

# Boltzmann Sampling of Unlabelled Structures

Philippe Flajolet \*      Éric Fusy \*      Carine Pivoteau \*

## Abstract

Boltzmann models from statistical physics combined with methods from analytic combinatorics give rise to efficient algorithms for the random generation of unlabelled objects. The resulting algorithms generate in an unbiased manner discrete configurations that may have nontrivial symmetries, and they do so by means of real-arithmetic computations. We present a collection of construction rules for such samplers, which applies to a wide variety of combinatorial classes, including integer partitions, necklaces, unlabelled functional graphs, dictionaries, series-parallel circuits, term trees and acyclic molecules obeying a variety of constraints, and so on. Under an abstract real-arithmetic computation model, the algorithms are, for many classical structures, of linear complexity provided a small tolerance is allowed on the size of the object drawn. As opposed to many of their discrete competitors, the resulting programs routinely make it possible to generate random objects of sizes in the range  $10^4$ – $10^6$ .

## Introduction

In combinatorics, a random generation algorithm (also called a “sampler”) produces objects under the constraint that two objects of the same size should have equal chances of being drawn. The objects of interest here are the usual ones of discrete mathematics, for instance, words, tilings, trees, graphs, and permutations of various sorts. In the literature, this topic is approached under different perspectives, including abstract complexity theory [13], combinatorics, algorithmics (design and/or engineering), as well as probability theory (Markov chains and Monte-Carlo methods).

Random generation, in a spirit close to ours, is explored in the recently published fascicles of Knuth’s *The Art of Computer Programming*, Volume 4, dedicated to combinatorial algorithms [19]. See also a manuscript of the book *Combinatorial Generation* by Frank Ruskey that is available on the web. A prime motivation in this area is the testing of combinatorial properties of structures (e.g., conjectured structural properties, quantitative aspects) as well as properties of the corresponding

algorithms (with respect to either correctness or efficiency). In the case of parameters that are not exactly, i.e., analytically, solvable, random generation makes it possible to launch simulations studies: Denise *et al.* have for instance developed combinatorial generators for simplified models of genetic sequences, with the goal of aiding users to isolate signal (unexpected events) from noise (statistically unavoidable regularities); see the **GenRGenS** prototype [5]. The approach known as *random testing* in software engineering creates the need to generate random instances of program inputs that obey various sorts of syntactic and semantic constraints, some of the corresponding problems being amenable to random generation as we intend it here (Gouraud *et al.* [4]).

Our objective here is to come up with a reasonably general methodology that is adapted to the design of reasonably efficient samplers. What we address is the collection of all combinatorial structures that can be described by means of a basic set of *constructors*. Precisely, we focus in this paper on the *unlabelled classes*, which are defined from basic elements by means of the fundamental constructions that form disjoint unions, cartesian products, sequences, sets or multisets, and cycles, this possibly combined with additional conditions on the number of components. Such constructions are basic to modern presentations of combinatorial enumeration [2, 9, 14, 23]; see Fig. 1. The difficulty is that the objects are not “rigid”—in general they possess internal symmetries—so that specific algorithms must be designed to generate them in an unbiased way. A general method of random generation dealing with symmetric classes was proposed by Jerrum [17], based on Markov chains—however, the distribution is bound to *approximate* uniformity. In contrast, our approach provides *perfect* uniformity, drawing some of its inspiration from the “recursive method” pioneered by Nijenhuis and Wilf [21], and later extended in [10]. It bases itself in an essential manner on the notion of Boltzmann samplers, as developed by Duchon *et al.* in [8]. It is however appreciably different since we treat here the unlabelled case, where symmetries (automorphisms) are to be suitably handled. A striking outcome of this orbit of ideas is the possibility of generating plane partitions [3] and

---

\*Projet Algo, INRIA Rocquencourt, B. P. 105, 78153 Le Chesnay Cedex, France

|               |  |  |
|---------------|--|--|
| Neutral class | $\mathbf{1}$                                   | composed of a unique element ( $\epsilon$ ) of size 0        |
| Atom          | $\mathcal{Z}$                                  | composed of a unique element (vertex, letter, ...) of size 1 |
| Sum           | $\mathcal{C} = \mathcal{A} + \mathcal{B}$      | Union of disjoint copies of $\mathcal{A}, \mathcal{B}$       |
| Product       | $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ | Cartesian product, forms ordered pairs                       |
| Sequence      | $\mathcal{C} = \text{SEQ}(\mathcal{A})$        | Forms all sequences $(\alpha_1, \dots, \alpha_\ell)$         |
| Cycle         | $\mathcal{C} = \text{CYC}(\mathcal{A})$        | Forms cycles, i.e., sequences up to cyclic shift             |
| Multiset      | $\mathcal{C} = \text{MSET}(\mathcal{A})$       | Forms multisets, i.e., sets with repetitions allowed         |
| Powerset      | $\mathcal{C} = \text{PSET}(\mathcal{A})$       | Forms sets, i.e., multisets with all multiplicities 1.       |

Figure 1: A description of basic constructions.

labelled planar graphs in small polynomial time [11].

**Plan.** Section 1 introduces the general framework of unlabelled constructions and Boltzmann models under which we are operating throughout. *Generating functions* play an essential rôle. Then, we provide a *complete* collection of *rules* that make it possible to produce a Boltzmann generator *automatically* from a structural specification (Section 2). Extensions to sets without repetitions and to cardinality constraints are given next (Section 3). Several classes can be sampled in this way, and the corresponding generated objects are displayed in Figure 8. Then, we discuss in Section 4 the way Boltzmann samplers can be adapted to produce objects within a given range of sizes. Our algorithms are often of linear complexity under an abstract *real-arithmetic* model, as soon as a tolerance is allowed on size (typically, a few percent); they are usually at most quadratic if exact-size random generation is imposed. Issues relative to the real-arithmetic model of computation and implementation are briefly discussed in the final section, Section 5.

## 1 Combinatorial classes and Boltzmann models

A *combinatorial class*  $\mathcal{C}$  is a finite or denumerable set endowed with a *size function* (denoted by  $|\cdot|$ ). We systematically let  $\mathcal{C}_n$  represent the subclass of objects of size  $n$  and  $C_n$  be the corresponding cardinality. The (ordinary) *generating function* (GF, for short) of class  $\mathcal{C}$  is

$$C(z) := \sum_{n \geq 0} C_n z^n \equiv \sum_{\gamma \in \mathcal{C}} z^{|\gamma|}.$$

The *Boltzmann model* associated to  $\mathcal{C}$  and to the positive parameter  $x \in \mathbb{R}$  is the probability distribution that assigns to an element  $\gamma \in \mathcal{C}$  a probability proportional to an exponential of its size:

$$\mathbb{P}(\gamma) = \frac{x^{|\gamma|}}{C(x)}.$$

(Only values of  $x$  in the range  $[0, \rho_C[$ , where  $\rho_C$  is the radius of convergence of  $C$  are considered.) Thus, in

a Boltzmann sampler, the size of the object produced becomes a random variable, denoted by  $N$ , with probability distribution

$$(1.1) \quad \mathbb{P}(N = n) = \frac{C_n x^n}{C(x)}.$$

Relaxing *a priori* the constraint of operating with objects of a fixed size induces, as we shall see, tangible algorithmic gains.

