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

Andy Lorenz[a], Yann Ponty[b,*]

[a]*Mathematics Departement*
Denison University
Granville, USA
[b]*CNRS/Inria AMIB*
Ecole Polytechnique
Palaiseau, France

Abstract

We address the non-redundant random generation of k words of length n in 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 allow for a non-redundant generation of k words of length n in $\mathcal{O}(k \cdot n \cdot \log n)$ arithmetic operations, after a 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.

Keywords: Context-free languages; Random generation; Weighted grammars; Non-redundant generation; Unranking; Recursive random generation

1. Introduction

The random generation of combinatorial objects has many direct applications in areas ranging from software testing [5] to bioinformatics [20]. 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 *recursive* framework introduced by Wilf [24], very elegant and general algorithms for the uniform random generation have been designed [16] and implemented. Many optimizations of this approach have been developed, using specificities of certain classes of combinatorial structures [17], or floating-point

*Corresponding author
Email address: `yann.ponty@lix.polytechnique.fr` (Yann Ponty)

arithmetics [8]. More recently, Boltzmann sampling [12], an algebraic approach based on analytic combinatorics, has drawn much attention, mostly owing to its minimal memory consumption and its intrinsic theoretical elegance.

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 folding of Ribo-Nucleic Acids (RNAs). 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 the entire set of conformations. 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.* Recursive random generation algorithms were derived, with time and space complexities equivalent to that observed within the uniform distribution [16]. 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 (i.e. most probable) objects. This redundancy, which can be useful in some context, such as the estimation 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, i.e. the generation of a set of **distinct** objects.

The non-redundant random generation has, to the best of our knowledge, only been addressed indirectly 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 generated set of structures was achieved respectively through *rejection* or an *unranking* algorithms. Unfortunately, these approaches do not transpose well to the case of weighted languages. Indeed, the former rejection algorithm may have exponential time-complexity in the average-case, as is shown later in the article. The unranking approach benefits from recent contributions by Martinez and Molinero [19], who gave generic unranking procedures for labeled combinatorial classes, generalized by Weinberg and Nebel [23] to rule-weighted context-free grammars. However, the latter algorithm is restricted to integral weights, and requires a transformation of the grammar which may impact its complexity. Furthermore, the question of figuring out a rank which avoids a set of words was completely ignored by these works.

In this paper, we address the non-redundant generation of words from a context-free language. We remind or 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 may lead to prohibitive average-case complexities in the case of degenerate grammars, large sets of forbidden words, large weights values, or large sets of generated words. 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 resulting algorithm is based on the recursive method, coupled with a custom data structure to perform a generation of 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, coupled with a dedicated data structure that stores and helps avoid any forbidden word, also yields 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 **Binary Chomsky Normal Form** (BCNF) iff each of its non-terminals $N \in \mathcal{N}$ is productive and can only be derived using a limited number of production rule (two for *union* type non-terminals, and one otherwise):

- Product type: $N \to N' \,.\, N''$ with $N', N'' \in \mathcal{N}$;
- Union type: $N \to N' \,|\, N''$ with $N', N'' \in \mathcal{N}$;
- Terminal type: $N \to t$ with $t \in \Sigma$;
- Epsilon type: $N \to \varepsilon$, iff N cannot be derived from self-referential non-terminals.

In the following, it will be assumed that the input grammar is given in BCNF. This restriction does not cause any loss of generality or performance, as it can be shown that any Chomsky Normal Form grammar can be transformed in linear time into an equivalent BCNF grammar, having equal number of rule up to a constant ratio.

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

$$\mathcal{L}(N) = \begin{cases} \mathcal{L}(N') \times \mathcal{L}(N'') & \text{If } N \to N' \,.\, N'' \\ \mathcal{L}(N') \cup \mathcal{L}(N'') & \text{If } N \to N' \,|\, N'' \\ \{t\} & \text{If } N \to t \\ \{\varepsilon\} & \text{If } N \to \varepsilon \end{cases} \tag{1}$$

The language $\mathcal{L}(\mathcal{G})$ generated by a grammar $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{P}, \mathcal{S})$ is then 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 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 $(\Sigma, \mathcal{N}, \mathcal{P}, \mathcal{S})$ define 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 symbols t.

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

$$\mathbb{P}(w \mid \pi, \mathcal{L}) = \frac{\pi(w)}{\displaystyle\sum_{w' \in \mathcal{L}} \pi(w')} = \frac{\pi(w)}{\pi(\mathcal{L})},\ \forall w \in \mathcal{L}. \tag{2}$$

The random generation of words of a given length n with respect to a weighted probability distribution has been addressed by previous works, and an algorithm in $\mathcal{O}(n \log n)$ after $\mathcal{O}(n^2)$ arithmetic operations was described [7] and implemented [20].

2.3. Problem statement

In the following, we consider algorithmic solutions for the non-redundant generation of a collection of words of a given length, generated by an unambiguous weighted context-free grammar. Our precise goal is to simulate efficiently a sequence of independent calls to a random generation algorithm until a set of exactly k distinct words in a language $\mathcal{L}$ are obtained. The returned subset $\mathcal{R} \subseteq \mathcal{L}_n$, $|\mathcal{R}| = k$, can be generated in any order, and the random generation scenarios leading to an ordering σ of $\mathcal{R}$ can be decomposed as:

$$\sigma_1 \to \sigma_1^* \to \sigma_2 \to (\sigma_1 \mid \sigma_2)^* \to \ldots \to \sigma_{k-1} \to (\sigma_1 \mid \cdots \mid \sigma_{k-1})^* \to \sigma_k.$$

The successive calls made to the weighted random generator are independent, therefore the total probability p_σ of getting a set $\mathcal{R}$ in a given order σ is given

