Chapter 3

Random Generation

3.1 Context and motivations

In the coming chapter, we consider methods for the random generation of objects from a given combinatorial class $\mathcal{C} = \cup_{n\geq 0}\mathcal{C}_n$ indexed by a *size parameter* n. In general, if not explicitly mentioned, we aim at uniform generation at a fixed size n. In other words, for a given size n, the algorithm has to return an object in $\mathcal{C}_n$ uniformly at random (u.a.r), that is, under the distribution:

$$\mathbb{P}(\gamma) = \frac{1}{\#\mathcal{C}_n} \quad \text{for each } \gamma \in \mathcal{C}_n.$$

More generally, if $\mathcal{C}$ is endowed with a probability law, we can also be interested in sampling according to this probability.

3.1.1 Motivations

There are many motivations to design such algorithms. Here are a few of them :

- To test the robustness of a programm: a random generator can produce a set of instances that can be used to test that the programm behaves in the expected way.
- To check the validity of a model. For instance, one of the big challenge of today's research is to get a better understanding of the graph world wide web. As of December, 31st 2013, 48.10^9 webpages were indexed by Google and the total number of webpages is believed to be of order 14.10^{12}.
  It is beyond hope to get an exact picture of the graph, but to check whether a random model of graphs makes a suitable approximation, one can study the probability law of the degree of a node (which is believed to follow a power-law, i.e $\mathbb{P}(\deg = n) \approx n^{-\alpha}$), or to study the resistance to attacks by suppressing a random number of servers.
- To get statistical datas on abstract structures.
- In a research perspective: to establish or check conjectures on the typical behavior of large random objects.

3.1.2 Context and Hypotheses

We assume all along the course that we have at our disposal a *perfect generator of bits* $b_1, b_2, b_3, \ldots$ that are independent and unbiased: $\mathbb{P}(b = 0) = \mathbb{P}(b = 1) = 1/2$. (In practice, there are efficient deterministic procedures that return a binary string having statistical

properties close to perfect randomness, see the detailed study in [Knuth, *The art of computer programming, vol.2*]).

We can interpret the sequence of random bits as a real value x taken uniformly at random in $[0,1]$, which we denote by $x \leftarrow \text{rnd}(0,1)$. From this, it is easy to draw a random integer in an interval $\{1,\dots,n\}$:

$$\text{rnd}[1..n] : \textbf{return} \lfloor \text{rnd}(0,1) * n \rfloor + 1).$$

We can also draw a *Bernoulli law* of a parameter $p \in (0,1)$, which returns "true" (or 1) with probability p and "false" (or 0) with probability $1-p$:

$$\text{Bern}(p) : \textbf{return } \text{rnd}(0,1) \leq p.$$

A note on complexity: if not explicitly mentioned, we adopt an arithmetic complexity model (in contrast to a bit complexity), assuming cost $\mathcal{O}(1)$ for operations such as rnd[1..n]), Bern(p), access to a pointed element, ... Note that with a bit complexity model, operations such as rnd[1..n]) would require order of $\log_2(n)$ bits; indeed $\log_2(n)$ bits are already necessary to write down the development of n in base 2.

In the next sections, we are going to investigate different methods to get efficient (complexity linear or in the worst case quadratic in the size of the object) random samplers. We start with simple ad-hoc method that rely on bijections, surjections and rejections before moving to more systematic ways of designing random samplers such as the recursive method and Boltzmann samplers.

3.2 Elementary methods

3.2.1 Permutations

Algorithm 1: RandPerm(n)

Input: n an integer
Output: a random permutation of $\{1,\dots,n\}$

let tab $= [1,..,n]$;
for $i \leftarrow 2$ **to** n **do**
  swap (tab[i],tab[rnd[1..i]]);
return tab;

Proposition 31. *The algorithm* RandPerm(n) *returns a uniform permutation of* $\mathfrak{S}n$ *and its bit-complexity is optimal.*

Proof. For $i \in \mathbb{N}$, the bit complexity of [rnd[1..i]] is of order $\log_2(i)$. Hence the total bit complexity of RandPerm(n) is equal to $\sum_{i=2}^{n} \log_2(i) = \log_2(n!)$. Since $\#\mathfrak{S}_n = n!$, it is optimal.

The fact that the result of RandPerm(n) is a uniform permutation follows directly from the next lemma. □

Lemma 32. *For all $\sigma \in \mathfrak{S}_n$, there exists a unique $(\tau_2, \dots, \tau_n)$, such that for all $2 \leq k \leq n$, $\tau_k = (i_k, k)$, with $i_k \leq k$ and $\tau = \tau_n \dots \tau_2$.*

Proof. We start by proving the existence part by induction. The result is clear if $n \leq 2$. Let $\sigma \in \mathfrak{S}_n$ be fixed and set $\tau_n = (\sigma(n), n)$. Then $(\tau_n\sigma)(n) = (n)$ and $\tau_n\sigma$ can be seen as a permutation of $\mathfrak{S}_{n-1}$ and by the induction hypothesis, there exists $(\tau_2, \dots, \tau_{n-1})$ satisfying the properties of the lemma and such that $\tau_n\sigma = \tau_2, \dots, \tau_{n-1}$, which concludes the existence part.

The uniqueness comes from the fact that the number of $(n-1)$-tuples of permutations satisfying the assumptions of the Lemma is equal to $n!$. Thus the construction described above is a bijection between such $(n-1)$-tuples and the set of permutations on n elements. □

3.2.2 Generation by permutations: compositions and balanced binary words

Let $\mathcal{C}_{n,k}$ be the set of compositions of n in k parts. In other words,

$$\mathcal{C}_{n,k} = \big\{(n_1, \dots, n_k) \text{ such that } \forall 1 \leq i \leq k, n_i \in \mathbb{N}^\star \text{ and } \sum_{i=1}^{k} n_i = n\big\}.$$

It is convenient to see elements of $\mathcal{C}_{n,k}$ as binary words with $n-k$ letters a and $k-1$ letters b in the following way. To an element $(n_1, \dots, n_k)$ of $\mathcal{C}_{n,k}$, we associate the word $a^{n_1-1}ba^{n_2-1}b \dots ba^{n_k-1}$. This is clearly a bijection which enables in particular to see that $\#\mathcal{C}_{n,k} = \binom{n-1}{k-1}$.

Note now that for any fixed n and k, there is a simple surjective mapping from $\mathfrak{S}_{n-1}$ to $\mathcal{C}_{n,k}$: associate with $\sigma \in \mathfrak{S}_{n-1}$ the word $\Phi_k(\sigma) = w = w_1 \dots w_{n-1}$, where $w_i = a$ if $\sigma(i) \leq n-k$ and $w_i = b$ otherwise. For instance for $n = 10$ and $k = 4$, the permutation 2 6 7 3 8 9 4 1 5 is mapped to *aababbaaa*, which corresponds to the composition $10 = 3 + 2 + 1 + 4$.

Since Φ_k is surjective and each $c \in \mathcal{C}_{n,k}$ has $(n-k-1)!(k-1)!$ preimages under Φ_k, the uniform distribution on $\mathfrak{S}_{n-1}$ is projected by Φ_k to the uniform distribution on $\mathcal{C}_{n,k}$. Therefore the following algorithm is a uniform random generator on $\mathcal{C}_{n,k}$ of complexity $\mathcal{O}(n)$ (in an arithmetic model):

$$\text{Gen}\mathcal{C}_{n,k}: \quad \sigma \leftarrow \text{RandPerm}(n-1); \quad \textbf{return } \Phi_k(\sigma).$$

Remark 7. In terms of bit complexity, this algorithm is not optimal. The bit complexity is of order $\mathcal{O}(n \log n)$ (which comes from the sampling of a uniform permutation of size n), whereas the size of $\mathcal{C}_{n,k}$ is for instance of order 2^n when $k \sim n$ (and is then encoded by $\mathcal{O}(n)$ bits).