The constructions we deal with are summarized in Figures 1 and 2; details can be found in the publicly available book *Analytic Combinatorics* [9]. Roughly SEQ forms linear lists, CYC forms circular lists, MSET (resp. PSET) form heaps of objects with (resp. without) repetitions allowed (Figure 1). A class is *constructible* if it admits a complete specification in terms of the basic classes  $\mathbf{1}, \mathcal{Z}$  involving only the basic constructors. We also consider constrained constructions, where, e.g.,  $\text{CYC}_k$  forms cycles consisting of exactly  $k$  components. Recursive specifications of classes are allowed. This language of constructions then makes it possible to describe an infinite variety of combinatorial types by means of “grammars”, which resemble context-free grammars augmented with commutation or cyclic-shift rules. Our main result here is that for all such types, a Boltzmann sampler having good complexity-theoretic properties can be automatically obtained.

**THEOREM 1.1.** *Let  $\mathcal{C}$  be any combinatorial class specifiable (possibly recursively) from finite sets using the constructions  $\{+, \times, \text{SEQ}, \text{SEQ}_k, \text{MSET}, \text{MSET}_k, \text{CYC}, \text{CYC}_k\}$ . Assume given an oracle for values of generating functions at positive points. Then, there exists an effective process that produces a Boltzmann sampler  $\Gamma_C(x)$  for  $\mathcal{C}$  such that the time complexity of a generation is, in the worst-case, linear in the size of the object produced.*

In this statement an *oracle* is an external procedure that, given a specification of a class  $\mathcal{F}$  and a value  $x$ , outputs the value  $F(x)$  of the GF of  $\mathcal{F}$  at  $x$ , provided

|  |                                      |   |
|--|--------------------------------------|---|
| $\mathcal{C} = \mathbf{1}$                     | $C(z) = 1$                           |   |
| $\mathcal{C} = \mathcal{Z}$                    | $C(z) = z$                           |   |
| $\mathcal{C} = \mathcal{A} + \mathcal{B}$      | $C(z) = A(z) + B(z)$                 |   |
| $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ | $C(z) = A(z) \times B(z)$            |   |
| $\mathcal{C} = \text{SEQ}(\mathcal{A})$        | $C(z) = (1 - A(z))^{-1}$             | (quasi-inverse)   |
| $\mathcal{C} = \text{CYC}(\mathcal{A})$        | $C(z) = \text{Log } A(z)$            | $\text{Log } f(z) = \sum (\varphi(k)/k) \log(1 - f(z^k))^{-1}$  |
| $\mathcal{C} = \text{MSET}(\mathcal{A})$       | $C(z) = \text{Exp}(A(z))$            | $\text{Exp}(f(z)) = \exp \sum (1/k) f(z^k)$                     |
| $\mathcal{C} = \text{PSET}(\mathcal{A})$       | $C(z) = \overline{\text{Exp}}(A(z))$ | $\overline{\text{Exp}}(f(z)) = \exp \sum ((-1)^{k-1}/k) f(z^k)$ |

Figure 2: Translation of the constructions of Fig. 1 into GFs.

$0 < x < \rho_F$  (with  $\rho_F$  the radius of convergence of  $F$ ). In the core of this paper, we adopt the real domain  $\mathbb{R}$  as our abstract computation domain, so that *complexity results stated here are under an exact real-arithmetic model*. (We defer to the last section practical realizability considerations.) For clarity of the discussion, we also assume that the oracle returns its result in unit time: more sophisticated complexity models taking into account bit complexity and exactness of representations could be investigated (see Denise and Zimmermann [6] for a parallel discussion), but doing so here would exceed the page limitations of this abstract.

Observe finally that Theorem 1.1 describes the linear time complexity of a Boltzmann generator operating *freely* under the sole effect of its parameter  $x \in \mathbb{R}_{\geq 0}$ . Section 4 will discuss the way to add to it some *size control*, so as to obtain objects in a predetermined range of size values, while preserving low polynomial time complexity.

## 2 Design rules for basic constructions

The goal of this section is to describe the rules by which a Boltzmann sampler can be assembled, given the specification of a combinatorial class in terms of the constructions listed in the statement of Theorem 1.1. *This whole section thus constitutes the proof of Theorem 1.1.* We make use here of classical discrete probability distributions listed in Figure 3, for which generators having a linear-time complexity (in terms of the value of the output) are well known [7, 8]. Generically, for a class  $\mathcal{C}$ , we let  $\Gamma C(x)$  denote a Boltzmann sampler that returns an object of  $\mathcal{C}$  according to the Boltzmann model of parameter  $x$ .

**2.1 Unions, products, and sequences** ( $+$ ,  $\times$ , **Seq**). The constructions of disjoint union and cartesian product are treated here in the same way as in the labelled case, following the principles of [8]. To wit:

— *Disjoint union*. If  $\mathcal{C} = \mathcal{A} + \mathcal{B}$ , then  $\Gamma C$  obtains by

a Bernoulli switch based on the probabilities  $\frac{A}{C}, \frac{B}{C}$  that triggers either a  $\Gamma A$  or a  $\Gamma B$ :

(2.2)

$$\Gamma C(x) := \text{Bern} \left( \frac{A(x)}{C(x)}, \frac{B(x)}{C(x)} \right) \longrightarrow \Gamma A(x) \mid \Gamma B(x).$$

— *Cartesian product*. If  $\mathcal{C} = \mathcal{A} \times \mathcal{B}$ , then

$$(2.3) \quad \Gamma C(x) := \langle \Gamma A(x), \Gamma B(x) \rangle.$$

The verification by elementary probability theory is immediate from the definition. The remarkable property, which is at the origin of large algorithmic benefits, is that cartesian products are produced *unconditionally* by *two independent calls* to the component samplers.

Next, for  $X$  an integer values random variable and  $f$  a procedure, note

$$(2.4) \quad X \Longrightarrow f$$

to mean: “if  $X = r$ , then launch  $f_1, \dots, f_r$  where each  $f_j$  is an independent call of  $f$ ”. A consequence of the definitions is that sequences are easily produced:

— *Sequences*. If  $\mathcal{C} = \text{SEQ}(\mathcal{A})$ , then, with notation (2.4):

$$(2.5) \quad \Gamma C(x) := [\text{Geom}(A(x)) \Longrightarrow \Gamma(A(x))].$$

(Proof [8]: Write  $\mathcal{C} = \mathbf{1} + \mathcal{A} \times \mathcal{C}$  and unwind the recursion.)

Equipped with the design rules of Eq. (2.2)–(2.5), it is possible to generate in linear time in the size of the result any object of a class described by an unambiguous regular expression or a deterministic finite automaton. (More generally, transfer matrix models can be accommodated.)