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(G_π, k, n, F):
  Perform some precomputations...
  R ← ∅
  while |R| ≤ k do
    x ← DrawNonRed(S_n, π(N_n), G_π, F) {Any non-redundant algorithm}
    Update some data structure...
    (R, F) ← (R ∪ {x}, F ∪ {x})
  end while
  return R
```

by

$$p_\sigma = \frac{\pi(\sigma_1)}{\pi(\mathcal{L}_n)} \cdot \prod_{i=2}^{k} \left(\sum_{m \geq 0} \left(\frac{\sum_{j=1}^{i-1} \pi(\sigma_j)}{\pi(\mathcal{L}_n)} \right)^m \cdot \frac{\pi(\sigma_i)}{\pi(\mathcal{L}_n)} \right)$$

$$= \frac{\pi(\sigma_1)}{\pi(\mathcal{L}_n)} \cdot \prod_{i=2}^{k} \left(\frac{1}{1 - \frac{\sum_{j=1}^{i-1} \pi(\sigma_j)}{\pi(\mathcal{L}_n)}} \cdot \frac{\pi(\sigma_i)}{\pi(\mathcal{L}_n)} \right) = \prod_{i=1}^{k} \left(\frac{\pi(\sigma_i)}{\pi(\mathcal{L}_n) - \sum_{j=1}^{i-1} \pi(\sigma_j)} \right).$$

Summing over every possible permutation of the elements in $\mathcal{R}$, one obtains

$$\mathbb{P}(\mathcal{R} \mid k, n) = \sum_{\sigma \in \mathfrak{S}(\mathcal{R})} \prod_{i=1}^{k} \frac{\pi(\sigma_i)}{\pi(\mathcal{L}_n) - \sum_{j=1}^{i-1} \pi(\sigma_j)} \tag{3}$$

where $\mathfrak{S}(\mathcal{R})$ is the set of all permutations over the elements of $\mathcal{R}$. The problem can then be restated as:

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

Note that the distribution described by Equation (3) naturally arises from a sequence of dependent calls $(r_1, \ldots, r_k)$ to weighted generators for $\mathcal{L}$, avoiding sets of words $\varnothing$, $\{r_1\}$, $\ldots$, $\{r_1, \ldots, r_{k-1}\}$ respectively, as implemented in Algorithm 1. It is therefore sufficient to address the generation of a single word w, while avoiding a prescribed set $\mathcal{F}$, in the weighted probability distribution $\mathbb{P}(w \mid \pi, \mathcal{L} \backslash \mathcal{F})$.

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

3. Naive rejection algorithm

A **naive rejection strategy** for this problem consists in drawing words at random in an unconstrained way, rejecting those from the forbidden set until a valid word is generated, as implemented in Algorithm 2. As noted by Zimmermann [26], this approach is suitable for the uniform distribution of objects in general recursive specifications. This rejection strategy relies on an auxiliary generator **draw**$(\cdots)$ of words from a (weighted) context-free languages, and we refer to previous works by Flajolet *et al* [16, 12], or Denise *et al* [8] for efficient solutions for this problem.

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**, 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 us analyze the complexity of Algorithm 2, given $\mathcal{L}$ a context-free language, $n \in \mathbb{N}^+$ a positive integer and $\mathcal{F} \subset \mathcal{L}_n$ a set of forbidden words, assuming a uniform distribution on $\mathcal{L}_n$.

One first remarks that the worst-case time-complexity of the algorithm is unbounded, as nothing prevents the algorithm from repeatedly generating the same word. An average-case analysis, however, draws a more contrasted picture of the time complexity.

Theorem 3.2. *In the uniform distribution, the naive rejection implemented in Algorithm 2 leads to an average-case complexity in* $\mathcal{O}\left(\left(\frac{|\mathcal{L}_n|}{|\mathcal{L}_n|-|\mathcal{F}|}\right)\cdot k\log k\cdot \mathit{draw}(n)\right)$, *where* $\mathit{draw}(n)$ *is the complexity of drawing a single word.*

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 expected number $X_{n,k}$ of attempts for k distinct 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* [14]. It follows that $\mathbb{E}(X_{n,k})$ is trivially increasing with k, while remaining upper bounded by $k\cdot\mathcal{H}_k\in\Theta(k\log(k))$ 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 $\frac{|\mathcal{L}_n|}{|\mathcal{L}_n|-|\mathcal{F}|}$. □

It follows that, unless the forbidden set dominates the set of words, the *per-sample* complexity of the naive rejection strategy remains largely unaffected (at most a factor $\mathcal{O}(\log k)$, i.e. $\Omega(n)$ since $k\in\Omega(|\Sigma|^n)$) by the cumulated cost of rejections.

3.2. Complexity analysis: Weighted languages

Turning towards **weighted context-free languages**, one shows that a rejection strategy may have average-case complexity which is **exponential on** k, even in the most favorable case of an empty initial set of forbidden words.

Proposition 3.3. *The generation of* k *distinct words, starting from an empty initial forbidden set* $\mathcal{F}=\varnothing$, *may require a number of calls to* **draw** *that is 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 generating any element of $\mathcal{V}_{n,k}$ is greater than 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 a naive rejection approach is in $\Omega(n \cdot \alpha^k)$, i.e. exponential on k the number of words. □

However, the above example is based on a regular language, and may not be typical of the rejection algorithm's behavior on general context-free languages. Indeed, it can be shown that, under a natural assumption, no single word can asymptotically contribute a significant portion 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*[1]*. Assume that $\omega^\triangle_n$ the most probable (i.e. largest weight w.r.t. π) word of length n has weight $\pi(\omega^\triangle_n) \in \Theta(\alpha^n)$, for some $\alpha > 0$.*
Then the probability of $\omega^\triangle$ decreases exponentially as $n \to \infty$:

$$\exists\, \beta < 1 \text{ such that } \mathbb{P}(\omega^\triangle \mid \pi) = \frac{\pi(\omega^\triangle)}{\pi(\mathcal{L}(\mathcal{G}_\pi)_n)} \in \Omega(\beta^n).$$

Proof. The Drmota-Lalley-Woods theorem [10, 18, 25] establishes that the generating function of any simple type grammar has a *square-root type* singularity. This powerful result relies on properties of the underlying system of functional equations, and therefore also holds for the coefficients of weighted generating functions [6]. Therefore the overall weights $W_n := \pi(\mathcal{L}(\mathcal{G}_\pi)_n)$ – the coefficients of the weighted generating function – follow an expansion of the form $\frac{\kappa' \cdot \alpha'^n}{n\sqrt{n}}(1 + \mathcal{O}(1/n))$, $\alpha', \kappa' > 0$. Since $\omega^\triangle_n$ is contributing to W_n, then one has $\pi(\omega^\triangle_n) \leq \pi(\mathcal{L}(\mathcal{G}_\pi)_n)$ and therefore $\alpha < \alpha'$. The proposition follows directly from taking $\beta := \alpha'/\alpha$. □