Another classical application of generation by permutations is the random generation of a balanced binary word. The set $\mathcal{B}_{2n}$ of balanced binary words of size n is the set of words with n letters a and n letters b. Again there is a simple surjective mapping from $\mathfrak{S}_{2n}$ to $\mathcal{B}_{2n}$: associate with $\sigma \in \mathfrak{S}_{2n}$ the word $\Phi_k(\sigma) = w = w_1 \dots w_{2n}$, where $w_i = a$ if $\sigma(i) \leq n$ and $w_i = b$ otherwise. We'll come back to this über-classical example in Section3.3.2 and will explain how to achieve linear bit complexity on average.

3.2.3 Cycle lemma and Dyck/Lukaciewicz words

We can combine the construction given in previous section and the cycle Lemma (see Lemma 25) to obtain a random sample for a binary tree of size n as described in Algorithm 2.

Algorithm 2: RandDyckWord(n)

Input: n an integer
Output: a random Dyck word with n up-steps and $n+1$ down-steps

let $\sigma = \texttt{RandPerm}\ (2n+1)$, w = empty word, $\mathtt{u}$ = empty word ;
for $i \leftarrow 1$ **to** $2n+1$ **do**
  if $\sigma(i) \leq n$ **then**
    $\mathsf{w} = \mathsf{w}\ u$
  else
    $\mathsf{w} = \mathsf{w}\ d$
let CurrentHeight $= 0$, min $= 0$, ind $= 0$;
for $i \leftarrow 1$ **to** $2n+1$ **do**
  if $\mathtt{w}_i = u$ **then**
    CurrentHeight = CurrentHeight + 1
  else
    CurrentHeight = CurrentHeight- 1;
    if CurrentHeight $<$ min **then**
      min = CurrentHeight;
      ind $= i$
for $k \leftarrow 1$ **to** $2n+1$ **do**
  if $\mathtt{ind}+k \leq 2n+2$ **then**
    $\mathtt{u} = \mathtt{u}\cdot\mathtt{w}_{\mathtt{ind}+k-1}$
  else
    $\mathtt{u} = \mathtt{u}\cdot\mathtt{w}_{\mathtt{ind}+k-2n-2}$
return u

Remark 8. This construction can be generalized to get a uniform sampler of random forests with a fixed passport. We let to the reader the care to check the details.

3.3 Generation by rejection

The general principle of generation by rejection is very simple. Suppose we want to design a uniform sampler for a class $\mathcal{C}$ and that we have a uniform sample Gen$\mathcal{S}$ of a larger combinatorial class say $\mathcal{S}$, which contains $\mathcal{C}$. Then the following algorithm gives a uniform sampler for $\mathcal{C}$:

$$\begin{aligned} \mathrm{Gen}\mathcal{C}_n : \quad & \textbf{repeat } c \leftarrow \mathrm{Gen}\mathcal{S}_n \\ & \textbf{until } c \in \mathcal{C}_n \\ & \textbf{return } c. \end{aligned}$$

For each call to the random sampler, Gen$\mathcal{S}_n$, the probability that its result belong to $\mathcal{C}_n$ is equal to $|\mathcal{C}_n|/|\mathcal{S}_n|$. So, for a fixed element c of $\mathcal{C}_n$, we have:

$$\mathbb{P}\Big(\mathrm{Gen}\mathcal{C}_n = c\Big) = \frac{1}{|\mathcal{S}_n|}\sum_{k=0}^{\infty}\left(1-\frac{|\mathcal{C}_n|}{|\mathcal{S}_n|}\right)^k = \frac{1}{|\mathcal{C}_n|}.$$

Hence, when the algorithm terminates, it produces a uniform random element of $\mathcal{C}_n$. Since it may happen (with probability 0 but still... remember that usually we don't want to wait an infinite amount of time for an algorithm to terminate) that the algorithm does not terminate, this algorithm is in fact what is called a *pseudo-algorithm*.

Remark 9. The complexity of a rejection (pseudo-)algorithm is driven by the complexity of the sampler $\text{Gen}\mathcal{S}_n$ and the expected number of times it has to be called.
Recall that a random variable X is said to follow a *geometric law* of parameter p if :

$$\mathbb{P}(X = k) = p(1-p)^{k-1} \quad \text{for } k \geq 1.$$

In other words, the variable X counts the number of trials needed to obtain a success, when the probability of success is equal to p, (be careful that the definition of a geometric law may fluctuate slightly by allowing the variable X to be equal to 0, in this case, X represents the number of failures before getting a success).
The expected value and the variance of X are given by :

$$\mathbb{E}[X] = \frac{1}{p} \quad \text{and} \quad \text{var}(X) = \frac{1-p}{p^2}.$$

3.3.1 Simple connected labeled graphs

Suppose we want to sample a random simple connected labeled graph on n vertices such as the one represented in Figure 3.1. This is *a priori* not an easy task and in particular the number of such graphs is not known.

Figure 3.1: An example of a simple labeled connected graph .

A much easier task is to sample a random (not-necessarily connected) simple graph on n vertices. The number of such graphs is easily seen to be $2^{\binom{n}{2}}$. Indeed, for each pair of vertices $\{i, j\}$, we can add the edge in the graph or not, resulting in $2^{\binom{n}{2}}$ possibilities. This construction also gives a random sampler $\text{Gen}\mathcal{G}_n$ for simple graphs on n vertices: for each pair of vertices $\{i, j\}$, add the edge $\{i, j\}$ to the graph with probability $1/2$. $\text{Gen}\mathcal{G}_n$ produces a uniform graph of a given size, indeed for a fixed simple graph g_n,

$$\mathbb{P}(\text{Gen}\mathcal{G}_n = g_n) = \sum_{1 \leq i < j \leq n} \frac{1}{2} = \frac{1}{2^{\binom{n}{2}}}.$$

In fact, $\text{Gen}\mathcal{G}_n$ is a special case of a classical construction called the *Erdös-Renyi graph*, defined as follows. Fixed $p \in [0, 1]$ and $n \in \mathbb{N}$, then the so-called *Erdös-Renyi graph* $\mathcal{G}_{n,p}$ of parameter p and size n denotes a random graph on n vertices where each edge is kept with probability p. This model of graph has been much studied and in particular if $p = p(n) >> \log n/n$, then

$$\mathbb{P}(\mathcal{G}_{n,p(n)} \text{ is connected}) \to 1 \quad (\text{which implies in particular } \mathbb{P}(\mathcal{G}_{n,1/2)} \text{ is connected}) \to 1).$$

Therefore, the rejection method in this case works extraordinary well, with probability tending to 1 with n, $\text{Gen}\mathcal{G}_n$ actually produces a random *connected* simple graph.

3.3.2 Dyck words and Florentine rejection

Recall that at the end of Section 3.2.2, we explain how to project a permutation on $2n$ elements to a balanced binary word. But even if this algorithm has arithmetic complexity $\mathcal{O}(n)$, the number of random bits it actually requires is of order $n\log_2(n)$, which are required to draw a permutation on $2n$ elements. However, the entropy (defined as the $\log_2$ of the cardinality of $\mathcal{B}_{2n}$ is only linear, so it is a pity to project from $\mathfrak{S}_{2n}$ which is much larger than $\mathcal{B}_{2n}$; and hopefully one can generate in $\mathcal{B}_{2n}$ with only $\mathcal{O}(n)$ random bits. In this section, relying on simple rejection principles, we describe a random generation algorithm on $\mathcal{B}_{2n}$ that needs $\mathcal{O}(n)$ random bits on average.

