Biased Boltzmann samplers and generation of extended linear languages with shuffle

Alexis Darrasse[1], Konstantinos Panagiotou[2], Olivier Roussel[1], and Michèle Soria[1]

[1]*Laboratoire d'Informatique de Paris 6 — Équipe APR — UPMC — 4, Place Jussieu — Paris, France*
[2]*Max-Planck-Institute for Informatics — Campus E1.4 — 66123 Saarbrücken, Germany*

This paper is devoted to the construction of Boltzmann samplers according to various distributions, and uses stochastic bias on the parameter of a Boltzmann sampler, to produce a sampler with a different distribution for the size of the output. As a significant application, we produce Boltzmann samplers for words defined by regular specifications containing shuffle operators and linear recursions. This sampler has linear complexity in the size of the output, where the complexity is measured in terms of real-arithmetic operations and evaluations of generating functions.

Keywords: random sampling, Boltzmann samplers, biased distributions, generating functions, combinatorial specifications, shuffle operator

1 Introduction

In combinatorial modeling, random sampling is an important tool for exploring properties of objects, validating models and software testing. The framework of Analytic Combinatorics [FS08] provides two generic methods for uniform sampling of objects belonging to combinatorial classes described by their specifications (uniform sampling means that all objects of same size have the same probability). In the recursive method, given a combinatorial class $\mathcal{C}$ and an integer n, the samplers produce a random object of $\mathcal{C}$ with size n, by computing enumeration sequences [FZVC94]. In the Boltzmann method, given a class $\mathcal{C}$ and a parameter x, the samplers produce a random object of $\mathcal{C}$ with a fluctuating size, by computing generating functions [DFLS04].

For Boltzmann samplers, the output distribution depends on the parameter x. This property is classically used for aiming at a certain expected size of the output, but it can also be used in order to reach a particular distribution. In this paper we investigate cases when x is biased with a stochastic value: starting with a random sampler that produces objects of a class $\mathcal{C}$ with output distribution, a stochastic perturbation of x leads to a random sampler that produces the same objects, with a different output distribution. This transformation, that only operates on the parameter, can be seen as a "random samplers transform" that uses samplers as black boxes (the sampling algorithms are not modified).

An important motivation for this work is to propose Boltzmann samplers capable to deal with the shuffle operator on languages. Languages, specified by recursive grammars with *union* and *product* operators,

Funded by project MAGNUM — ANR 2010 BLAN 0204

do belong to the Boltzmann model of random generation, but the *shuffle product* cannot be handled in the classical Boltzmann framework. However, the shuffle product is an important and useful operator on languages, both for its structural properties and its applications, for example in hashing and program verification (see *e.g.* [FGT92, DGG^{+}06, MZ08]).

The main difficulty with the shuffle product is to deal at the same time with both ordinary and exponential generating functions. Indeed, languages are naturally associated with ordinary generating functions $A(z) = \sum a_n z^n$, where a_n is the number of words of size n, whereas the shuffle product only translates nicely in terms of exponential generating functions $\widehat{A}(z) = \sum a_n z^n/n!$: the exponential generating function of the shuffle product of languages is the product of the exponential generating functions of its components.

Regarding Boltzmann samplers, these two worlds correspond to different probabilities for random generation: in the case of an ordinary Boltzmann sampler with parameter x, which we denote by $\Gamma\mathcal{A}(x)$, a word γ of size n in $\mathcal{A}$ is generated with probability $\mathbb{P}(\gamma) = \frac{x^n}{A(x)}$, whereas in the case of an exponential Boltzmann sampler $\widehat{\Gamma}\mathcal{A}(x)$, the probability is $\mathbb{P}(\gamma) = \frac{x^n}{n!\widehat{A}(x)}$.

The ordinary and exponential generating functions of a language are related by the Borel-Laplace transform. Using this transform, it is easy to see that there is an appropriate density for biasing an exponential Boltzmann sampler $\widehat{\Gamma}$ into an ordinary Boltzmann sampler Γ. Unfortunately the inverse transformation leads to an impossible problem of moments.

Our solution is to make a topdown decomposition according to the specification of the language, and construct an exponential Boltzmann sampler from biased exponential Boltzmann samplers for its components. The central idea in all our algorithms is that of biasing: when a Boltzmann sampler for a class $\mathcal{A}$ cannot be constructed directly, we construct instead a Boltzmann sampler for a derived class $\mathcal{B}$ and then bias the parameter of the Boltzmann sampler for $\mathcal{B}$ with a probabilistic value in order to get back a correct generation on $\mathcal{A}$.

The family of languages that we consider in this paper consists in languages specifiable in terms of operators *union*, *product* and *shuffle*, with linear recursions. An important property of these languages is that their ordinary generating functions are rational functions. We construct an ordinary Boltzmann sampler for a language in this family by following its specification; whenever a *shuffle* operator appears, there is no simple way to construct an ordinary Boltzmann sampler, but it is easy to obtain an exponential sampler. The situation is reversed for the classical *product* operator: the ordinary sampler is elementary whereas the construction of an exponential sampler is complex and entails the computation of derivative samplers. Our solution consists in going back and forth between the ordinary and exponential worlds, and create bridges between ordinary and exponential Boltzmann samplers. The realization of these Boltzmann samplers relies on the evaluation of generating functions, and since all generating functions of the considered languages are rational functions, we can devise an evaluation process based on some computations of resultants.

In a previous paper [DPRS10] we worked on regular specifications containing shuffle operators, and designed Boltzmann generation algorithms with linear complexity for approximate size sampling. In that case, we were restricted to regular (rational) languages, since this family is closed under the shuffle operator (see *e.g.* [Eil74]), and it is always possible to avoid dealing directly with the shuffle operator, via rephrasing the specification, or by giving an automaton description, and very efficient algorithms have been designed [Den94, DZ99, BG12].

In this paper, we deal with a more general family of languages, strictly containing the class of linear languages (which is not closed under the shuffle operator), and we still propose Boltzmann samplers with linear complexity.

Outline of the paper Section 2 presents the general principle of biasing Boltzmann samplers with stochastic values, and shows three cases of application: biasing Boltzmann sampler for pointed objects into a Boltzmann sampler for non pointed objects; transforming an exponential Boltzmann sampler into an ordinary Boltzmann sampler; and providing Boltzmann samplers for the box product and the ordered product. Section 3 is devoted to the presentation of Boltzmann samplers for extended linear languages with shuffle; we first recall the definition and some useful properties of the shuffle operator; then we define our family of extended linear languages, which are languages specifiable in terms of operators *union*, *product* and *shuffle*, with linear recursions; and finally we state the general sampling algorithm for such specifications, which implies to derive parametrized exponential Boltzmann samplers for the various operators of the specifications. In Section 4, we give the precise algorithms for sampling languages composed with union, product, sequence and shuffle operators, according to parametrized exponential Boltzmann distributions; and Section 5 shows a combinatorial interpretation of these algorithms, explaining how objects are constructed and affected by the various operators. Section 6 discusses complexity and implementation issues: we first state that the sampling algorithm of Section 3 has linear complexity, where the complexity is measured in terms of real-arithmetic operations and evaluations of generating functions; we also discuss the cost of drawing the various random variables that appear in the stochastic biasing process; and finally we show how to rely on the rationality of the involved generating functions in order to make efficient computations.