Furthermore, one can easily design disconnected grammars such that, for any fixed length n, a subset of words $\mathcal{M} \subset \mathcal{L}(\mathcal{G}_\pi)_n$ having maximal number of occurrences of a given symbol t has total cumulated probability $1 - \alpha^n$, $0 < \alpha < 1$, in the weighted distribution. It follows that sampling more than

[1] 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 [13] 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.

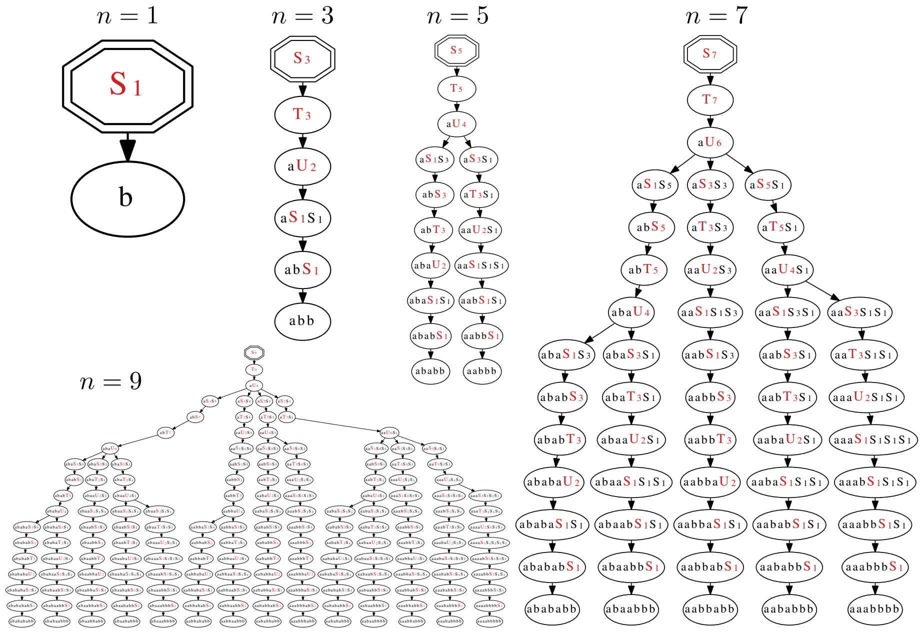


Figure 1: Trees of all walks associated with prefix notations of binary trees, having length $n \in [1, 9]$ and generated by the BCNF grammar $\{S \to T \,|\, b,\ T \to a\,.\,U,\ U \to S\,.\,S\}$, under the *leftmost first* derivation policy ϕ_L.

$|\mathcal{M}|$ words (e.g. a polynomial number of such words) can be extremely time-consuming (typically requiring exponential-time in n).

Finally, it is worth noticing that, in non-degenerate context-free languages, the weight of the least probable word ω_n^∇ grows like $\Theta(\alpha^n)$, $\alpha < 1$, where the exact value of α depends on a subtle trade-off between structural properties of the language and its weight function π. In particular, α can become arbitrarily close to 0, by adequately increasing the weight $\pi(t)$ of some terminal symbols. Sampling $k = |\mathcal{L}(\mathcal{G}_\pi)_n|$ words (Coupon Collector) then requires an expected $\Omega(\alpha^{-n})$ number of calls to **draw**, since the waiting time of the least probable word is clearly a lower-bound for the full collection. Since the number of words in a context-free language is bounded by $|\Sigma|^n$ and does not depend on the weight, then the *average cost per generation* may grow exponentially on n. This observation generalizes to many weighted languages, as shown by in Boisberranger *et al* [11].

4. A step-by-step recursive algorithm

A common approach to random generators for combinatorial objects [16, 7] consists of treating non-terminal symbols as **independent generators**. For instance, generating from an union-type non-terminal $N \to N'.N''$, involves two independent calls to dedicated generators for N' and N'', either directly

(Boltzmann sampling), or after figuring out suitable lengths for N' and N'' (Recursive method). Unfortunately, avoiding a predefined set of words breaks the independence assumption.

For instance, consider an unweighted grammar $\mathcal{G}$, having axiom N, and rules:

$$N \to N'.N'', \qquad N' \to a \mid b, \qquad \text{and} \qquad N'' \to a \mid b.$$

Remark that, starting from either N' or N'', both the recursive method and Boltzmann sampling would chose one of the rules with probability $1/2$. Assume now that some set $\mathcal{F} = \{aa\}$ has to be avoided, and that a sequential choice of derivations is adopted such that N' is fully derived before taking N'' into consideration. In this case, the derivation $N'' \to a$ must be forbidden iff $N' \to a$ was chosen. Moreover, the probabilities assigned to the derivations of N' must reflect the future unavailability of some choices for N''. One possibility is to use altered probabilities such that $\{N' \to_{1/3} a, N' \to_{2/3} b\}$, and introduce conditional probabilities such that $\{N'' \to_0 a, N'' \to_1 b\}$ 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 dependency sequentially, by considering random generation scenarios as random (parse) walks. This perspective allows to determine the total contribution of all forbidden (i.e. previously encountered) words for each of the locally-accessible alternatives. These contributions can then be used to modify conditionally the precomputed probabilities, leading to an uniform (resp. weighted) generation within $\mathcal{L}(\mathcal{G})_n/\mathcal{F}$, while keeping the computational cost to a reasonable level.

4.1. Immature words: A compact description of 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

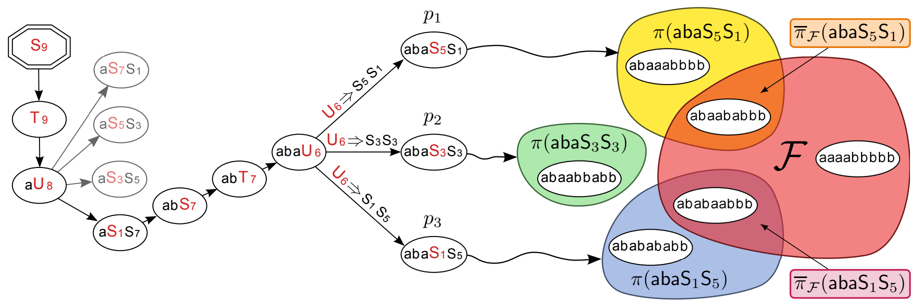
$$\omega \in \mathcal{L}^{\triangleleft}(\mathcal{G}) \subseteq \left((\Sigma \cup \mathcal{N}) \times \mathbb{N}^{+}\right)^{*},$$

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 is derived from the languages of its symbols through

$$\mathcal{L}(\omega) = \prod_{\substack{i \in [1, |\omega|] \\ s_m = \omega_i}} \mathcal{L}(s)_m \tag{4}$$

where $\mathcal{L}(s)$ is defined as in Equation 1 with $s \in \mathcal{N}$, and naturally extended on terminal symbols $t \in \Sigma$ through $\mathcal{L}(t) = \{t\}$. In the following, we use the notation $\pi(\omega)$ as a natural shorthand for $\pi(\mathcal{L}(\omega))$ and denote by $\overline{\pi}_{\mathcal{F}}(\omega) := \pi(\mathcal{L}(\omega) \cap \mathcal{F})$ the total weight of all forbidden words in $\mathcal{L}(\omega)$.



$$p_1 = \frac{\pi(\mathsf{abaS_5S_1}) - \overline{\pi}(\mathsf{abaS_5S_1})}{\pi(\mathsf{abaU_6}) - \overline{\pi}(\mathsf{abaU_6})} = \frac{\pi(\mathsf{abaaabbbb})}{\pi(\mathsf{abaU_6}) - \overline{\pi}(\mathsf{abaU_6})} \propto \pi(\mathcal{L}(\mathsf{abaS_5S_1}) - \mathcal{F})$$

$$p_2 = \frac{\pi(\mathsf{abaS_3S_3}) - \overline{\pi}(\mathsf{abaS_3S_3})}{\pi(\mathsf{abaU_6}) - \overline{\pi}(\mathsf{abaU_6})} = \frac{\pi(\mathsf{abaabbabb})}{\pi(\mathsf{abaU_6}) - \overline{\pi}(\mathsf{abaU_6})} \propto \pi(\mathcal{L}(\mathsf{abaS_3S_3}) - \mathcal{F})$$

$$p_3 = \frac{\pi(\mathsf{abaS_1S_5}) - \overline{\pi}(\mathsf{abaS_1S_5})}{\pi(\mathsf{abaU_6}) - \overline{\pi}(\mathsf{abaU_6})} = \frac{\pi(\mathsf{abababbbb})}{\pi(\mathsf{abaU_6}) - \overline{\pi}(\mathsf{abaU_6})} \propto \pi(\mathcal{L}(\mathsf{abaS_1S_5}) - \mathcal{F})$$

Figure 2: Snapshot of a *step-by-step* random scenario for the language consisting of prefix notations of binary trees of length 6, generated while avoiding $\mathcal{F}$. The step-by-step algorithm chooses one out of three possible derivations for $\mathsf{abaU_6}$ using probabilities proportional to the overall weights of accessible/admissible words.

