Non-redundant random generation algorithms for weighted context-free languages

Andy Lorenz, Yann Ponty

Non-redundant random generation algorithms for weighted context-free languages

Andy Lorenz
Mathematics Departement
Denison University, USA

Yann Ponty[1]
CNRS/INRIA AMIB
Ecole Polytechnique, France

Abstract

We address the non-redundant random generation of k words of length n from a context-free language. Additionally, we want to avoid a predefined set of words. We study a rejection-based approach, whose worst-case time complexity is shown to grow exponentially with k for some specifications and in the limit case of a coupon collector. We propose two algorithms respectively based on the recursive method and on an unranking approach. We show how careful implementations of these algorithms allows for the non-redundant generation of k words of size n in $\mathcal{O}(k \cdot n \cdot \log n)$ arithmetic operations, after precomputation of $\Theta(n)$ numbers. The overall complexity is therefore dominated by the generation of k words, and the non-redundancy comes at a negligible cost.

1. Introduction

The random generation of combinatorial objects has many direct applications in areas ranging from software testing [5] to bioinformatics [19]. It can help formulate conjectures on the average-case complexity of algorithms [2], raises new fundamental mathematical questions, and motivates new developments on its underlying objects. These include, but are not limited to, generating functionology, arbitrary precision arithmetics and bijective combinatorics. Following the so-called *recursive* framework introduced by Wilf [24], very elegant and general algorithms for the uniform random generation have been designed [15] and implemented. Many optimizations of this approach have been developed, using specificities of certain classes of combinatorial structures [16], or floating-point arithmetics [8]. More recently a probabilistic approach to this problem, the so-called Boltzmann generation [11], has drawn much attention both because its very low memory complexity and its underlying theoretical beauty.

For many applications, it is necessary to depart from *uniform* models [9, 4]. A striking example lies in a recent paradigm for the *in silico* analysis of the RNA molecule's folding. Instead of trying to predict a conformation of minimal free-energy, current approaches tend to focus on the *ensemble properties* of realizable conformations, assuming a Boltzmann probability distribution [9] on each and every compatible structures. Random generation is then performed, and complex structural features are evaluated in a statistical manner. In order to capture such features, a general non-uniform scheme was introduced by Denise *et al* [7], based on the concept of *weighted context-free grammars*.

[1]To whom correspondence should be addressed.

Recursive random generation algorithms were derived, with time and space complexities equivalent to that observed within the uniform distribution [15]. This initial work was later completed toward general decomposable classes [6] and a Boltzmann weighted sampling scheme, used as a preliminary step within a rejection-based algorithm for the multidimensional sampling of languages [3].

In a weighted probability distribution, the probability ratio between the most and least frequent words typically grows exponentially on the size of the generated objects. Therefore a typical set of independently generated objects may feature a large number of copies of the heaviest objects. This redundancy, which can be useful in some context to estimate the probability of each sample from its frequency, is completely uninformative in the context of weighted random generation, as the exact probability of any sampled object can be derived in a straightforward manner. Consequently it is a natural question to address the **non-redundant random generation** of combinatorial objects, that is the generation of a set of **distinct** objects.

This non-redundant version of random generation has, to the best of our knowledge, only been addressed through the introduction of the `PowerSet` construct by Zimmermann [26]. An algorithm in $\Theta(n^2)$ arithmetic operations, or a practical $\Theta(n^4)$ complexity in this case, was derived for recursive decomposable structures. The absence of redundancy in the set of generated structures was achieved respectively through *rejection* or an *unranking* algorithms. While the former is discussed later on in the document, the latter cannot be transposed directly to the case of weighted languages, since the assumption that different integral ranks correspond to different objects does not hold.

Another approach for a non-redundant generation may be based on the so-called unranking method, which starts by picking a number at random, and then uses a backtracking procedure to determine which word bijectively correspond to the number in a total ordering of the language. Martinez and Molinero [18] give a general approach for unranking labeled combinatorial classes, which includes context-free grammars. In the context of weighted grammar, one additionally wants to Weinberg and Nebel [23] generalize this to non-uniform generation involving weighted context-free grammars. However, their approach was restricted to integral weights, which required a sometimes complex transformation of the grammar, modification which impacts the complexity of the algorithm.

In this paper, we address the non-redundant generation of words from a context-free language. We remind an introduce in Section 2 some concepts and definitions related to weighted languages, and define our objective. In Section 3, we analyze the shortcomings of a naive rejection approach. We show that, although well-suited for the uniform distribution, the rejection approach can yield high average-case complexities for large sets of forbidden words and or large values of weights. Then in Section 4 we introduce the concept of immature words, which allows us to rephrase the random generation process as a *step-by-step* process. The algorithm uses the so-called recursive method, coupled with a custom data structure, to perform a generation k sequences of length n at the cost of $\mathcal{O}(k \cdot n \log(n))$ arithmetic operations after a precomputation in $\Theta(n)$ arithmetic operations. We also propose in Section 5 an unranking algorithm for weighted grammars, which we again couple with a dedicated data structure that stores and helps avoid any forbidden word, giving rise again to a $\mathcal{O}(k \cdot n \log(n))$ algorithm after $\Theta(n)$ arithmetic operations. We conclude in Section 6 with a summary of our propositions and results, and outline some perspectives and open questions.

2. Notations and concepts

2.1. Context-free grammars

Let us remind, for the sake of completeness, some basic language-theoretic definitions. A **context-free grammar** is a 4-tuple $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{P}, \mathcal{S})$ where

- Σ is the alphabet, i.e. a finite set of terminal symbols.
- $\mathcal{N}$ is a finite set of non-terminal symbols.
- $\mathcal{P}$ is the finite set of production rules, each of the form $N \to X$, for $N \in \mathcal{N}$ any non-terminal and $X \in \{\Sigma \cup \mathcal{N}\}^*$.
- $\mathcal{S}$ is the **axiom** of the grammar, i. e. the initial non-terminal.

A grammar $\mathcal{G}$ is then said to be in **Chomsky Normal Form** (CNF) iff the rules associated to each non-terminal $N \in \mathcal{N}$ are of the form:

- Product case: $N \to N' \,.\, N''$
- Union case: $N \to N' \mid N''$
- Terminal case: $N \to t$

for $N, N', N'' \in \mathcal{N}$ non-terminal symbols and $t \in \Sigma$ a terminal symbol. In addition our grammars will be ε-free, i.e. only the axiom $\mathcal{S} \in \mathcal{N}$ is allowed to derive the empty word ε, provided that $\mathcal{S}$ never appears in the right-hand side of a production. Since it is a classic result that any context-free grammar $\mathcal{G}$ admits a CNF ε-free grammar $\mathcal{G}'$ generating the same language, then our algorithms will be defined on CNF ε-free grammars without loss of generality.

Let $\mathcal{L}(N)$ be the **language** associated to $N \in \Sigma$ within a grammar $\mathcal{G}$, that is the set of words composed of terminal symbols that can be generated starting from N through a sequence of derivations, recursively defined as

$$\mathcal{L}(N) = \begin{cases} \mathcal{L}(N') \times \mathcal{L}(N'') & \text{If } N \to N' \,.\, N'' \\ \mathcal{L}(N') \cup \mathcal{L}(N'') & \text{If } N \to N' \mid N'' \\ \{t\} & \text{If } N \to t \\ \varepsilon & \text{If } N \to \varepsilon \end{cases} \tag{2.1}$$

Then the language $\mathcal{L}(\mathcal{G})$ generated by a grammar $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{P}, \mathcal{S})$ is defined as $\mathcal{L}(\mathcal{S})$ the language associated with the axiom $\mathcal{S}$. Finally, let us denote by $\mathcal{L}_n$ the restriction of a language $\mathcal{L}$ to its words of length n.

2.2. Weighted context-free grammars

Definition 2.1 (Weighted Grammar [7])**.** A weighted grammar $\mathcal{G}_\pi$ is a 5-tuple $\mathcal{G}_\pi = (\pi, \Sigma, \mathcal{N}, \mathcal{P}, \mathcal{S})$ where Σ, $\mathcal{N}$, $\mathcal{P}$ and $\mathcal{S}$ are the members of a context-free grammar, and $\pi : \Sigma \to \mathbb{R}$ is a weighting function, that associates a real-valued weight π_t to each terminal symbol t.

This notion of weight naturally extends to a mature word w in a multiplicative fashion, i.e. such that $\pi(w) = \prod_{i=1}^{|w|} \pi_{w_i}$, and additively on any set of word $\mathcal{L}$ through $\pi(\mathcal{L}) = \sum_{w \in \mathcal{L}} \pi(w)$. One can then define a π-**weighted probability distribution** over a set $\mathcal{L}$ of words, such that

$$\mathbb{P}(w \mid \pi, \mathcal{L}) = \frac{\pi(w)}{\sum\limits_{w' \in \mathcal{L}} \pi(w')} = \frac{\pi(w)}{\pi(\mathcal{L})}, \ \forall w \in \mathcal{L}. \tag{2.2}$$

Algorithm 1 Non-redundant sequential meta-algorithm for the generation of k distinct words of length n, from a (weighted) context-free grammar $\mathcal{G}_\pi = (\pi, \Sigma, \mathcal{N}, \mathcal{P}, \mathcal{S})$, avoiding a forbidden set of words $\mathcal{F}$.

NonRedundantSequential$(\mathcal{G}_\pi, k, n, \mathcal{F})$:
  Perform some precomputations...
  $\mathcal{R} \leftarrow \varnothing$
  while $|\mathcal{R}| \leq k$ **do**
    $x \leftarrow$ **DrawNonRed**$(\mathcal{S}_n, \pi(N_n), \mathcal{G}_\pi, \mathcal{F})$ // Any non-redundant algorithm
    Update some data structure...
    $(\mathcal{R}, \mathcal{F}) \leftarrow (\mathcal{R} \cup \{x\}, \mathcal{F} \cup \{x\})$
  end while
  return $\mathcal{R}$

In previous works, the random generation of words of a given length n in a context-free language with respect to a weighted probability distribution has been addressed and an algorithm in $\mathcal{O}(n \log n)$ after $\mathcal{O}(n^2)$ arithmetic operations was derived [7] and implemented [19].

2.3. Problem statement

In the following, we will consider algorithmic solutions for the non-redundant generation of words from a weighted context-free language. In other words, one wants to generate any subset $\mathcal{S}$ of a language $\mathcal{L}$ with respect to the probability distribution naturally induced by random independent generations, conditioned to stopping when k distinct samples are obtained. For any $\mathcal{R} \subseteq \mathcal{L}$, this targeted probability is given by

$$\mathbb{P}(\mathcal{R} \mid k, n) = \begin{cases} \displaystyle\sum_{\sigma \in \mathfrak{S}(\mathcal{R})} \prod_{i=1}^{k} \frac{\mathbb{P}(\sigma_i \mid \pi, \mathcal{L}_n)}{1 - \sum_{j=1}^{i-1} \mathbb{P}(\sigma_j \mid \pi, \mathcal{L}_n)} & \text{If } |\mathcal{R}| = k, \\ 0 & \text{Otherwise.} \end{cases} \tag{2.3}$$

Let us then formally restate the problem as:

> Weighted-Non-Redundant-Generation (wnrg)
>
> Input: A weighted grammar $\mathcal{G}_\pi$, a length n and a targeted number of words k.
>
> Output: A set of words $R \subseteq \mathcal{L}(\mathcal{G})_n$ of cardinality k with probability $\mathbb{P}(\mathcal{R} \mid k, n)$.