EXAMPLE 1. *Vertically convex (V.C.) polyominoes*. These are connected assemblies of unit squares with vertices at the points of the discrete plane  $Z \times Z$ , such that the intersection with any vertical line is an interval. Pólya and Temperley

| Distribution | Notation                  | Definition   |
|--------------|---------------------------|--|
| Bernoulli    | Bern( $p_1, \dots, p_m$ ) | $\mathbb{P}(k) = p_k$ (with $\sum p_j = 1$ )   |
| Geometric    | Geom( $\lambda$ )         | $\mathbb{P}(k) = \lambda^k(1 - \lambda)$ (with $0 \leq \lambda < 1$ )  |
| Poisson      | Pois( $\lambda$ )         | $\mathbb{P}(k) = e^{-\lambda} \frac{\lambda^k}{k!}$  |
| Logarithmic  | Loga( $\lambda$ )         | $\mathbb{P}(k) = \frac{1}{L(\lambda)} \frac{\lambda^k}{k}$ (with $L(\lambda) = \log(1/(1 - \lambda))$ , $0 \leq \lambda < 1$ ) |

Figure 3: Distributions of use in Boltzmann sampling.

Define the probability distribution relative to  $\mathcal{A}$  and  $x$ :

$$(2.6) \quad \mathbb{P}(K \leq k) = \prod_{j \leq k} \exp\left(\frac{1}{j} A(x^j)\right).$$

Let `MAX_INDEX(A; x)` be a generator according to this distribution (using the classical “inversion method” [7, §2.1] and [18, §4.1]).

**Algorithm** `Γ MSET[ $\mathcal{A}$ ]( $x$ )` :

```

 $\gamma \leftarrow \emptyset$ ;  $k_0 \leftarrow \text{MAX\_INDEX}(A; x)$ ;
for  $j$  from 1 to  $k_0 - 1$  do
     $\gamma \leftarrow \gamma, \left[ \text{Pois}\left(\frac{A(x^j)}{j}\right) \Rightarrow \text{copy}(j, \Gamma A(x^j)) \right]$ 
 $\gamma \leftarrow \gamma, \left[ \text{Pois}_{\geq 1}\left(\frac{A(x^{k_0})}{k_0}\right) \Rightarrow \text{copy}(k_0, \Gamma A(x^{k_0})) \right]$ 
return  $\mu$ .

```

Figure 4: The rule producing a Boltzmann sampler for `MSET`.

first found, by means of certain functional equations, the rational generating function

$$VC(z) = \frac{z(1-z)^3}{1-5z+7z^2-4z^3}.$$

Hickerson [16] has provided what amounts to an unambiguous regular language description, which can then be translated automatically into a Boltzmann sampler. That sampler can equivalently be viewed as a stochastic automaton with transitions that are rational functions of the Boltzmann parameter  $x$ . Here is for instance a polyomino of size 275,



obtained in this way upon using a value of  $x$  close to  $\rho_{VC} \doteq 0.31195$ . . . . . □

**2.2 Multiset construction (MSet)** The multiset construction applied to  $\mathcal{A}$  builds the class  $\mathcal{C} = \text{MSET}(\mathcal{A})$  of all finite multisets, which can alternatively be viewed, up to combinatorial isomorphism, as an infinite product

$$(2.7) \quad \mathcal{C} = \text{MSET}(\mathcal{A}) \cong \prod_{\alpha \in \mathcal{A}} \text{SEQ}(\alpha).$$

(Sweep over all elements  $\alpha \in \mathcal{A}$  and retain, for each  $\alpha$ , an arbitrary sequence of copies of  $\alpha$ .) This gives rise to

the generating function equations

$$(2.8) \quad \begin{aligned} C(z) &= \prod_{\alpha \in \mathcal{A}} (1 - z^{|\alpha|}) \equiv \prod_{n \geq 1} (1 - z^n)^{-A_n} \\ &= \exp \sum_{k \geq 1} \frac{1}{k} A(x^k) \equiv \prod_{k \geq 1} \exp\left(\frac{A(x^k)}{k}\right). \end{aligned}$$

There the *exp-log transformation*,  $f \equiv \exp(\log(f))$ , is used to reorganize the product. The equations (2.7) and (2.8) are central in the construction of samplers for multisets. We establish here:

**PROPOSITION 2.1.** *The generator `Γ MSET[ $\mathcal{A}$ ]( $x$ )` as described in Figure 4 is a valid Boltzmann sampler for `MSET( $\mathcal{A}$ )`.*

*Proof.* From the infinite product representation (2.7) and the construction rule for sequences (2.5), a process equivalent to  $\Gamma C(x)$  is obtained as follows: scan sequentially all elements  $\alpha \in \mathcal{A}$  and output a random number of copies of  $\alpha$ , given by a law  $\text{Geom}(x^{|\alpha|})$ . In symbols  $\Gamma C(x) \cong \prod_{\alpha \in \mathcal{A}} \alpha^{\text{Geom}(x^{|\alpha|})}$ . This is of course *not* an algorithm as the loop is in general infinite.

Next, the following lemma expresses the decomposition of a geometric random variable as an infinite sum of Poisson variables.

**LEMMA 2.1.** *Let  $(Y_i)_{i \geq 1}$  be a sequence of independent random variables such that  $Y_i \in \text{Pois}(\lambda^i/i)$  with  $\lambda < 1$ . Then the sum  $S = \sum_{i \geq 1} iY_i$  satisfies  $S \in \text{Geom}(\lambda)$ .*

(The verification is by characteristic functions or probability generating functions, using the exp-log transformation.)

Equipped with Lemma 2.1, we can transform the abstract infinite-product Boltzmann sampler as follows: (2.9)

$$(2.9) \quad \begin{aligned} \Gamma C(x) &= \prod_{\alpha \in \mathcal{A}} \alpha^{\sum_i i \text{Pois}\left(\frac{x^{i|\alpha|}}{i}\right)} \quad [\text{Lemma 2.1}] \\ &= \prod_i \prod_{\alpha \in \mathcal{A}} \alpha^{i \text{Pois}\left(\frac{x^{i|\alpha|}}{i}\right)} \quad [\text{interchange of } \Pi\text{'s}] \\ &= \prod_i \prod_{\beta \in \mathcal{A}^{\odot i}} \beta^{\text{Pois}\left(\frac{x^{|\beta|}}{i}\right)} \quad [\mathcal{A}^{\odot i} := \underbrace{\{\alpha, \alpha, \dots, \alpha\}}_{i \text{ copies}}] \end{aligned}$$

Next we need a basic program transformation.

LEMMA 2.2. Given a class  $\mathcal{B}$  and a constant  $c > 0$ , the process  $\mathfrak{P} : \prod_{\beta \in \mathcal{B}} \beta^{\text{Pois}(cx^{|\beta|})}$  is realized by the algorithm  $\mathfrak{A} : [\text{Pois}(cB(x)) \implies \Gamma B(x)]$ , with the notation (2.4).