This is an extended abstract, and the proofs are omitted because of the lack of space.

2 Boltzmann generation with a biased parameter

In this section, we present the general principle of biasing Boltzmann samplers with stochastic values, and show three cases of application. First biasing Boltzmann sampler for generating standard objects from pointed objects. Second we show how to transform an exponential Boltzmann sampler into an ordinary Boltzmann sampler, on the basis of the Borel-Laplace transform. Third we provide exponential Boltzmann samplers for two additional operators: the box product and the ordered product.

2.1 Boltzmann samplers

Let $\mathcal{C}$ be a combinatorial class, with a size function $|\cdot| : \mathcal{C} \to \mathbb{N}$. Given a parameter x, a *Boltzmann sampler* for class $\mathcal{C}$ is a probabilistic algorithm that generates any object γ in $\mathcal{C}$ with probability which is essentially proportional to $x^{|\gamma|}$. More precisely, if f is a function depending only on the size, and g a function depending only on the parameter, the output distribution is

$$\mathbb{P}(x,\gamma) = \frac{f(|\gamma|)g(x)\, x^{|\gamma|}}{\sum_{\gamma\in\mathcal{C}} f(|\gamma|)g(x)\, x^{|\gamma|}} \ .$$

This definition covers the two classical Boltzmann samplers:

- the ordinary sampler $\Gamma\mathcal{C}(x)$, for non labelled objects and ordinary generating series, that samples with probability $\frac{x^{|\gamma|}}{C(x)}$, with $C(x) = \sum_\gamma x^{|\gamma|}$, where $f(|\gamma|) = 1 = g(x)$;

- the exponential sampler $\widehat{\Gamma}\mathcal{C}(x)$, for labelled objects and exponential generating series, that samples with probability $\frac{x^{|\gamma|}}{|\gamma|!} \cdot \frac{1}{\widehat{C}(x)}$, with $\widehat{C}(x) = \sum_{\gamma} \frac{x^{|\gamma|}}{|\gamma|!}$, where $f(|\gamma|) = 1/|\gamma|!$ and $g(x) = 1$.

But we can also define Boltzmann samplers associated with various distributions. For example we shall be dealing with distributions involving derivatives.

- An algorithm $\widehat{\Gamma}\mathcal{C}(x,k)$ is said to be an (x,k)-*exponential Boltzmann sampler* if the output distribution is equivalent to (2.1), the normalizing constant being $\widehat{C}^{(k)}(x) = \sum_{\gamma;|\gamma|\geq k} \frac{x^{|\gamma|-k}}{(|\gamma|-k)!}$, with $g(x) = x^{-k}$ and $f(|\gamma|) = \frac{1}{(|\gamma|-k)!}$ if $|\gamma| \geq k$, otherwise $f(|\gamma|) = 0$.

$$\mathbb{P}_k(x,\gamma) = \frac{1}{\widehat{C}^{(k)}(x)} \cdot \frac{x^{|\gamma|-k}}{(|\gamma|-k)!} \text{ if } |\gamma| \geq k \text{ , and otherwise } 0. \tag{2.1}$$

- An algorithm $\overset{\bullet}{\Gamma}\mathcal{C}(x,k)$ is said to be an (x,k)-*exponential pointed Boltzmann sampler* if the output distribution is equivalent to (2.2), with normalizing constant $\widehat{G}^{(k)}(x) = \sum_{\gamma;|\gamma|\geq k} \frac{x^{|\gamma|}}{(|\gamma|-k)!}$, with $g(x) = 1$ and $f(|\gamma|) = \frac{1}{(|\gamma|-k)!}$ if $|\gamma| \geq k$, otherwise $f(|\gamma|) = 0$.

$$\mathbb{P}_k(x,\gamma) = \frac{1}{\widehat{G}^{(k)}(x)} \cdot \frac{x^{|\gamma|}}{(|\gamma|-k)!} \text{ if } |\gamma| \geq k \text{ , and otherwise } 0. \tag{2.2}$$

Notice that $\widehat{\Gamma}\mathcal{C}(x) \sim \widehat{\Gamma}\mathcal{C}(x,0) \sim \overset{\bullet}{\Gamma}\mathcal{C}(x,0)$, where $A \sim B$ means that the two algorithms A and B output the same objects with the same probabilities.
(Also notice that a similar definition can be given for (x,k)-*ordinary Boltzmann samplers, pointed or not.*)

2.2 Biasing with a stochastic value

Given a combinatorial class $\mathcal{C}$, and a Boltzmann sampler $\Gamma\mathcal{C}(x)$ with output distribution $p(\gamma,x)$, we shall stochastically perturbate the parameter x in order to produce a Boltzmann sampler for $\mathcal{C}$ with a different output distribution $q(\gamma,x)$. Consider $\delta_x^{\mathcal{C}}(t)$ to be a probability distribution on domain D, $R_x^{\mathcal{C}}$ a random variable following the distribution defined by $\delta_x^{\mathcal{C}}$, and u a realization of $R_x^{\mathcal{C}}$; then by calling $\Gamma\mathcal{C}(ux)$, we get a random object γ with probability $q(\gamma,x) = \int_D p(\gamma,tx)\,\delta_x^{\mathcal{C}}(t)\,\mathrm{d}t$.

Algorithm 1 Biased Boltzmann sampler

```
Input: A Boltzmann sampler $\Gamma\mathcal{C}(x)$ with output distribution $p(\gamma,x)$,
    and a sampler for the variable $R_x^{\mathcal{C}}$ following a probability distribution $\delta_x^{\mathcal{C}}$ on domain $D$
Output: An object $\gamma \in \mathcal{C}$ with probability $q(\gamma,x) = \int_D p(\gamma,tx)\,\delta_x^{\mathcal{C}}(t)\,\mathrm{d}t$
    $u \leftarrow \mathsf{draw}\left(R_x^{\mathcal{C}}\right)$
    return $\Gamma\mathcal{C}(ux)$
```

Remark. Beware that, in this algorithm, one can pick the input distribution $p(\gamma,x)$ and the distribution $\delta_x^{\mathcal{C}}$ freely, but we got as a result the output distribution $q(\gamma,x)$. In practive we are more often faced with the following problem: given the input and output distributions p and q, what is the distribution δ which performs the transformation? This problem is known as a moments problem, and unfortunately has often no solution.

2.2.1 Biasing a pointed Boltzmann sampler into a non pointed Boltzmann sampler

Given a pointed Boltzmann sampler of order k, it is possible to derive a non pointed Boltzmann sampler of order $k-1$. The proposition below states this result in the case of exponential samplers, but it also holds for ordinary samplers (with the appropriate definitions of derivating and pointing).