First, notice that for any set of binary words $\mathcal{E} \subset \{a,b\}^{2n}$, the algorithm that generates a string of $2n$ random bits until the resulting binary word is in $\mathcal{E}$ is a uniform (naive) random generator on $\mathcal{E}$. The probability of success when $\mathcal{E} = \mathcal{B}_{2n}$ is equal to:

$$\text{probability of success } = \frac{\binom{2n}{n}}{2^{2n}} \approx \frac{1}{\sqrt{n}},$$

where we used Stirling approximation to get $\binom{2n}{n} \approx 2^{2n}/\sqrt{n}$.

The computation above indicates that in average we need $\mathcal{O}(\sqrt{n})$ calls to the algorithm that produces a string of $2n$ random bits and hence on average, the total of random bits used is $\mathcal{O}(n^{3/2})$. This is way worse that the approach using permutations and an intuitive reason to explain this (bad) behavior is that the probability of success is low and above all the cost of each call to the random sampler is high (linear in n).

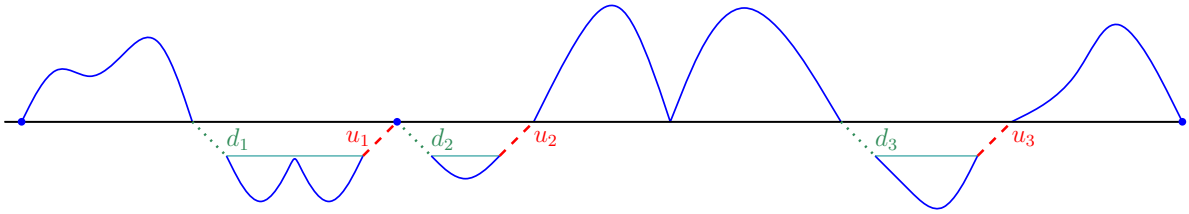
The principle of *Florentine rejection* is to lower the cost of each failure by determining as soon as possible whether the string of random bits is going to produce a valid candidate or not. In the algorithm described above, we have to wait until the last bit is sampled to decide whether the word belongs to $\mathcal{B}_{2n}$. Is there a way to modify the problem in an equivalent one for which we could decide earlier ?

There is indeed a way, described by Poulalhon and Schaeffer in *Combinatorics on words*. The first idea is to establish a bijection between $\mathcal{B}_{2n}$ and $\mathcal{P}_{2n}$, where $\mathcal{P}_{2n}$ is the set of non-negative walks of length $2n$, that is walks with $2n$ steps, where these steps are either upsteps $(+1,+1)$ and downsteps $(+1,-1)$ starting from $(0,0)$ and such that the ordinate remains nonnegative. Then we can perform a rejection algorithm on $\mathcal{P}_{2n}$ instead of $\mathcal{B}_{2n}$. The advantage is that the generation can be aborted very early in $\mathcal{P}_{2n}$, at the first time the walk visits negative ordinates.

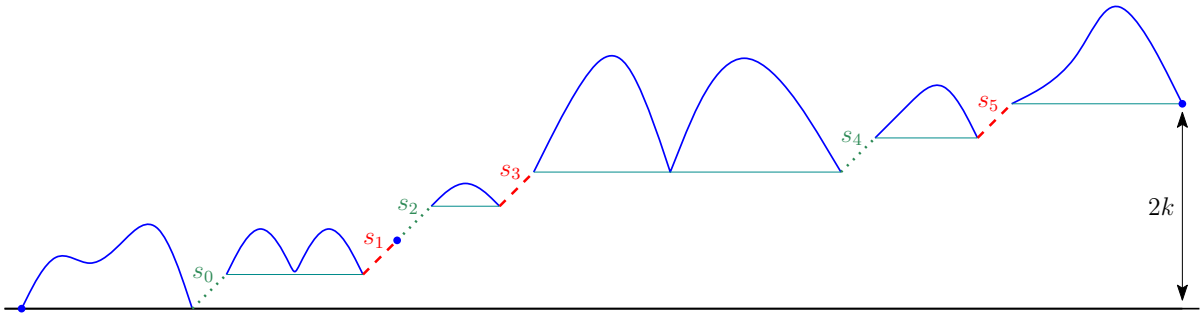
Proposition 33. *There exists an explicit bijection Φ between $\mathcal{B}_{2n}$ and $\mathcal{P}_{2n}$ for $n \geq 0$. The complexity of the bijection in both directions is $\mathcal{O}(n)$.*

Proof. For $k \geq 0$, let $\mathcal{B}^{(k)}$ be the subfamily of walks in $\mathcal{B}$ with k downsteps from ordinate 0 to ordinate -1. For $k \geq 0$, let $\mathcal{P}^{(k)}$ be the subfamily of walks in $\mathcal{P}$ with endpoint at ordinate $2k$. To prove the proposition, we prove the stronger result that there is an explicit bijection between $\mathcal{B}^{(k)}_{2n}$ and $\mathcal{P}^{(k)}_{2n}$.

Observe first that $\mathcal{B}^{(k)}$ admits a description as a regular language given by $(\mathcal{D}d\mathcal{D}^{-}u)^k\mathcal{D}$, where $\mathcal{D}$ represents the set of Dyck excursions (i.e $\mathcal{D} = \mathcal{B} \cap \mathcal{P}$), d is a downstep, u an upstep and $\mathcal{D}^{-}$ represents the set of mirrored Dyck excursions. In other words, if we denote $d_1, \ldots, d_k$ the k downsteps from ordinate 0 to ordinate -1 in an element B of $\mathcal{B}^{(k)}$, they alternate with



(a) The decomposition of a walk of $\mathcal{B}^{(k)}$.



(b) The decomposition of a walk of $\mathcal{P}^{(k)}$.

Figure 3.2: The bijection between paths of $\mathcal{P}^{(k)}$ and of $\mathcal{B}^{(k)}$.

k upsteps $u_1, \ldots, u_k$ and these $2k$ steps separate B into $2k+1$ walks $D_0, D_1, \ldots, D_{2k}$, where $D_0, D_2, \ldots, D_{2k}$ are Dyck excursions and $D_1, D_3, \ldots, D_{2k-1}$ are mirrored Dyck excursions (see Fig.3.2(a)).

Similarly, we can decompose a path W of $\mathcal{P}^{(k)}$. For $0 \leq i \leq 2k-1$, let s_i be the last step from ordinate i to ordinate $i+1$. Then the steps $s_0, \ldots, s_{2k-1}$ split W into $2k+1$ Dyck paths (see Fig.3.2(b)).

Then, to go from $W \in \mathcal{P}^{(k)}$ to $W' \in \mathcal{B}^{(k)}$. For $0 \leq i \leq k-1$, we apply an horizontal mirror to the walk between s_{2i} and s_{2i+1} and flip the step s_{2i}. The inverse bijection is similar and is left to the reader. The complexity of the bijection is clearly linear : to find the last-passage steps $s_0, \ldots, s_{2k-1}$, we read the path from right to left and record the down steps s_i of first arrival to ordinate i for i from $2k-1$ to 0. This takes time $\mathcal{O}(n)$ and the flip and mirror operations can also be performed in linear time. □

Thanks to the bijection Φ from $\mathcal{P}_{2n}$ to $\mathcal{B}_{2n}$, generating uniformly in $\mathcal{B}_{2n}$ boils down to generating uniformly in $\mathcal{P}_{2n}$, which is efficiently done by rejection:

Then the generator Gen$\mathcal{B}_{2n}$ is

Gen$\mathcal{B}_{2n}$: **return** Φ(RandPositiveWord(n))

Algorithm 3: RandPositiveWord(n)

Input: n an integer
Output: a random positive word with $2n$ steps

```
let CurrentHeight = 0, w = empty word ;
for i ← 1 to 2n + 1 do
    if Bern(1/2) then
        CurrentHeight = CurrentHeight + 1;
        w = w·u;
    else
        CurrentHeight = CurrentHeight − 1;
        w = w·d;
    if CurrentHeight < 0 then
        break;
return tab
```

Theorem 34. *The algorithm* Gen$\mathcal{B}_{2n}$ *is a uniform random sampler for* $\mathcal{B}_{2n}$ *of expected complexity* $\mathcal{O}(n)$*. The number of random bits required is also* $\mathcal{O}(n)$ *in average.*

Proof. Since Φ is a bijection and RandPositiveWord(n) produces a random positive word with $2n$ steps, Gen$\mathcal{B}_{2n}$ is clearly a uniform random sampler for $\mathcal{B}_{2n}$. We thus only need to check that the bit complexity of RandPositiveWord(n) is $\mathcal{O}(n)$.