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, to some non-terminal to be rewritten through an atomic derivation. Formally, a derivation policy is a function $\phi : \mathcal{L}(\mathcal{G}) \cup \mathcal{L}^{\triangleleft}(\mathcal{G}) \to \mathbb{N} \cup \{\varnothing\}$ such that

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

The **unambiguity** of a grammar requires that any generated word be generated by a unique 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 explicit reference to parse trees.

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 an unambiguous 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

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 of the language 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 $1, n-1, 2, n-2 \ldots$}
      $\mu_i \leftarrow \mu \cdot \pi(N'_i) \cdot \pi(N''_{m-i})/\pi(N_m)$
      $r \leftarrow r - (\mu_i - \overline{\pi}_{\mathcal{F}}(\omega'.N'_i.N''_{m-i}.\omega''))$
      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 $\mathcal{L}(\omega)$

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 extended using one of the derivations acting on X_k (See Figures 1 and 2), until a mature word in Σ^* is reached.

4.3. A step-by-step algorithm

Let us now describe and validate Algorithm 3, based on the recursive method introduced by Wilf [24], which uses the concepts of immature words to linearize the generation of words. More specifically, the algorithm draws a random word through a sequence of local choices (atomic derivations) using probabilities that are proportional to the cumulated weight of accessible non-forbidden words, as illustrated by Figure 2. To grant access to such weights in reasonable time, the cumulated weights of languages generated by non-terminals are precomputed recursively [6], and a dedicated *tree-like* data structure is introduced to gain efficient access to the contribution of forbidden words.

Theorem 4.2. *Algorithm 3 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 storing $\mathcal{O}(n \cdot |\mathcal{N}| + k)$ numbers, and a data structure consisting of $\Theta(n \cdot k)$ nodes.*

Proof. As discussed in Section 4.5, Algorithm 3 generates a word in $\mathcal{O}(n \log(n))$ arithmetic operations, assuming that some correcting terms $\overline{\pi}_{\mathcal{F}}(\omega)$ are available at runtime. 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 $\mathcal{O}(k \cdot n \log(n))$ arithmetic operations. After each generation, the data structure is updated in $\Theta(n)$ arithmetic operations, and the complexity is therefore dominated by the cost of the generation.

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. *Assuming that $\mu = \pi(\omega)$, Algorithm 3 draws a word at random according to the π-weighted distribution on $\mathcal{L}(\omega)\backslash\mathcal{F}$, or returns* `error` *iff $\mathcal{L}(\omega)\backslash\mathcal{F} = \varnothing$.*

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

$$\mu' = \pi(\omega) \cdot \frac{\pi(N'_m)}{\pi(N_m)} = \frac{\pi(\omega').\pi(N_m).\pi(\omega'') \cdot \pi(N'_m)}{\pi(N_m)} = \pi(\omega'.N'_m.\omega'') \qquad (5)$$

$$\mu_i = \pi(\omega) \cdot \frac{\pi(N'_i) \cdot \pi(N''_{m-i})}{\pi(N_m)} = \pi(\omega'.N'_i.N''_{m-i}.\omega''). \qquad (6)$$

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 word in $\mathcal{L}(\omega)$ is generated with respect to a weighted distribution on $\mathcal{L}(\omega)\backslash\mathcal{F}$. Let d be the maximum number of recursive calls needed for the generation of a mature word from a given immature word ω, then 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 $d \leq n$, 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

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

  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 $\mathcal{L}(\omega)\backslash\mathcal{F}$, and the argument applies symmetrically to 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

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

  A recursive call is then made on an immature word $\omega'.N'_i.N''_{m-i}.\omega''$, using weight μ_i. As established in Equation 6, one has $\mu_i = \pi(\omega'.N'_i.N''_{m-i}.\omega'')$, therefore the induction hypothesis applies, and any word $x \in \mathcal{L}(\omega'.N'_i.N''_{m-i}.\omega'')$ is generated by the recursive call with probability

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

  The emission probability of $x \in \mathcal{L}(\omega'.N'_i.N''_{m-i}.\omega'')$ from ω is then 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 emission probability for any word x emitted from ω equals that of the word issued from $\omega'.t.\omega''$. It is then given by $\frac{\pi(x)}{\pi(\mathcal{L}(\omega'.t.\omega'')\backslash\mathcal{F})} = \frac{\pi(x)}{\pi(\mathcal{L}(\omega)\backslash\mathcal{F})}$ according to 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 the *partitions* $N^*_m \Rightarrow N'_i \,.\, N''_{m-i}$ for *product rules*.
4. Handling 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 at generation time. A precomputation of these numbers can be performed in $\Theta(n)$ arithmetic operations, thanks to the algebraic, therefore holonomic, nature of the weighted counting generating functions. Indeed, the coefficients of an holonomic generating function obey a linear recurrence with polynomial coefficients in n. Such a recurrence can be algorithmically determined from the system of functional equations induced by the context-free grammar (e.g. using the `Maple` package `GFun` [21]).

4.5.2. A data structure for forbidden words

Proposition 4.4. *The total weight $\overline{\pi}_{\mathcal{F}}(\omega)$ of all forbidden words generated from an immature word ω can be accessed by Algorithm 3 in $\mathcal{O}(\log(n))$ time, at the cost of an update operation in $\Theta(n)$ arithmetic operations, while storing $\Theta(|\mathcal{F}|)$ additional numbers.*

