Génération aléatoire et uniforme de mots

A. Denise *

LaBRI, Université Bordeaux I
URA CNRS 1304

Résumé

Il existe une méthode de génération aléatoire de mots facteurs gauches de Motzkin et de chemins sous-diagonaux, méthode due à Barcucci, Pinzani et Sprugnoli et opérant en temps et espace linéaires en moyenne. Nous proposons une généralisation de ce procédé à un ensemble plus vaste de langages. Nous définissons une classe de langages, les fg-langages, pour lesquels sa complexité moyenne est aisément calculable, notamment de façon automatique à l'aide de logiciels de calcul formel. Des applications sont données pour quelques langages particuliers.

1 Introduction

Soit un ensemble de *lettres* $A = \{a_1, a_2, \ldots, a_k\}$ appelé *alphabet*. On note A^* le *monoïde libre* engendré par A, c'est à dire l'ensemble des mots finis composés de lettres de A. Un *langage* est un sous-ensemble de A^*. Etant donnés un langage L et un entier naturel n, notons L_n l'ensemble des mots de L de longueur n, et l_n le cardinal de L_n. La génération aléatoire et uniforme d'un mot consiste à tirer au hasard un mot de L_n avec une probabilité égale à $\frac{1}{l_n}$.

Les travaux concernant la génération aléatoire et uniforme de mots s'inscrivent dans une problématique plus vaste : la génération d'objets combinatoires. Celle-ci s'avère utile pour l'aide à la résolution de certains problèmes, comme par exemple l'étude de la distribution d'un paramètre sur une classe d'objets combinatoires. Dans la plupart des cas, le nombre d'objets de taille donnée n croît exponentiellement; générer tous les éléments de taille n devient vite une tâche impossible. Apparaît alors le besoin d'une étude statistique, via l'observation d'un "échantillon représentatif" de ces objets. Un tel échantillon ne peut être construit que par génération aléatoire et uniforme d'un nombre suffisant d'objets de taille fixée. La génération aléatoire et uniforme d'objets combinatoires trouve donc son utilité dans des domaines tels que :

- l'étude de la distribution de paramètres, ou de valeurs moyennes de certains paramètres dans des objets combinatoires;
- la visualisation d'objets combinatoires "moyens" de grande taille;
- les tests de complexité moyenne d'algorithmes portant sur des structures combinatoires.

*E-mail : denise@labri.u-bordeaux.fr

Ces champs d'application peuvent fournir une aide appréciable à la formulation ou à l'affinement de conjectures sur le comportement d'objets de très grande taille.

L'étude statistique d'objets combinatoires entraîne la génération d'un nombre important d'objets de grande taille. Il est donc nécessaire que les méthodes de génération offrent une complexité raisonnable en fonction de la longueur des mots. Nijenhuis et Wilf présentent dans [10] une méthodologie de tirage aléatoire et uniforme d'objets combinatoires, pouvant s'appliquer à de nombreuses classes d'objets. Cette méthode a été récemment systématisée et généralisée dans [7]. Il s'agit d'un algorithme dont la complexité est d'ordre polynomial en fonction de la taille des objets générés. Cependant son déroulement fait intervenir explicitement le nombre d'objets de taille n, ce qui altère sensiblement les performances lorsque ce nombre est exponentiel par rapport à n. De ce fait, la complexité d'un tel algorithme ne pourra en aucun cas être linéaire.

Dans certains cas, il est possible de réduire le problème de génération d'objets combinatoires à un problème de génération de mots. On sait énumérer des familles d'objets combinatoires en utilisant des bijections avec des langages, et particulièrement des langages algébriques [13]. Ces bijections sont en général constructives, de sorte qu'à tout mot du langage on sait associer son image dans la famille d'objets combinatoires; de plus, un certain nombre de paramètres sont "transportés" par la bijection, de l'objet combinatoire vers le mot associé.

Exemple 1.1 *Considérons la famille des animaux dirigés à une source sur réseau carré. Un animal dirigé est un objet combinatoire issu de la physique statistique, composé d'une configuration de points sur un réseau à coordonnées entières, deux à deux adjacents et se développant par ajout de nouveaux points voisins dans une direction privilégiée. On montre dans* [11] *que cette famille est en bijection avec le langage des facteurs gauches des mots de Motzkin. Le paramètre aire (nombre de points) d'un animal correspond au paramètre longueur du mot associé.*

Plus généralement, pour une famille d'objets combinatoires donnée, on définit le paramètre *taille* dont la valeur s'exprime comme une fonction de la longueur des mots associés. Ainsi, le nombre l_n de mots de longueur n est égal au nombre d'objets correspondants de taille $f(n)$. Il s'avère parfois grandement profitable de scinder le problème initial en deux parties : il s'agit d'abord de produire un mot, puis de construire l'objet combinatoire correspondant.