At each trial to get a positive word on $2n$ steps, either we fail after $2i+1$ steps (with $0 \leq i < n$) or we succeed after $2n$ steps. Since a failure after $2i+1$ consists of a Dyck path of length $2i$ followed by a downstep, we get:

$$\mathbb{E}[\text{cost of a failure}] = \sum_{i=0}^{n-1} \frac{|\mathcal{D}_{2i}|}{2^{2i+1}}(2i+1),$$

where $\mathcal{D}_{2i}$ is the set of Dyck paths of length $2i$. Since $|\mathcal{D}_{2i}| \approx 4^i i^{-3/2}$, we get:

$$\mathbb{E}[\text{cost of a failure}] = \sum_{i=0}^{n-1} \frac{i^{-3/2}}{2}(2i+1) \approx \sum_{i=0}^{n-1} \frac{1}{\sqrt{i}} = \theta(\sqrt{n}).$$

On the other hand, the probability of success is of order $\frac{1}{\sqrt{n}}$ (as for the balanced binary words). In conclusion, we start on average with $\sqrt{n}$ failures, each of them has an average bit complexity of $\sqrt{n}$ and end with a success with has bit complexity of n. That gives a total bitcomplexity of n. □

Remark 10. This idea of "florentine rejection" can be applied to other combinatorial classes: for instance to Motzkin paths, directed animals, ...
The main difficulty is to find a criterion to get an anticipated rejection (here it required a non-trivial bijection between two family of paths). Also, one has to keep in mind, that the cost of checking at each step of the construction that the object still belongs to the appropriate step may be higher that generating the whole object and check at the last step. There is in fact often a trade-off between the complexity in time and the number of random bits required.

3.4 Targetting method

3.4.1 Balanced binary words revisited

We describe a third method to generate walks from $\mathcal{B}_{2n} \approx \mathfrak{S}(a^n b^n)$. Actually, we describe more generally a method to generate from the set $\mathcal{W}_{i,j}$ of walks with i up-steps and j down-steps, hence $\mathcal{W}_{i,j} \approx \mathfrak{S}(a^i b^j)$ and $W_{i,j} = \binom{i+j}{j}$. The name of the method is due to the fact that we fix the endpoint $(i+j, i-j)$ to arrive : the *target* of the walk.

Clearly, for $w \in \mathcal{W}_{i,j}$ taken uniformly at random:

$$\mathbb{P}(w \text{ starts with } 'a') = \frac{W_{i-1,j}}{W_{i,j}} = \frac{(i+j-1)!}{(i-1)!j!} \cdot \frac{i!j!}{(i+j)!} = \frac{i}{i+j},$$

which yields the following random generation algorithm:

$$\begin{aligned} \text{Gen}\mathcal{W}_{i,j} : \quad & \textbf{if } (i=j=0) \textbf{ return } \text{empty word} \\ & \textbf{if } \text{Bern}(i/(i+j)) \textbf{ return } ('a' \ + \ \text{Gen}\mathcal{W}_{i-1,j}) \\ & \textbf{else return } ('b' \ + \ \text{Gen}\mathcal{W}_{i,j-1}) \textbf{ end if}, \end{aligned}$$

that is clearly uniform on $\mathcal{W}_{i,j}$ by recurrence on $i+j$. Then a generator for $\mathcal{B}_{2n}$ is simply obtained as the particular case $\text{Gen}\mathcal{W}_{n,m}$. and gives an algorithm with linear arithmetic complexity to sample balanced binary words.

3.4.2 Binary trees and Remy's algorithm

We start this section by a bijection due to Remy between binary trees with $n-1$ nodes and a marked side of edge and binary trees with n nodes and a marked leaf. Let $\mathcal{A}_n$ denote the set of binary trees with n nodes. In the following, it will be convenient to imagine the tree as hanging from a root, with an edge connecting the root to the "topmost" vertex of the tree as shown in Figure 3.3 (these trees are usually called *planted* binary trees). Note, that with this convention a tree $t \in \mathcal{A}_n$ has $n+1$ leaves and $2n+1$ edges.

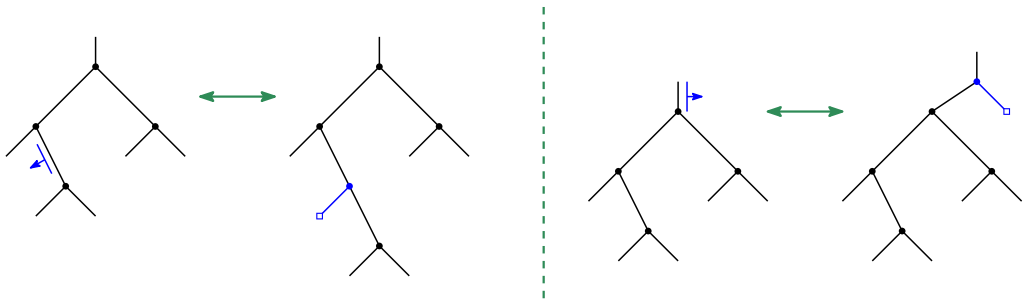


Figure 3.3: Two examples of Remy's bijection .

The bijection goes as follows. Starting from a tree with n nodes and a marked leaf ℓ, we erase the leaf and its parent edge. The parent of ℓ lies now in the middle of an edge, we erase it and mark this edge on the side where the parent edge of ℓ used to lie, see Figure 3.3. This construction admits an obvious inverse by adding a vertex on the middle of the marked edge

and grafting on it an edge and a leaf on the marked side. This gives in particular a nice proof of the fact that:

$$2(2n-1)\#\mathcal{A}_{n-1} = (n+1)\#\mathcal{A}_n.$$

This also yields an elegant method to sample a binary tree :

Gen$\mathcal{A}_n$: **if** $(n = 0)$ **return** the unique tree in $\mathcal{A}_0$
$t \leftarrow$ Gen$\mathcal{A}_{n-1}$;
choose a side of edge e of t uniformly at random;
attach at the middle of e of t (and toward the chosen side of e) an edge ended by a leaf;
return the obtained tree.

Theorem 35. *The algorithm* Gen$\mathcal{A}_n$ *is a uniform random generator on* $\mathcal{A}_n$ *of complexity* $\mathcal{O}(n)$.

Proof. The proof of uniformity goes by induction. For $n = 0$, this is evident, assume it is true for $n-1$, the second and third line of the algorithm draw a uniform element on the set of trees with $n-1$ nodes with a marked side of edge. The fourth line is Remy's algorithm which guarantees that the result is uniform in the set of trees with n nodes and a marked leaf. Since every binary tree with n nodes has $n+1$ leaves, forgetting the marked leaf gives a uniform element of $\mathcal{A}_n$.