Proposition 2.1 *Given $\overset{\bullet}{\Gamma}\mathcal{C}(x,k)$, an exponential pointed sampler of order k for a class $\mathcal{C}$, one gets an exponential sampler of order $k-1$ for $\mathcal{C}$, by applying Algorithm 1, with the probability distribution $\delta_x^{\mathcal{C}}(u) = x\frac{\widehat{G}^{(k)}(ux)}{\widehat{C}^{(k-1)}(x)}x^{-k}u^{-k}$ on* $[0,1]$.

Remark. Notice that the inverse transformation (get $\overset{\bullet}{\Gamma}\mathcal{C}(x,k)$ from $\widehat{\Gamma}\mathcal{C}(x,k-1)$) is impossible (moments problem with no solution). However other methods do exist for obtaining a pointed Boltzmann sampler from a non pointed Boltzmann sampler (see e.g. the way of pointing specifications used for the generation of trees in [DFLS04]).

Example. The exponential generating function of the class of (rooted) Cayley trees satisfies the recursive equation $\widehat{C}(z) = z\,\exp(\widehat{C}(z))$. Hence a Boltzmann sampler $\widehat{\Gamma}\mathcal{C}(x)$ for rooted Cayley trees is simply the following [DFLS04]: make a root, draw the number k of root-subtrees according to a Poisson law with parameter $\widehat{C}(x)$, and recursively call k independant $\widehat{\Gamma}\mathcal{C}(x)$ for generating the root-subtrees.
It is now easy to design an exponential Boltzmann sampler $\widehat{\Gamma}\mathcal{D}(x)$ for the class $\mathcal{D}$ of *unrooted Cayley trees*: the number of unrooted Cayley trees is n^{n-2}, and the generating functions satisfy $z\widehat{D}'(z) = \widehat{C}(z)$. Hence we get a sampler for $\widehat{\Gamma}\mathcal{D}(x)$ by drawing u with density $\widehat{C}(tx)/t\widehat{D}(x)$, and launching $\widehat{\Gamma}\mathcal{C}(ux)$.

2.2.2 Biasing an exponential Boltzmann sampler into an ordinary Boltzmann sampler

An exponential Boltzmann sampler $\widehat{\Gamma}\mathcal{C}(x)$ for a class $\mathcal{C}$ can be biased appropriately, so that the output follows the ordinary Boltzmann distribution.

Proposition 2.2 *Let $\widehat{\Gamma}\mathcal{C}(x)$ be an exponential sampler for a class $\mathcal{C}$. The biased sampler of Algorithm 1, with the probability distribution $\delta_x^{\mathcal{C}}(u) = \frac{\widehat{C}(xu)}{C(x)}e^{-u}$ on $[0,+\infty)$, is an ordinary Boltzmann sampler for $\mathcal{C}$.*

The fact that $\delta_x^{\mathcal{C}}(u)$ is a probability distribution on $[0,+\infty)$ is a direct consequence of the well-known Borel-Laplace transform, and the result follows from evaluating $\displaystyle\int_0^\infty \frac{(xu)^n}{n!\widehat{C}(xu)}\delta_x^{\mathcal{C}}(u)\,\mathrm{d}u$. The algorithm transforming an exponential Boltzmann sampler into an ordinary Boltzmann sampler will be of important use in the rest of the paper; we thus repeat it here, though it is a simple instanciation of Algorithm 1.

Algorithm 2 $\Gamma\mathcal{C}(x)$: Boltzmann sampler for $\Gamma\mathcal{C}$ being given $\widehat{\Gamma}\mathcal{C}$

```
Input: A real number $x \in [0, \rho_C)$, and and exponential Boltzmann sampler for a class $\mathcal{C}$
Output: An object $\gamma \in \mathcal{C}$ with probability $\mathbb{P}_x(\gamma) = x^{|\gamma|}/C(x)$
  $u \leftarrow$ draw $\left(R_x^{\mathcal{C}}\right)$
  return $\widehat{\Gamma}\mathcal{C}(ux)$
```

2.2.3 Other transformations: box-product and ordered-product

By combining Boltzmann samplers biased with an appropriate stochastic value, we can derive Boltzmann samplers for new operators on labelled objects: the box-product [FS08] and the ordered-product [BLL98]. Both operators add extra constraints in the way of labelling a pair of objects. Recall that in a standard labelled product $\mathcal{C} = \mathcal{A} \star \mathcal{B}$, a pair (α, β) is labelled by a permutation of $\{1, \cdots, |\alpha| + |\beta|\}$ (which is consistent with both permutations labelling α and β), and the resulting exponential generating function is $\widehat{C}(z) = \widehat{A}(z) \cdot \widehat{B}(z)$. In the case of a box-product $\mathcal{C} = \mathcal{A}^{\square} \times \mathcal{B}$, the labels of a pair (α, β) satisfy the constraints of the standard product. In addition, the smallest label must be attached to an atom of α; the resulting exponential generating function satisfies $\widehat{C}(z) = \int_0^z \widehat{A'}(t)\widehat{B}(t)\,\mathrm{d}t$. For an ordered-product $\mathcal{C} = \mathcal{A} \odot \mathcal{B}$, in a pair (α, β) the $|\alpha|$ smallest labels must be attached to α; and the generating function shows a convolution $\widehat{C}(z) = A_0\widehat{B}(z) + \int_0^z \widehat{A'}(z-t)\widehat{B}(t)\,\mathrm{d}t$.

Proposition 2.3 *The two following algorithms are correct exponential Boltzmann samplers for respectively the box-product and the ordered-product.*

Algorithm 3 Boltzmann sampler for box-product $\mathcal{C} = \mathcal{A}^{\square} \times \mathcal{B}$

```
Input: Boltzmann samplers $\widehat{\Gamma}\mathcal{A}(x,1)$ and $\widehat{\Gamma}\mathcal{B}(x)$, and a sampler for the random variable $R_x^{\mathcal{C}}$ following a
    probability distribution $\delta(t) = \frac{x\widehat{A'}(tx)\widehat{B}(tx)}{\widehat{C}(x)}$ on $[0,1]$.
Output: An object $\gamma = (\alpha, \beta) \in \mathcal{C}$ with probability $\frac{x^{\gamma}}{\gamma!\,\widehat{C}(x)}$.
    $u \leftarrow \mathsf{draw}\left(R_x^{\mathcal{C}}\right)$
    $\alpha \leftarrow \widehat{\Gamma}A(xu, 1)$ ; $\beta \leftarrow \widehat{\Gamma}B(xu)$
    return $\gamma = \mathrm{boxProduct}(\alpha, \beta)$
```

Algorithm 3 is the basis for generating alternating permutations [RS09], and more generally, structures defined by combinatorial differential equations of first order [BRS10].

Algorithm 4 will be used in this paper for generating words with shuffle (see Section 4.2).