Notice that the distribution of Equation 2.3 naturally arises from a sequence of dependent calls $(r_1, r_2, \ldots, r_k)$ to weighted generators for $\mathcal{L}$, avoiding sets of words $\varnothing$, $\{r_1\}, \ldots, \{r_1, r_2, \ldots, r_{k-1}\}$ respectively, as implemented in Algorithm 1. It follows that the problem of generating a **set of words** with respect to the distribution of Equation 2.3 **can be reduced to drawing a single word** w within a weighted probability distribution avoiding a given set of forbidden words $\mathcal{F}$, i.e. such that $\mathbb{P}(w \mid \pi, \mathcal{L} \backslash \mathcal{F})$.

3. Naive rejection algorithm

A **naive rejection approach** for this problem consists in drawing words at random in an unconstrained way, rejecting those from the forbidden set until a valid word is

Algorithm 2 Naive rejection algorithm for generating a word of length n, from a (weighted) context-free grammar $\mathcal{G}_\pi$, avoiding a forbidden set of words $\mathcal{F}$.

NaiveRejection$(\mathcal{G}_\pi, n, \mathcal{F})$:
  repeat
    $t \leftarrow$ **draw**$(\mathcal{G}_\pi, n)$ // One may use any available generation algorithm.
  until $t \notin \mathcal{F}$
  return t

generated. This strategy was prescribed by Zimmermann [26] for the uniform distribution of objects in general recursive specifications. This rejection strategy is implemented by Algorithm 2 relies on an auxiliary generator **draw(**$\cdots$**)** of words from a (weighted) context-free languages, for which we refer to previous works of Flajolet *et al* [15, 11], or Denise *et al* [8] that achieves a $\mathcal{O}(n^{1+\varepsilon})$ complexity through non-trivial floating-point arithmetics.

Proposition 3.1 (Correctness of a naive rejection algorithm)**.** *Any word returned by Algorithm 2 is drawn with respect to the weighted distribution on* $\mathcal{L}(\mathcal{G})_n \backslash \mathcal{F}$.

Proof. Let w be the word returned by the algorithm, and $\mathcal{F} = \{f_i\}_{i=1}^{|\mathcal{F}|}$. Let us characterize the sequences of words generated by **draw** and leading to the generation of w, by mean of a rational expression over an alphabet $\mathcal{F} \cup \{w\}$:

$$\mathcal{R}_w = (f_1 \mid f_2 \mid \cdots \mid f_{|\mathcal{F}|})^*.w.$$

Let $p_x = \pi(x)/\pi(\mathcal{L}(\mathcal{G})_n)$ the probability of emission of any – possibly forbidden – word $x \in \mathcal{L}(\mathcal{G})_n$, then the cumulated probability of the sequences of calls to **draw**, leading to the generation of w, is such that

$$\begin{aligned}\mathbb{P}(\mathbf{w}) &= p_w + \left(\sum_{i=1}^{|\mathcal{F}|} p_{f_i}\right) \cdot p_w + \left(\sum_{i=1}^{|\mathcal{F}|} p_{f_i}\right) \cdot \left(\sum_{i=1}^{|\mathcal{F}|} p_{f_i}\right) \cdot p_w + \cdots \\ &= \frac{p_w}{1 - \sum_{i=1}^{|\mathcal{F}|} p_{f_i}} = \frac{\pi(w)}{\pi(\mathcal{L}(\mathcal{G})_n) - \sum_{i=1}^{|\mathcal{F}|} \pi(f_i)} = \frac{\pi(w)}{\pi(\mathcal{L}(\mathcal{G})_n \backslash \mathcal{F})}.\end{aligned}$$

□

3.1. Complexity analysis: Uniform distribution

Let $\mathcal{L}$ be a context-free language, $n \in \mathbb{N}^+$ a positive integer and $\mathcal{F} \subset \mathcal{L}_n$ a set of forbidden words.

Theorem 3.2. *In the uniform distribution, a naive rejection implemented in Algorithm 2 leads to an average-case complexity in* $\mathcal{O}\left(\left(\frac{|\mathcal{L}_n|}{|\mathcal{L}_n| - |\mathcal{F}|}\right) \cdot n^{1+\varepsilon} \cdot k \cdot \log k\right)$.

Proof. In the uniform model when $\mathcal{F} = \varnothing$, the number of attempts required by the generation of the i-th word only depends on i and is independent from prior events. Thus the average total number $X_{n,k}$ of trials for generating of k words of size n is given by

$$\mathbb{E}(X_{n,k}) = \sum_{i=0}^{k-1} \frac{l_n}{l_n - i} = l_n(\mathcal{H}_{l_n} - \mathcal{H}_{l_n - k})$$

where $l_n := |\mathcal{L}_n|$ is the number of words of size n in the language and $\mathcal{H}_i$ the harmonic number of order i, as pointed out by Flajolet *et al* [13]. It follows that $\mathbb{E}(X_{n,k})$ is

trivially increasing with k, while remaining upper bounded by $k\mathcal{H}_k \in \Theta(k\log(k))$ when $k = l_n$ (Coupon collector problem). Since the expected number of rejections due to a non-empty forbidden set $\mathcal{F}$ remains the same throughout the generation, and does not have any influence over the generated sequences, it can be considered independently and contributes to a factor $\frac{|\mathcal{L}_n|}{|\mathcal{L}_n|-|\mathcal{F}|}$. Finally, each call to **draw** takes time $\mathcal{O}(n^{1+\varepsilon})$, regardless of the generated sequence and cardinality of $\mathcal{L}_n$. □

The complexity of the naive rejection then remains largely unaffected by the cost of rejections. Furthermore, let us mention that, for any context-free language $\mathcal{L}$, there exists a constant $\alpha \in \mathbb{R}^+$ such that $\mathcal{L}_n \in O(\alpha^n)$. It follows that the *per-sample* cost of all rejections only increases the generation by a factor which is linear on n, even for generating the whole language (coupon collector). Also, it is worth mentioning that the worst-case time complexity of the algorithms remains trivially unbounded.

3.2. Complexity analysis: Weighted languages

By contrast with the uniform case, the rejection approach to the non-redundant random generation for **weighted context-free languages** can yield an **exponential complexity** on k, even when starting from an empty set of forbidden words $\mathcal{F} = \varnothing$.

Proposition 3.3. *There exists grammars such that the generation of k distinct words, starting from an empty initial forbidden set $\mathcal{F} = \varnothing$, requires a number of call to* **draw** *exponential on k.*

Proof. Consider the following grammar, generating the language denoted by the regular expression a^*b^*:

$$S \to a\ .\ S \mid T \qquad\qquad T \to b\ .\ T \mid \varepsilon$$

We adjoin a weight function π to this grammar, such that $\pi(b) := \alpha > 1$ and $\pi(a) := 1$. The probability of any word $\omega_m := a^{n-m}b^m$ in the language is

$$\mathbb{P}(\omega_m) = \frac{\pi(\omega_m)}{\sum_{\substack{\omega\in\mathcal{L}(S)\\|\omega|=n}}\pi(\omega)} = \frac{\alpha^m}{\sum_{i=0}^{n}\alpha^i} = \frac{\alpha^{m+1}-\alpha^m}{\alpha^{n+1}-1} < \alpha^{m-n}.$$

Now consider the set $\mathcal{V}_{n,k} \subset \mathcal{S}_n$ of words having less than $n-k$ occurrences of the symbol b. The probability of generating a word from $\mathcal{V}_{n,k}$ is then

$$\mathbb{P}(\mathcal{V}_{n,k}) = \sum_{i=0}^{n-k}\mathbb{P}(\omega_{n-k-i}) = \frac{\alpha^{n-k+1}-1}{\alpha^{n+1}-1} < \alpha^{-k}$$

The expected number of generations before a sequence from $\mathcal{V}_{n,k}$ is generated is then lower-bounded by α^k. Since any non-redundant set of k sequences issued from $\mathcal{S}_n$ must contain at least one sequence from $\mathcal{V}_{n,k}$, then the average-case time complexity of the rejection approach is in $\Omega(n\cdot\alpha^k)$, i.e. exponential on k the number of words. □

However the above example, being a regular language, is not very typical of the rejection algorithm's behavior on a general context-free language. By contrast, it can be shown that, under a natural assumption, no word can asymptotically contribute up to a significant proportion of the distribution in simple type grammars.

Proposition 3.4. *Let $\mathcal{G}_\pi = (\pi, \Sigma, \mathcal{N}, \mathcal{S}, \mathcal{P})$ be a weighted grammar of simple type*[2]*. Let ω_n^0 be the word of length n generated from $\mathcal{G}_\pi$ with highest probability (i.e. weight) in the*

[2] A grammar of simple type is mainly a grammar whose dependency graph is strongly-connected and whose number of words follow an aperiodic progression (See [12] for a more complete definition). Such a grammar can easily be found for the avatars of the algebraic class of combinatorial structures (Dyck words, Motzkin paths, trees of fixed degree,...), all of which can be interpreted as trees.

π-weighted distribution. Additionally, let us assume that there exists $\alpha, \kappa \in \mathbb{R}^+$ positive constants such that $\pi(\omega_n^0) \underset{n\to\infty}{\longrightarrow} \kappa\alpha^n$.
Then the probability of ω^0 tends to 0 when $n \to \infty$:

$$\mathbb{P}(\omega^0 \mid \pi) = \frac{\pi(\omega^0)}{\pi(\mathcal{L}(\mathcal{G}_\pi)_n)} \underset{n\to\infty}{\longrightarrow} 0$$

Proof. From the Drmota-Lalley-Woods theorem [10, 17, 25], we know that the generating function of a simple type grammar has a *square-root type* singularity, and its coefficients admits an expansion of the form $\frac{\kappa\beta^n}{n\sqrt{n}}(1 + \mathcal{O}(1/n))$. This property holds in the case of weighted context-free grammars, where the coefficients are now the overall weights $W_n := \pi(\mathcal{L}(\mathcal{G}_\pi)_n)$. Since ω_n^0 is contributing to W_n, then $\pi(\omega_n^0) \leq \pi(\mathcal{L}(\mathcal{G}_\pi)_n)$ and therefore $\beta > \alpha$. □

Lastly one can exhibit grammars such that, for any fixed length n, the set of words $\mathcal{M}$ having maximal number of occurrences of a given symbol t has total probability 1 when $\pi(t) \to \infty$. It follows that sampling more than $|\mathcal{M}|$ words can be extremely time-consuming for finite lengths if one of the weights dominates the others by several orders of magnitude.

Finally, it is worth noticing that in non-degenerate context-free languages, the weight ratio between the heaviest and lightest words grows like $\Theta(\alpha^n)$, where $\alpha > 1$ will depend on structural properties of the language and on the weight function π. In particular, α can be made arbitrarily large by increasing the weight $\pi(t)$ of some terminal t. For instance in Motzkin words, presented in Figure 1, setting $\pi(a) = \pi(b) = 1$ and $\pi(c) = x$ will give a weight ratio of x^n between the words c^n and $(a.b)^{n/2}$. It follows that sampling $k = |\mathcal{L}(\mathcal{G}_\pi)_n|$ (Coupon Collector) will require at least $\Theta(x^n)$ calls to **draw**, since the lightest words has probability at most x^{-n}. Since the number of words in a context-free language is bounded by $|\Sigma|^n$ independently of the weight, then the *average cost per generation* can be made to grow exponentially with n.

4. A step-by-step recursive algorithm

A common feature of generic algorithms for the random generation of combinatorial objects [15, 7] is to consider non-terminal symbols as **independent generators**. Namely they draw words from a $N \to N'.N''$ non-terminal by making two independent calls to dedicated generators for N' and N'' (either directly for Boltzmann sampling, or after figuring out suitable lengths for N' and N'' in the case of the recursive method). Unfortunately when one wants to avoid a set of words, these two generators become correlated.