The linear complexity is also clear if, at each stage, the tree is encoded in such a way that each node has two pointers towards its two childen and a global array stores the edges (identified to the pointers from a node to a child and including the root-edge). Thus choosing a side of edge uniformly at random takes time $'\mathcal{O}(1)$ and the tree modifications, which are very local, take also time $\mathcal{O}(1)$. Thus the overall complexity over the n steps is $\mathcal{O}(n)$. □

3.5 Recursive method

3.5.1 Introduction

The recursive method is an automatic method introduced by Nijenhuis and Wilf in 1978 an formalized (and developed) later by Flajolet, Van Custem and Zimmermann (1994). For any combinatorial class described by a recursive specification and any fixed size, this method produces a random sampler of the given size.

In theses notes, we illustrate only the method for unlabeled combinatorial classes whose specification involves $+, \star$ and Seq. We refer the reader to the work of Flajolet et al. for a more general setting. As before, we use the notation $\Gamma C[n]$ for a uniform random sampler on $\mathcal{C}_n$.

This method is illustrated on Figure 3.4 for the combinatorial class of complete binary trees of size 4.

3.5.2 Products and sums

Assume $\mathcal{A}$ and $\mathcal{B}$ are two combinatorial classes for which we already have fixed-size uniform generators $\Gamma A[i]$ and $\Gamma B[i]$. How can we construct a uniform random generator $\Gamma C[n]$ for $\mathcal{C} = \mathcal{A} + \mathcal{B}$ or $\mathcal{C} = \mathcal{A} \star \mathcal{B}$?

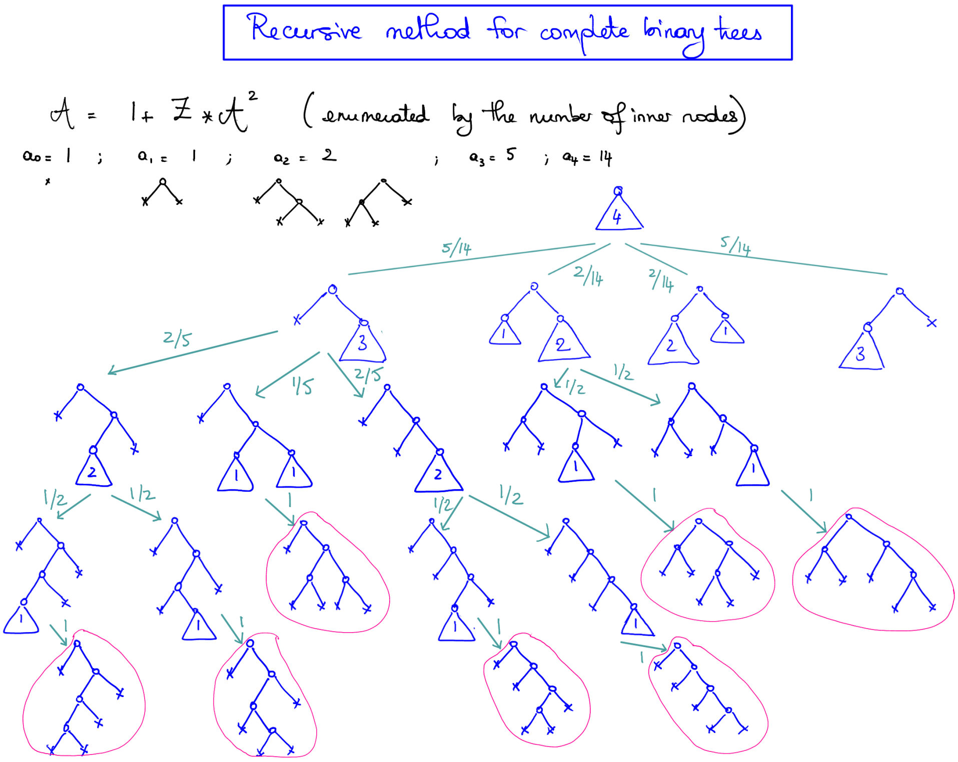


Figure 3.4: The sampling procedure to produce a uniform random binary tree with 4 nodes.

Disjoint union

Consider $\mathcal{C} = \mathcal{A}+\mathcal{B}$. Note that an object taken uniformly in $\mathcal{C}_n$ belongs to $\mathcal{A}_n$ with probability a_n/c_n. Hence the following random sampler is uniform in $\mathcal{C}_n$:

$$\Gamma C_n \ (\mathcal{C} = \mathcal{A} + \mathcal{B}) : \quad \textbf{if } \mathrm{Ber}(a_n/c_n) \textbf{ return } \Gamma A_n \\ \textbf{else return } \Gamma B_n$$

Proof. Let $\gamma \in \mathcal{C}_n$, assume wlog that $\gamma \in A_n$, then:

$$\mathbb{P}(\Gamma C_n = \gamma) = \mathbb{P}\big(\mathrm{Ber}(a_n/c_n) = 1\big) \cdot \mathbb{P}(\Gamma A_n = \gamma) = \frac{a_n}{c_n} \cdot \frac{1}{a_n} = \frac{1}{c_n}.$$

□

Product

Consider $\mathcal{C} = \mathcal{A} \star \mathcal{B}$. Note that for any object $\gamma = (\alpha, \beta)$ taken uniformly in $\mathcal{C}_n$, the probability that $|\alpha|_{\mathcal{A}} = k$ is equal to $a_k b_{n-k}/c_n$. Hence the following random sampler is uniform in $\mathcal{C}_n$:

$$\Gamma C_n \ (\mathcal{C} = \mathcal{A} \star \mathcal{B}) : \quad \text{Draw } k \text{ under the distribution } P(k = \ell) = a_\ell b_{n-\ell}/c_n \\ \textbf{return } (\Gamma A_k, \Gamma B_{n-k})$$

Proof. Let $\gamma = (\alpha, \beta) \in \mathcal{C}_n$, assume wlog that $|\alpha|_{\mathcal{A}} = \ell$, then:

$$\mathbb{P}(\Gamma C_n = \gamma) = \mathbb{P}(k = \ell) \cdot \mathbb{P}(\Gamma A_k = \alpha) \cdot \mathbb{P}(\Gamma B_{n-k} = \beta) = \frac{a_k b_{n-k}}{c_n} \cdot \frac{1}{a_k} \cdot \frac{1}{b_{n-k}} = \frac{1}{c_n}.$$

□

The size k of the first component is drawn by the following procedure :

ΓP : $p \leftarrow \text{Rand}[c_n]$; $k \leftarrow 0$; $d \leftarrow a_0 b_n$
while$(p > d)$ **do**
$k \leftarrow k + 1$; $d \leftarrow d + a_k b_{n-k}$;
od;
return k.

Note that the (arithmetic) cost of drawing k is equal to the number of iterative steps, i.e. is equal to k.

3.5.3 Complexity analysis

Example of the complete binary trees

The combinatorial class $\mathcal{A}$ of binary trees enumerated by their number of (inner) nodes is specified recursively by

$$\mathcal{A} = 1 + \mathcal{Z} \star \mathcal{A}^2,$$

since it is either empty or has a root-node with a left subtree and a right subtree.

For $a \in \mathcal{A}$, let $\lambda(a)$ be the cost of sampling a. If a is of size 0, then $\lambda(a) = 0$. Otherwise, denote a_1 the left subtree of a and a_2 its right subtree, then:

$$\lambda(a) = |a_1| + \lambda(a_1) + \lambda(a_2), \tag{3.1}$$

where the term $|a_1|$ comes from the cost of sampling the appropriate size for a_1 and a_2.