Il existe deux types de procédés de génération aléatoire et uniforme de mots, pour lesquels les calculs de complexité diffèrent.

Les algorithmes dits *déterministes* autorisent le calcul de la complexité "dans le pire des cas". Pour certains langages particuliers comme celui de Dyck, il existe des algorithmes performants, de complexité linéaire en temps et en espace mémoire [1, 12]. Plus généralement, l'algorithme de Hickey et Cohen [8], qui permet de tirer aléatoirement et uniformément des mots de tout langage algébrique non ambigu, est similaire à la méthode de Nijenhuis et Wilf [10]. De ce fait, il possède les qualités et les défauts de cette méthode, à savoir une complexité polynômiale dans tous les cas, mais des performances altérées du fait de l'utilisation explicite du nombre de mots de taille n. De ce fait, une génération efficace de mots de très grande taille par ce procédé semble exclue.

Les méthodes dites *à tirage*[1] procèdent en quelque sorte par une suite d'essais-erreurs, et

[1] Terminologie due à Jean Berstel.

de ce fait le nombre d'opérations nécessaires pour générer un mot n'est pas borné. Seule une complexité moyenne peut alors être prise en compte. Elles peuvent néanmoins constituer une alternative intéressante, comme en témoignent les travaux de Barcucci, Pinzani et Sprugnoli. Ceux-ci ont présenté [2] un tel procédé appliqué à la génération aléatoire et uniforme de facteurs gauches de Motzkin. Sa complexité en temps de calcul s'avère linéaire en moyenne. Quant à l'espace mémoire nécessaire, il est linéaire dans tous les cas. Elle est aussi appliquée à la génération de chemins sous-diagonaux [3], qui constituent une généralisation des facteurs gauches de Motzkin. Cette méthode est notamment utilisée pour générer en temps moyen et espace linéaires des animaux dirigés à une source en réseau carré.

Nous nous intéressons ici à l'extension de ce procédé à un ensemble plus vaste de langages. Nous évaluons sa complexité moyenne dans un cadre général, puis nous définissons une famille de langages, les *fg-langages*, pour lesquels elle peut être obtenue de façon simple en fonction de leur série génératrice. Enfin, à l'aide notamment des logiciels de calcul formel ΛΥΩ [6] et Maple, nous calculons la complexité de la méthode pour quelques langages particuliers.

2 Le procédé de génération de mots

2.1 Présentation

Considérons tout d'abord le pseudo-programme "naïf" suivant, qui génère un mot w de L_n.

Méthode 0:
Entrée: Un langage L, un entier naturel n.
Sortie: Un mot w de L_n.
début
  $w \leftarrow \epsilon$
  tant que $|w| < n$ **faire**
  début
    $a \leftarrow hasard(A)$
    $w \leftarrow wa$
    si $|w| = n$ **et** $w \notin L_n$
    alors $w \leftarrow \epsilon$
  fin
fin

Si w est un mot de L, $|w|$ représente la longueur de w. L'appel de la fonction $hasard(A)$ a pour effet de retourner une lettre choisie aléatoirement dans A avec la probabilité $\frac{1}{k}$ (où k est le cardinal de A). Nous supposerons que l'appel de cette fonction s'effectue en temps et espace constants.

La méthode 0 consiste donc à construire un mot de A^n avec la probabilité $\frac{1}{k^n}$, et recommencer le traitement tant que ce mot n'appartient pas à L_n.

Soit v un mot de L_n. La probabilité d'obtenir v après avoir généré i mots de $A^n \backslash L_n$ est égale à

$$p_i(v) = \left(1 - \frac{l_n}{k^n}\right)^i \frac{1}{k^n} \tag{2.1}$$

donc la probabilité pour que v soit résultat du procédé est

$$p(v) = \sum_{i \geq 0} p_i(v) = \frac{1}{l_n} \tag{2.2}$$

et la génération s'effectue bien de façon uniforme.