Algorithm 4 Boltzmann sampler for ordered-product $\mathcal{C} = \mathcal{A} \odot \mathcal{B}$

```
Input: Boltzmann samplers $\widehat{\Gamma}\mathcal{A}(x,1)$ and $\widehat{\Gamma}\mathcal{B}(x)$, and a sampler for the random variable $R_x^{\mathcal{C}}$ following
    the probability distribution $\delta(t) = \frac{\widehat{A'}(x-t)\widehat{B}(t)}{\widehat{C}(x) - A_0\widehat{B}(z)}$ on $[0,x]$
Output: An object $\gamma = (\alpha, \beta) \in \mathcal{C}$ with probability $\frac{x^{|\gamma|}}{|\gamma|!\,\widehat{C}(x)}$.
    if Bernoulli $\left(\frac{A_0\widehat{B}(x)}{\widehat{C}(x)}\right)$ then
        Draw $\alpha \in \mathcal{A}_0$ uniformly; $\beta \leftarrow \widehat{\Gamma}\mathcal{B}(x)$
    else
        $u \leftarrow \mathsf{draw}\left(R_x^{\mathcal{C}}\right)$
        $\alpha \leftarrow \widehat{\Gamma}A(x-u, 1)$ ; $\beta \leftarrow \widehat{\Gamma}B(u)$
    end if
    return $\gamma = \mathrm{orderedProduct}(\alpha, \beta)$
```

3 Samplers for extended linear languages with shuffle

In this section, we will present an application of the ideas presented previously to the random sampling of words in some formal languages. We are interested in a Boltzmann sampler for the shuffle product of languages. We will first define the shuffle operator, and then recall some facts about formal languages. Eventually, we will define a class of languages on which our algorithms are correct.

3.1 The shuffle product

We present a common and useful operator, the *shuffle product* (or Hurwitz product). It is a well-known and classical operator on languages. The shuffle product of two words u and v is the set of all the possible words obtained by interleaving the letters of u and v.

Definition 3.1 *Let Σ denote an alphabet, and consider two words u and v in Σ^*. We define their shuffle product as $u \sqcup\!\sqcup\, v = \{u_1 v_1 u_2 v_2 \dots u_n v_n \mid u = u_1 \dots u_n, v = v_1 \dots v_n, \forall 1 \leq i \leq n : u_i, v_i \in \Sigma^*\}$, or equivalently — with $x, y \in \Sigma$ — by the recursive definition: $u \sqcup\!\sqcup\, \epsilon = \epsilon \sqcup\!\sqcup\, u = u$ and $xu \sqcup\!\sqcup\, yv = x(u \sqcup\!\sqcup\, yv) + y(xu \sqcup\!\sqcup\, v)$.*

Given two languages $\mathcal{A}$ and $\mathcal{B}$ on disjoint alphabets, their *shuffle* product is the language

$$\mathcal{C} = \mathcal{A} \sqcup\!\sqcup\, \mathcal{B} = \{u \sqcup\!\sqcup\, v \mid u \in \mathcal{A}, v \in \mathcal{B}\}.$$

The condition for the alphabets to be disjoint is technical, and is the same as for the classical disjoint union operator $+$ on languages. The minimal condition is that the language $\mathcal{A} \sqcup\!\sqcup\, \mathcal{B}$ be unambiguous. It is well-known that the counting sequence for $\mathcal{C}$ satisfies $c_n = \sum_{p=0}^{n} \binom{n}{p} a_p b_{n-p}$, and this relation translates into the product of *exponential generating functions* $\widehat{C}(z) = \widehat{A}(z) \times \widehat{B}(z)$.

However, the shuffle operator applies to words, which are unlabelled objects, associated to *ordinary generating functions*. But there exists no direct equation linking the ordinary generationg function $C(z)$ of $\mathcal{C}$ and $A(z)$ and $B(z)$. Nonetheless we have the following lemma.

Lemma 3.2 *Let $A(z)$ and $B(z)$ be two generating function which are rational over $\mathbb{Z}$. Let $C(z)$ be the generating function corresponding to $c_n = \sum_{p=0}^{n} \binom{n}{p} a_p b_{n-p}$. Then $C(z)$ is rational over $\mathbb{Z}$.*

This lemma will be really useful later (see Section 6.2) as it gives some sort of closure under the shuffle product in the same way as the classical product, the sum and the sequence.

In this paper, our solution for sampling consists in going back and forth between the ordinary and exponential worlds, and create bridges between an ordinary Boltzmann sampler $\Gamma\mathcal{C}(x)$ and its corresponding exponential sampler $\widehat{\Gamma}C(x)$. The way from exponential to ordinary, *i.e.* constructing an ordinary Boltzmann sampler out of an exponential one, can be achieved by biasing the exponential sampler with an appropriate density arising from the Laplace transform (see Proposition 2.2). On the contrary, it is not feasible to make a direct converse transformation, since this leads to an impossible moment problem. We thus recursively rely on the specification and study the construction of an exponential Boltzmann sampler for the sum, product and shuffle of two languages, given exponential Boltzmann samplers for the components.

3.2 Extended linear languages

In this subsection, we define the class of languages we are interested in, which stricly contains linear languages. First we recall some long-known facts about languages.

Regular languages The class of regular languages is one of the simplest classes of languages. Kleene's theorem shows that this class of languages can be defined in several equivalent ways, by finite automata and by regular expressions. For our purpose, we shall use the definition in terms of an unambiguous regular expression (or specification), where the term unambiguous means that for every word in the language there is exactly one way of parsing it according to the specification.
A regular language can be described unambiguously by a (non-recursive) regular expression that only involves letters of a finite alphabet, and the three combinatorial operations of disjoint union $+$, concatenation $\cdot$, and the Kleene star *.
Regular languages share the properties that their generating function is rational, and they are closed under the shuffle operator $\shuffle$ and substitution.

Linear languages Next we define the class of linear language.
A language is said to be linear if it can be generated using a linear grammar, that is a (finite) set of production rules only involving union and product, with only linear recursion allowed: the production rule $S \overset{*}{\to} \alpha S \beta S \gamma$ where $\alpha, \beta, \gamma \in (\Sigma \cup Nonterminal)^*$ never occurs.
These languages have the properties that their generating function is rational. However, they are not closed under the shuffle product nor under substitution. For example, the well known language $\mathcal{L}_{a,b} = \{a^n b^n \mid n \in \mathbb{N}\}$ is linear, but $\mathcal{L}_{a,b} \shuffle \mathcal{L}_{c,d}$ is not even context-free.

Extended linear languages with substitution and shuffle We define a convenient class of languages for our problem. It is the smallest class closed by shuffle product, usual product, union and linear recursions, which contains linear languages.

