat{\mathcal{B}}$ as given in lemma 4.4, we use the relation $\mathrm{EGF}\left(\widehat{\mathcal{A}} \odot \widehat{\mathcal{Z}} \odot \widehat{\mathcal{B}}\right)(z) = \int_0^z \widehat{A}(z-t)\widehat{B}(t)\,\mathrm{d}t$ given in [BLL98] and the transformation

$$\widehat{\mathcal{A}} \odot \widehat{\mathcal{B}} = \widehat{\mathcal{A}^{(1)}} \odot \widehat{\mathcal{Z}} \odot \widehat{\mathcal{B}} + \widehat{\mathcal{A}_0} \star \widehat{\mathcal{B}} \tag{5.1}$$

This last relation being true by a simple case analysis: for objects of size 0 in $\mathcal{A}$, the labelled product is equivalent to the labelled product. And for any object of size > 0, it can be uniquely decomposed as itself without it first label, and one isolated label. We now can translate any specification of a regular language into a specification of canonicaly labelles objects with derivatives.

In order to construct the $\widehat{\Gamma}$ sampler for regular languages, we can use the usual exponential Boltzmann samplers extended with ordinal product and derivatives. We will travel through the algorithms of section 4.

Algorithm 8 is not much more than the classic Boltzmann sampler for labelled product where we independantly draw an element from each component of the product (lines 2 and 3). Le rest of the algorithm simply deals with the labelling of the final object, where the shuffle really takes place.

Algorithm 9 is exactly the classic Bolzmann sampler for disjoint union, where we choose with a Bernoulli law one of the two components of the sum.

The algorithm 10 for the ordinal product follows the transformation 5.1 and its k-th derivative: $\left(\widehat{\mathcal{A}} \odot \widehat{\mathcal{B}}\right)^{(k)} = \widehat{\mathcal{A}}^{(k+1)} \odot \widehat{\mathcal{Z}} \odot \widehat{\mathcal{B}} + \widehat{\mathcal{A}}_k^{(k)} \star \widehat{\mathcal{B}}^{(0)} + \cdots + \widehat{\mathcal{A}}_0^{(0)} \star \widehat{\mathcal{B}}^{(k)}$. All but the first case of the sum are recursively dealt with; the first one is a special case of a product, where we bias the two calls for the generator of the constituents using a well-chosen random variable.

Eventually, for algorithm 11, we use the recursive definition of sequence, and the two previous algorithms.

References

[BLL98] F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial species and tree-like structures*. Cambridge University Press, 1998.

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

[DGG+06] A. Denise, M.C. Gaudel, S.D. Gouraud, R. Lassaigne, and S. Peyronnet. Uniform random sampling of traces in very large models. In *Proceedings of the 1st international workshop on Random testing*, page 19. ACM, 2006.

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

[FFP07] P. Flajolet, É. Fusy, and C. Pivoteau. Boltzmann sampling of unlabelled structures. *Proceedings of ANALCO*, 7, 2007.

[FGT92] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):229, 1992.

[FS08] P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, 2008.

[MZ08] M. Mishna and M. Zabrocki. Analytic aspects of the shuffle product. In *25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, volume 1, pages 561–572. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2008.

[NW75] A. Nijenhuis and H.S. Wilf. *Combinatorial algorithms*. Academic Press New York, 1975.