As a minimal example illustrating this claim, let us consider the following unweighted grammar $\mathcal{G}$, whose axiom is N:

$$N \to N'.N'' \qquad\qquad N' \to a \mid b \qquad\qquad N'' \to a \mid b.$$

Both the recursive method and Boltzmann sampling would assign probabilities to the derivations of N' such that $(N' \to_{1/2} a, N' \to_{1/2} b)$ (resp. $(N'' \to_{1/2} a, N'' \to_{1/2} b)$ to N''). Let us assume that a set $\mathcal{F} = \{aa\}$ has to be avoided, and that a sequential choice of derivations is adopted such that N' is fully derived before considering N''. Then the probabilities assigned to the derivations of N' must reflect the future unavailability of some derivations for N''. Hence generating uniformly within $\mathcal{L}(\mathcal{G})/\mathcal{F} = \{ab, ba, bb\}$ will require using probabilities $(N' \to_{1/3} a, N' \to_{2/3} b)$. Moreover the probabilities used to derive N'' while remaining unbiased will depend on the word produced from N', and

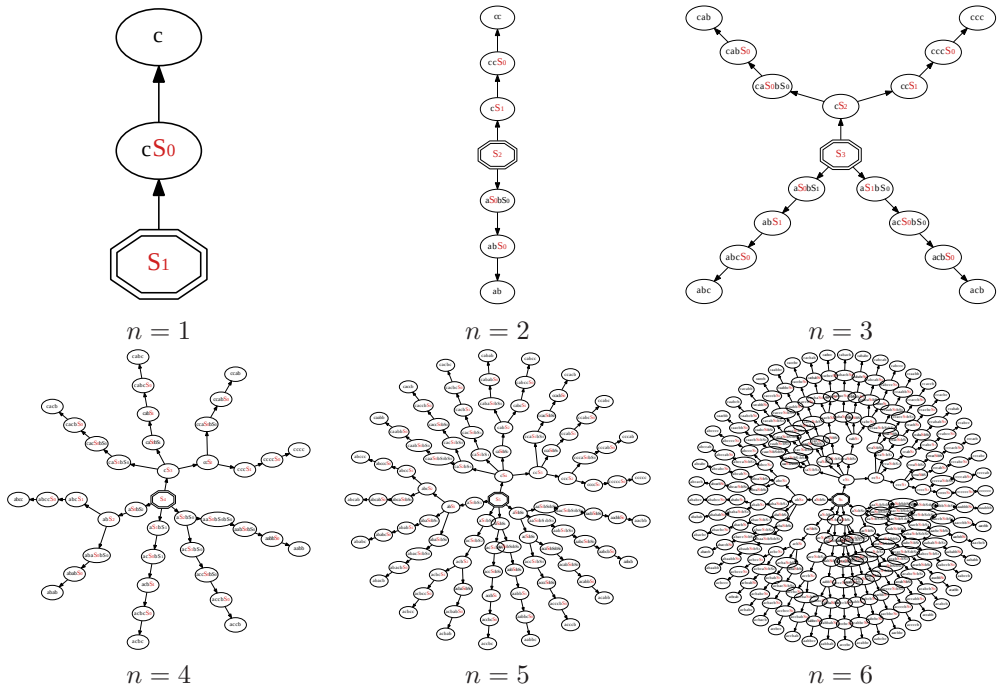


Figure 1: Trees of all walks associated with Motzkin words of size $n \in [1, 6]$ generated by the grammar $S \to a\ S\ b\ S \mid c\ S \mid \varepsilon$ under the *leftmost first* derivation policy ϕ_L.

will respectively be $(N'' \to_0 a, N'' \to_1 b)$ when $N' \to a$, and $(N'' \to_{1/2} a, N'' \to_{1/2} b)$ when $N' \to b$.

The idea behind our step-by-step algorithm is to capture this sequential dependency, by considering random generation scenarii as random (parse) walks. This perspective will allow to determine the total contribution of all forbidden (or already generated) words to each of the locally-accessible choices. This contributions will be used to modify the probabilities computed in the absence of forbidden words, leading to a uniform (resp. weighted) generation, while keeping the computational overhead to a reasonable level.

4.1. Immature words: A compact description for fixed-length sublanguages

Let us introduce the notion of **immature** words, defined as words on both the terminal and non-terminal alphabets, where **prescribed lengths** are additionally attached to any occurrence of a symbol. Formally, let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{P}, \mathcal{S})$ be a context-free grammar, then an immature word is any word

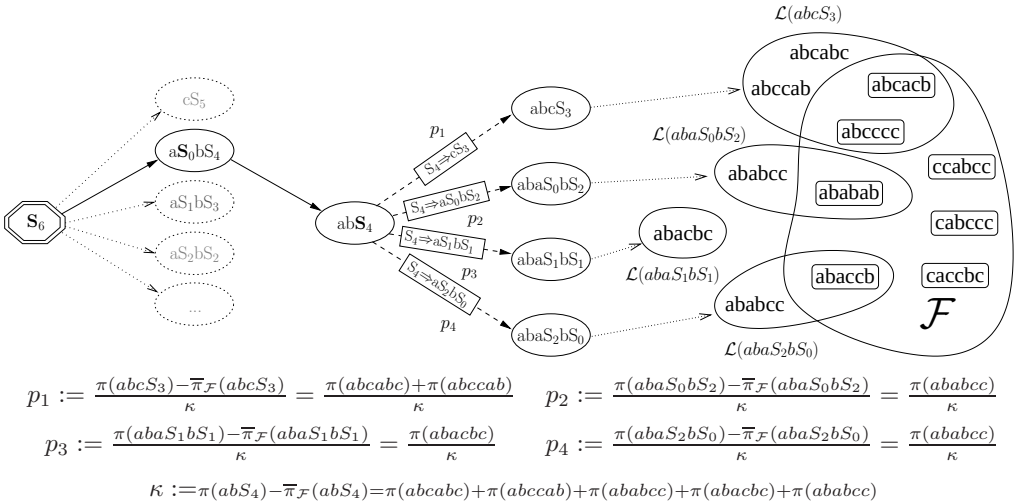
$$\omega \in \mathcal{L}^{\triangleleft}(\mathcal{G}) \subseteq \left((\Sigma \cup \mathcal{N}) \times \mathbb{N}^+\right)^*,$$

where $\mathcal{L}^{\triangleleft}(\mathcal{G})$ is the set of immature words generated from the axiom $\mathcal{S}$. Such words may contain non-terminal symbols, and potentially require some further derivations before becoming a word on the terminal alphabet, or mature word. Intuitively, immature words correspond to intermediate states in a random generation scenario.

The language associated with an immature word ω can be related to the languages of its symbols through

$$\mathcal{L}(\omega) = \prod_{\substack{i \in [1, |\omega|] \\ s_m = \omega_i}} \mathcal{L}(s)_m \tag{4.1}$$

where $\mathcal{L}(s)$ is defined as in Equation 2.1 when $s \in \mathcal{N}$, and naturally extended on terminal symbols $t \in \Sigma$ through $\mathcal{L}(t) = \{t\}$. In the following, we will use the notation $\pi(\omega)$ as



$$p_1 := \frac{\pi(abcS_3)-\overline{\pi}_{\mathcal{F}}(abcS_3)}{\kappa} = \frac{\pi(abcabc)+\pi(abccab)}{\kappa} \qquad p_2 := \frac{\pi(abaS_0bS_2)-\overline{\pi}_{\mathcal{F}}(abaS_0bS_2)}{\kappa} = \frac{\pi(ababcc)}{\kappa}$$

$$p_3 := \frac{\pi(abaS_1bS_1)-\overline{\pi}_{\mathcal{F}}(abaS_1bS_1)}{\kappa} = \frac{\pi(abacbc)}{\kappa} \qquad p_4 := \frac{\pi(abaS_2bS_0)-\overline{\pi}_{\mathcal{F}}(abaS_2bS_0)}{\kappa} = \frac{\pi(ababcc)}{\kappa}$$

$$\kappa := \pi(abS_4)-\overline{\pi}_{\mathcal{F}}(abS_4) = \pi(abcabc)+\pi(abccab)+\pi(ababcc)+\pi(abacbc)+\pi(ababcc)$$

Figure 2: Snapshot of a *step-by-step* random scenario for a Motzkin word of length 6, generated while avoiding $\mathcal{F}$. The step-by-step algorithm chooses one of the four possible derivations for abS_4 with probabilities proportional to the overall weights of accessible and admissible words.

a natural shorthand for $\pi(\mathcal{L}(\omega))$. Finally we will denote by $\overline{\pi}_{\mathcal{F}}(\omega) := \pi(\mathcal{L}(\omega) \cap \mathcal{F})$ the total weight of all forbidden words in $\mathcal{L}(\omega)$.

4.2. Random generation as a random walk in language space

An **atomic derivation**, starting from a word $\omega = \omega' \,.\, N \,.\, \omega'' \in \{\Sigma \cup \mathcal{N}\}^*$, is the application of a production $N \to X$ to ω, that replaces N by the right-hand side X of the production, yielding $\omega \Rightarrow \omega'.X.\omega''$. Let us call **derivation policy** a deterministic strategy that points, in an immature word, the first non-terminal to be rewritten through an atomic derivation. Formally a derivation policy is a function $\phi : \mathcal{L}(\mathcal{G}) \cup \mathcal{L}^{\triangleleft}(\mathcal{G}) \to \mathbb{N} \cup \{\emptyset\}$ such that

$$\begin{array}{rrcl} \phi: & \omega \in \mathcal{L}(\mathcal{G}) & \to & \varnothing \\ & \omega' \in \mathcal{L}^{\triangleleft}(\mathcal{G}) & \to & i \in [1, |\omega'|] \text{ such that } \omega_i \in \mathcal{N}. \end{array}$$

The **unambiguity** of a grammar requires that any given word be uniquely generated by a sequence of derivation. A sequence of atomic derivations is then said to be **consistent with a given derivation policy** if the non-terminal rewritten at each step is the one pointed by the policy. This notion provides a convenient framework for defining the **unambiguity** of a grammar without making reference to parse trees as is usually done.

Definition 4.1 (Unambiguity)**.** Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{P}, \mathcal{S})$ be a context-free grammar and ϕ a derivation policy acting on $\mathcal{G}$. The grammar $\mathcal{G}$ is said to be **unambiguous** if and only if, for each $\omega \in \Sigma^*$, there exists at most one sequence of atomic derivations that is consistent with ϕ and produces ω from $\mathcal{S}$.

Any derivation leading to a mature word $\omega \in \mathcal{L}(\mathcal{G})$ in a grammar $\mathcal{G}$ can then be associated in a one-to-one fashion with a walk in the space of sublanguages associated with immature words, or **parse walk**, taking steps consistent with a given derivation policy ϕ. More precisely, such a walk starts from the axiom $\mathcal{S}$ and, for any intermediate immature word $X \in \mathcal{L}^{\triangleleft}(\mathcal{G})$, the derivation policy ϕ points at a position $\phi(X)$, where a non-terminal X_k can be found. The parse walk can then be prolonged using one of the derivations acting on X_k (See Figures 1 and 2), until a mature word over Σ^* is reached.

Algorithm 3 Step-by-step random generation algorithm. $\mathcal{G}_\pi$ is a weighted grammar, ω an immature word, $\mu = \pi(\omega)$ is the **precomputed** weight generated from ω, and $\mathcal{F} \subset \mathcal{L}(\omega)$ is a set of forbidden words.