Le nombre moyen de mots tirés pour aboutir à un mot de L est $\frac{k^n}{l_n}$. Si on sait reconnaître un mot de L_n en temps $T(n)$, alors la complexité moyenne en temps est égale à $\frac{T(n)k^n}{l_n}$. En particulier, si L est un langage algébrique déterministe, c'est à dire s'il existe un automate à pile déterministe qui reconnaît les mots de L, alors elle est égale à $\frac{nk^n}{l_n}$.

Soit le mot w en cours de construction par la méthode 0, tel que $|w| < n$. Pour améliorer la complexité du procédé sans altérer l'uniformité du tirage, nous utilisons l'idée simple mais féconde exprimée dans [2] : détecter dès que possible si la construction en cours peut ou non aboutir à un mot du langage. Supposons que l'on puisse décider s'il existe ou non un mot u de L_n tel que w est facteur gauche de u. En cas de réponse négative, le traitement peut alors être recommencé sans attendre que la longueur de w atteigne n.

Notons $fg(L_n)$ l'ensemble des mots facteurs gauches d'un mot de L_n; la méthode 1 décrite ci-après tient compte de la remarque précédente.

Méthode 1:

`Entrée:` Un langage L, un entier naturel n.
`Sortie:` Un mot w de L_n.

```
début
    w ← ϵ
    tant que |w| < n faire
    début
        a ← hasard(A)
        w ← wa
        si w ∉ fg(L_n)
        alors w ← ϵ
    fin
fin
```

2.2 Notations

Les définitions et notations introduites ici nous permettront d'une part de montrer que la méthode 1 génère des mots de L de façon uniforme, et d'autre part d'étudier cette méthode du point de vue de sa complexité.

Soit L un langage sur un alphabet A. On note L_n l'ensemble des mots de L de longueur n, et $L_{\leq n}$ l'ensemble des mots de L de longueur inférieure ou égale à n. Soit $fg(L)$ l'ensemble des mots facteurs gauches d'un mot de L:

$$fg(L) = \{u \in A^* / \exists w \in L, \exists v \in A^*, w = uv\}$$

On note $\widehat{L}$ le langage des mots n'appartenant pas à L, mais dont le plus grand facteur gauche propre est dans L.

$$\widehat{L} = LA \backslash L$$

Les mots qui déterminent l'exécution de l'affectation $w \leftarrow \epsilon$ dans la boucle de la méthode 1, donc qui font recommencer la construction de w, sont exactement ceux du langage $fg(\widehat{L_n})_{\leq n}$. Pour plus de clarté dans les notations, nous écrirons

$$R = fg(\widehat{L_n})_{\leq n}$$

On définit les séries énumératrices de ces langages, $\mathcal{L}(t)$ pour L et $\mathcal{R}(t)$ pour R.

$$\mathcal{L}(t) = \sum_{n \geq 0} l_n t^n$$

$$\mathcal{R}(t) = \sum_{i=1}^{n} r_{n,i} t^i$$

2.3 Uniformité

Montrons que la génération d'un mot de longueur n par la méthode 1 s'effectue de façon uniforme. Un mot w sera cause d'une nouvelle exécution de la méthode 1 si et seulement si $w \in R$. La probabilité de cet événement est

$$\begin{aligned} p(w \in R) &= \sum_{i=1}^{n} p(w \in R_i) \\ &= \sum_{i=1}^{n} \frac{r_{n,i}}{k^i} \end{aligned} \tag{2.3}$$

Si l'on remarque que L_n et $\{R_i A^{n-i} \ / \ i \in [1,n]\}$ forment une partition de A^n, alors on a:

$$k^n - l_n = \sum_{i=1}^{n} r_{n,i} k^{n-i} \tag{2.4}$$

et donc d'après (2.3) et (2.4)

$$p(w \notin L_n) = 1 - \frac{l_n}{k^n}$$

Soit maintenant $v \in L_n$. On montre alors comme en (2.1) et (2.2) que la probabilité pour que v soit généré par la méthode 1 est égale à

$$p(v) = \frac{1}{l_n}$$

2.4 Complexité moyenne

Il est manifeste que la complexité moyenne en temps de la méthode 1 dépend étroitement du nombre moyen de lettres tirées pour obtenir un mot de L_n. Soit $a \in A$. Le test de la méthode 1 consiste à chercher si $wa \in fg(L_n)$, sachant que le mot w appartient à $fg(L_n)$. Dans ce qui suit, nous l'appellerons *test de maintien d'un mot dans* $fg(L_n)$. Il est effectué à chaque fois qu'une lettre est tirée. Pour que la méthode 1 soit raisonnable, il est donc nécessaire que la complexité en temps de ce test soit négligeable par rapport à celle du test de la méthode 0 (qui n'est effectué que tous les n tirages). Nous nous préoccuperons ici essentiellement des langages pour lesquels le test de maintien peut s'effectuer en temps constant. Le résultat suivant est alors immédiat :