*Proof.* The probability that  $\mathfrak{P}$  produces the multiset  $\{\gamma_1^{r_1}, \dots, \gamma_s^{r_s}\}$  (all  $r_i > 0$ ) is

$$(2.10) \quad \prod_{i=1}^s \frac{(cx^{|\gamma_i|})^{r_i}}{r_i!} \prod_{\gamma \in \mathcal{B}} e^{-cx^{|\beta|}} = c^\ell x^n e^{-cB(x)} \prod_{i=1}^s \frac{1}{r_i!},$$

where  $\ell = \sum r_i$  is the number of components and  $n = \sum r_i |\gamma_i|$  is the size of the multiset. This same multiset is produced by  $\mathfrak{A}$  as a sequence  $\beta_1, \dots, \beta_\ell$  in  $\binom{\ell}{r_1, \dots, r_s}$  ways, each having probability

$$(2.11) \quad e^{-cB(x)} \frac{(cB(x))^\ell}{\ell!} \frac{x^{|\beta_1|}}{B(x)} \cdots \frac{x^{|\beta_\ell|}}{B(x)}.$$

By (2.10) and (2.11), the two distributions induced by  $\mathfrak{P}$  and  $\mathfrak{A}$  coincide.

We can now apply Lemma 2.2 to the last line of (2.9), which describes an abstract Boltzmann sampler for  $\mathcal{C} = \text{MSET}(\mathcal{A})$ . This gives us an algorithm that consists of an infinite loop over all indices  $i$  of (2.9) controlling Poisson generators. The final algorithm of Figure 4 then results after computing separately the largest size  $k_0$  for which an element is generated (Eq. (2.6) of Fig. 4). Proposition 2.1 is established.

EXAMPLE 2. *Nonplane unlabelled trees.* These have been enumerated by Cayley and Pólya. They are specified by  $\mathcal{U} = \mathcal{Z} \times \text{MSET}(\mathcal{U})$ , with the multiset construction expressing the absence of planar embedding. Algorithms to generate such trees have been first given by Nijenhuis and Wilf [21], but their method necessitates the maintenance of large integer tables (of bit size  $\Theta(n^2)$  for trees of size  $n$ ). Here a recursive Boltzmann sampler results directly from the multiset construction of Fig. 4. .... □

**2.3 Cycle construction (Cyc)** A cycle is a sequence of elements, taken up to circular permutation. With  $\varphi(\cdot)$  the Euler totient function, the generating function corresponding to  $\mathcal{C} = \text{CYC}(\mathcal{A})$  is

$$(2.12) \quad C(z) = \sum_{k \geq 1} \frac{\varphi(k)}{k} \log \frac{1}{1 - A(z^k)},$$

due to Pólya, Read, De Bruijn, Klarner, and others). The design rule for cycles is given in Figure 5.

PROPOSITION 2.2. *The generator  $\Gamma \text{CYC}[\mathcal{A}](x)$  of Figure 5 is a valid Boltzmann sampler for  $\text{CYC}(\mathcal{A})$ .*

Consider the probability distribution

$$(2.13) \quad \mathbb{P}(K = k) = \frac{\varphi(k)}{kC(x)} \log \left( (1 - A(x^k))^{-1} \right).$$

Let  $\text{REPLICORDER}(A; x)$  be a generator of this distribution.

**Algorithm**  $\Gamma \text{CYC}[\mathcal{A}](x)$   
 $k \leftarrow \text{REPLICORDER}(x);$   
 $j \leftarrow \text{Loga}(A(x^k));$   
 $w \leftarrow$  a sequence of  $j$  calls to  $\Gamma A(x^k);$   
**return** the cycle made of  $k$  copies of  $w$  cyclically chained.

Figure 5: The rule producing a Boltzmann sampler for CYC.

The idea behind the design rule of Fig. 5 is to build a cycle as follows: (i) Generate a sequence  $w \in \text{SEQ}(\mathcal{A})$  of some length  $j$ , which is called the pattern; (ii) Copy the pattern  $w$  a certain number  $k$  of times (the “replication order”) and produce the cycle associated to  $w^k$ . The problem is to draw the pair  $j, k$  with the right probabilities since a cycle can be obtained in various ways (e.g.,  $[(ab)^6] = [(ba)^6] = [(abab)^3]$ ). We choose the replication order according to the probability distribution (2.13) and the length of the pattern according to a logarithmic distribution (Fig. 3 and 5).

*Proof.* By construction, the probability of the replication order to be  $k$  and of the length of the pattern to be  $j$  is:  $\mathbb{P}(k, j) = \frac{\varphi(k)}{kj} \frac{A(x^k)^j}{C(x)}$ . A cycle  $\gamma \in \text{CYC}(\mathcal{A})$  is maximally decomposed as  $\gamma = (u^r)$ , with  $u = (u_1, \dots, u_s)$  a primitive sequence (i.e., having no symmetry under shift). Then, any decomposition  $\gamma = w^k$  is such that  $k$  is a divisor of  $r$  and  $w = \hat{u}^{r/k}$  with  $\hat{u}$  a cyclic shift of  $u$ . Since  $u$  has  $s$  different cyclic shifts, the cycle  $\gamma$  is drawn with probability

$$\begin{aligned} \mathbb{P}(\gamma) &= s \sum_{k|j=l, k|r} \frac{\varphi(k)}{kj} \frac{A(x^k)^j}{C(x)} \left( \frac{x^{k|u_1|}}{A(x^k)} \cdots \frac{x^{k|u_s|}}{A(x^k)} \right)^{r/k} \\ &= \frac{s}{l} \frac{x^{|\gamma|}}{C(x)} \sum_{k|r} \varphi(k) = \frac{x^{|\gamma|}}{C(x)}, \end{aligned}$$

where use is made of the formula  $\sum_{k|m} \varphi(k) = m$ .

EXAMPLE 3. *Cyclic compositions and necklaces.* A necklace is a sequence of words over a finite alphabet taken up to cyclic shift. In the binary case, a specification is  $\mathcal{N} = \text{CYC}(\mathcal{Z} + \mathcal{Z})$ . Thus, a generator is derived automatically from Fig. 5. Similarly for cyclic compositions described by  $\mathcal{C} = \text{CYC}(\mathcal{Z} \text{SEQ}(\mathcal{Z}))$ : a logarithmic generator triggers a geometric generator. .... □

**MSet<sub>k</sub>**: Define the polynomials  $M_k(z)$  as in (2.15).  
 • Define a *partition-sequence* of size  $k$  as an integer sequence  $(n_i)$  such that  $\sum_{i=1}^k in_i = k$ , and denote by  $\mathcal{P}_k$  the set of partition-sequences of size  $k$ . For  $P \in \mathcal{P}_k$ , introduce  $M_P(z) := \prod_{i=1}^k A(z^i)^{n_i} / (n_i! i^{n_i})$  and define the corresponding sampler:  
 $\Gamma M_P(x) := \mu \leftarrow \emptyset;$   
     **for**  $i$  **from** 1 **to**  $k$  **do**  
         **for**  $j$  **from** 1 **to**  $n_i$  **do**  
              $\mu \leftarrow \mu, \text{copy}(i, \Gamma A(x^i))$   
         **return**  $\mu$ .  
 • Observe that  $M_k(z) = \sum_{P \in \mathcal{P}_k} M_P(z)$ . The sampler  $\Gamma \text{MSET}_k[\mathcal{A}](x)$  is defined as follows: draw  $P \in \mathcal{P}_k$  under the Bernoulli choice  $\mathbb{P}(P) = M_P(x)/M_k(x)$ , and return  $\Gamma M_P(x)$ .