Proof. Let us first remark that, in any BCNF grammar, any parse walk p_n that produces a mature word of length n, involves $\Theta(n)$ derivations (i.e. has length in $\Theta(n)$). To that purpose, let us discuss the number of occurrences of each type of rules in p_n, by reasoning on the associated parse tree. First, let us observe that each letter in the mature word can be bijectively associated with the application of a terminal rule, thus p_n contains exactly n applications of terminal rules. Then, product rules induce a binary structure in the parse tree, whose leaves correspond to the n terminal letters. Therefore, p_n contains exactly $n-1$ applications of product rules. Finally, sequences of union-type rules can be found before any occurrence of a product or terminal rule. However, it should be noted that the unambiguity of the grammar forbids derivations of the form $N \Rightarrow^* N$. The length of any union-type derivations sequence therefore cannot

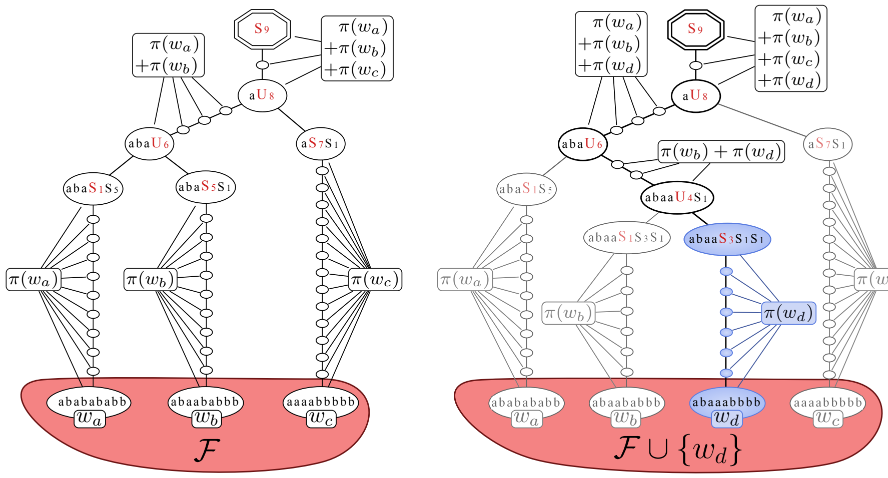


Figure 3: Illustration of the update operation for the weighted tree for forbidden walks, for our running example. Initial tree (Left): Each node is associated with an immature word ω and its overall weight of forbidden words $\overline{\pi}_{\mathcal{F}}(\omega)$ (some unary nodes are contracted for the sake of readability). During the execution of Algorithm 3, the tree is traversed to grant efficient access to $\overline{\pi}_{\mathcal{F}}(\omega)$. Updated tree (Right): After generating a new (mature) word $w_d :=$ abaaabbbb, the proper suffix of the parse walk is added to the tree (Blue nodes), associated with the additional weight πw_d, which must then be propagated back to the root (bold branch), using at most $\Theta(n)$ arithmetic operations.

exceed $|\mathcal{N}| + 1$. Treating $|\mathcal{N}|$ as a constant, the total number of occurrences of union rules is then in $\mathcal{O}(n)$, and we conclude that the total number of derivations involved in p_n is indeed $\Theta(n)$.

Assume now that the parse walks of the elements of $\mathcal{F}$ are available as a set $\mathcal{T}$ of sequences of immature words. We introduce a data structure, the **weighted tree of forbidden walks**, a decorated prefix-tree whose nodes are in bijection with the set of immature words in $\mathcal{T}$, and such that the overall weight $\overline{\pi}_{\mathcal{F}}(\omega) := \pi(\mathcal{L}(\omega) \cap \mathcal{F})$ is attached to each node labeled ω.

The idea is to descend into the tree during the execution of Algorithm 3, simply fetching the precomputed contributions $\overline{\pi}_{\mathcal{F}}(\omega)$ of forbidden words, that are attached to local nodes. Implementation-wise, an argument g is added to Algorithm 3 (omitted in the pseudocode for the sake of readability), holding the node associated with ω if any, or $\varnothing$ otherwise. One then 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 at random, 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.

A **tree update** operation must then be performed, as illustrated by Figure 3:

- First, a *top-down* stage descends into the tree, ensuring efficient access to $\overline{\pi}_{\mathcal{F}}(\omega)$, until a new mature word w_d is generated. Absent nodes are then added, corresponding to the proper suffix of the parse walk (Blue nodes). At each step, one needs to test the presence/absence of a given immature word within the children of the current node. Since the degree of a node is bounded by $\Theta(n)$, then this operation can be performed in $\Theta(\log(n))$ time, using a dedicated AVL tree to store the children of a node. The total time complexity of a single top-down descent is therefore in $\Theta(n \log(n))$ basic instructions.

- Then, a unique new weight $\overline{\pi}_{\mathcal{F}}(w_d)$ is created and attached to the nodes in the proper suffix of the parse walk (Blue nodes). A *bottom-up* stage propagates the weight of the generated (mature) word to his ancestors, all the way up to the root. The weights associated with branching nodes along the path are incremented by $\overline{\pi}_{\mathcal{F}}(w_d)$. 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 at most in $\Theta(n)$ arithmetic operations.

Note that the immature words used to label each node do not require an explicit encoding (which may otherwise result in a $\Theta(n^2)$ time complexity). Indeed, the immature words found on consecutive nodes may only differ on at most two positions, owing to the binary nature of products. Therefore, one may only store the difference between consecutive immature words, leading to a space complexity in $\Theta(|\mathcal{F}| \cdot n)$ bits. By the same token, the memory requirement can be limited to $2 \cdot |\mathcal{F}|$ large numbers, by observing that a unary node and its unique successor have same value for $\overline{\pi}_{\mathcal{F}}(\cdot)$, and that the memory representation this number can be shared. □

4.5.3. Boustrophedon order for product non-terminals

For product-type non-terminal rules, one may possibly have to investigate $\Theta(n)$ possible candidate partitions of the length, leading to a worst-case complexity in $\Theta(n^2)$ arithmetic operations. Therefore, we use a Boustrophedon order [16] $(1, n-1, 2, n-2, \ldots)$ to investigate possible decompositions $N_m \Rightarrow N'_i.N''_{m-i}$. As previously shown [16], this simple device reduces the total number of execution of the body of the innermost loop (Algorithm 3, line 18) to $\mathcal{O}(n \log(n))$ in the worst case scenario.

4.5.4. Arbitrary precision arithmetics

Although efficient algebraic generators exist even for some classes of transcendent probabilities [15], it is reasonable, for all practical purpose, to assume that weights are provided as floating point numbers of bounded (yet arbitrarily large) precision. Since the language is 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. It follows that operations performed on such numbers may take time $\mathcal{O}(n \log(n) \log\log(n))$ [22], while the space occupied by their encoding grows like $\mathcal{O}(n)$.

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. *Using an unranking approach, 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 $\mathcal{O}(n \cdot |\mathcal{N}| + k)$ large numbers.*

Proof. In Section 5.1.4, we introduce Algorithm 4, a general unranking procedure which transforms, 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 $\mathcal{L}(\mathcal{G}_\pi)_n$ with respect to a weighted distribution. Furthermore, Section 5.2 introduces a dedicated data structure, coupled with Algorithm 5 which draws numbers in the subset $[0, \pi(\mathcal{L}(\mathcal{G}_\pi)_n)[$ while avoiding contributions of forbidden words, and uses $\mathcal{O}(k \log(k))$ arithmetic operations.