Proposition 2.1 *Si le test de maintien d'un mot dans $fg(L_n)$ peut être effectué en temps constant, alors la complexité moyenne en temps de la méthode 1 est égale au nombre moyen de lettres tirées pour générer un mot de L.*

D'autre part, nous pouvons dès maintenant évaluer le gain maximum en temps occasionné par la méthode 1 par rapport à la méthode 0. En effet, soit m le nombre moyen de mots générés par la méthode 0 pour construire un mot de L_n; le nombre moyen de lettres tirées est donc égal à mn. Remarquons que m est aussi le nombre moyen de mots générés par la méthode 1. Supposons qu'une exécution de celle-ci a permis la génération d'un mot de L_n en i étapes. Dans le meilleur des cas, les $i-1$ premiers mots contiendront une seule lettre, et le nombre total de lettres tirées sera égal à $i-1+n$. Notons $C_0(n)$ (resp. $C_1(n)$) le nombre moyen de lettres tirées par la méthode 0 (resp. la méthode 1). Alors

$$C_1(n) = \frac{m-1+n}{mn} C_0(n) > \frac{C_0(n)}{n}$$

et on en déduit l'encadrement qui suit pour $C_1(n)$.

Proposition 2.2 *Soit $C_1(n)$ le nombre moyen de lettres tirées lors de la génération de mots de L_n par la méthode 1. Alors*

$$\frac{k^n}{l_n} < C_1(n) \leq n\frac{k^n}{l_n}$$

Autrement dit, pour que la méthode 1 soit efficace, il est nécessaire que la méthode 0 s'effectue en un temps raisonnable, donc que $\frac{k^n}{l_n}$ soit d'ordre polynômial. En particulier, $C_1(n)$ ne pourra être linéaire que si $\frac{k^n}{l_n} = O(n^\alpha)$, avec $\alpha < 1$.

Nous nous intéressons maintenant à la valeur de $C_1(n)$ en fonction des séries énumératrices $\mathcal{L}$ et $\mathcal{R}$. Pour ce faire, il nous faut introduire les notions qui suivent, empruntées à la Théorie des Codes[2] [4].

Soit π_0 la *distribution uniforme* sur A^*, ainsi définie :

$$\forall w \in A^* \quad \pi_0(w) = \frac{1}{k^{|w|}}$$

On étend π_0 à tout langage $L \subset A^*$ de la façon suivante :

$$\pi_0(L) = \sum_{w \in L} \pi_0(w)$$

$\pi_0(L)$ est appelée *mesure* du langage L.

Un *code préfixe* est un langage X tel que, si $w \in X$, alors aucun mot facteur gauche de w n'appartient à X. La *longueur moyenne* d'un code X (relativement à π_0) est définie par

$$\lambda(X) = \sum_{w \in X} |w| \pi_0(w)$$

Nous pouvons maintenant énoncer le résultat suivant

[2]Les notions de mesure et de longueur moyenne sont habituellement définies de façon plus générale, en fonction d'une distribution appelée *mesure cylindrique*.

Proposition 2.3 *Le nombre moyen de tirages de lettres nécessaires pour générer un mot de L_n par la méthode 1 est*

$$C_1(n) = n + \frac{\lambda(R)}{\pi_0(L_n)}$$

Preuve. Le nombre moyen de mots générés pour aboutir à un mot de L_n est, nous l'avons vu, $\frac{k^n}{l_n}$, soit $\frac{1}{\pi_0(L_n)}$. Chacun de ces mots est tiré dans $L_n \cup R$ selon la distribution uniforme π_0. Donc

$$\begin{aligned} C_1(n) &= \frac{1}{\pi_0(L_n)} \lambda(L_n \cup R) \\ &= \frac{\lambda(L_n) + \lambda(R)}{\pi_0(L_n)} \qquad \text{car } L_n \cap R = \emptyset \\ &= n + \frac{\lambda(R)}{\pi_0(L_n)} \end{aligned}$$

Notons que les longueurs moyennes $\lambda(L_n \cup R)$, $\lambda(L_n)$ et $\lambda(R)$ n'ont un sens que parce que $L_n \cup R$, L_n et R sont des codes préfixes, ce que l'on peut vérifier aisément. □

3 fg-langages

L'expérience montre que, pour un langage L quelconque, la structure de $\widehat{fg(L_n)}$, et donc celle de R, varient de façon extrêmement irrégulière en fonction de n. Le champ d'application de la proposition 2.3 se trouve donc limité du fait qu'on ne peut espérer trouver, dans le cas général, une expression de $\lambda(R)$ en fonction de L et n.