---

**Cyc<sub>k</sub>**: Define the sampler as follows:  
 $\Gamma \text{CYC}_k[\mathcal{A}](x) :=$  • draw a divisor  $i$  of  $k$  with distribution  $\mathbb{P}(i) = \varphi(i)A(x^i)^{k/i} / (kC_k(x))$ ;  
 • return the cycle made of  $i$  cyclically chained copies of a sequence of length  $k/i$ , obtained by  $k/i$  independent calls to  $\Gamma A(x^i)$ .

Figure 6: The rules producing Boltzmann samplers for  $\text{MSET}_k, \text{CYC}_k$ .

EXAMPLE 4. *Unlabelled functional graphs.* A functional graph is a directed graph in which each vertex has out-degree 1. The unlabelled version appears as equivalence class of mappings from a finite set to itself, also known as a “mapping pattern” [20]. The specification is

$$\mathcal{F} = \text{MSET}(\mathcal{K}), \quad \mathcal{K} = \text{CYC}(\mathcal{U}), \quad \mathcal{U} = \mathcal{Z} \times \text{MSET}(\mathcal{U}).$$

A Boltzmann sampler then results from the cycle and multiset constructions, given the sampler for nonplane trees of Example 2. □

**2.4 Constructions with  $k$  components**

(**Seq<sub>k</sub>, MSet<sub>k</sub>, Cyc<sub>k</sub>**) In order to complete the proof of Theorem 1.1, there only remains to treat the case of constructions of type  $\mathfrak{R}_k$ , where  $\mathfrak{R}$  is any of **SEQ, MSET, CYC** and the subscript  $k$  indicates the restriction to  $k$  components in the construction.

**Seq<sub>k</sub>.** Since  $\text{SEQ}_k(\mathcal{A}) = \mathcal{A} \times \dots \times \mathcal{A}$  ( $k$  times), it suffices to generate independently the components:

$$(2.14) \quad \Gamma \text{SEQ}_k[\mathcal{A}](x) := \langle \Gamma A(x), \dots, \Gamma A(x) \rangle \quad (k \text{ times}).$$

**MSet<sub>k</sub>:** The generating function  $M_k(z)$  of  $\text{MSET}_k(\mathcal{A})$

is<sup>1</sup>

$$(2.15) \quad M_k(z) = [u^k] \exp \left( \sum_{i \geq 1} \frac{u^i}{i} A(z^i) \right).$$

This is a polynomial in  $A(z), A(z^2), \dots, A(z^k)$ . In particular, we have

$$(2.16) \quad M_2(z) = \frac{1}{2}A(z)^2 + \frac{1}{2}A(z^2),$$

$$M_3(z) = \frac{1}{6}A(z)^3 + \frac{1}{2}A(z)A(z^2) + \frac{1}{3}A(z^3).$$

Remember that  $B(z) = A(z^\ell)$  is the GF of the class  $\mathcal{B} = \mathcal{A}^{\odot \ell}$  composed of  $\ell$ -tuples of identical elements,  $\mathcal{B} = \{ \langle \alpha, \dots, \alpha \rangle | (\ell \text{ copies}), \alpha \in \mathcal{A} \}$ . Then, the idea of the algorithm is to interpret  $M_k(z)$  as a weighted union, perform the corresponding Bernoulli switch to pick up a term, and, once a monomial has been chosen, return a tuple composed of repeated elements. For instance, in the case of  $M_3$ , we may generate elements of one of the three types  $\langle \alpha, \alpha', \alpha'' \rangle, \langle \alpha, \alpha', \alpha' \rangle, \langle \alpha, \alpha, \alpha \rangle$ , in accordance with (2.16). The formal description is given in Figure 6.

**Cyc<sub>k</sub>** : The generating function of  $\text{CYC}_k$  is

$$(2.17) \quad C_k(z) = [u^k] \sum_{i \geq 1} \frac{\varphi(i)}{i} \log \frac{1}{1 - u^i A(z^i)}$$

$$(2.18) \quad = \frac{1}{k} \sum_{i|k} \varphi(i) A(z^i)^{\frac{k}{i}}.$$

We derive from it a sampler along the same lines as  $\text{MSET}_k$ , only simpler (Fig. 6).

PROPOSITION 2.3. *The generators  $\Gamma \text{SEQ}_k[\mathcal{A}](x), \Gamma \text{MSET}_k[\mathcal{A}](x), \Gamma \text{CYC}_k[\mathcal{A}](x)$  of Eq. (2.14) and Fig. 6 are valid Boltzmann sampler for sequences, multisets, and cycles of a fixed number of components  $k$ .*

*Proof.* (Sketch) Obvious for sequences. For multisets, it can be observed that the algorithm emulates  $\Gamma \text{MSET}[\mathcal{A}](x)$  conditioned upon the value of  $k$  (alternatively, use basic Burnside-Pólya theory [15, 22]). Similarly for cycles.

The combination of Propositions 2.1–2.3, and Eq. (2.2)–(2.5) completes the proof of Theorem 1.1.

Let us remark finally that one can realize samplers for constructions  $\mathfrak{R}_{\leq k}$  ( $\mathfrak{R} = \text{SEQ, MSET, CYC}$ ) involving at most  $k$  components by decomposing them as disjoint unions,  $\sum \mathfrak{R}_j$ , for  $j \leq k$ . (Dually, for  $\mathfrak{R}_{\geq k}$ , one can proceed by piling up a rejection procedure on top of unconstrained  $\mathfrak{R}$  samplers.)

<sup>1</sup>The notation  $[u^k]\Phi(u)$  represents the coefficient of  $u^k$  in the  $u$ -expansion of  $\Phi(u)$ .

```

Algorithm  $\Gamma \text{PSET}[\mathcal{A}](x)$ 
 $\mu \leftarrow \Gamma \text{MSET}[A](x); \varpi := \emptyset;$ 
for  $\gamma \in \mu$  do
    if the multiplicity of  $\gamma$  in  $\mu$  is odd
    then  $\varpi := \varpi, \gamma;$ 
return  $\varpi$ 

```

Figure 7: The rule for producing a Boltzmann sampler for PSET.

EXAMPLE 5. *Series-parallel circuits.* These are a classical abstraction of electrical circuits, where both parallel ( $\mathcal{P}$ ) and serial ( $\mathcal{S}$ ) compositions are allowed. The recursive specification is  $\mathcal{C} = \mathcal{Z} + \mathcal{S} + \mathcal{P}$ ,  $\mathcal{S} = \text{SEQ}_{\geq 2}(\mathcal{Z} + \mathcal{P})$ ,  $\mathcal{P} = \text{MSET}_{\geq 2}(\mathcal{Z} + \mathcal{S})$ . Circuits of large sizes (e.g., size  $n = 10^4$  in  $\approx 10^{11}$  machine cycles) are then easily generated. ...  $\square$

### 3 The powerset construction (PSet)

A *powerset* is a multiset in which multiplicities of elements are all equal to 1. In other words, an element of  $\mathcal{C} = \text{PSET}(\mathcal{A})$  is a finite subset (in the usual sense) of  $\mathcal{A}$ . Generating powersets is more complicated than in the case of multisets, because of the distinctness condition imposed on elements.