The precomputation required by the Algorithm 4 involves $\Theta(n \cdot |\mathcal{N}|)$ arithmetic operations, and a storage of $\Theta(n \cdot |\mathcal{N}|)$ numbers. Maintaining the data structure used by Algorithm 5 requires the storage of $\Theta(k)$ numbers. $\square$

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 random object.

Generic unranking algorithms have been proposed for the uniform generation of words from a context-free language [19]. Through grammar transformations aiming at the introduction a controlled ambiguity, Weinberg and Nebel [23] extended their construct to special cases of non-uniform generation. For the sake of self-completeness, we reformulate, and mildly generalize, the above algorithms.

5.1.1. Statement of the problem

For a given length n, let us assume a total ordering on the words in $\mathcal{L}(S)_n$, and denote by $w_1, \ldots, w_{|\mathcal{L}(S)|}$ the ordered list of words in $\mathcal{L}(S)_n$. One can then split the interval $[0, \pi(\mathcal{L}(S))[$ into $|\mathcal{L}(S)|$ pieces of width $\pi(w_1), \pi(w_2), \ldots, \pi(w_{|\mathcal{L}(S)|})$

respectively, each piece being associated to a particular word. Denoting the j-th interval by I_j , one has

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

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 $[0, \pi(\mathcal{L}(S))[$, this word is to be selected with probability proportional to the width of its interval, i.e. its weight. It follows that the whole procedure constitutes a random generation algorithm for the weighted probability distribution presented in Equation 2.

5.1.2. Total ordering for words of length n

For each non-terminal $N \in \mathcal{N}$, let us introduce a dedicated order relation $\cdot \preccurlyeq_N \cdot$, defining an arbitrary notion of precedence on $\mathcal{L}(N)_{m\leq n}$ the set of words of length m generated from N. For the sake of simplicity, let us write $\mathcal{A} \preccurlyeq_N \mathcal{B}$ as a shorthand for $a \preccurlyeq_N b, \forall (a, b) \in \mathcal{A} \times \mathcal{B}$. The order relation $\cdot \preccurlyeq_N \cdot$ is defined by $w \preccurlyeq_N w, \forall w \in \mathcal{L}(N)_{m\leq n}$, and recursively defined by:

- **Union type** $N \to N' \mid N''$. Then, $\forall m \leq n$, one has:
  - $\mathcal{L}(N'_m) \preccurlyeq_N \mathcal{L}(N''_m)$;
  - $\forall w_1, w_2 \in \mathcal{L}(N'_m)$ (resp. $\mathcal{L}(N''_m)$), $w_1 \preccurlyeq_N w_2$ iff $w_1 \preccurlyeq_{N'} w_2$ (resp. $w_1 \preccurlyeq_{N''} w_2$).
- **Product type** $N \to N'.N''$. Then, $\forall m \leq n$, $\forall j, j' \in [1, m-1]$, one has:
  - If $j < j'$, then $\mathcal{L}(N'_j.N''_{m-j}) \preccurlyeq_N \mathcal{L}(N'_{j'}.N''_{m-j'})$;
  - If $j = j'$ then $\forall (u, v), (u', v') \in \mathcal{L}(N'_j) \times \mathcal{L}(N''_{m-j})$:
    - If $u \preccurlyeq_{N'} u'$, then $u.v \preccurlyeq_N u'.v'$;
    - If $u = u'$, then $u.v \preccurlyeq_N u'.v'$ iff $v \preccurlyeq_{N''} v'$.
- **Terminal type** $N \to t$: $\mathcal{L}(N_n) = \{t\}$, and one has $t \preccurlyeq_N t$.

Let us then denote by $\cdot \preccurlyeq_r \cdot := \cdot \preccurlyeq_{\mathcal{S}} \cdot$ the order induced on the language generated by the axiom $\mathcal{S}$ of the grammar. It is easily verified that $\cdot \preccurlyeq_r \cdot$ constitutes a total order over $\mathcal{L}(\mathcal{S}_n)$.

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:

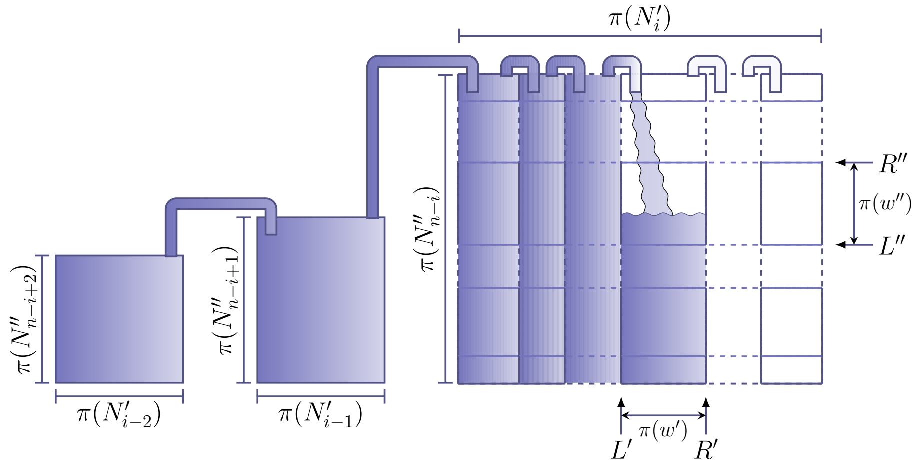


Figure 4: Water filling illustration of the ranking/unranking principle for the $\cdot \preccurlyeq_r \cdot$ order in product type non-terminals. Each word $w'.w'' \in \mathcal{L}(N'_i.N''_{n-i})$ is uniquely associated with a rectangular compartment of total area $\pi(w') \cdot \pi(w'')$. The ranking of a word $w = w'.w''$ can be adequately compared to the interval on the volume of water (in blue), which upon injection in the matrix, partly fills the compartment associated with w, assuming a water flow in a left-to-right/top-to-bottom lexicographic order. The unranking stage simply consists in searching for the compartment which is partly filled upon injection of a given volume r.

- **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 then define

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

  As illustrated by Figure 4, the returned interval must then be

$$[L, R[:= \left[\sum_{i=1}^{|w'|-1} \pi(N'_i.N'_{n-i}) + L' \cdot \pi(N''_{n-i}) + L'' \cdot \pi(w'), L + \pi(w') \cdot \pi(w'')\right].$$

- **Terminal type** $N \to t$: Return $[0, \pi(t)[$.

5.1.4. Unranking algorithm

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

Proposition 5.2. *Given a real number $r \in [0, \pi(\mathcal{L}(\mathcal{G})_n)[$ Algorithm 4 produces the word associated with an interval I, $r \in I$, in $O(n \log(n))$ arithmetic operations after a precomputation in $\Theta(|\mathcal{N}| \cdot n)$ arithmetic operations involving storage of $\Theta(|\mathcal{N}| \cdot n)$ 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}
        $(w', [L', R'[) =$ **Unrank**$(N'_i, \frac{r}{\pi(N''_{m-i})})$
        $(w'', [L'', R'[) =$ **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. First 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 either 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 one is 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 Algorithm 4, 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 tight (reached by the first and last words). 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 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.

The complexity of the algorithm is established by the following observations:

- 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, as discussed in Section 4.5.1. They can then be precomputed in $\Theta(n)$ arithmetic operations, requiring storage for $|\mathcal{N}| \cdot \Theta(n)$ large numbers.
- The order of investigation of possible decompositions can be modified in Algorithm 4, line 10 to adopt a Boustrophedon order as discussed in Section 4.5.3, 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 is easily adapted.