StepByStep$(\omega, \mu, \mathcal{G}_\pi, \mathcal{F}, \phi)$:
  if $\mu \leq \overline{\pi}_\mathcal{F}(\omega)$ **then**
    return `Error`
  else if $\phi(\omega) = \varnothing$ **then**
    return ω // ω is a mature word, generation is over
  end if
  $(\omega', N_m, \omega'') \leftarrow \left(\omega_{[1,\phi(\omega)-1]}, \omega_{\phi(\omega)}, \omega_{[\phi(\omega)+1,|\omega|]}\right)$
  $r \leftarrow$ **rand**$(\mu - \overline{\pi}_\mathcal{F}(\omega))$ // r is random, uniformly in $[0, \pi(\mathcal{L}(\omega)/\mathcal{F}))$
  if $N \to N' \mid N''$ **then** // **Union** type
    $\mu' \leftarrow \mu \cdot \pi(N'_m)/\pi(N_m)$
    $r \leftarrow r - (\mu' - \overline{\pi}_\mathcal{F}(\omega'.N'_m.\omega''))$
    if $r < 0$ **then**
      return **StepByStep**$(\omega'.N'_m.\omega'', \mu', \mathcal{G}_\pi, \mathcal{F})$
    else
      return **StepByStep**$(\omega'.N''_m.\omega'', \mu \cdot \pi(N''_m)/\pi(N_m), \mathcal{G}_\pi, \mathcal{F})$
    end if
  else if $N \to N' \,.\, N''$ **then** // **Product** type
    for all $i \in [1, n-1]$ **do** // Boustrophedon order $1, n-1, 2, n-2 \ldots$
      $\mu_i \leftarrow \mu \cdot \pi(N'_i) \cdot \pi(N''_i)/\pi(N_i)$
      $r \leftarrow r - (\mu_i - \overline{\pi}_\mathcal{F}(\omega'.N'_i.N''_{m-i}.\omega''))$
      if $r < 0$ **then**
        return **StepByStep**$(\omega'.N'_i.N''_{m-i}.\omega'', \mu_i, \mathcal{G}_\pi, \mathcal{F})$
      end if
    end for
  else if $N \to t$ **then** // **Terminal** type
    return **StepByStep**$(\omega'.t.\omega'', \mu, \mathcal{G}_\pi, \mathcal{F}, \phi)$
  end if

Where: **rand**(x): Draws a random number uniformly in $[0, x)$
$\overline{\pi}_\mathcal{F}(\omega) := \pi(\mathcal{L}(\omega) \cap \mathcal{F})$: Total weight of forbidden words in $\mathcal{L}(\omega)$

Let us observe that the parse walk has the same number of nodes as its corresponding parse tree, therefore in any CNF grammar the length of a parse walk is proportional to that of its final mature word, that is in $\Theta(n)$.

4.3. A step-by-step algorithm

Let us now propose Algorithm 3 based on the so-called recursive method introduced by Wilf [24], which uses the concepts of immature words to linearize the generation of words. More specifically, the algorithm will draw a random word by performing a sequence of local choices (atomic derivations) with probabilities proportional to the cumulated weight of accessible non-forbidden words, as illustrated by Figure 2. To gain access to these weights in reasonable time, cumulated weights of non-terminals will be precomputed, and a dedicated tree-like data structure is introduced.

Theorem 4.2. *Through an adaptation of the recursive approach, the* **StepByStep** *algorithm generates k distinct words of length n from a weighted grammar $\mathcal{G}_\pi$ in $\mathcal{O}(n \cdot |\mathcal{N}| + k \cdot n \log n)$ arithmetic operations, while using memory for $\mathcal{O}(n \cdot |\mathcal{N}| + k)$ numbers and a data structure over $\Theta(n \cdot k)$ nodes.*

Proof. As discussed in Section 4.5, Algorithm 3 generates a word in $\mathcal{O}(n\log(n))$ arithmetic operations, assuming that some values $\overline{\pi}_{\mathcal{F}}(\omega)$ are available during generation. In Section 4.5.2, a data structure is introduced that returns this value in $\mathcal{O}(\log(n))$ time. Namely, one has that $\mathcal{O}(n\log(n))$ times, a search in $\mathcal{O}(\log(n))$ is performed followed by an arithmetic operation involving large (at least polynomial on n, usually exponential) numbers. It follows that the cost of accessing the data structure is dominated by the cost of the following arithmetic operations, and the overall cost of generating k words is in $\mathcal{O}(k \cdot n\log(n))$. After each generation, the data structure can be updated in $\Theta(n)$ arithmetic operations, dominated by the cost of the generation itself.

The precomputation required by the **StepByStep** algorithm involves $\Theta(n \cdot |\mathcal{N}|)$ arithmetic operations, and the storage of $\Theta(n \cdot |\mathcal{N}|)$ numbers. The data structure for $\overline{\pi}_{\mathcal{F}}(\omega)$ has $\Theta(n \cdot k)$ nodes and contains $\Theta(k)$ different numbers, thus the overall complexity. □

4.4. Correctness

Proposition 4.3. *Under the hypothesis that $\mu = \pi(\omega)$, then Algorithm 3 draws a mature word at random according to the π-weighted distribution restricted to $\mathcal{L}(\omega)\backslash\mathcal{F}$ or returns* `error` *when $\mathcal{L}(\omega)\backslash\mathcal{F} = \varnothing$.*

Proof. Let us start with some observations to simplify the proof. Firstly since $\mu = \pi(\omega)$, then the variables μ' and μ_i of Algorithm 3 respectively obey

$$\mu' = \pi(\omega) \cdot \frac{\pi(N'_m)}{\pi(N_m)} = \frac{\pi(\omega').\pi(N_m).\pi(\omega'') \cdot \pi(N'_m)}{\pi(N_m)} = \pi(\omega'.N'_m.\omega'') \tag{4.2}$$

$$\mu_i = \pi(\omega) \cdot \frac{\pi(N'_i) \cdot \pi(N''_{m-i})}{\pi(N_m)} = \pi(\omega'.N'_i.N''_{m-i}.\omega''). \tag{4.3}$$

Secondly for any immature word ω, one has

$$\pi(\omega) - \overline{\pi}_{\mathcal{F}}(\omega) = \pi(\mathcal{L}(\omega)) - \pi(\mathcal{L}(\omega) \cap \mathcal{F}) = \pi(\mathcal{L}(\omega)\backslash\mathcal{F}).$$

We now show that, provided that $\mu = \pi(\omega)$ holds, then any emitted word is generated with respect to a weighted distribution on $\mathcal{L}(\omega)\backslash\mathcal{F}$. By induction on d the maximum number of recursive calls needed for the generation of a mature word from a given immature word ω, one has:

Base: The $d = 0$ case corresponds to an already mature word ω, for which the associated language is limited to $\{\omega\}$. In this case, ω has probability 1 in the weighted distribution, and is indeed always generated.

Inductive step: Assuming that the theorem holds for $d \geq n$, we investigate the probabilities of emission of words that require $d = n + 1$ derivations. Let N_m be the non-terminal pointed by ϕ, then:

- $N \to N' \mid N''$: Let us first assume that the derivation $N_m^* \Rightarrow N'_m$ is chosen with probability

$$\begin{aligned}\frac{\mu' - \overline{\pi}_{\mathcal{F}}(\omega'.N'_m.\omega'')}{\mu - \overline{\pi}_{\mathcal{F}}(\omega)} &= \frac{\pi(\omega'.N'_m.\omega'') - \overline{\pi}_{\mathcal{F}}(\omega'.N'_m.\omega'')}{\pi(\omega) - \overline{\pi}_{\mathcal{F}}(\omega)} \\ &= \frac{\pi(\mathcal{L}(\omega'.N'_m.\omega'')\backslash\mathcal{F})}{\pi(\mathcal{L}(\omega)\backslash\mathcal{F})}.\end{aligned}$$

  The recursive call to **StepByStep**$(\omega'.N'_m.\omega'', \mu', \mathcal{G}_\pi, \mathcal{F})$ indeed satisfy $\mu' = \pi(\omega'.N'_m.\omega'')$, and subsequently generates a mature word x using at most n recursive calls. The induction hypothesis holds, and the emission probability of $x \in \mathcal{L}(\omega'.N'_m.\omega'')\backslash\mathcal{F}$

is therefore given by $\pi(x)/\pi(\mathcal{L}(\omega'.N'_m.\omega'')\backslash\mathcal{F})$. The overall probability of issuing x starting from ω is then

$$\frac{\pi(\mathcal{L}(\omega'.N'_m.\omega'')\backslash\mathcal{F})}{\pi(\mathcal{L}(\omega)\backslash\mathcal{F})}.\frac{\pi(x)}{\pi(\mathcal{L}(\omega'.N'_m.\omega'')\backslash\mathcal{F})} = \frac{\pi(x)}{\pi(\mathcal{L}(\omega)\backslash\mathcal{F})}$$

in which one recognizes the weighted distribution on $\mathcal{L}(\omega'.N'_m.\omega'')\backslash\mathcal{F}$.
A similar proof can be given for the words issued from N''_m.

- $N \to N' \ . \ N''$: A repartition $N_m \Rightarrow N'_i \ . \ N''_{m-i}, i \in [1, m-1]$ is chosen with probability

$$\begin{aligned}\frac{\mu' - \overline{\pi}_{\mathcal{F}}(\omega'.N'_i.N''_{m-i}.\omega'')}{\mu - \overline{\pi}_{\mathcal{F}}(\omega)} =& \frac{\pi(\omega'.N'_i.N''_{m-i}.\omega'') - \overline{\pi}_{\mathcal{F}}(\omega'.N'_i.N''_{m-i}.\omega'')}{\pi(\omega) - \overline{\pi}_{\mathcal{F}}(\omega)} \\ =& \frac{\pi(\mathcal{L}(\omega'.N'_i.N''_{m-i}.\omega'')\backslash\mathcal{F})}{\pi(\mathcal{L}(\omega)\backslash\mathcal{F})}.\end{aligned}$$

  A recursive call is made, taking as arguments an immature word $\omega'.N'_i.N''_{m-i}.\omega''$ and a weight μ_i. As established in Equation 4.3, one has $\mu_i = \pi(\omega'.N'_i.N''_{m-i}.\omega'')$, and the induction hypothesis applies. It follows that any word $x \in \mathcal{L}(\omega'.N'_i.N''_{m-i}.\omega'')$ is generated by the recursive call with probability

$$\frac{\pi(x)}{\pi(\mathcal{L}(\omega'.N'_i.N''_{m-i}.\omega'')\backslash\mathcal{F})}.$$

  It follows that the total probability of generating $x \in \mathcal{L}(\omega'.N'_i.N''_{m-i}.\omega'')$ from ω is given by

$$\frac{\pi(\mathcal{L}(\omega'.N'_i.N''_{m-i}.\omega'')\backslash\mathcal{F})}{\pi(\mathcal{L}(\omega)\backslash\mathcal{F})}.\frac{\pi(x)}{\pi(\mathcal{L}(\omega'.N'_i.N''_{m-i}.\omega'')\backslash\mathcal{F})} = \frac{\pi(x)}{\pi(\mathcal{L}(\omega)\backslash\mathcal{F})}$$

- $N \to t$: The probability of any word x issued from ω is that of the word issued from $\omega'.t.\omega''$, that is $\frac{\pi(x)}{\pi(\mathcal{L}(\omega'.t.\omega'')\backslash\mathcal{F})} = \frac{\pi(x)}{\pi(\mathcal{L}(\omega)\backslash\mathcal{F})}$ by the induction hypothesis, which applies since $\pi(\omega'.t.\omega'') = \pi(\omega'.N.\omega'')$.

□

4.5. Complexities and data structures

The overall complexity of Algorithm 3 depends critically on efficient algorithms and data structures for:

1. Accessing the weights of languages associated with non-terminals.
2. Computing the total weight $\overline{\pi}_{\mathcal{F}}(\omega) := \pi(\mathcal{L}(\omega) \cap \mathcal{F})$ of all forbidden words accessible from an immature word ω.
3. Investigating of the *partitions* $N^*_m \Rightarrow N'_i \ . \ N''_{m-i}$ for *product rules*.
4. Manipulating large numbers.