Now let $\Lambda(z)$ be the generating function of cost, that is $\Lambda(z) = \sum_{a \in \mathcal{A}} \lambda(a) z^{|a|}$. Using Equation (3.1), we get:

$$\begin{aligned}
\Lambda(z) &= \sum_{a_1, a_2} (|a_1| + \lambda(a_1) + \lambda(a_2)) z^{|a_1|+|a_2|+1} \\
&= z\Big[\sum_{a_1, a_2} (|a_1| z^{|a_1|} z^{|a_2|}) + \sum_{a_1, a_2} (\lambda(a_1) z^{|a_1|} z^{|a_2|}) + \sum_{a_1, a_2} (\lambda(a_2) z^{|a_1|} z^{|a_2|})\Big] \\
&= z^2 A'(z) A(z) + 2A(z)\Lambda(z)z.
\end{aligned}$$

Since, $A(z) = 1 + zA(z)^2$,

$$A'(z) = \frac{A(z)^2}{1 - 2zA(z)},$$

we get:

$$\Lambda(z) = \frac{z^2 A(z)^3}{(1 - 2zA(z))^2}.$$

Using the fact that the coefficients of $A(z)$ are the Catalan numbers, we get $a_n \sim 4^n/\sqrt{\pi n^3}$, hence $\lambda_n \sim 4^n/2$ and finally:

$$\mathbb{E}_n(cost) = \frac{\lambda_n}{a_n} \sim \frac{\sqrt{\pi}}{2} n^{3/2}.$$

Remark 11. These computations can be extended to all combinatorial classes, with the help of complex analysis to determine the asymptotics of the coefficients of $A(z)$ and $\Lambda(z)$, when no closed form is known. We refer again the interested reader to the work of Flajolet et al. for more details.

3.5.4 How to lower the complexity ?

Boustrophedonic strategy

When the specification of a combinatorial class includes a product, the main contribution to the total cost of the sampling procedure comes from the sampling of the distribution of sizes within the two components. Indeed, for large binary trees, most of the size n is concentrated either on the left subtree or on the right subtree. Since the complexity of sampling the size of the left subtree is linear in its size, the cases where it concentrates most of the mass contributes a huge cost to the total complexity.

Therefore, a (simple) idea to improve the complexity (of the sampling of the distribution of sizes) is to explore the possible values of the size of the left subtree in the following order: 0, n, 1, n-1, 2, n-2 , ... instead of the natural order : 0, 1, 2, 3, 4, ...

If we now compute the cost $\lambda_b(a)$ of sampling a tree in $\mathcal{A}$ with this new procedure, instead of Equation (3.1), we get:

$$\lambda_b(a) \leq 2\min(|a_1|, |a_2|) + \lambda_b(a_1) + \lambda_b(a_2) + 2.$$

Recurrences of a similar flavor have been studied by Knuth and his results enable to show that in this case $\Lambda_b(n) = \theta(n \log n)$ in the worst case.

Pointing

Recall that for any combinatorial class $\mathcal{A}$, we can consider its pointed version $\mathcal{A}^\bullet$, which is defined as $\mathcal{A}^\bullet = \cup_n \mathcal{A}_n \times [1..n]$ and corresponds to the set of objects in $\mathcal{A}$ with a marked atom (such as a tree with a marked node). The pointing operator obeys simple rules with respect to the classical specifications:

$$(\mathcal{A} + \mathcal{B})^\bullet = \mathcal{A}^\bullet + \mathcal{B}^\bullet \qquad \text{and} \qquad (\mathcal{A} \star \mathcal{B})^\bullet = \mathcal{A}^\bullet \star \mathcal{B} + \mathcal{A} \star \mathcal{B}^\bullet.$$

For the class $\mathcal{A}$ of binary trees (this time counted according to their number of leaves) specified by $\mathcal{A} = \mathcal{Z} + \mathcal{A}^2$, we get:

$$\mathcal{A}^\bullet = \mathcal{Z}^\bullet + \mathcal{A}^\bullet \star \mathcal{A} + \mathcal{A} \star \mathcal{A}^\bullet \approx \mathcal{Z} + 2\mathcal{A} \star \mathcal{A}^\bullet.$$

Since, for a given size n, there are n more pointed trees than trees, the distribution of the size $K^\bullet$ of the first component in $\mathcal{A} \star \mathcal{A}^\bullet$ is biased toward smaller values. Indeed

$$P(K^\bullet = \ell) = \frac{a_\ell (n-\ell) a_{n-\ell}}{n a_n},$$

so that $\mathbb{E}(K^\bullet) = \theta(\sqrt{n})$, instead of $n/2$ for the non-pointed version.

Therefore, on average the complexity is improved by a factor $\sqrt{n}$ at each step of the algorithm. A careful analysis of the cost in the pointed version enables to show that the total cost of the pointed version of a recursive method has complexity $\mathcal{O}(n \log n)$ on average.

Remark 12. A uniform sampler in the pointed class can immediately be transformed into a uniform sampler in the non-pointed class, just by forgetting the pointing. Since any (non-pointed) object of size n has exactly n pointed versions, pointing or depointing does not alter the uniformity of the sampling.

Cost of the pre-computation of the coefficients

Let us finally comment on the complexity of computing the coefficients $a_0, a_1, \ldots, a_n$ which are required to draw the sizes of the subtrees at a node. A naive recursive method, using the expression $a_n = \sum_{k=0}^{n-1} a_k a_{n-1-k}$ would require $\mathcal{O}(n^2)$ operations ($\mathcal{O}(n)$ operations for each coefficient). A faster way is to derive a linear differential equation for the specification.

For instance, from the algebraic equation $A = 1 + zA^2$, one obtains a linear differential equation (with coefficient polynomial in z) satisfied by $A(z)$. From which, one obtains, by coefficient extraction, a linear recurrence (with coefficients that are fixed polynomials in n) satisfies by the a_n's :

$$\begin{aligned} A(z) = 1 + zA(z)^2 &\implies z(1-4z)A'(z) + (1-2z)A(z) = 0 \\ &\implies (n+1)a_n - 2(2n-1)a_{n-1} = 0 \text{ for } n > 0. \end{aligned}$$

In this, we recover the recurrence satisfied by Catalan numbers (already obtained by Remy's bijection, see Section3.4.2). But this method applies more generally for any algebraic generating function, i.e. satisfying an equation of the form $P(f(z), z) = 0$, with P a polynomial. Using such a recurrence, the cost of computing each coefficient a_n is $\mathcal{O}(1)$, so the cost of computing all the n first coefficients is reduced to $\mathcal{O}(n)$.

3.6 Boltzmann sampling

We end this chapter about random generation by describing another automatic method called Boltzmann sampling. This section borrows its material from the article *"Boltzmann samplers for the random generation of combinatorial structures"* by Duchon, Flajolet, Louchard and Schaeffer published in 2004. This article is of course much more complete that what is presented here and we encourage the reader to refer to it for more details.

The great strength of the recursive method is that it is *automatic*: meaning that it can be applied verbatim to any decomposable class. Its main drawback however is its complexity when it comes to the pre computation of coefficients and the sampling of the size of one of the component when a product appears in the specification. Boltzmann sampling solves these two issues at the expense of getting only *approximate size* sampling.

3.6.1 Bolzmann model

Let $\mathcal{C} = \cup_n \mathcal{C}_n$ be a combinatorial class with generating series $C(x) = \sum_n c_n x^n = \sum_{\gamma \in \mathcal{C}} x^{|\gamma|}$. Let x in $(0, \rho_C)$ be a fixed parameter (where ρ_C is the radius of convergence of $C(z)$), then

the *free Boltzmann model* with parameter x is the probability law $\mathbb{P}_x$ on $\mathcal{C}$ given by:

$$\mathbb{P}_x(\gamma) = \frac{x^{|\gamma|}}{C(x)}, \text{ for any } \gamma in \mathcal{C}.$$

In the following, we will denote $\Gamma\mathcal{C}(x)$ a so-called *Boltzmann sampler* which draws an element of $\mathcal{C}$ under the probability law $\mathbb{P}_x$.