Dans cette section, nous définissons une classe de langages, les fg-langages, pour lesquels la longueur moyenne de R s'exprime de façon simple en fonction de la série énumératrice de L, et de n. Ainsi, le nombre moyen de lettres tirées lors de l'application de la méthode 1 à un fg-langage sera aisément calculable.

3.1 Définitions

Définition 3.1 *On dit qu'un langage L est* préfixiel *si $fg(L) \subset L$.*

Définition 3.2 *On dit qu'un langage L est* préfixe-complet *si pour tout mot $u \in L$, il existe un mot $v \in L$ tel que u est un préfixe (ou facteur gauche)* propre *de v (i.e $u \neq v$ et u est préfixe de v).*

Les deux définitions précédentes sont classiques en Théorie des Langages, et permettent d'énoncer la suivante, qui décrit la notion nouvelle de *fg-langage*.

Définition 3.3 *On dit qu'un langage L est un* fg-langage *s'il est préfixiel* et *préfixe-complet.*

On montre aisément la propriété suivante :

Propriété 3.4 *Si L est un fg-langage, alors pour tout mot $u \in L$ et pour tout entier $n > |u|$, il existe un mot $v \in L_n$ tel que u est préfixe de v.*

3.2 Complexité

Voici maintenant notre résultat principal, qui permettra un calcul aisé du nombre moyen de lettres tirées par la méthode 1 pour un fg-langage L, en fonction de sa série génératrice.

Théorème 3.5 *Si L est un fg-langage, alors le nombre moyen de lettres tirées par la méthode 1 pour la génération d'un mot de L_n est égal à*

$$C_1(n) = \frac{\pi_0(L_{\leq n-1})}{\pi_0(L_n)} \tag{3.5}$$

ou de façon équivalente

$$C_1(n) = \frac{[t^n]\left(\frac{t}{1-t}\mathcal{L}(\frac{t}{k})\right)}{[t^n]\left(\mathcal{L}(\frac{t}{k})\right)} \tag{3.6}$$

Ce théorème est une conséquence de la proposition suivante, qui présente une propriété plus générale des fg-langages.

Proposition 3.6 *Si L est un fg-langage, alors*

$$\forall n > 0, \quad \lambda(R) = \pi_0(L_{\leq n-1}) - n\pi_0(L_n) \tag{3.7}$$

Preuve. Soit $u \in (fg(L))_{\leq n}$. Alors $u \in L_{\leq n}$ car L est préfixiel. Donc il existe $v \in L_n$ tel que u est facteur gauche de v, car L satisfait la propriété 3.4. En conséquence, $u \in fg(L_n)$, et on en déduit que $fg(L_n) = (fg(L))_{\leq n} \quad \forall n \in I\!N$. Finalement, $\widehat{fg(L)}_i = \widehat{fg(L_n)}_i \quad \forall i \leq n$ et donc

$$\lambda(R) = \lambda(\widehat{fg(L)}_{\leq n})$$

D'autre part, $LA = \widehat{fg(L)} \cup L$, du fait que L est préfixiel. En particulier, $L_{i-1}A = \widehat{fg(L)}_i \cup L_i \quad \forall i \leq n$ donc

$$\begin{aligned}
\lambda(R) &= \lambda(L_{\leq n-1}A \backslash L_{\leq n}) \\
&= \sum_{i=0}^{n} i\frac{kl_{i-1} - l_i}{k^i} \\
&= \sum_{i=0}^{n-1} \frac{l_i}{k^i} - n\frac{l_n}{k^n} \\
&= \pi_0(L_{\leq n-1}) - n\pi_0(L_n)
\end{aligned}$$