Definition 3.3 *The class V of* extended linear languages with substitution and shuffle *is the smallest class of languages containing the linear languages, and closed upon the following schemas:*

- $X, Y \in V \Longrightarrow L = X \shuffle Y \in V$*;*
- $X_1, \cdots, X_i \in V \Longrightarrow L = \phi(L, X_1, \cdots X_i) \in V$ *where ϕ is any function* linear in L *and using atoms, union $+$ and product $\times$.*

Thus, we are working on languages defined by linealy recursive specifications involving the union $+$, the product $\times$ and the shuffle $\shuffle$. An example of language in V is $\{(ab^*)^n(cd^* \shuffle (\varepsilon + e^6))^n \mid n \in \mathbb{N}\}$. Indeed, $\{x^n y^n \mid n \in \mathbb{N}\}$ is linear, hence in V; any rational language is linear hence in V; and V is closed under shuffle and substitution. The class V shares many nice properties with regular languages:

Proposition 3.4 *The class V is closed under union, product, shuffle product, and substitution. Moreover, any language in this class has a* rational *generating function.*

3.3 Boltzmann sampling

We now write the main algorithm, a Boltzmann sampler for languages in V. The algorithm switches to the corresponding Boltzmann sampler according to the form of the specification, with a simple pattern-matching. Furthermore, it biases the recursive calls in order to get an ordinary Boltzmann sampler. The difficult part is concentrated on line 8 of this algorithm. Indeed, in order to have an ordinary Boltzmann sampler for the shuffle, we have to bias and call an *exponential* Boltzmann sampler (see Section 2.2.2). But this, in turn, implies the need of an exponential Boltzmann sampler for every other operator (union, product and sequence). Moreover, the construction of an exponential sampler for the product leads to

Algorithm 5 $\Gamma\mathcal{C}(x)$: Ordinary Bolzmann sampler for $\mathcal{C}$ a regular language with shuffle

```
Input: A real $x$
Output: A word $\gamma$ from $\mathcal{C}$ with a Boltzmann probability
if $\mathcal{C} = \mathbf{1}$ then return $\varepsilon$
else if $\mathcal{C} = \mathcal{Z}_l$ where $l \in \Sigma$ then return $l$
else if $\mathcal{C} = \mathcal{A} + \mathcal{B}$ then return $\Gamma(\mathcal{A} + \mathcal{B})(x)$
else if $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ then return $\Gamma(\mathcal{A} \times \mathcal{B})(x)$
else if $\mathcal{C} = \mathsf{Seq}(\mathcal{A})$ then return $\Gamma(\mathsf{Seq}(\mathcal{A}))(x)$
else if $\mathcal{C} = \mathcal{A} \sqcup\!\sqcup \mathcal{B}$ then
    $u \leftarrow \mathrm{draw}(R_x^{\mathcal{C}})$   where $R_x^{\mathcal{C}}$ follows the density $\delta_x^{\mathcal{C}}(u) = \frac{\widehat{C}(xu)}{C(x)} e^{-u}$
    return $\widehat{\Gamma}(\mathcal{A} \sqcup\!\sqcup \mathcal{B})(ux, 0)$
end if
```

the apparition of derivatives in the combinatorial classes as well as in the generating functions, hence the necessity of constructing exponential Boltzmann samplers of order k (see Section 4). To sum up,

$$\text{we want } \Gamma\mathcal{C}(x) \xrightarrow[\text{because of } \sqcup\!\sqcup]{\text{we need}} \widehat{\Gamma}\mathcal{C}(x) \xrightarrow[\text{because of } \times]{\text{we need}} \widehat{\Gamma}\mathcal{C}(x, k).$$

Theorem 3.5 *Suppose that a language $\mathcal{C} \in V$ is given by a finite specification. Then, Algorithm 5, assembled from the specification of $\mathcal{C}$, is an ordinary Boltzmann generator for $\mathcal{C}$.*

Moreover, Algorithm 5 has a linear complexity: the Boltzmann generator $\Gamma\mathcal{C}(x)$ assembled from the specification of $\mathcal{C}$ given by the procedures from the next section has a complexity, measured in the number of real-arithmetic operations, that is linear in the size of the generated string (see Section 6).

4 Construction of exponential Boltzmann samplers of order k

This section is devoted to the construction of exponential Boltzmann samplers of order k for the operations of union, product, sequence and shuffle, as they are required by Algorithm 5. In each case, considering $\mathcal{C} = \mathcal{A}\ Op\ \mathcal{B}$, we derive the expression $\widehat{C^{(k)}}(z)$ of the exponential generating function of order k, and compute the probabilities that drive the sampler in order to get the output distribution $\mathbb{P}_{x,k}(\gamma) = \frac{1}{\widehat{C^{(k)}}(x)} \frac{x^{|\gamma|-k}}{(|\gamma|-k)!}$ for $|\gamma| \geq k$. The general statement of this section is the following:

Proposition 4.1 *Algorithms 6, 7, 8 and 9 define exponential Boltzmann samplers of order k respectively for the constructors of union, product, sequence and shuffle.*

4.1 Disjoint union

Let the language $\mathcal{C}$ be defined as the disjoint union of two other languages, *i.e.* $\mathcal{C} = \mathcal{A} + \mathcal{B}$. The exponential generating function of order k satisfies $\widehat{C^{(k)}}(z) = \widehat{A^{(k)}}(z) + \widehat{B^{(k)}}(z)$, and the sampler works very similar to the ordinary case.

4.2 Product

Let $\mathcal{C} = \mathcal{A} \times \mathcal{B}$. The description of exponential Boltzmann samplers for this construction relies on expressing the exponential generating functions, and the involved probabilities.

Algorithm 6 $\widehat{\Gamma}\mathcal{C}(x,k)$: Exponential Bolzmann sampler for $\mathcal{C} = \mathcal{A} + \mathcal{B}$

```
Input: A real $x$ and an integer $k$
Output: A word $\gamma$ from $\mathcal{C} = \mathcal{A} + \mathcal{B}$ with probability $\mathbb{P}_{x,k}(\gamma)$
  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
```

The exponential generating function for the product is $\widehat{C}(z) = |\mathcal{A}_0|\widehat{B}^{(0)}(z) + \int_0^z \widehat{A}^{(1)}(z-t)\widehat{B}(t)\,\mathrm{d}t$. By differentiating both sides of the equation k times with respect to z, we obtain

$$\widehat{C}^{(k)}(z) = \sum_{\ell=0}^{k} |\mathcal{A}_\ell|\widehat{B}^{(k-\ell)}(z) + \int_0^z \widehat{A}^{(k+1)}(z-t)\widehat{B}(t)\,\mathrm{d}t. \tag{4.1}$$

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

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

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

$$t \mapsto \frac{\widehat{A}^{(k+1)}(x-t)\widehat{B}(t)}{\int_0^x \widehat{A}^{(k+1)}(x-u)\widehat{B}(u)\,\mathrm{d}u} \quad \text{where } 0 \le t \le x \tag{4.2}$$

With the above notation at hand, an exponential Boltzmann sampler for $\mathcal{C}$ works as in Algorithm 7.

Algorithm 7 $\widehat{\Gamma}\mathcal{C}(x,k)$: Exponential Bolzmann sampler for $\mathcal{C} = \mathcal{A} \times \mathcal{B}$

```
Input: A real $x$ and an integer $k$
Output: A word $\gamma$ from $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ with probability $\mathbb{P}_{x,k}(\gamma)$
  $\ell \leftarrow$ draw $\left(P^{\mathcal{C}}_{k,x}\right)$
  if $0 \le \ell \le k$ then
      Draw $\gamma_{\mathcal{A}} \in \mathcal{A}_\ell$ uniformly
      $\gamma_{\mathcal{B}} \leftarrow \widehat{\Gamma}\mathcal{B}(x, k-\ell)$
  else
      $t \leftarrow$ draw $\left(B^{\mathcal{C}}_{k,x}\right)$
      $\gamma_{\mathcal{A}} \leftarrow \widehat{\Gamma}\mathcal{A}(x-t, k+1)$; $\gamma_{\mathcal{B}} \leftarrow \widehat{\Gamma}\mathcal{B}(t, 0)$
  end if
  return the word $\gamma_{\mathcal{A}}\gamma_{\mathcal{B}}$
```

4.3 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 Boltzmann sampler for $\mathcal{C}$ can be easily composed from the results in Sections 4.1 and 4.2.