A serendipitous consequence of the close relationship that Boltzmann models entertain with generating functions is that suitable GF identities can often guide the design of Boltzmann samplers. From the GF expressions of Fig. 2, one deduces the fundamental identity (Vallée’s identity [9]):  $P(z)M(z^2) = M(z)$ , which reflects a fundamental combinatorial isomorphism

$$(3.19) \quad \text{MSET}(\mathcal{A}) \cong \text{PSET}(\mathcal{A}) \times \text{MSET}(\mathcal{A}^{\odot 2}).$$

(Elements can be grouped by pairs of identical elements, plus possibly an isolated element, depending on the parity of their multiplicity.) This suggests the following scheme to generate powersets: “draw a multiset; retain all elements of odd multiplicity; discard the rest” (Figure 7). Notice that this procedure has an additional cost, called *overhead*, which is the total size of the discarded elements.

THEOREM 3.1. (i) The sampler  $\Gamma \text{PSET}[\mathcal{A}](x)$  of Fig. 7 is a valid Boltzmann sampler for  $\text{PSET}(\mathcal{A})$ .

(ii) If the generating function  $A(z)$  of  $\mathcal{A}$  has a radius of convergence  $\rho$  that satisfies the condition<sup>2</sup>  $\rho < 1$ , then, for all  $x \in (0, \rho)$ , the expectation of the overhead is uniformly bounded from above by the constant  $K = \frac{2\rho^2 A'(\rho^2)}{1 - \rho^2}$ .

<sup>2</sup>This condition is satisfied by almost all specifications (in a precise measure-theoretic sense) and is algorithmically testable [9].

*Proof.* Correctness follows from a general *division lemma* (Lemma 3.1) applied to the decomposition of multisets given in (3.19).

LEMMA 3.1. (DIVISION) Let  $\mathcal{H}, \mathcal{K}, \mathcal{L}$  satisfy the isomorphism  $\mathcal{H} \cong \mathcal{K} \times \mathcal{L}$ . Given a Boltzmann sampler  $\Gamma H(x)$  for  $\mathcal{H}$ , the process of extracting the first component of an object generated by  $\Gamma H(x)$  is a valid Boltzmann sampler  $\Gamma K(x)$  for  $\mathcal{K}$ .

The complexity analysis results from the fact that the probability generating function of the overhead relative to sampler  $\Gamma \text{PSET}[\mathcal{A}](x)$  is

$$\prod_n \left( \frac{1+x^n+u^{2n}(x^{2n}+x^{3n})+\dots}{1+x^n+x^{2n}+x^{3n}+\dots} \right)^{A_n} = \prod_n \left( \frac{1-x^{2n}}{1-u^{2n}x^{2n}} \right)^{A_n}.$$

Differentiation followed by the specialization  $u = 1$  then gives  $\mathbb{E}(\text{overhead}) = \sum_{n=1}^{\infty} \frac{2nA_n x^{2n}}{1-x^{2n}}$ , a quantity easily verified to be majorized by  $K$ .

EXAMPLE 6. *Partitions of integer.* These are specifiable as  $\mathcal{P} = \text{MSET}(\mathcal{Z} \text{SEQ}(\mathcal{Z}))$ . The subclass of partitions into *distinct* summands is  $\mathcal{Q} = \text{PSET}(\mathcal{Z} \text{SEQ}(\mathcal{Z}))$ . A sampler for  $\mathcal{P}$  results from Proposition 2.1; a sampler for  $\mathcal{Q}$  derives automatically from Theorem 3.1. ....  $\square$

EXAMPLE 7. *Nonplane unlabelled trees without automorphisms.* They are specified by  $\mathcal{U} = \mathcal{Z} \times \text{PSET}(\mathcal{U})$ , with the powerset construction expressing the fact that two subtrees pending from the same node are structurally different (“identity trees” of [15, p. 64]). These trees are less ramified than unlabelled trees; see Figure 8. ....  $\square$

Generators for  $\text{PSET}_k$  and variants are then obtained by methods akin to those of the previous section.

### 4 Size-controlled samplers

Random generation usually requires us to draw objects with a target size either of a fixed value  $n$ —*exact-size sampling*—or in a range of the form  $[(1-\epsilon)n, (1+\epsilon)n]$ —*approximate-size sampling*. In the latter case, the parameter  $\epsilon$ , called the *tolerance*, is a small real number ( $\epsilon = \frac{1}{10}$  or  $\epsilon = \frac{1}{100}$  suffices for many practical purposes). In all cases, *uniformity amongst objects of the same size must be preserved*. Given the uniformity inherent in the definition of Boltzmann models, one can achieve this goal by plainly controlling size by means of *rejection*:

$$(4.20) \quad \Gamma C(x \mid \Omega_n) := \begin{array}{l} \mathbf{repeat} \ \gamma \leftarrow \Gamma C(x) \\ \mathbf{until} \ |\gamma| \in \Omega_n; \\ \mathbf{return} \ \gamma. \end{array}$$

There,  $\Omega_n = \{n\}$  (exact size) or  $\Omega_n = [(1-\epsilon)n, (1+\epsilon)n]$  (approximate size). In addition, by maintaining a global variable that records at each stage the size of

the partial object generated and aborting execution if necessary, it is possible to *avoid building any object with size exceeding the upper limit*  $\max \Omega_n$ .

The rejection technique of (4.20) can be coupled with the tuning of the value  $x_n$  of the control parameter  $x$ , which maximizes the chances of success. This tuning relies on the analysis of the random size  $N \equiv N_x$  of the object produced by a free Boltzmann sampler  $\Gamma C(x)$ :

$$\mathbb{E}_x(u^N) = \frac{C(xu)}{C(x)}, \quad \mathbb{E}_x(N) = \frac{x C'(x)}{C(x)},$$

$$\mathbb{V}_x(N) = \frac{x^2 C''(x)}{C(x)} + \frac{x C'(x)}{C(x)} - \left( \frac{x C'(x)}{C(x)} \right)^2.$$

A fruitful optimisation technique is the *targetting heuristic*: “choose  $x := x_n$  which satisfies the relation  $\mathbb{E}_x(N) = n$ ”. (Due to properties of Boltzmann distributions an approximate solution is normally sufficient.) In all cases, one needs to take  $x_n \rightarrow \rho_C$  as  $n \rightarrow \infty$ .

The behaviour of the rejection sampler (4.20) for a class  $\mathcal{C}$  then tightly depends on the singularity type of the generating function  $C(x)$ , that is, on the behaviour of  $C(x)$  as  $x$  approaches the critical (singular) value  $\rho_C$ . We refer to the discussion in [8]. For several commonly encountered singularity types, a combination of rejection and the targetting heuristic leads to *approximate-size random generators having linear-time complexity*. This applies in particular to regular languages.