4.5.1. Weights of non-terminals

As is usual within the recursive approach [7], the total weights $\pi(N_i)$ of languages generated from each non-terminal N must be readily available during the generation, for any length $i \in [0, n]$. A precomputation of these numbers can be performed in $\Theta(n)$ arithmetic operations, thanks to the holonomic nature of the generating functions at stake. Indeed, the coefficients of an holonomic generating function obey to a linear recurrence with polynomial coefficients, which can be algorithmically determined from the system of functional equations induced by a context-free grammar (Using the `Maple` package `GFun` [20], for instance). Such a technique could also be used to limit the memory requirement, by recomputing *on the fly* necessary values.

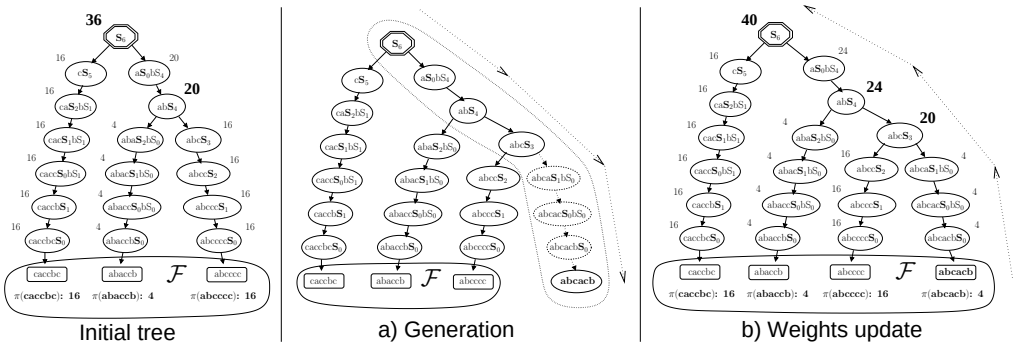


Figure 3: Illustration of the weighted tree of forbidden walks built for $\mathcal{F} = \{\mathbf{caccbc}, \mathbf{abaccb}, \mathbf{abcccc}\}$, using weights $\pi(a) = \pi(b) = 1$ and $\pi(c) = 2$. Generation (**a**) of a new word **abcacb** and update (**b**) of the contributions of $\mathcal{F}$ to the immature words involved in the generation.

4.5.2. A weighted tree for forbidden words

Proposition 4.4. *Through a dedicated data structure, any contribution $\overline{\pi}_{\mathcal{F}}(\omega)$ of forbidden words to an immature word can be accessed in $\mathcal{O}(\log(n))$ time within Algorithm 3 at the cost of an update operation in $O(n)$ arithmetic operations after each generation, while storing $\mathcal{O}(|\mathcal{F}|)$ additional numbers.*

Proof. Let us first assume that the parse walks of the elements of $\mathcal{F}$ are initially available as a set $\mathcal{T}$ of sequences of immature words. We introduce a data structure, the **weighted tree of forbidden walks**, a prefix tree whose nodes are in bijection with the immature words found in $\mathcal{T}$. Additionally, any node in correspondance with an immature word ω will hold the forbidden weight $\overline{\pi}_{\mathcal{F}}(\omega) := \pi(\mathcal{L}(\omega) \cap \mathcal{F})$ of all forbidden words reachable from ω.

The idea is to traverse the tree while Algorithm 3 is executed, fetching forbidden words contributions from local nodes. Namely, one simply adds an argument g to Algorithm 3 (omitted in the pseudocode for the sake of readability), which represents the node associated with ω if any, or $\varnothing$ otherwise. That way, one gets access in $O(1)$ operations to the forbidden weight $\overline{\pi}_{\mathcal{F}}(\omega)$ of ω, and in $O(\log(n))$ to that of its children nodes $\overline{\pi}_{\mathcal{F}}(\omega'.N'.\omega'')$, $\overline{\pi}_{\mathcal{F}}(\omega'.N''.N''_{m-i}.\omega'')$, or $\overline{\pi}_{\mathcal{F}}(\omega'.N'_i.N''_{m-i}.\omega'')$, e.g. using AVL trees [1] to store the children of a node. Once an atomic derivation $\omega \Rightarrow \omega'$ is chosen, the suitable child g' of g (or $\varnothing$, if no word from $\mathcal{F}$ can be computed from ω), is fed to the recursive call.

An **update of the tree**, posterior to a generation, can be performed in two steps, as illustrated by Figure 3:

- First, a *top-down* stage traverses the tree until a new immature word is found, and then adds nodes for each subsequent immature word traversed during the generation (**a**). Its requires to test the presence of a given child $\Theta(n)$ times, an operation that can be performed in $\Theta(n \log(n))$ time using AVL trees. The time complexity is therefore in $\Theta(n \log(n))$ instructions (All the numbers involved are bounded by a polynomial on k).

- Then, a *bottom-up* stage will propagate the weight of the sampled mature word to his ancestors up to the root(**b**). It will update the weights featured on branch points by adding the weight of the sampled word. Since $\Theta(n)$ nodes can be found from the leaf to the initial immature word $\mathcal{S}_n$, then the complexity of this stage is in $\Theta(n)$ arithmetic operations.

Many of the internal nodes in the trees have degree 1, which means that their value for $\overline{\pi}_{\mathcal{F}}(\omega)$ could just a copy of their single child's own. Since in any tree the number of

internal nodes of degree at least two is strictly smaller than the total number of leaves, then the number of *different* values (indicated by bold numbers in Figure 3) for $\overline{\pi}_{\mathcal{F}}(\omega)$ is bounded by $2 \cdot |\mathcal{F}|$. It follows that the memory used to store these – possibly large – numbers can be shared between consecutive unary nodes. Consequently the memory required to store the prefix tree consists in $\mathcal{O}(|\mathcal{F}|)$ numbers. □

4.5.3. Boustrophedon order for products

For product rules, one can use the so-called Boustrophedon [15] order during the investigation of the different decompositions $N_m \Rightarrow N'_i.N''_{m-i}$ of product rules. This yields a total number of investigation of the product loop (Algorithm 3, line 18) in $\mathcal{O}(n \log(n))$ in the worst case scenario, leading to a total complexity in $\mathcal{O}(n \log(n))$.

4.5.4. Arbitrary precision arithmetics

Although binary choices generators are now available even for transcendant probabilities [14], it is reasonable, for all practical purpose, to assume that the weights are going to be floating point numbers of bounded (yet arbitrarily large) precision. Since the language are context-free, the numbers involved in the precomputations of N_i and in the tree of forbidden words scale like $\mathcal{O}(\alpha^n)$ for some explicit α, since the resulting language is context-free. It follows that operations performed on such numbers will take time $\mathcal{O}(n \log(n) \log\log(n))$ [21], while the space occupied by their encoding will be in $\mathcal{O}(n)$.

5. Non-redundant unranking algorithm

As an alternative approach, let us propose a weighted *unranking algorithm*, which consists in two distinct parts:

- An unranking algorithm for generating words from a weighted context free grammar, presented in Section 5.1
- An algorithm that samples random numbers uniformly within a *gapped* union of intervals, to be used in the unranking algorithm to ensure non-redundant generation, presented in Section 5.2.

Our main result is summarized by the following theorem.

Theorem 5.1. *Through an unranking of carefully generated numbers, k distinct words of length n can be generated from a weighted grammar $\mathcal{G}_\pi$ in $\mathcal{O}(n \cdot |\mathcal{N}| + k \cdot n \log n)$ arithmetic operations, while storing $\mathcal{O}(n \cdot |\mathcal{N}| + k)$ numbers.*

Proof. In Section 5.1.4, we introduce a general weighted **Unranking** algorithm to transform in $\mathcal{O}(n \log(n))$ arithmetic operations any random number drawn uniformly in the interval $[0, \pi(\mathcal{L}(\mathcal{G}_\pi)_n)[$ into a random word in $\mathcal{L}(\mathcal{G}_\pi)_n$ while respecting the weighted distribution. Furthermore Section 5.2 introduces a data structure and an algorithm **ModRandom** for drawing numbers in of $[0, \pi(\mathcal{L}(\mathcal{G}_\pi)_n)[$ while avoiding intervals associated with forbidden words using $\mathcal{O}(k \log(k))$ arithmetic operations.

The precomputation required by the **Unranking** algorithm involves $\Theta(n \cdot |\mathcal{N}|)$ arithmetic operations, and the storage of $\Theta(n \cdot |\mathcal{N}|)$ numbers, while maintaining the data structure of **ModRandom** requires the storage of $\Theta(k)$ numbers. □

5.1. Weighted Unranking algorithm

Unranking algorithms, formalized by Wilf [24], usually take as input a **rank** in the interval $[0, |\mathcal{L}|)$, for $|\mathcal{L}|$ the number of words in a language, and output a word from the language that is uniquely associated to this rank according to some predefined ordering. It follows that calling an unranking procedure, starting from a uniformly-generated rank, immediately gives a uniformly generated object.

Generic unranking algorithms have been proposed for the uniform generation of words from a context-free language [18]. Through grammar transformations aiming at the introduction a controlled ambiguity, Weinberg and Nebel [22] extended their construct to special cases of non-uniform generation. Here we propose a generalization of the above algorithms, taking as input any weighted grammar and performing an unranking operation such that any word is obtained with probability proportional to its weight.

5.1.1. Statement of the problem

Let us assume an ordering on the words of the language $\mathcal{L}(S)$, Denote the ordered list of words generated from $\mathcal{L}(S)_n$ as $w_1, \ldots, w_{|\mathcal{L}(S)|}$. On can then split the interval of the real line $[0, \pi(\mathcal{L}(S))]$ into $|\mathcal{L}(S)|$ pieces, of width $\pi(w_1), \pi(w_2), \ldots, \pi(w_{|\mathcal{L}(S)|})$ respectively, each piece corresponding to a particular word. Let us denote by I_j the j-th interval, such that in general

$$I_j = \left[\sum_{k=1}^{j-1} \pi(w_j), \sum_{k=1}^{j} \pi(w_k)\right[.$$

The goal of our generalized unranking is to take as input a number $r \in [0, \pi(\mathcal{L}(S))]$, to figure out the interval $I_j = [L_j, R_j[$ such that $L_j \leq r < R_j$, and to return the corresponding word w_k. Upon starting the unranking procedure from a uniformly generated random real number in $\pi(\mathcal{L}(S))$, this word will be selected with probability proportional to the width of its interval, i.e. its weight. It follows that the whole procedure is a random generation algorithm for the weighted probability distribution presented in Equation 2.2.