By equation 4.1, and using the fact $\mathcal{A}_0 = \varnothing$ we infer that $\widehat{C}^{(k)}(z)$ satisfies the relation

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

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

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

Moreover, let $B^{\mathcal{C}}_{k,x}$ be a random variable as in Equation (4.2), where the function $\widehat{B}$ is replaced by $\widehat{C}$. Then an exponential Boltzmann sampler for the sequence construction is given by:

Algorithm 8 $\widehat{\Gamma}\mathcal{C}(x,k)$: Exponential Bolzmann sampler for $\mathcal{C} = \mathsf{Seq}(\mathcal{A})$

```
Input: A real $x$ and an integer $k$
Output: A word $\gamma$ from $\mathcal{C} = \mathsf{Seq}(\mathcal{A})$ with probability $\mathbb{P}_{x,k}(\gamma)$
  $\ell \leftarrow \mathsf{draw}\left(S^{\mathcal{C}}_{k,x}\right)$
  if $\ell = 0$ then return $\varepsilon$
  else if $1 \le \ell \le k$ then
      Draw $\gamma_{\mathcal{A}} \in \mathcal{A}_\ell$ uniformly
      $\gamma_{\mathcal{C}} \leftarrow \widehat{\Gamma}\mathcal{C}(x, k-\ell)$
  else
      $t \leftarrow \mathsf{draw}\left(B^{\mathcal{C}}_{k,x}\right)$
      $\gamma_{\mathcal{A}} \leftarrow \widehat{\Gamma}\mathcal{A}(x-t, k+1)$; $\gamma_{\mathcal{C}} \leftarrow \widehat{\Gamma}\mathcal{C}(t, 0)$
  end if
  return the word $\gamma_{\mathcal{A}}\gamma_{\mathcal{C}}$
```

4.4 Shuffle

Finally, we consider the shuffle operation, and construct exponential Boltzmann samplers for the shuffle of two languages. As already mentionned, if $\mathcal{C} = \mathcal{A} \sqcup\!\sqcup \mathcal{B}$, where the alphabets of $\mathcal{A}$ and $\mathcal{B}$ are disjoint, then $\widehat{C}(x) = \widehat{A}(x) \cdot \widehat{B}(x)$. In order to sample from the shuffle of two languages we need to express the involved distribution, obtained by differentiating k times the previous equation. 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 Boltzmann sampler for the shuffle of two languages is given by the following algorithm (where we assume that we have an algorithm for shuffling two *given* words).

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

```
Input: A real $x$ and an integer $k$
Output: A word $\gamma$ from $\mathcal{C} = \mathcal{A} \sqcup \mathcal{B}$ with probability $\mathbb{P}_{x,k}(\gamma)$
  $\ell \leftarrow \mathsf{draw}\left(D^{\mathcal{C}}_{k,x}\right)$
  $\gamma_{\mathcal{A}} \leftarrow \widehat{\Gamma}\mathcal{A}(x,\ell)$; $\gamma_{\mathcal{B}} \leftarrow \widehat{\Gamma}\mathcal{B}(x,k-\ell)$
  $n_{\mathcal{A}} \leftarrow |\gamma_{\mathcal{A}}|$ and $n_{\mathcal{B}} \leftarrow |\gamma_{\mathcal{B}}|$
  Let $b_{\mathcal{A}}e_{\mathcal{A}} = \gamma_{\mathcal{A}}$ where $|b_{\mathcal{A}}| = \ell$, and $b_{\mathcal{B}}e_{\mathcal{B}} = \gamma_{\mathcal{B}}$ where $|b_{\mathcal{B}}| = k - \ell$
  $b \leftarrow \mathrm{shuffle}(b_{\mathcal{A}}, b_{\mathcal{B}})$; $e \leftarrow \mathrm{shuffle}(e_{\mathcal{A}}, e_{\mathcal{B}})$
  return the word $be$
```

5 Combinatorial interpretation

In this section, we show how to combinatorially interpret the constructions and algorithms presented in Section 4. Usually, when an object of size n is labelled, any of the $n!$ possible labelling are allowed (apart from symmetry). On the contrary, our interpretation here is based on a *unique* labelling of the words. Using this vision, we interpret algorithms from the previous section, explaining how objects are constructed and affected by the various combinatorial operators.

5.1 Canonically labelled classes

Definition 5.1 *Let $\mathcal{A}$ be a language. We define the* canonically labelled class $\widehat{\mathcal{A}}$ *as the class of words in $\mathcal{A}$ with the unique labelling associating to each letter in a word its position. Moreover, we define a* labelling *function L and an* unlabelling *function U: for a word $w \in \mathcal{A}$, $L(w)$ is the word w where each letter is labelled by its position; and from any labelled word γ, $U(\gamma)$ is the same word, but without any label.*

The function L performs the unique labeling defined previously, and U removes the labels from an object. Note that $U \circ L = id$, but $L \circ U \neq id$ in general. In the following, when there is no risk of ambiguity, we will consistently write $\widehat{\mathcal{A}}$ for $L(\mathcal{A})$. We will use the notation $\mathrm{EGF}(\widehat{\mathcal{A}})$ for the exponential generating function of the class $\widehat{\mathcal{A}}$. These labelled classes share most of their properties with the ordinary ones.

Lemma 5.2 *Let $\mathcal{A}$ be a language, and $\widehat{\mathcal{A}} = L(\mathcal{A})$. Then $\mathrm{EGF}(\widehat{\mathcal{A}}) = \mathrm{EGF}(\mathcal{A})$ and $\widehat{\Gamma}\widehat{\mathcal{A}} = \widehat{\Gamma}\mathcal{A}$.*

For example, if $\mathcal{A} = \{ab, bb, bac\}$ then $L(\mathcal{A}) = \widehat{\mathcal{A}} = \{a_1b_2, b_1b_2, b_1a_2c_3\}$. The ordinary generating function of $\mathcal{A}$ is $2z^2 + z^3$, and we have $\mathrm{EGF}(\mathcal{A})(z) = 2\frac{z^2}{2!} + \frac{z^3}{3!} = \mathrm{EGF}(\widehat{\mathcal{A}})(z)$.

5.2 Operators and samplers of labelled classes

This section presents the transfer of combinatorial operators by the function L, and the resulting exponential Boltzmann samplers. These results are summarized in the following table.