**THEOREM 4.1.** *Given a regular language  $\mathcal{L}$  and a fixed  $\epsilon > 0$ , there exists an approximate-size sampler with expected linear-time complexity for  $\mathcal{L}$ . If the automaton recognizing  $\mathcal{L}$  corresponds to a strongly connected digraph, then there exists an exact-size sampler with expected linear-time complexity.*

*Proof.* From the automaton, a recursive specification for  $\mathcal{L}$  involving  $\{+, \times\}$  is derived. This gives rise to a Boltzmann sampler  $\Gamma L(x)$  (using Eq. (2.2)-(2.3)), which is equivalent to a Markov chain with suitable transition probabilities. (The transformation is nontrivial—since the automaton is not necessarily complete, we emulate in fact a substochastic matrix.) The approximate-size (resp. exact-size) sampler is obtained by running the derived Markov chain for  $x$  chosen according to the targetting heuristic (resp. with  $x = \rho_L$  and with rejection if the state at the  $n$ th step is not the final state).

The next theorem applies to any finite collection of classes of trees with degrees constrained to fixed finite sets, where the classes are bound by a context-free like grammar. This covers term trees in logic and symbolic calculation (even in the case of typed operators) and it

allows for various commutativity rules. It also covers the case of acyclic molecules formed from atoms endowed with specific valencies (see [12] for the cyclic case).

**THEOREM 4.2.** *Consider a class  $\mathcal{C}$  having a recursive specification*

$$\{\mathcal{F}_1 = \Psi_1(\mathcal{Z}; \mathcal{F}_1, \dots, \mathcal{F}_m), \dots, \mathcal{F}_m = \Psi_m(\mathcal{Z}; \mathcal{F}_1, \dots, \mathcal{F}_m)\},$$

where the  $\Psi_i$ 's are taken from the collection of constructors  $\{+, \times, \text{SEQ}, \text{MSET}, \text{CYC}, \text{SEQ}_k, \text{MSET}_k, \text{CYC}_k\}$ . Assume that the dependency graph of the  $\mathcal{F}_i$ 's is strongly connected. Then an approximate-size sampler and an exact-size sampler can be derived for  $\mathcal{C}$ , having respectively linear and quadratic expected running time.

*Proof.* We make use of a singular sampler defined by the limit value  $x = \rho$  (see [8]). The coefficients of  $\mathcal{C}$  obey the universal asymptotic estimate  $C_n \sim c\rho^{-n}n^{-3/2}$ . This results from the Drmota-Lalley-Woods theorem [9], with an adaptation to Pólya theory of which [1] is typical. The expected complexity of  $\Gamma C(\rho)$  turns out to be  $O(\sqrt{n})$ , while the success probabilities for approximate-size and exact-size sampling are of respective order  $O(n^{-1/2})$  and  $O(n^{-3/2})$ . The statement follows.

**EXAMPLE 8.** *Acyclic alcohols.* These molecules, enumerated by Pólya [22], correspond to the specification  $\mathcal{A} = \mathcal{Z} + \mathcal{Z} \times \text{MSET}_3(\mathcal{A})$ . An example is drawn in Fig. 8. A related example is the class of Otter trees (nonplane binary trees),  $\mathcal{O} = \mathcal{Z} + \mathcal{Z} \times \text{MSET}_2(\mathcal{O})$ , which correspond to terms built on a commutative operation. . . . .  $\square$

## 5 Realization of Boltzmann samplers

There are two basic choices for actually implementing computations over the real abstract domain  $\mathbb{R}$  used so far.

**EIA** *Exact interval arithmetic* is a way to implement exact real computations, refining estimates in an adaptive manner (see [6] for similar context).

**FPA** *Fixed precision arithmetic* consists in adopting a large enough (but fixed) floating point precision for all real computations.

We have implemented the second strategy, **FPA**, using 20 digits of accuracy in the calculations of values of generating functions. Statistical tests conducted on various simulation results indicate no detectable bias. In other words, the deviation from uniformity, though mathematically nonzero, is “small” (and, in a sense, of the order of  $10^{-20}$ ). Also, the oracle is, under such circumstances, efficiently realized by a combination of recursive calls mimicking the functional

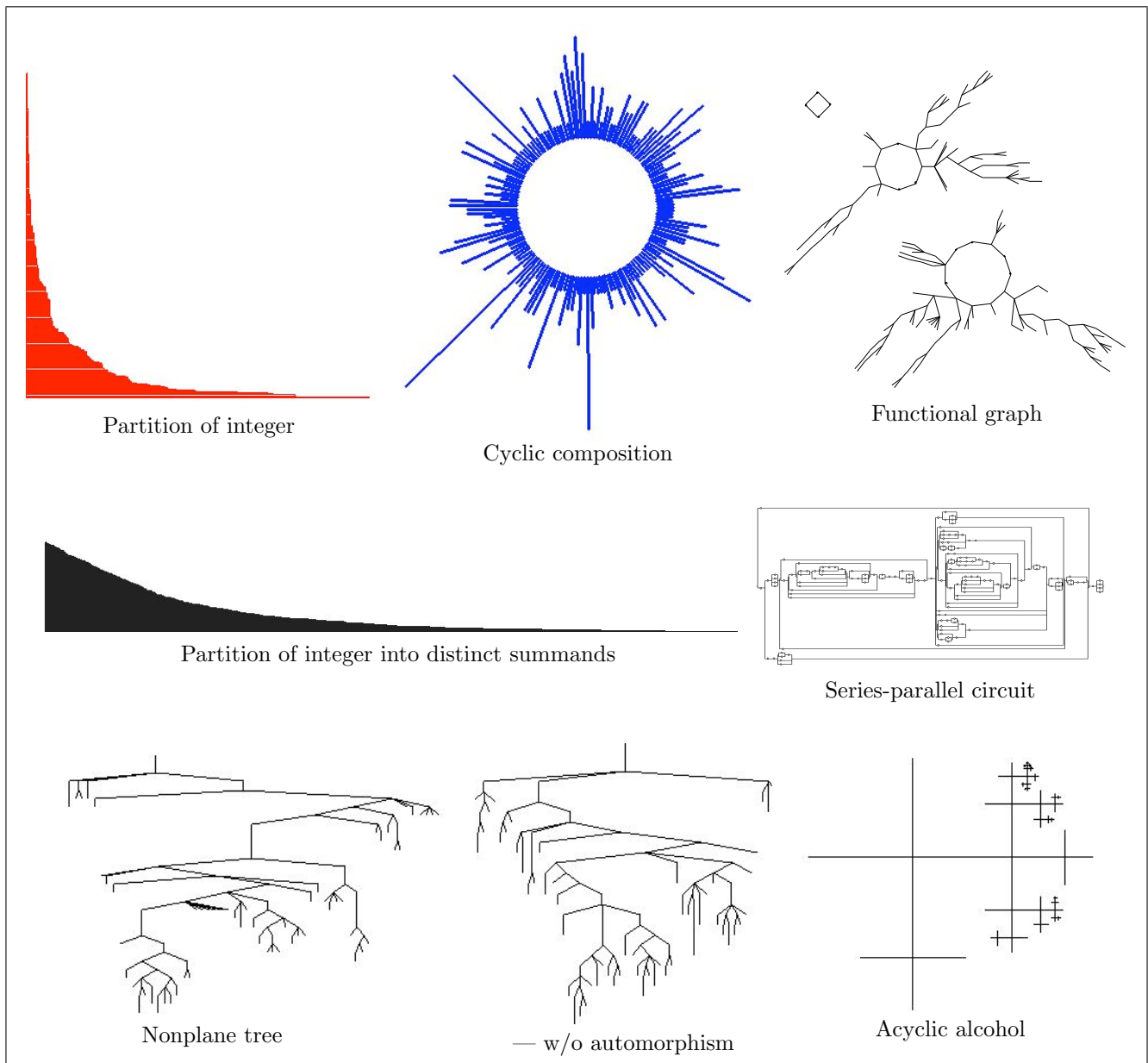


Figure 8: Examples of generated objects.