5.1.2. Total ordering for the words of $\mathcal{L}(N)_n$

Let N be a non-terminal and n a length, let introduce a total order $\cdot \leq_r \cdot$ on $\mathcal{L}(N)_n$. For the sake of simplicity, let us write $\mathcal{A} \leq_r \mathcal{B}$ as a shorthand for $a \leq_r b, \forall a \in \mathcal{A}, b \in \mathcal{B}$. Let us then recursively define $\cdot \leq_r \cdot$ by $w \leq_r w, \forall w \in \mathcal{L}(N)_n$ and, :

- **Union type** $N \to N' \mid N''$: $\forall w' \in N'_n, w'' \in N''_n$, one has $N'_n \leq_r N''_n$.
  If $\forall w'_1, w'_2 \in N'_n$ and $w'_1 \leq_r w'_2$ within $\mathcal{L}(N'_n)$, then $w'_1 \leq_r w'_2$.
  If $\forall w''_1, w''_2 \in N''_n$ and $w''_1 \leq_r w''_2$ within $\mathcal{L}(N''_n)$, then $w''_1 \leq_r w''_2$.
- **Product type** $N \to N'.N''$, then $\forall i, j \in [1, n-1]$:
  - If $i < j$ then $w_i \leq_r w_j, \forall w_i, w_j \in \mathcal{L}(N'_i.N''_{n-i}) \times \mathcal{L}(N'_j.N''_{n-j})$
  - If $i = j$ then $\forall (w'_i, w''_i), (w'_j, w''_j) \in \mathcal{L}(N'_i) \times \mathcal{L}(N''_{n-i})$:
    If $w'_i \leq_r w'_j$ within $\mathcal{L}(N'_i)$ and $w'_i \neq w'_j$, then $w'_i.w''_i \leq_r w'_j.w''_j$.
    If $w'_i = w'_j$ and $w''_i \leq_r w''_j$ within $\mathcal{L}(N''_{n-i})$, then $w'_i.w''_i \leq_r w'_j.w''_j$.
- **Terminal type** $N \to t$: $\mathcal{L}(N_n) = \{t\}$ so this case is already captured by the first condition $w \leq_r w, \forall w \in \mathcal{L}(N)_n$.

It is easily verified that this order is total over $\mathcal{L}(N_n)$.

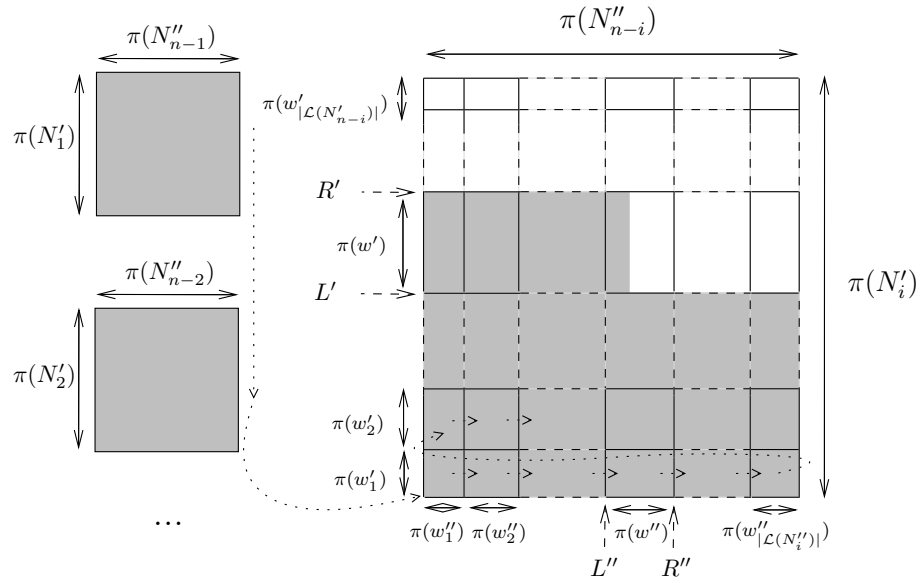


Figure 4: Matrix filling illustration of the $\cdot \leq_r \cdot$ order for product type non-terminals. Each word $w'.w'' \in \mathcal{L}(N'_i.N''_{n-i})$ is uniquely associated with a rectangle of total area $\pi(w') \cdot \pi(w'')$. The ranking of a word w can then be redefined as the volume of water (in grey) that has to be injected in the matrix such that the rectangle associated with w is partly filled, assuming the water flows in a left-to-right and bottom-to-top order. The unranking rephrases as the search of the rectangle which is partly filled by a given volume.

5.1.3. Ranking algorithm

Let $x \in \mathbb{R}^+$ be a positive real number, and $I = [L, R[\subset \mathbb{R}$ an interval, let us overload the sum operator through $I + x := [L + x, R + x[$ for the sake of simplicity. Then an algorithm **Rank** for computing the ranking interval of any word $w \in \mathcal{L}(N)_n$ can be outlined as:

- **Union type** $N \to N' \mid N''$: if $w \in N'_n$ then return $\mathbf{Rank}(w, N'_n)$. Otherwise $w \in N''_n$, and return $\pi(N'_n) + \mathbf{Rank}(w, N''_n)$.
- **Product type** $N \to N'.N''$: Since the grammar is unambiguous, then there only exists one decomposition $w = w'.w''$ such that $w' \in \mathcal{L}(N')$ and $w'' \in \mathcal{L}(N'')$. Let us define

$$[L', R'] := \mathbf{Rank}(w', N'_{|w'|}) \quad \text{and} \quad [L'', R''] := \mathbf{Rank}(w'', N''_{|w''|})$$

  It should be then clear from Figure 4 that the return value should be $[L, R[$ such that $L := \sum_{i=1}^{|w'|-1} \pi(N'_i.N'_{n-i}) + L' \cdot \pi(N''_{n-i}) + L'' \cdot \pi(w')$ and $R := L + \pi(w') \cdot \pi(w'')$.
- **Terminal type** $N \to t$: Return $[0, \pi(t)[$.

5.1.4. Unranking algorithm

Let us now propose Algorithm 4, which implements unranking for the relation $\cdot \leq_r \cdot$ and mostly consists in inverting the calculation presented in the Section 5.1.3.

Proposition 5.2. *Given an real number $r \in [0, \pi(\mathcal{L}(\mathcal{G})_n)[$ the* **Unrank** *algorithms retrieves the word associated with the interval I, such that $r \in I$ in $O(n \log(n))$ arithmetic operations after a precomputation in $\Theta(n)$ arithmetic operations involving storage of $\Theta(n)$ numbers.*

Algorithm 4 Unranking algorithm. Returns a word w and an interval $[I_L,\, I_R]$

Unrank(N_m, r):
  if $N \to N' \mid N''$ **then** // **Union** type
    if $r < \pi(N'_m)$ **then**
      return **Unrank**(N'_m, r)
    else
      $(w'', [I_L, I_R])$ =**Unrank**$(N''_m, r - \pi(N'_m))$
      return $(w'', [I_L + \pi(N'_m), I_R + \pi(N'_m)])$
    end if
  else if $N \to N'.N''$ **then** // **Product** type
    $L \leftarrow 0$
    for all $i \in [1, m-1]$ **do**
      if $\pi(N'_i) \cdot \pi(N''_{m-i}) \leq r$ **then**
        $r \leftarrow r - \pi(N'_i) \cdot \pi(N''_{m-i})$
        $L \leftarrow L + \pi(N'_i) \cdot \pi(N''_{m-i})$
      else // Found the right decomposition
        $(w', [L', R']) = \mathbf{Unrank}(N'_i, \frac{r}{\pi(N''_{m-i})})$
        $(w'', [L'', R']) = \mathbf{Unrank}\left(N''_{m-i}, \frac{r - L_{N'} \cdot \pi(N''_{m-i})}{\pi(w')}\right)$
        $I_L = L + L' \cdot \pi(N''_{n-i}) + L'' \cdot \pi(w')$
        $I_R = I_L + \pi(w') \cdot \pi(w'')$
        return $(w'.w'', [I_L, I_R])$
      end if
    end for
  else if $N \to t$ **then** // **Terminal** type
    return $(t, [0, \pi(t)])$
  end if

Sketch of proof. Firstly let us outline a proof of correctness by induction for the unranking procedure, starting from the initial case of terminal rules, where the algorithm returns the only word t, associated with an interval $[0, \pi(t)[$.

In the case of union rules, one may need to remove the added contribution $\pi(N'_m)$ when $r \geq \pi(N'_m)$ before proceeding to unrank within $\mathcal{L}(N''_m)$, or directly unrank within $\mathcal{L}(N'_m)$ otherwise.

For products rules, one first remarks that $\sum_{i=1}^{|w'|-1} \pi(N'_i.N'_{n-i})$ is exactly the quantity computed within L in section 5.1.3, so we are left to ensure that the remaining part of r indeed generates its corresponding word. Namely let us assume that $w = w'.w'' \in \mathcal{L}(N'_i.N''_{n-i})$, where w' and w'' are associated with intervals $[L', L' + \pi(w')[$ in $\mathcal{L}(N'_i)$ and $[L'', L'' + \pi(w'')[$ in $\mathcal{L}(N''_{n-i})$ respectively. Therefore the interval associated with w (after subtraction of L) is $I = [x, x + \pi(w') \cdot \pi(w'')[$ with $x := L' \cdot \pi(N''_{n-i}) + L'' \cdot \pi(w')$. Therefore computing, as done by the algorithm, the quantity $r' := r/\pi(N''_{m-i})$ for any $r \in I$ gives

$$L' + \frac{L''}{\pi(N''_{n-i})} \cdot \pi(w') \leq r' < L' + \frac{(L'' + \pi(w''))}{\pi(N''_{n-i})} \cdot \pi(w')$$

Since L'' is a partial sum of the weights in $\mathcal{L}(N''_{n-i})$, one has $0 \leq L'' \leq \pi(N''_{n-i}) - \pi(w_{|\mathcal{L}(N''_{n-i})|})$ and both bounds are actually reached (e.g. respectively by the first and last words in the order). It follows that

$$L' \leq r' < L' + \pi(w')$$

in which one recognizes the interval associated with w' within $\mathcal{L}(N'_i)$. The recursive

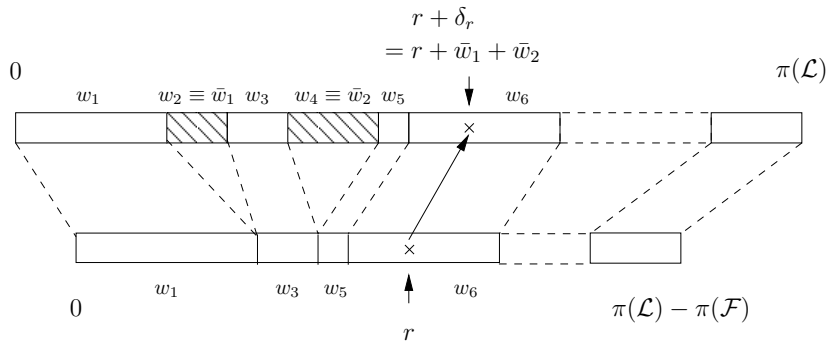


Figure 5: Illustration of the shift function δ. In order to avoid any forbidden words, one needs to *shift to the right* the random number r by the total weight of forbidden words (grayed area) *to the left* of its image.

unranking on $\mathcal{L}(N''_i)$ is given as argument $r'' := \frac{r - L' \cdot \pi(N''_{m-i})}{\pi(w')}$ which, for $r \in I$, gives

$$L'' \leq r'' < L'' + \pi(w'')$$

in which one recognizes the interval associated with w'' within $\mathcal{L}(N''_i)$.

We conclude on the correctness of the algorithm by reminding that the unambiguity of the grammar prevents multiple parsings (i.e. different intervals) to contribute to the generation of a given word.

Secondly, the complexity claimed in our proposition can be established by reminding the following facts:

- The numbers $\pi(N_m)$ involved in the unranking procedure can be precomputed thanks to the existence of linear recurrences for the coefficients of holonomic generating functions. We refer to Section 4.5.1 for a detailed discussion. Therefore they can be available in $\mathcal{O}(1)$ time after a precomputation of $\Theta(n)$ numbers involving $\Theta(n)$ arithmetic operations.

- The order of investigation of possible decompositions can be modified in Algorithm 4, line 10 to adopt a Boustrophedon order, decreasing the worst-case complexity of the algorithm from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log(n))$ in the worst-case. The total ordering on words can then be redefined to account for such a change, and the proof of correctness easily transposes.