□

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 needs to avoid any interval associated with a forbidden word. 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 $\delta_{r,\mathcal{F}}$ that sends $r + \delta_{r,\mathcal{F}}$ into $I_{\bar{\mathcal{F}}}$. Considering Figure 5, one observes that $\delta_{r,\mathcal{F}}$ can be inductively defined as the total weight of all forbidden words smaller than the word found at $r + \delta_{r,\mathcal{F}}$. In general, one could order the forbidden words in $\mathcal{F}$ and traverse $\mathcal{F}$ to compute $\delta_{r,\mathcal{F}}$, but this would

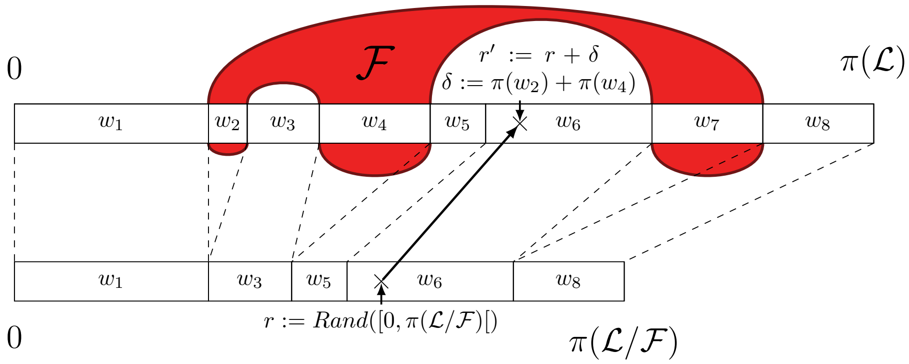


Figure 5: Illustration of the shift function δ. In order to avoid any forbidden words, one needs to *shift rightward* a random number r by the total weight of forbidden words (red area) that are found *leftward*.

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 $v = \varnothing$ **then**
    return r
  end if
  $(\mathcal{T}_L, \mathcal{T}_R, \bar{w}, [L_{\bar{w}}, R_{\bar{w}}[, \mu_{\bar{w}}) \leftarrow v$
  if $r < L_{\bar{w}} - \mu_{\bar{w}}$ **then**
    ModRandom$(r, \mathcal{T}_L)$
  else
    ModRandom$(r + \mu_{\bar{w}_i} + \pi(\bar{w}_i), \mathcal{T}_R)$
  end if

---

induce spending an additional $\mathcal{O}(|\mathcal{F}|)$ arithmetic operations per generation. For this reason, the intervals of forbidden words are gathered in a balanced binary tree structure that grants access to $\delta_{r,\mathcal{F}}$ 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 [1], i.e. a self-balancing binary search tree, whose height after k insertions can be limited to $\Theta(\log(k))$ through balancing operations. Since the intervals associated with the forbidden set are non overlapping, then they can be compared and therefore stored within an AVL tree. 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 tree 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

$$\mu'_{\bar{w}} = \begin{cases} \mu_{\bar{w}} + \pi(\bar{w}) & \text{If } \bar{w}' \preccurlyeq_r \bar{w}, \text{ i.e. } \bar{w}' \text{ is inserted in the left subtree } \mathcal{T}_L \text{ of } v \\ \mu_{\bar{w}} & \text{Otherwise} \end{cases} \tag{7}$$

Assuming the tree is correctly built, Algorithm 5 simply descends into the tree, and computes $\delta_{r,\mathcal{F}}$ incrementally. For a given node $v = (\mathcal{T}_L, \mathcal{T}_R, \bar{w}, I_{\bar{w}}, \mu_{\bar{w}})$, the algorithm determines 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 within 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 are dominated by r, in which one recognizes the definition of $\delta_{r,\mathcal{F}}$.

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 establish a technical invariant on the subset of values passed to Algorithm 5. Using this invariant, we show that the final value returned by **ModRandom** avoids every forbidden interval, and that any interval can be reached. Let us start with some notations, followed by a technical lemma.

Let v_i be the i-th node in the tree and 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 possible values for r is 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 r can be any value in $\mathcal{B}$, then any value $r' \in H_j$ can 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 **ModRandom**$(r', \varnothing)$ and **ModRandom**$(r'', \varnothing)$ from v_i and v_j respectively, $i \neq j$, give rise to distinct intervals. Recall 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 a single 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 7, 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 of $\mathcal{O}(\log(m))$ *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 was the first node of the left subtree of v_j, then v_j becomes 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, and 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 a first algorithm for this problem, based on the recursive approach. The correctness of the algorithm was demonstrated, and its efficient implementation discussed. This algorithm was showed to perform a non-redundant generation of k distinct structures in $\mathcal{O}(k \cdot n \log(n))$, after a precomputation in $\Theta(n \cdot |\mathcal{N}|)$ arithmetic operations, and requires storage of $\mathcal{O}(n \cdot |\mathcal{N}| + k)$ large numbers, and a data structure consisting of $\Theta(n \cdot k)$ nodes. We explored a second approach, based on a ranking/unranking approach for the same task, and obtained an algorithm in $\mathcal{O}(n \cdot |\mathcal{N}| + k \cdot n \log n)$ complexity, with the slightly decreased memory consumption of $\mathcal{O}(n \cdot |\mathcal{N}| + k)$ large numbers. These complexities hold in the worst-case scenario, and remain mostly unaffected by the magnitude of weights being used.

6.1. Different impact of fixed-precision arithmetics implementations

When using arbitrary (or sufficient) precision arithmetics, the complexity and storage of the two algorithms are the same. However, practical implementations may involve using fixed-precision arithmetic, in which case significant differences between the two methods arise. The complexity of both algorithms can be improved significantly if one uses fixed-precision arithmetic. However, in both cases, the algorithms suffer from a quantifiable loss of precision.

If the ratio between the weight of the smallest word and the weight of the language is small, then built-in floating point operations may be used, giving some advantages to the unranking approach with respect to its memory consumption. Indeed, the cost of storing the data structure will then dominate the memory consumption of the recursive version ($\mathcal{O}(n \cdot |\mathcal{N}| + n \cdot k)$), while the memory complexity of the unranking algorithm gently decreases to ($\mathcal{O}(n \cdot |\mathcal{N}| + k)$).