□

Preuve du Théorème 3.5. On déduit aisément des propositions 2.3 et 3.6 la formule (3.5). Soit maintenant

$$\mathcal{T} = \sum_{n\geq 1}\left(\sum_{i=0}^{n-1}\frac{l_i}{k^i}\right)t^n$$

alors

$$\begin{aligned}
\mathcal{T} &= \sum_{n\geq 0}\frac{l_n}{k^n}\left(\sum_{m\geq 1} t^m\right)t^n \\
&= \frac{t}{1-t}\sum_{n\geq 0} l_n\left(\frac{t}{k}\right)^n
\end{aligned}$$

et on obtient le résultat (3.6). □

Le théorème 3.5 permet de calculer un équivalent asymptotique du nombre moyen de lettres tirées lorsque l'on connaît le comportement à l'infini de l_n. En particulier, si $l_n \sim cn^{-\alpha}k^n$, avec $\alpha \geq 0$ et $c > 0$, alors le tableau suivant, extrait de [14], donne des équivalents pour la moyenne et la variance du nombre de lettres générées, en fonction de α (K est une constante dépendant de $\mathcal{L}(t)$).

α	moyenne	variance
$\alpha < 1$	$\frac{n}{1-\alpha}$	$\frac{\alpha n^2}{(1-\alpha)^2(2-\alpha)}$
$\alpha = 1$	$n\log n$	$(n\log n)^2$
$\alpha > 1$	Kn^α	$K^2n^{2\alpha}$

4 fg-langages algébriques

Il est naturel de s'intéresser au cas particulier des langages algébriques, car ceux-ci interviennent dans le codage de nombreux objets combinatoires. Les facteurs gauches de Motzkin et les chemins sous-diagonaux sont notamment des mots de langages algébriques. Nous nous intéressons plus particulièrement aux langages algébriques déterministes, car ils présentent la propriété suivante :

Proposition 4.1 *Si L est un fg-langage algébrique déterministe, alors le test de maintien dans $fg(L)$ peut être effectué en temps et en espace constants.*

et sa conséquence immédiate

Corollaire 4.2 *Si L est un fg-langage algébrique déterministe sur un alphabet A à k lettres, alors*

- *la complexité en espace de la méthode 1 est linéaire,*
- *la complexité moyenne en temps de la méthode 1 est égale à*

$$C_1(n) = \frac{[t^n]\left(\frac{t}{1-t}\mathcal{L}(\frac{t}{k})\right)}{[t^n]\left(\mathcal{L}(\frac{t}{k})\right)}$$

La preuve de la proposition 4.1 découle du résultat suivant.

Lemme 4.3 *Si L est un fg-langage algébrique déterministe, alors il est reconnu par états d'acceptation par un automate à pile déterministe $\mathcal{A}$ vérifiant les propriétés suivantes:*
- *l'état initial de $\mathcal{A}$ est un état d'acceptation,*
- *si q est un état d'acceptation de $\mathcal{A}$, alors q est origine d'une transition au moins vers un état d'acceptation,*
- *si q n'est pas un état d'acceptation de $\mathcal{A}$, alors il n'existe aucune transition de q vers un état d'acceptation.*

Preuve. Soit L un fg-langage, reconnu par états d'acceptation par un automate à pile déterministe $\mathcal{A} = < A, Z, Q, z_1, q_1, \lambda >$ Nécessairement, q_1 est un état d'acceptation car $\epsilon \in L$.

Supposons qu'il existe un état d'acceptation q qui n'est origine d'aucune transition vers un état d'acceptation. S'il n'existe pas de calcul valide menant à q, alors q peut-être ôté de l'automate $\mathcal{A}$. Nous pouvons donc supposer qu'il existe un mot $w \in L$ dont le seul calcul valide est de la forme

$$(w, q_1, z_1) \overset{*}{\vdash} (\epsilon, q, h)$$

et comme L est un fg-langage, il existe au moins un mot u de longueur $|w|+1$ appartenant à L, et dont w est facteur gauche. Donc il existe au moins une transition menant de q à un état d'acceptation, ce qui contredit l'hypothèse.

Supposons maintenant qu'il existe un état q qui n'est pas un état d'acceptation, et qui est origine d'une transition vers un état d'acceptation. Si aucun calcul valide ne passe par q, alors q peut être supprimé de l'automate. Sinon, il existe un mot $w \in L$ tel que $w = ux$, $u \in A^*$, $v \in A^*$ et donnant lieu au calcul

$$(w, q_1, z_1) \overset{*}{\vdash} (v, q, g) \overset{*}{\vdash} (\epsilon, q_F, h)$$

où q_F est un état d'acceptation.