□

5.2. Random generation of numbers in gapped intervals

In the previous section, a simple weighted unranking algorithm was proposed. Therefore by generating a random number r uniformly in $[0, \pi(\mathcal{L})]$, and using the **Unranking** algorithm, a word w can be generated with respect to the weighted distribution over a language $\mathcal{L}$. However when a forbidden set $\mathcal{F}$ is given, one additionally has to avoid the intervals associated with forbidden words. In other words, one can no longer draw a random number uniformly in $[0, \pi(\mathcal{L})[$, but rather in $I_{\bar{\mathcal{F}}} := \cup_{w \notin \mathcal{F}} I_w = [0, \pi(\mathcal{L})[\backslash (\cup_{\bar{w} \in \mathcal{F}} I_{\bar{w}})$.

Since the intervals I_w are mutually disjoint subsets of $[0, \pi(\mathcal{L})[$, a possible strategy consists in drawing a random number $r \in [0, \pi(\mathcal{L}) - \pi(\mathcal{F})[$, and increment r by some quantity that sends $r + \delta_r$ into $I_{\bar{\mathcal{F}}}$. Considering Figure 5, one observes that δ_r can be inductively defined as the total weight of all forbidden words smaller than the word found at $r + \delta_r$. In general, one could order the forbidden words in $\mathcal{F}$ and traverse $\mathcal{F}$ to compute δ_r, but this would induce spending an additional $\mathcal{O}(\mathcal{F})$ arithmetic operations

Algorithm 5 ModRandom: Takes a uniform random number and a node, and returns a uniform random number that avoids any interval associated with already generated words.

```
ModRandom(r, v)
    if v = ∅ then
      return r
    end if
    (T_L, T_R, w̄, [L_w̄, R_w̄[, μ_w̄) ← v
5:  if r < L_w̄ − μ_w̄ then
      ModRandom(r, T_L)
    else
      ModRandom(r + μ_w̄i + π(w̄_i), T_R)
    end if
```

per generation. For this reason we gather the intervals of forbidden words in a balanced binary tree structure that will grant access to δ_r in $\mathcal{O}(\log(\mathcal{F}))$ operations.

5.2.1. AVL tree for forbidden intervals

For each (word, interval) pair produced by the unrank algorithm, a corresponding node is inserted into an AVL tree (named after its original inventors). Let us remind that AVL trees [1] are self-balancing binary search trees, whose height can be maintained in $\Theta(\log(k))$ after k insertions through balancing operations. Since the intervals associated with the forbidden set are non overlapping, then they are ordered and can be compared and stored within an AVL. It follows that the insertion and lookup of k intervals can be performed in $\Theta(k \log(k))$ comparisons in the worst-case scenario.

Let us then define recursively our tree as either the empty tree, denoted by $\varnothing$, or a 5-tuple $v = (\mathcal{T}_L, \mathcal{T}_R, \bar{w}, I_{\bar{w}}, \mu_{\bar{w}})$ where:

- $\mathcal{T}_L$ and $\mathcal{T}_R$ are respectively the left and right children of the tree. Both can possibly be empty trees.
- $\bar{w}$ and $I_{\bar{w}} := [L_{\bar{w}}, R_{\bar{w}}[$ are a forbidden word and its corresponding interval.
- $\mu_{\bar{w}}$ is the total weight of forbidden intervals in the left subtree.

Let us remind that the nodes of an AVL are such that any node in a left subtree is less than or equal to its root, itself being less than or equal to any node of its right subtree. Also let us remark that, upon inserting in a tree $v_{\bar{w}}$ a new word $\bar{w}' \neq \bar{w}$ associated with an interval $I_{\bar{w}'} = [L_{\bar{w}'}, R_{\bar{w}'}[)$, the value $\mu_{\bar{w}}$, initialized at 0, can be easily updated into a new value $\mu'_{\bar{w}}$ such that

$$\mu'_{\bar{w}} = \begin{cases} \mu_{\bar{w}} + \pi(\bar{w}) & \text{If } \bar{w}' \leq_r \bar{w}, \text{ i.e. } \bar{w}' \text{ is inserted in the left subtree } \mathcal{T}_L \text{ of } v \\ \mu_{\bar{w}} & \text{Otherwise} \end{cases} \tag{5.1}$$

Assuming the tree is correctly built, Algorithm 5 will simply traverse the tree, and compute δ_r incrementally. For a given node $v = (\mathcal{T}_L, \mathcal{T}_R, \bar{w}, I_{\bar{w}}, \mu_{\bar{w}})$, the algorithm will determine if r corresponds to a word in the interval covered by $\mathcal{T}_L$, by comparing r to $L_{\bar{w}} - \mu_{\bar{w}}$ the total mass of allowed words in $\mathcal{T}_L$. If smaller, then r remains unmodified and the algorithm is run recursively on $\mathcal{T}_L$. If greater, then the final interval reached by r is greater than $I_{\bar{w}}$, and fits in the right subtree $\mathcal{T}_R$. The value r is then incremented by the total mass $\mu_{\bar{w}} + \pi(\bar{w})$ of forbidden words smaller than $\mathcal{T}_R$, and this value is used in a recursive call on $\mathcal{T}_R$. This process is terminated when the empty tree $\varnothing$ is reached, and the current value of r is returned. In other words, the returned value r is distant from its original value by the sum of weights $\mu_{\bar{w}}$ on the left subtrees whose intervals were dominated by r, in which one recognizes δ_r.

5.2.2. Correctness

Proposition 5.3. *The function **ModRandom** computed by Algorithm 5 is a bijection from* $[0, \pi(\mathcal{L}) - \pi(\mathcal{F}))[$ *onto* $[0, \pi(\mathcal{L})[\backslash (\cup_{\bar{w} \in \mathcal{F}} I_{\bar{w}})$ *with uniform density.*

Proof. The outline of the proof is as follows: First we will establish a technical invariant on the subset of values passed to the Algorithm **ModRandom**. Using this invariant, we will show that the final value returned by **ModRandom** will always fall out of any forbidden intervals, and that any interval gap between consecutive intervals can be reached. Let us start with some notations, followed by a technical lemma.

Let v_i be a the i-th node in the tree, let us denote by $[a, \ldots, i, \ldots, b]$ the indices of nodes accessible from v_i. Then let us denote by H_i the interval that is *dominated* by v_i, defined as

$$H_i = \left[R_{\bar{w}_{a-1}}, L_{\bar{w}_{b+1}} - \sum_{k=a}^{b} \pi(\bar{w}_k) \right[$$

where $R_{\bar{w}_i}, i \in [1, |\mathcal{F}|]$, the upper bound (resp. $L_{\bar{w}_i}, i \in [1, |\mathcal{F}|]$, the lower bound) of the forbidden interval of index i is extended by $R_{\bar{w}_0} = 0$ (resp. $L_{\bar{w}_{|\mathcal{F}|}} = \pi(\mathcal{L})$).

Lemma 5.4. *Let* $v_i = (\mathcal{T}_L, \mathcal{T}_R, \bar{w}, [L_{\bar{w}_i}, R_{\bar{w}_i}[, \mu_{\bar{w}})$ *be a node in the tree. Then the set of values* r *passed as argument to **ModRandom** jointly with* v_i *is exactly* H_i*.*

Proof. Let us prove this claim by induction on the depth D of recursive calls. Clearly in the initial call ($D = 0$), v_i is the root node and H_i is the whole interval $[0, \pi(\mathcal{L}) - \pi(\mathcal{F})[$ from which r is drawn uniformly, so our claim holds. Assume now that the set of values reached used for r are exactly $H_i := [R_{\bar{w}_{a-1}}, L_{\bar{w}_{b+1}} - \sum_{k=a}^{b} \pi(\bar{w}_k)[$ at a given depth $D = M$, then let us investigate the recursive calls. Two cases arise, depending on the value of r:

- When $r \in \mathcal{A} = [R_{\bar{w}_{a-1}}, L_{\bar{w}_i} - \mu_{\bar{w}_i}[$, then **ModRandom** is called on $v_j := \mathcal{T}_L$ with unmodified value $r' := r$. Thanks to the binary search tree structure, the indices of the forbidden nodes on the left subtree are $[a, \ldots, i-1]$, and $H_j = [R_{\bar{w}_{a-1}}, L_{\bar{w}_i} - \sum_{k=a}^{i-1} \pi(\bar{w}_k)[$. Since $\mu_{\bar{w}_i} = \sum_{k=a}^{i-1} \pi(\bar{w}_k)$ (def.), then $H_j = \mathcal{A}$, and any value $r' \in H_j$ can therefore be passed to the subsequent call.

- When $r \in \mathcal{B} = [L_{\bar{w}_i} - \mu_{\bar{w}_i}, L_{\bar{w}_{b+1}} - \sum_{k=a}^{b} \pi(\bar{w}_k)[$, then **ModRandom** is called on $v_j := \mathcal{T}_R$ with value $r' := r + \mu_{\bar{w}_i} + \pi(\bar{w}_i)$. The indices of the forbidden nodes on the right subtree are $[i+1, \ldots, b]$, so one has $H_j = [R_{\bar{w}_i}, L_{\bar{w}_{b+1}} - \sum_{k=i+1}^{b} \pi(\bar{w}_k)[$. The image $\mathcal{B}'$ of the interval $\mathcal{B}$ through a shift of value $\mu_{\bar{w}_i} + \pi(\bar{w}_i)$ is then

$$\begin{aligned} \mathcal{B}' &= \left[L_{\bar{w}_i} + \pi(\bar{w}_i), L_{\bar{w}_{b+1}} - \sum_{k=a}^{b} \pi(\bar{w}_k) + \sum_{k=a}^{i-1} \pi(\bar{w}_k) + \pi(\bar{w}_i) \right[\\ &= \left[R_{\bar{w}_i}, L_{\bar{w}_{b+1}} - \sum_{k=i+1}^{b} \pi(\bar{w}_k) \right[= H_j. \end{aligned}$$

  Finally, since any value is $r \in \mathcal{B}$, then any value $r' \in H_j$ can and will be passed to **ModRandom** for some value $r \in \mathcal{B}$.

Consequently at depth $D = M + 1$, the values r' provided to **ModRandom** over a subtree v_j are exactly H_j, and this property therefore holds for any $D \geq 0$. □

Let us show that forbidden intervals are indeed avoided. Let us consider a node $v_i = (\mathcal{T}_L, \mathcal{T}_R, \bar{w}, [L_{\bar{w}}, R_{\bar{w}}[, \mu_{\bar{w}})$, giving rise to a call **ModRandom**$(r', \varnothing)$, itself returning the final value. Since, for this node, Lemma 5.4 holds, then the value passed to this

call is any $r \in [R_{\bar{w}_{i-1}}, L_{\bar{w}_{i+1}} - \pi(\bar{w}_i)[$. Therefore either $r < L_{\bar{w}_i}$ and $r \in [R_{\bar{w}_{i-1}}, L_{\bar{w}_i}[$ is returned, or $r \geq L_{\bar{w}_i}$ and $r + \pi(\bar{w}_i) \in [R_{\bar{w}_i}, L_{\bar{w}_{i+1}}[$ is returned. It follows that any returned value r' falls between two consecutive forbidden intervals (resp. within the ending intervals $[0, L_{\bar{w}_1}[$ or $[R_{\bar{w}_{|\mathcal{F}|}}, \pi(\mathcal{L})[$), and therefore cannot fall in a forbidden interval.