Note that the a Boltzmann sampler can return an object of any size. But, since $\mathbb{P}_x$ depends only on the size of γ, given the size of the object produced, the latter is uniform among the objects of this size.

In the case where $C(\rho_C) < \infty$, we can define the free Boltzmann model with parameter ρ_C, which we call the *critical Boltzmann model*.

The size of an object drawn by a Boltzmann sampler of parameter x is a random variable that we denote N. Then we have:

Proposition 36. *The random variable N satisfies:*

$$\mathbb{E}_x(N) = x\frac{C'(x)}{C(x)}, \qquad \mathbb{E}_x(N^2) = \frac{xC''(x) + xC'(x)}{C(x)}$$

and furthermore $\mathbb{E}_x(N)$ is an increasing function of x.

Proof. By definition:

$$\mathbb{E}_x(N) = \frac{\sum_n nc_n x^n}{C(x)} = \frac{xC'(x)}{C(x)}.$$

A similar computation gives the expression of $\mathbb{E}_x(N^2)$. Now, differentiating the latter equation gives:

$$x\frac{\partial}{\partial x}\mathbb{E}_x(N) = x\frac{C'(x)}{C(x)} + \frac{x^2C''(x)}{C(x)} - \frac{x^2C'(x)^2}{C(x)^2} = \mathbb{E}_x(N^2) - \big(\mathbb{E}_x(N)\big)^2 = \mathrm{var}(N) > 0.$$

□

Remark 13. Even if a Boltzmann sampler produces an object of approximate size, its expected value can be controlled by picking an appropriate value of x.
In the following, we always assume that the evaluation of the series at x are exactly known.

3.6.2 Boltzmann samplers for elementary constructions

Assume $\mathcal{A}$ and $\mathcal{B}$ are two combinatorial classes for which we already have Boltzmann samplers $\Gamma\mathcal{A}(x)$ and $\Gamma\mathcal{B}(x)$. How can we construct a Boltzmann sampler $\Gamma\mathcal{C}(x)$ for $\mathcal{C} = \mathcal{A}+\mathcal{B}$, $\mathcal{C} = \mathcal{A}\star\mathcal{B}$ or $\mathcal{C} = \mathrm{Seq}(\mathcal{A})$?

Disjoint union

Consider $\mathcal{C} = \mathcal{A} + \mathcal{B}$. Note that for $\alpha \in \mathcal{A}$,

$$\mathbb{P}(\Gamma\mathcal{C}(x) = \alpha) = \frac{x^{|\alpha|}}{C(x)} \quad \text{and so } \mathbb{P}(\Gamma\mathcal{C}(x) \in \mathcal{A}) = \frac{A(x)}{C(x)}.$$

Hence the following sampler is a Boltzmann sampler for $\mathcal{C}$ of parameter x:

$$\begin{aligned}\Gamma\mathcal{C}(x)\ (\mathcal{C} = \mathcal{A} + \mathcal{B}): \quad &\textbf{if } \mathrm{Ber}(A(x)/C(x)) = 1 \textbf{ return } \Gamma\mathcal{A}(x)\\ &\textbf{else return } \Gamma\mathcal{B}(x)\end{aligned}$$

Product

Consider $\mathcal{C} = \mathcal{A} \star \mathcal{B}$. Let $\gamma = (\alpha, \beta)$ be an element of $\mathcal{C}$, then :

$$\mathbb{P}(\Gamma\mathcal{C}(x) = (\alpha, \beta)) = \frac{x^{|\alpha|+|\beta|}}{C(x)} = \frac{x^{|\alpha|}}{A(x)} \cdot \frac{x^{|\beta|}}{B(x)},$$

Hence the following sampler is a Boltzmann sampler for $\mathcal{C}$ of parameter x:

$$\Gamma\mathcal{C}(x)\ (\mathcal{C} = \mathcal{A} \star \mathcal{B}) : \quad \textbf{return}\ (\Gamma\mathcal{A}(x), \Gamma\mathcal{B}(x))$$

Sequence

If $\mathcal{C} = \text{SEQ}(\mathcal{A})$, then $\mathcal{C} = 1 + \mathcal{A} \star \mathcal{C}$. A combination of Boltzmann samplers for the disjoint union and the product hence yields the following algorithm :

$$\begin{aligned} \Gamma\mathcal{C}(x)\ (\mathcal{C} = \text{SEQ}\ \mathcal{A}) : \quad & \textbf{if}\ \text{Ber}(A(x)) = 1\ \textbf{return}\ (\Gamma\mathcal{A}(x), \Gamma\mathcal{C}(x)) \\ & \textbf{else return}\ 1. \end{aligned}$$

We can reformulate this first sampler by noticing that the number of times that the Bernoulli variable of parameter $A(x)$ is going to be equal to 1 follows a geometric law of parameter $A(x)$. We obtain the following equivalent second sampler :

$$\begin{aligned} \Gamma\mathcal{C}(x)\ (\mathcal{C} = \text{SEQ}\mathcal{A}) : & k \leftarrow \text{Geom}\ (A(x)) \\ & \textbf{return}(\Gamma\mathcal{A}(x), \ldots, \Gamma\mathcal{A}(x))\ \text{(with k independant calls to } \Gamma\mathcal{A}(x)) \end{aligned}$$

3.6.3 Complexity

A Boltzmann sampler for a disjoint union or a cartesian products requires two independent calls to a Boltzmann sampler of the "simple" classes. Hence, its complexity is easy to analyse. On the other hand, a Boltzmann sampler for a sequence first requires sampling a variable according to a geometrical law. Denote p_k the probability that a geometric law of parameter λ is equal to k. Then $p_k = (1-\lambda)\lambda^k = \lambda p_{k-1}$ and $p_0 = 1 - \lambda$. It gives the following sampler for a geometric law of parameter λ:

$$\begin{aligned} \Gamma\text{Geom}(\lambda) : \quad & r \leftarrow \text{Rand}(0,1);\ k \leftarrow 0;\ s \leftarrow p_0 \\ & \textbf{while}(r \geq s)\ \textbf{do} \\ & \quad s \leftarrow s + p_k;\ k \leftarrow k + 1; \\ & \textbf{od}; \\ & \textbf{return}\ k. \end{aligned}$$

Note that the (arithmetic) cost of drawing k is equal to the number of iterative steps, i.e. is equal to k.

The only element of complexity left to discuss is the evaluation of the generating series at x. We assume that the values of the generating series at x are known up to an arbitrary big precision, this is the so-called *oracle assumption*. In parctive, one evaluates the generating functions with a fixed precision, say up to N digits (where typically $N = 20$) and in the unlikely case we need more digits during the sampling, we compute a few more digits. With this assumption, we have :

Theorem 37. *Consider a decomposable class $\mathcal{A}$ (i.e. such that $\mathcal{A}$ admits a recursive specification involving $\{+, \star, \mathrm{SEQ}\}$ and the basic classes $\{1, \mathcal{Z}\}$) and let $\Gamma A(x)$ be the Boltzmann sampler obtained from the sampling rules. Then, under the oracle assumption, the sampling of a uniform element alpha of $\mathcal{A}$ is linear in $|\alpha|$.*

3.6.4 Singular Boltzmann samplers

Usually, a Boltzmann sampler for a combinatorial class $\mathcal{A}$ is defined for a real x smaller than the radius of convergence ρ_A of $A(x)$. In some cases though, it makes sense to take x equal to ρ_A.