However, we believe the recursive method to be more stable numerically than the unranking approach. Indeed, the weights accessible on the alternative choice in the usual generation are typically comparable. Therefore, it will typically take an large number of generations for the recursive algorithm to fully deplete one of the alternatives. By contrast, the unranking algorithm may very quickly isolate a poorly contributing set of words after very few generation. For instance, if the second word in the ordering is generated first, then the data structure may practically forbid the first element, choosing it with 0 probability because of the rounding error. This point therefore seems favorable to the recursive algorithm.

6.2. Perspectives

Let us briefly outline a few perspectives to the current work:

- **Decomposable structures:** One natural extension of the current work concerns the random generation of the more general class of decomposable structures [16]. 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 one may extend the notion of parse tree in this context. Intrinsic ambiguity issues might arise, for instance while using the *unranking* operator.

- **Non-redundant Boltzmann sampling.** Another direction for an efficient implementation of the non-redundant generation may rely on an extension of Boltzmann samplers [12]. Indeed, the prefix-tree introduced by the step-by-step algorithm could, in principle, be used *as is* to correct the probabilities used by Boltzmann sampling. However, it is unclear how such a correction may impact the probability of rejection, and consequently degrade 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.

Acknowledgements

The author would like to thank A. Denise, C. Herrbach, and an anonymous reviewer for thought-provoking remarks and helpful suggestions. This work was supported by the French *Agence Nationale de la Recherche* as part of the MAGNUM project (`ANR 2010 BLAN 0204`).

[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] Jérémie du Boisberranger, Danièle Gardy, and Yann Ponty, *The weighted words collector*, 23rd Intern. Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms (AofA'12), DMTCS Proceedings, vol. AQ, 2012, pp. 243–264.

[12] 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.

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

[14] 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.

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

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

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

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

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

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

[21] 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.

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

[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.

Appendix A. Expressivity of the binary Chomsky normal form

Let us show that the assumption of a BCNF can be made without loss of generality (or performance). Indeed, it is a classic result that any context-free grammar $\mathcal{G}$ can be transformed into a Chomsky Normal Form (CNF) grammar that generates the same language.

Appendix A.1. From CNF to BCNF grammars: An algorithm

From such a grammar, an equivalent grammar in BCNF can be simply and efficiently obtained through the following transformation:

i) For each terminal t (resp. empty word ε) create a new non-terminal N_t (resp. N_ε) whose sole production is $N_t \to t$ (resp. $N_t \to \varepsilon$);
ii) Replace any occurrence of t (resp. ε) within a production rule with its dedicated non-terminal N_t (resp. N_ε);
iii) Replace any rule $N \to N'.N''$, where N has more than one derivation, with rules $N \to N^\bullet$ and $N^\bullet \to N'.N''$, where $N^\bullet$ is a newly created non-terminal;
iv) For any non-terminal N having multiple production rules ($N \to X_1 \mid \cdots \mid X_k$, $k > 1$), create $k-2$ dedicated non-terminals $\{N_i\}_{i=1}^{k-2}$, and replace the rules of N with a tree-like equivalent hierarchy of binary rules. For instance, one may create chained rules, such that $N \to X_1 \mid N_1$, $\{N_i \to X_{i+1} \mid N_{i+1}\}_{i=1}^{k-3}$, and $N_{k-2} \to X_{k-1} \mid X_k$;
v) Finally, remove every non-terminal whose sole production is $N \to N'$, replacing any occurrence of N by N' in any derivation rule.

Appendix A.2. Correctness

The equivalence of the resulting grammar to the input one in CNF trivially follows from the language-preserving nature of the substitutions performed at each step. Furthermore, it is easily verified that the resulting grammar is in BCNF. Indeed, consider the set $\mathcal{P}_N$ of derivation rules available for any former non-terminal N, along the transformation:

- Before executing the transformation: $\mathcal{P}_N$ consists an arbitrary number of terminal rules ($N \to N'$), binary-product ($N \to N' \,.\, N''$) rules, or possibly an epsilon rule for the axiom ($\mathcal{S} \to \varepsilon$);

- Steps i) and ii) remove terminal symbols: After their execution, $\mathcal{P}_N$ contains an arbitrary number of unary ($N \to N'$) or binary ($N \to N' \,.\, N''$) rules;

- Step iii) removes non-unary multiple rules: $\mathcal{P}_N = \{(N \to N' \,.\, N'')\}$, or $\mathcal{P}_N = \{N \to N' \mid N' \in \mathcal{N}' \subseteq \mathcal{N}\}$;

- Step iv) binarizes multiple rules: $\mathcal{P}_N = \{(N \to N')\}$, $\mathcal{P}_N = \{(N \to N' \,.\, N'')\}$, or $\mathcal{P}_N = \{(N \to N'), (N \to N'')\}$, where $N', N'' \in \mathcal{N}$;

- Finally, step v) removes extraneous unary non-terminals: $\mathcal{P}_N = \{(N \to N' \,.\, N'')\}$, or $\mathcal{P}_N = \{(N \to N'), (N \to N'')\}$, $N', N'' \in \mathcal{N}$.

The derivation rules available for the set of non-terminals, created during the transformation, are initially in BCNF. Note that the only modification performed on productions of new non-terminals substitute a non-terminal for another, thereby keeping the rules BCNF-compliant. Finally, the constraint on the initial CNF guarantees that the only epsilon rule is derived from the axiom, either in a single production or through a sequence of non-referential productions created at step iv). Therefore one concludes that the produced grammar is indeed in BCNF.

The proposed transformation from a CNF to an equivalent BCNF can be implemented in linear time, through a careful ordering of the removals performed by step v), and the number of rules is at most increased by a constant factor.