Furthermore let us prove that any two calls $\mathbf{ModRandom}(r', \varnothing)$ and $\mathbf{ModRandom}(r'', \varnothing)$ from v_i and v_j respectively, $i \neq j$, give rise to distinct intervals. Remind that, as pointed out in the previous paragraph, the possibly generated intervals from a node v_i are $[R_{\bar{w}_{i-1}}, L_{\bar{w}_i}[$ if $\mathcal{T}_L = \varnothing$, and $[R_{\bar{w}_i}, L_{\bar{w}_{i+1}}[$ if $\mathcal{T}_R = \varnothing$. Therefore, by contradiction, any two calls giving rise to similar intervals would have to involve consecutive nodes v_i and v_{i+1} such that the right subtree v_i is $\mathcal{T}_{Ri} = \varnothing$ and the left subtree of v_{i+1} is $\mathcal{T}_{Li+1} = \varnothing$. Since such two nodes would represent consecutive values, then one would appears in a subtree of the other, otherwise the first common ancestor v_j of v_i and v_{i+1} would be such that $v_i < v_j < v_i + 1$ and the two nodes would not be consecutive. Since $v_i < v_{i+1}$, then either v_i would be found in the left subtree of v_{i+1} (and then $\mathcal{T}_{Li+1} \neq \varnothing$), or v_{i+1} would be found in the right subtree of v_i (and then $\mathcal{T}_{Ri} \neq \varnothing$). Both situations contradict the premisses, thus any interval $[R_{\bar{w}_{i-1}}, L_{\bar{w}_i}[, i \in [1, |\mathcal{F}| + 1]$ is generated by at most a call over an empty tree node $\varnothing$.

We conclude with the remark that there are exactly $|\mathcal{F}| + 1$ leaves in a binary tree with $|\mathcal{F}|$ inner nodes. Since there are also $|\mathcal{F}| + 1$ intervals $[R_{\bar{w}_{i-1}}, L_{\bar{w}_i}[, i \in [1, |\mathcal{F}| + 1]$ which are generated by at most one leave, then any such interval is generated, and **ModRandom** is therefore a bijection of $[0, \pi(\mathcal{L}) - \pi(\mathcal{F})[$ into $\cup_{i=1}^{|\mathcal{F}|+1}[R_{\bar{w}_{i-1}}, L_{\bar{w}_i}[= [0, \pi(\mathcal{L})[\backslash(\cup_{i=1}^{|\mathcal{F}|} I_{\bar{w}_i})$.

Finally, since the map **ModRandom** involves only shifts and no scaling, it follows that the map is measure preserving. Thus the algorithm alters uniformly generated random numbers over $[0, \pi(\mathcal{L}) - \pi(\mathcal{F})[$ into uniform random numbers over $[0, \pi(\mathcal{L})[\backslash(\cup_{i=1}^{|\mathcal{F}|} I_{\bar{w}_i})$. □

5.2.3. Complexity considerations

As can be seen in Equation 5.1, updating the values μ_v in a tree with m nodes can be done in $\mathcal{O}(\log(m))$ arithmetic operations upon insertion of a new node. However the AVL structure also requires a post-processing consisting in $\mathcal{O}(\log(m))$ *shifts* to keep the tree balanced. The shift operation involves taking two nodes $v_i < v_j$ that are connected in the tree and switching their ancestrality. Namely, if v_i is the first node of the left subtree of v_j, then v_j will become the first node of the right subtree of v_i (and vice-versa). The effect of this operation is local, therefore in any pair (v_i, v_j) of nodes inverted by a shift operation, the values μ_{v_i} and μ_{v_j} can be updated in $\mathcal{O}(1)$ arithmetic operations. It follows that the overall cost of k insertions remains in $\mathcal{O}(k \log(k))$ arithmetic operations.

Each internal node maintains a possibly large number μ, therefore $\Theta(|\mathcal{F}|)$ numbers need be stored in the tree. The ratio of probability between the most and least probable structure grows like $\Omega(\alpha^n)$, therefore at least $\Theta(n)$ bits needs be used for the numbers.

6. Conclusion and perspectives

We addressed the random generation of non-redundant sets of sequences from context-free languages, while avoiding a predefined set of words. We first investigated the efficiency of a rejection approach. Such an approach was found to be acceptable in the uniform case. By contrast, for weighted languages, we showed that for some languages the expected number of rejections would grow exponentially on the desired number of generated sequences for at least two parameters. Furthermore, we showed that in typical context-free languages and for fixed length, the probability distribution can be dominated by a small number of sequences. We proposed an alternative algorithm solution

for this problem, based on the so-called recursive approach. The correctness of the algorithm was demonstrated, and its efficient implementation discussed. This algorithm was showed to achieve the generation of a non-redundant set of k structures with a in $\mathcal{O}(k \cdot n \log(n))$ after a precomputation in $\Theta(n \cdot |\mathcal{N}|)$ arithmetic operations, while using $\mathcal{O}((n+k) \cdot n)$ memory for storing numbers. These complexities hold in the worst-case scenario, and remain mostly unaffected by the magnitude of weights being used.

Decomposable structures. One natural extension of the current work concerns the random generation of the more general class of decomposable structures [15]. Indeed, such aspects like the *pointing* and *unpointing* operator are not explicitly accounted for in the current work. Furthermore, the generation of labeled structures might be amenable to similar techniques in order to avoid a redundant generation. It is unclear however how to extend the notion of parse tree in this context. Isomorphism issues might arise, for instance while using the *unranking* operator.

Non-redundant Boltzmann sampling. Another direction for an efficient implementation of this approach would be to investigate an extension of Boltzmann samplers [11]. Although the prefix-tree introduced by the step-by-step algorithm could be trivially used to correct the probabilities used by Boltzmann sampling, it is unclear how such a correction may influence the probability of rejection and consequently the performances of the resulting algorithm.

Accommodating general sets of forbidden words. Both the step-by-step and unranking algorithms require the preliminary insertion of the forbidden set $\mathcal{F}$ into a dedicated data structure (prefix tree/AVL tree), both requiring the parse trees/walks of any word in $\mathcal{F}$ to be available. When such an information is not available, one could in principle parse the words in $\mathcal{F}$ to build the tree. In general this may require $\mathcal{F}$ run of a $n^{3-\varepsilon}$ parsing algorithm, leading to an impractical $\mathcal{O}(n^{3-\varepsilon} \cdot |\mathcal{F}|)$ complexity. In practice, it seems more fruitful to simply run the algorithm starting from an empty tree, and to test after each generation if the generated word is found in $\mathcal{F}$. If so, reject it after adding its parse walk, available to the algorithm without further computation since the word was just created, to the tree. Since this update is made at most once for each word in $\mathcal{F}$, then the worst-case complexity of generating k words remains bounded by $\mathcal{O}(|\mathcal{F}| \cdot n \log(n))$ arithmetic operations.

Unbiased bounded-arithmetic representations through confidence intervals Following the remark that the random generation from reasonable specifications is a *numerically stable problem* [8], we could envision using arbitrary precision arithmetics to achieve a $\mathcal{O}(k \cdot n^{1+\varepsilon})$ complexity. However, such a study would only benefit our step-by-step algorithm as the unranking algorithm requires generation, and therefore manipulation, of $\Theta(n)$ bits numbers.

References

[1] G. M. Adelson-Velskii and E .M. Landis, *An algorithm for the organization of information*, Proceedings of the USSR Academy of Sciences **146** (1962), 263–266.

[2] F. Bassino, J. David, and C. Nicaud, *On the average complexity of Moore's state minimization algorithm*, 26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009) (Dagstuhl, Germany), Leibniz International Proceedings in Informatics (LIPIcs), vol. 3, 2009, pp. 123–134.

[3] O. Bodini and Y. Ponty, *Multi-dimensional Boltzmann sampling of languages*, Proceedings of AOFA'10 (Vienna), Discrete Mathematics and Theoretical Computer Science Proceedings, no. 113, June 2010, pp. 49–64.

[4] S. Brlek, E. Pergola, and O. Roques, *Non uniform random generation of generalized Motzkin paths*, Acta Informatica **42** (2006), no. 8, 603–616.

[5] A. Denise, M.-C. Gaudel, S.-D. Gouraud, R. Lassaigne, and S. Peyronnet, *Uniform random sampling of traces in very large models*, First ACM International Workshop on Random Testing (ISSTA), 2006, pp. 10–19.

[6] A. Denise, Y. Ponty, and M. Termier, *Controlled non uniform random generation of decomposable structures*, Theoretical Computer Science **411** (2010), no. 40-42, 3527–3552.

[7] A. Denise, O. Roques, and M. Termier, *Random generation of words of context-free languagesaccording to the frequencies of letters*, Mathematics and Computer Science: Algorithms,Trees, Combinatorics and probabilities (D. Gardy and A. Mokkadem, eds.), Trends in Mathematics, Birkhaüser, 2000, pp. 113–125.

[8] A. Denise and P. Zimmermann, *Uniform random generation of decomposable structures using floating-point arithmetic*, Theor. Comput. Sci. **218** (1999), no. 2, 233–248.

[9] Y. Ding and E. Lawrence, *A statistical sampling algorithm for RNA secondary structure prediction*, Nucleic Acids Research **31** (2003), no. 24, 7280–7301.

[10] M. Drmota, *Systems of functional equations*, Random Struct. Alg. **10** (1997), 103–124.

[11] P. Duchon, P. Flajolet, G. Louchard, and G. Schaeffer, *Boltzmann samplers for the random generation of combinatorial structures*, Combinatorics, Probablity, and Computing **13** (2004), no. 4–5, 577–625, Special issue on Analysis of Algorithms.

[12] P. Flajolet, E. Fusy, and C. Pivoteau, *Boltzmann sampling of unlabelled structures*, Proceedings of ANALCO'07 (SIAM Press, ed.), January 2007.

[13] P. Flajolet, D. Gardy, and L. Thimonier, *Birthday paradox, coupon collectors, caching algorithms and self-organizing search*, Discrete Appl. Math. **39** (1992), no. 3, 207–229.

[14] P. Flajolet, M. Pelletier, and M. Soria, *On buffon machines and numbers*, SODA, 2011, pp. 172–183.

[15] P. Flajolet, P. Zimmermann, and B. Van Cutsem, *Calculus for the random generation of labelled combinatorial structures*, Theoretical Computer Science **132** (1994), 1–35.

[16] M. Goldwurm, *Random generation of words in an algebraic language in linear binary space*, Information Processing Letters **54** (1995), 229–233.

[17] S. P. Lalley, *Finite range random walk on free groups and homogeneous trees*, Ann. Probab. **21** (1993), 2087–2130.

[18] C. Martinez and X. Molinero, *A generic approach for the unranking of labeled combinatorial classes*, Random Structures & Algorithms, vol. 19, 2001, pp. 472–497.

[19] Y. Ponty, M. Termier, and A. Denise, *GenRGenS: Software for generating random genomic sequences and structures*, Bioinformatics **22** (2006), no. 12, 1534–1535.

[20] B. Salvy and P. Zimmerman, *Gfun: a maple package for the manipulation of generating and holonomic functions in one variable*, ACM Transactions on Mathematical Softwares **20** (1994), no. 2, 163–177.

[21] J. van der Hoeven, *Relax, but don't be too lazy*, Journal of Symbolic Computation **34** (2002), 479–542.

[22] F. Weinberg and M. E. Nebel, *Non uniform generation of combinatorial objects*, Tech. report, Technical Report University of Kaiserslauter, 2010.

[23] Y. Weinberg and M. E. Nebel, *Non Uniform Generation of Combinatorial Objects*, Tech. report, University of Kaiserslauter, May 2010.

[24] H. S. Wilf, *A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects*, Advances in Mathematics **24** (1977), 281–291.

[25] A. R. Woods, *Coloring rules for finite trees, and probabilities of monadic second order sentences*, Random Struct. Alg. **10** (1997), 453–485.

[26] P. Zimmermann, *Uniform random generation for the powerset construction*, Proceedings of the 7th conference on Formal Power Series and Algebraic Combinatorics, 1995, pp. 589–600.