Supercritical sequences

Let $\mathcal{C} = \mathrm{SEQ}(\mathcal{A})$, with $\rho_A \geq \rho_C$, or in other words $A(\rho_C) = 1$. Clearly, $C(\rho_C) = \infty$, and the intuitive idea is that this divergence is due to the sequence operator and not to the sizes of the structures that belong to this sequence. More formally, $A(\rho_C)$ is finite, so we could sample according to $\Gamma A(\rho_C)$. But the issue comes from the geometric law which appears in the Boltmann sampling of a sequence, it would have parameter 1 in this example.

To avoid dealing with infinite sequences, an idea is to generate enough elements according to $\Gamma A(\rho_C)$ and to stop as soon as the desired total size is achieved. More formally, let n be the desired size of the output, sample $a_1, a_2, \ldots, a_k, a_{k+1}$ according to $\Gamma A(\rho_C)$, where $\sum_{i=1}^{k} |a_i| \leq n < \sum_{i=1}^{k+1} |a_i| > n$. Denote $\gamma = (a_1, \ldots a_k)$. The k-tuple γ is an element of $\mathcal{C}$, it remains to prove that this sampling procedure gives a uniform result for a fixed size:

$$\begin{aligned}\mathbb{P}\big(\Gamma C(\rho_C) = \gamma\big) &= \frac{\rho_C^{|a_1|}}{A(\rho_C)} \cdot \ldots \cdot \frac{\rho_C^{|a_k|}}{A(\rho_C)} \mathbb{P}\big(|a_{k+1}| > n - |\gamma|\big) \\ &= \rho_C^{|\gamma|} \cdot \mathbb{P}\big(|\Gamma A(\rho_C)| > n - |\gamma|\big),\end{aligned}$$

since the last quantity depends only on the size of γ, it concludes the proof.
Remark: It can be proved that asymptotically almost surely this sampling procedure produces a sequence of size $n + \mathcal{O}(1)$. A combination with some rejection hence gives a sampling procedure in exact size which runs in $\mathcal{O}(n)$.

Singular ceiled rejection

Consider a combinatorial class $\mathcal{C}$ such that $C(\rho_C) < \infty$, it could make sense to use a Boltzmann sampler evaluated at ρ_C. Let us see how this can be performed for the case of trees. Most of the families of trees we are interested with admit a generating function with a "square-root type" singularity. It implies in particular that the coefficients of their generating series behave as $c\rho^n n^{-3/2}$. Hence, the expected value of the size of the output of a Boltzmann sampler evaluated at ρ is infinite.

A simple modification of Boltzmann sampling uses ceiled rejection. The idea is to fix a maximum size, say M and perform Boltzmann sampling as usual, keeping only tracks of the number of atoms created. If this number happens to be larger than M at some point of the algorithm, then it is aborted and a new sampling starts.

It can be proved (and we refer to DFLS04 for the details) that such a procedure takes a linear time to produce a uniform tree of approximate size (see next section for the precise definition) and quadratic time for exact size.

3.6.5 Exact size vs approximate size

Let $n \in \mathbb{N}$, a random sampler can either produce a random object of size exactly n (and uniform among all the objects of size n of the combinatorial class of interest) or produce a random object of size between $n(1-\varepsilon)$ and $n(1+\varepsilon)$, (where $\varepsilon > 0$ is fixed). In the latter case, we still require the object to be uniform among all the objects of the same size.

The rejection paradigm enables to transform any sampler in approximate size to a sampler in exact size. The first step is of course to pick the value of the Boltzmann sampler so that the expected size of its output is equal to the "target size". If the distribution (i.e the law of the size of an element sampled via a Boltzmann procedure) we consider is concentrated (or flat) around its expected value, then the rejection method works reasonably well (e.g. partitions, binary words,...). Unfortunately in the case of trees, this is not at all the case: a Boltzmann sampler for trees is going to produce trees of very small size. A solution in this case is to use some pointed objects.

Let us illustrate this fact on the (surprisingly enough) combinatorial class $\mathcal{B}$ of binary trees counted according to their number of leaves. We know that $B(x) = (1-\sqrt{1-4x})/2$, $B'(x) = (1-4x)^{-1/2}$ and $b_n \sim c\rho^{-n}n^{-3/2}$, where $\rho = 1/4$ is the radius of convergence of B. Fix n the target size (and think of n as big) and $x = \rho(1-\varepsilon)$. The expected cost of sampling one binary tree is linear in the size of the produced tree and then is linear in $x\dfrac{B'(x)}{B(x)}$. On the other hand the probability that the size of the output is exaclty n is equal to $b_n x^n/B(x)$. Hence:

$$\mathbb{E}_x(\text{total cost}) = \frac{\mathbb{E}_x(\text{cost of sampling one element})}{\mathbb{P}(\text{exact size achieved})} \sim \frac{B'(x)}{b_n x^{n-1}}.$$

The latter expression is equivalent to $n^{3/2}/(\sqrt{\varepsilon}(1-\varepsilon)^{n-1})$. This is minimal for $\varepsilon = 1/2n$ and then gives a total quadratic complexity. This is of course far less efficient that the procedure we obtained with the recursive method. But observe that if we're interested in a sampling procedure in approximate size, the complexity becomes linear (which is quite cool) and is often what we are looking for.

If we want to improve the probability for the sampler to produce a tree which is of size approximately n, it can again be interesting to consider a pointed version. The combinatorial class $\mathcal{B}$ satisfies the decomposition $\mathcal{B} = \mathcal{Z} + \mathcal{B} \star \mathcal{B}$. Hence the pointed version $\mathcal{B}^\bullet$ of the class satisfies $\mathcal{B}^\bullet = \mathcal{Z} + \mathcal{B}^\bullet \star \mathcal{B} + \mathcal{B} \star \mathcal{B}^\bullet$. Recall that $B(x) = \frac{1-\sqrt{1-4x}}{2}$ and denote:

$$p_0 = \frac{2x}{1-\sqrt{1-4x}} \quad \text{and} \quad p_1 = \sqrt{1-4x}.$$

The Boltzmann samplers for $\mathcal{B}$ and $\mathcal{B}^\bullet$ can then be described as:

$$\begin{aligned} \Gamma B(x): \quad & \textbf{if } \mathrm{Ber}(p_0) = 1 \textbf{ return } \mathcal{Z} \\ & \textbf{else return } (\Gamma B(x), \Gamma B(x)). \end{aligned}$$

and

$$\begin{array}{ll} \Gamma B^\bullet(x): & \textbf{if } \mathrm{Ber}(p_1) = 1 \textbf{ return } \mathcal{Z} \\ & \textbf{else if } \mathrm{Ber}(1/2) = 1 \textbf{ return } (\Gamma B^\bullet(x), \Gamma B(x)) \\ & \quad \textbf{else return } (\Gamma B(x), \Gamma B^\bullet(x)) \end{array}$$

Let us illustrate the relative efficiency of simple Boltzmann samplers versus pointed Boltzmann samplers. Fix $n = 200$, then we must pick x to be equal to 0.2499984297 for the non-pointed version and to 0.2493734336 for the pointed version. Here are the sequences of sizes of trees obtained by several runs of the Boltzmann samplers, in the non-pointed case first:

$1, 5, 4, 1, 1, 393, 6, 28, 4, 1, 1, 1, 1, 1, 2, 110, 2204, 29, 1, 1, 2, 1, 83, 4, 1, 1, 1, 1, 1, 1, 4, 2, 15299, 1, 1, 1, ...$

and in the pointed case:

$166, 76, 96, 395, 21, 443, 36, 80, 842, 151, 12, 38, 71, 254, 163, 40, 37, 147, 194, 7, 12, 584, 135, 3, ...$

We let the numbers speak for themselves!