| Class                        | Approx.      | Exact        | Method  | $10^4 \pm 10\%$ | $10^5 \pm 10\%$ |
|------------------------------|--------------|--------------|---------|-----------------|-----------------|
| Partitions                   | $O(n^{1/2})$ | $O(n^{5/4})$ | target. | 0.1s            | 1s              |
| Partitions distinct summands | $O(n^{1/2})$ | $O(n^{5/4})$ | target. | 0.1s            | 1s              |
| VC polyominoes               | $O(n)$       | $O(n^2)$     | target. | 15s             | 130s            |
| Nonplane trees               | $O(n)$       | $O(n^2)$     | sing.   | 10s             | 100s            |
| Nonplane binary trees        | $O(n)$       | $O(n^2)$     | sing.   | 5s              | 90s             |
| Trees w/o automorphisms      | -            | -            | sing.   | 10s             | 180s            |
| Series-parallel circuits     | -            | -            | sing.   | 60s             | -               |
| Necklaces                    | $O(n)$       | $O(n^2)$     | target. | 1s              | 10s             |
| Circular compositions        | $O(n)$       | $O(n^2)$     | target. | 3s              | 30s             |
| Functional graphs            | $O(n)$       | $O(n^2)$     | target. | 10s             | 100s            |

Figure 9: Complexity and computation time of example classes.

equations of Fig. 2, memorization on the fly, and Newton’s method. Altogether, for each of our pilot examples, the computation time associated to the **FPA** implementation of the oracle appears to be negligible.

Figure 9 indicates, for a variety of classes, the complexity and the observed computation time<sup>3</sup> of a Boltzmann generation. The case of integer partitions is worthy of note since the complexity of approximate-size sampling is *sublinear*— this makes it possible to attain sizes well in range of  $10^9$ – $10^{10}$ . For all other classes listed in Fig. 9, the complexity of approximate-size sampling is linear. Objects with sizes of  $10^4$  to  $10^5$  are effectively drawn by means of at most  $10^{12}$  machine cycles. Figures 8 and 10 display some random objects generated under our implementation.

## References

- [1] Jason P. Bell, Stanley N. Burris, and Karen A. Yeats. Counting rooted trees: The universal law  $t(n) \sim cn^{-3/2}$ , July 2005. Available at <http://arxiv.org/abs/math.CO/0512432>.
- [2] F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial species and tree-like structures*. Cambridge University Press, Cambridge, 1998.
- [3] Olivier Bodini, Éric Fusy, and Carine Pivoteau. Random sampling of plane partitions. In Renzo Pinzani and Vincent Vajnovszki, editors, *Gascom 2006*, pages 124–135, Dijon, France, 2006. LE2I.
- [4] A. Denise, M.-C. Gaudel, and S.-D. Gouraud. A generic method for statistical testing. In *ISSRE ’04: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE’04)*, pages 25–34, Washington, DC, USA, 2004. IEEE Computer Society.
- [5] Alain Denise, Yann Ponty, and Michel Termier. Random generation of structured genomic sequences. In *RECOMB 2003*, page 3 pages (poster), Berlin, April 2003.

<sup>3</sup>Our prototype is implemented under the symbolic manipulation system MAPLE.

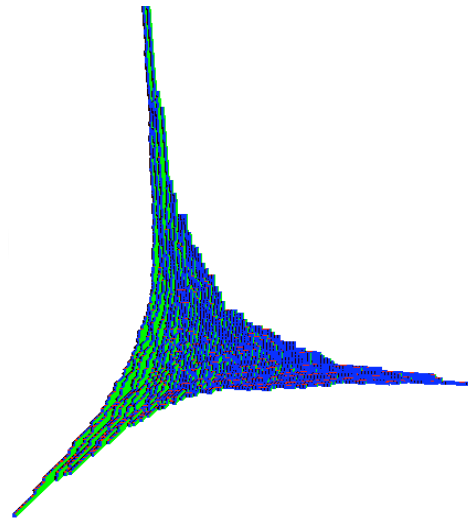


Figure 10: A plane partition of size about 15000, generated by a Boltzmann sampler [3].

- [6] Alain Denise and Paul Zimmermann. Uniform random generation of decomposable structures using floating-point arithmetic. *Theoretical Computer Science*, 218(2):233–248, 1999.
- [7] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer Verlag, 1986.
- [8] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4–5):577–625, 2004. Special issue on Analysis of Algorithms.
- [9] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. October 2005. Chapters I–IX of a book to be published, 688p.+x, available electronically from P. Flajolet’s home page.
- [10] Philippe Flajolet, Paul Zimmerman, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994.
- [11] Éric Fusy. Quadratic exact-size and linear approximate-size random sampling of planar graphs. *Discrete Mathematics and Theoretical Computer Science*, AD:125–138, 2005. Proceedings of 2005 International Conference on Analysis of Algorithms.
- [12] Leslie Ann Goldberg and Mark Jerrum. Randomly sampling molecules. 1997.
- [13] Vivek Gore, Mark Jerrum, Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Information and Computation*, 134(1):59–74, 1997.
- [14] Ian P. Goulden and David M. Jackson. *Combinatorial Enumeration*. John Wiley, New York, 1983.
- [15] Frank Harary and Edgar M. Palmer. *Graphical Enumeration*. Academic Press, 1973.
- [16] Dean Hickerson. Counting horizontally convex polyominoes. *Journal of Integer Sequences*, 2, 1999. Electronic.
- [17] Mark Jerrum. Uniform sampling modulo a group of symmetries using markov chain simulation. Technical Report ECS-LFCS-94-288, University of Edinburgh, 1994.
- [18] Donald E. Knuth. *The Art of Computer Programming*, volume 2: Seminumerical Algorithms. Addison-Wesley, 3rd edition, 1998.
- [19] Donald E. Knuth. *The Art of Computer Programming: Volume 4, Combinatorial Algorithms*. Addison Wesley, 2005–6. Fascicles 2-4.
- [20] A. Meir and J. W. Moon. On random mapping patterns. *Combinatorica*, 4(1):61–70, 1984.
- [21] Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms*. Academic Press, second edition, 1978.
- [22] G. Pólya and R. C. Read. *Combinatorial Enumeration of Groups, Graphs and Chemical Compounds*. Springer Verlag, New York, 1987.
- [23] Richard P. Stanley. *Enumerative Combinatorics*, volume I. Wadsworth & Brooks/Cole, 1986.

P.F., E.F., C.P.: Algorithms Project, INRIA Rocquencourt, F-78153 Le Chesnay (France)

C.P.: Université Pierre et Marie Curie, LIP6 -Équipe CalFor, 8, rue du Capitaine Scott, 75015 Paris (France)