Language $\mathcal{C}$	Labelled class $\widehat{\mathcal{C}}$	EGF $\widehat{C}$(z)	Exponential sampler $\widehat{\Gamma}\widehat{\mathcal{C}}$
$\mathcal{A}+\mathcal{B}$	$\widehat{\mathcal{A}}+\widehat{\mathcal{B}}$	$\widehat{A}(z)+\widehat{B}(z)$	$\widehat{\Gamma}\left(\widehat{\mathcal{A}}+\widehat{\mathcal{B}}\right)$
$\mathcal{A}\times\mathcal{B}$	$\widehat{\mathcal{A}}\odot\widehat{\mathcal{B}}$	$\lvert\mathcal{A}_0\rvert\widehat{B}(z)+\int_0^z\widehat{A^{(1)}}(z-t)\widehat{B}(t)\,\mathrm{d}t$	see Algorithm 4
$\mathcal{A}\sqcup\mathcal{B}$	$\widehat{\mathcal{A}}\star\widehat{\mathcal{B}}$	$\widehat{A}(z)\times\widehat{B}(z)$	$\widehat{\Gamma}\left(\widehat{\mathcal{A}}\star\widehat{\mathcal{B}}\right)$

Lemma 5.3 *Let $\mathcal{A}$ and $\mathcal{B}$ be two languages. Then $\widehat{\mathcal{A}+\mathcal{B}} = \widehat{\mathcal{A}} + \widehat{\mathcal{B}}$, and* $\mathrm{EGF}(\widehat{\mathcal{A}+\mathcal{B}}) = \mathrm{EGF}(\widehat{\mathcal{A}}) + \mathrm{EGF}(\widehat{\mathcal{B}})$.

From this lemma, we can deduce that an exponential Boltzmann sampler $\widehat{\Gamma}\left(\widehat{\mathcal{A}+\mathcal{B}}\right)$ is simply an exponential Boltzmann sampler $\widehat{\Gamma}\left(\widehat{\mathcal{A}}+\widehat{\mathcal{B}}\right)$, which is well-known: it consists of a Bernoulli trial depending on the values of the generating functions, and then it performs either a call to $\widehat{\Gamma}\widehat{\mathcal{A}}$ or to $\widehat{\Gamma}\widehat{\mathcal{B}}$.

Lemma 5.4 *Let $\mathcal{A}$ and $\mathcal{B}$ be two languages. Then $\widehat{\mathcal{A} \sqcup\!\sqcup \mathcal{B}} = \widehat{\mathcal{A}} \star \widehat{\mathcal{B}}$ (where $\star$ is the* labelled product *of two labelled classes), and* $\mathrm{EGF}(\widehat{\mathcal{A} \sqcup\!\sqcup \mathcal{B}}) = \mathrm{EGF}(\widehat{\mathcal{A}}) \times \mathrm{EGF}(\widehat{\mathcal{B}})$.

From this lemma, we can deduce that an exponential Boltzmann sampler $\widehat{\Gamma}\left(\widehat{\mathcal{A} \sqcup\!\sqcup \mathcal{B}}\right)$ is simply an exponential Boltzmann sampler $\widehat{\Gamma}\left(\widehat{\mathcal{A}} \star \widehat{\mathcal{B}}\right)$, which is well-known: it is just two independant recursive calls to $\widehat{\Gamma}\widehat{\mathcal{A}}$ and $\widehat{\Gamma}\widehat{\mathcal{B}}$.

Product of labelled classes To define the translation of the product, we need to introduce a new operator: the *ordered product* (or *ordinal product*) on labelled structures — more precisely on linear species as explained in [BLL98]. We define it here just with a clear example; and a formal and more general definition can be found in [BLL98]. The ordered product of two (labelled) words is 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.

Lemma 5.5 *Let $\mathcal{A}$ and $\mathcal{B}$ be two languages. Then $\widehat{\mathcal{A} \times \mathcal{B}} = \widehat{\mathcal{A}} \odot \widehat{\mathcal{B}}$ (where $\odot$ is the* ordered product *of two labelled classes), and*

$$\mathrm{EGF}\left(\widehat{\mathcal{A}} \odot \widehat{\mathcal{B}}\right)(z) = |\mathcal{A}_0|\widehat{B}(z) + \int_0^z \widehat{A'}(z-t)\widehat{B}(t)\,\mathrm{d}t$$

From this generating function we can deduce a Boltzmann sampler for the ordered product: the sum leads to a Bernoulli sampling with one outcome leading to a zero-sized $\mathcal{A}$ object and a generic $\mathcal{B}$ object; the second case corresponds to a convolution product of series, which is out of the scope of the original Boltzmann framework. To create a Boltzmann sampler for this, one can draw a random real value t between 0 and the parameter x following a well-chosen distribution; then call independently the sampler for $\widehat{\mathcal{A}}'$ with parameter $x - t$ and the sampler for $\widehat{\mathcal{B}}$ with parameter t (see Algorithm 4).

Derivative of labelled classes We saw in the previous sections that in order to devise an exponential Boltzmann sampler for the ordered product, we need to deal with derivatives of classes. 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, if $\mathcal{A} = \{ab, bb, bac\}$, we have $\widehat{\mathcal{A}} = \{a_1b_2, b_1b_2, b_1a_2c_3\}$, $\widehat{\mathcal{A}}^{(1)} = \{ab_1, bb_1, ba_1c_2\}$, $\widehat{\mathcal{A}}^{(2)} = \{ab, bb, bac_1\}$ and $\widehat{\mathcal{A}}^{(3)} = \{bac\}$. Their EGF is $\mathrm{EGF}\left(\widehat{\mathcal{A}}\right)(z) = 2\frac{z^2}{2} + \frac{z^3}{6}$, $\mathrm{EGF}\left(\widehat{\mathcal{A}}^{(1)}\right)(z) = 2z + \frac{z^2}{2}$, $\mathrm{EGF}\left(\widehat{\mathcal{A}}^{(2)}\right)(z) = 2 + z$ and $\mathrm{EGF}\left(\widehat{\mathcal{A}}^{(3)}\right)(z) = 1$. As usual with labelled classes, the actual size of an object is the number of labels it holds (and *not* its number of letters).

6 Complexity and implementation issues

The complexity of sampling is linear in the size of the output; and it also stays linear for an aimed size, provided that a small tolerance is allowed, as in the classical case [DFLS04].

Theorem 6.1 *For any $\mathcal{C} \in V$ there is a Boltzmann generator $\Gamma\mathcal{C}(x)$ with complexity 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 $\mathcal{C}$.*

This theorem ensures a linear complexity, given that it takes a *constant number of operations* for both evaluating any generating function, and drawing a random variable according to various distributions. Regarding the evaluation of the generating functions, we can partly use an oracle [PSS08]. Still, since all considered generating functions are rational functions, we can do better, by devising an *ad-hoc* evaluation process relying on some computations of resultants (see Section 6.2). The issue of drawing random variables according to the discrete and continuous probability distributions that are needed, is discussed in the next subsection.

6.1 Drawing continuous and discrete random variables

The main idea of the algorithms presented in this paper is to change the value of the parameter according to a probability distribution depending on the specification of the objects. The major problem which needs to be addressed is a way of drawing the various random variables according to nonstandard density functions. For example, if we are interested in the shuffle of two languages, then we need to be able to draw the random variable $D^{\mathcal{C}}_{k,x}$ following the probability law given in Subsection 4.4. Extensively, when dealing with extended linear languages with shuffle, three types of drawings are needed:

- Bernoulli choice with a given parameter (in Algorithms 6 and 4);
- *discrete* random variables with a finite support, and a given probability for each possibility (in Algorithms 7 and 9);
- *continuous* distributions (in Algorithms 2 and 7).

The first item is easy to perform; for the second item, we simply compute the finite number of probabilities over the whole (discrete and finite) support for the variable, and choose the value following these proportions; the third item is more tricky, as it involves continuous distributions, however both distributions are increasing over their domain (respectively $[0, +\infty)$ and $[0, x]$ for Algorithms 2 and 7). Maple knowns how to draw random variable from such distributions (unimodal, or strictly monotone, over a compact domain). For example, a simple rejection scheme works quite well; or a tuned algorithm based on numerical inversion, or based on the ziggurat method – which both have been long-studied and optimized in this case (see for example [Dev86]) – could be used if necessary. To sum up, as our random variables are fortunately either discrete and with finite support, or with a non-implicit probability function, Maple is able to draw automatically according to these given distributions (but we could devise our own specialized versions of these classical algorithms to gain some performances).

However, note that in the general case presented in Section 2, the distribution are arbitrary. In many real-life case, a sampler can be derived using those from [Dev86]. But Theorem 6.1 does not not hold anymore, the complexity is no longer linear.

6.2 Efficient computations of generating functions

The second issue when dealing with an actual implementation is the computations of the generating functions of the involed languages. Indeed, for the operators $+$, $\times$ and Seq, the symbolic method (see [FS08]) gives a direct translation in terms of ordinary generating functions. But such a translation is missing for the shuffle.

A first idea is to compute first the product of exponential generating functions $\widehat{C}(z) = \widehat{A}(z) \times \widehat{B}(z)$ and then compute a Borel-Laplace transform to eventually get the function $C(z)$. However, this path leads to a lot of symbolic computations (including a massive utilisation of differentiations, convolutions and integrations) back and forth the worlds of ordinary and exponential generating functions.

The good idea comes from the fact that for all languages we are dealing with, the ordinary generating function is rational. All of our operators keep this property: in particular the shuffle, if $\mathcal{C} = \mathcal{A} \shuffle \mathcal{B}$, and if $A(z)$ and $B(z)$ are rational functions of z, so will be $C(z)$. Using this property, we devised (with the help and insights of Bruno Salvy) an efficient algorithm using only computations of resultants and quotients over polynomials in $\mathbb{Q}[X]$, based on Algorithm 10 for partial fractions. Using this algorithm, we no longer need to perform heavy symbolic integrations in order to get the ordinary generating function for the shuffle product.

Algorithm 10 Computation of the ordinary generating function for the shuffle product $\mathcal{C} = \mathcal{A} \shuffle \mathcal{B}$

```
Input: Two rational ordinary generating functions $A(z)$ and $B(z)$ for $\mathcal{A}$ and $\mathcal{B}$, given as partial fractions
Output: The rational generating function $C(z)$ for $\mathcal{C} = \mathcal{A} \shuffle \mathcal{B}$
  Let $A(z) = \frac{N_A(z)}{D_A(z)}$ and $B(z) = \frac{N_B(z)}{D_B(z)}$, where $N_X$ and $D_X \in \mathbb{Z}[X]$ are the numerator and the denominator of $X \in \{A, B\}$
  $d \leftarrow \mathrm{degree}(N_A) \cdot \mathrm{degree}(N_B)$
  $P_A(X) \leftarrow X^{\mathrm{degree}(D_A)} D_A\left(\frac{1}{X}\right)$
  $P_B(X)(z) \leftarrow (z - X)^{\mathrm{degree}(D_B)} D_B\left(\frac{1}{z-X}\right)$
  $R(z) \leftarrow \mathrm{resultant}_X\left(P_A(X), P_B(X)(z)\right)$
  $D(z) \leftarrow z^{\mathrm{degree}(R)} R\left(\frac{1}{z}\right)$
  $N(z) \leftarrow \left(\sum_{n=0}^{d} c_n z^n\right) D(z) \mod z^d$
  return $C(z) = \frac{N(z)}{D(z)}$
```

7 Conclusion

The main purpose of this paper is to show how to transform the size distribution of the output of a Boltzmann sampler by modifying its parameter according to a well-chosen density (without any bias on a given size, so that uniformity is preserved). From a combinatorial point of view, this can be seen as a general method for transforming a Boltzmann generator for structures with a given distribution of sizes, into a Boltzmann generator for the same combinatorial objects, with a different distribution of sizes.

As an application of this method, we present a linear time algorithm for the random generation of a class of languages defined by linear recursive specifications involving the union, product and shuffle operators. This class strictly contains the class of linear languages, and shares with the class of regular languages the properties of being closed under shuffle and having a rational generating function. Moreover, the sampling algorithms can be adapted to any language that can be specified with the same set of operators and whose ordinary generating function is a rational function.

Acknowledgments

The authors are grateful to the referees for their perceptive and encouraging comments.

References

[BG12] O. Bernardi and O. Giménez. A linear algorithm for the random sampling from regular languages. *Algorithmica*, 62:130–145, 2012. 10.1007/s00453-010-9446-5.

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

[BRS10] Olivier Bodini, Olivier Roussel, and Michele Soria. Boltzmann samplers for first-order differential specifications. *Special issue of Discrete Applied Mathematics – LAGOS 09*, 2010. 15 pages.

[Den94] A. Denise. *Méthodes de génération aléatoire d'objets combinatoires de grande taille et problèmes d'énumération*. PhD thesis, Université Bordeaux I, 1994.

[Dev86] L. Devroye. *Non-uniform random variate generation*, volume 4. Springer-Verlag New York, 1986.

[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*, pages 10–19. ACM, 2006.

[DPRS10] A. Darrasse, K. Panagiotou, O. Roussel, and M. Soria. Boltzmann generation for regular languages with shuffle. In *Conference GASCom 2010*, Montréal, Canada, September 2010. 12 pages.

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

[Eil74] S. Eilenberg. *Automata, languages and machines, volume A*. Academic press, 1974.

[FGT92] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992.

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

[FZVC94] P. Flajolet, P. Zimmerman, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994.

[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.

[PSS08] C. Pivoteau, B. Salvy, and M. Soria. Boltzmann oracle for combinatorial systems. In *Algorithms, Trees, Combinatorics and Probabilities*, pages 475 – 488. Discrete Mathematics and Theoretical Computer Science, 2008. Proceedings of the Fifth Colloquium on Mathematics and Computer Science. Blaubeuren, Germany. September 22-26, 2008.

[RS09] O. Roussel and M. Soria. Boltzmann sampling of ordered structures. *Electronic Notes in Discrete Mathematics*, 35:305–310, 2009. LAGOS'09 - V Latin-American Algorithms, Graphs and Optimization Symposium.