Soit $i = |u|$. Alors $u \in (fg(L))_i$, donc $u \in L_i$ car L est un fg-langage. Donc q est un état d'acceptation, ce qui est absurde. □

Le résultat de la proposition 4.1 est obtenu en parcourant l'automate à pile défini par le lemme 4.3 au fur et à mesure de la construction de w. Alors $w \in fg(L_n)$ si et seulement si le parcours mène à un état d'acceptation.

5 Applications

5.1 Facteurs gauches de Motzkin et chemins sous-diagonaux

Nous retrouvons ici de façon simple à l'aide de l'outil de calcul formel ΛΥΩ [6] les résultat de Barcucci, Pinzani et Sprugnoli [2, 3] concernant la génération de facteurs gauches de Motzkin, et plus généralement de chemins sous-diagonaux.

Rappelons que le langage des facteurs gauches de Motzkin est l'ensemble des mots sur l'alphabet $\{x, y, z\}$ dont les préfixes ont un nombre de y qui ne dépasse jamais le nombre de x. Ce langage est engendré par la grammaire non ambiguë suivante

$$L \rightarrow xMyL + xL + zL + \epsilon$$
$$M \rightarrow xMyM + zM + \epsilon$$

dont la donnée permet, avec l'aide de ΛΥΩ, de calculer directement la complexité de la méthode 1.

Voici le programme ΛΥΩ correspondant.

```
type L = x M y L | x L | z L | e ;
     M = x M y M | z M | e ;
     x = atom(1/3) ;
     y = atom(1/3) ;
     z = atom(1/3) ;
     e = atom(0) ;
     S = t S | t ;
     t = atom(1) ;
     F = L S ;
to_analyze: F, L ;
```

Lorsque ce programme s'exécute, ΛΥΩ calcule les fonctions énumératrices $\mathcal{L}(\frac{t}{3})$ et $S(t)\mathcal{L}(\frac{t}{3})$ (où $S(t) = \frac{t}{1-t}$)), ainsi que les valeurs asymptotiques des coefficients de leurs développements en série respectifs.

Cela donne ici:

$$[t^n]\Big(\mathcal{L}(\frac{t}{3})\Big) = \frac{\sqrt{3}}{\sqrt{\pi n}} + O(\frac{1}{n})$$

$$[t^n]\Big(\frac{t}{1-t}\mathcal{L}(\frac{t}{3})\Big) = \frac{2\sqrt{3n}}{\sqrt{\pi}} + O(1)$$

et donc nous avons bien

$$C_1(n) = 2n + O(1)$$

Comme $C_0(n) = n\frac{3^n}{l_n} = \frac{n\sqrt{\pi n}}{\sqrt{3}}$, alors le gain en temps par rapport à la méthode 0 est de l'ordre de $\sqrt{n}$.

Les chemins sous-diagonaux sont une généralisation des facteurs gauches de Motzkin. Ils sont codés par des mots sur un alphabet $\{x_1, \ldots, x_a, z_1, \ldots, z_b, y_1, \ldots, y_c\}$. Un mot w code un chemin sous-diagonal si et seulement si, pour tout préfixe v de w, $\sum_{i=1}^{c} |v|_{y_i} \leq \sum_{i=1}^{a} |v|_{x_i}$. De la même façon que précédemment, la donnée de la grammaire permet d'écrire aisément un programme ΛΥΩ calculant la complexité moyenne en temps de la méthode 1, en application du corollaire 4.2. Il est nécessaire cependant de fixer des valeurs pour les entiers a, b et c, ΛΥΩ ne permettant pas de traiter des grammaires à nombre de lettres terminales variable. Comme dans le cas des facteurs gauches de Motzkin, les résultats de Barcucci, Pinzani et Sprugnoli [3] sont confirmés par ΛΥΩ : la méthode 1 est de complexité moyenne linéaire si $a \geq c$, exponentielle si $a < c$. Remarquons toutefois que, dans ce dernier cas, le calcul de $[t^n]\mathcal{L}(t)$ suffit pour conclure, à l'aide de la proposition 2.2.

5.2 Autres fg-langages

Un fg-langage n'est pas forcément algébrique. Ainsi, le langage des *facteurs des suites de Sturm* est un fg-langage. On montre [5, 9] que le nombre de tels mots de taille n est asymptotiquement de l'ordre de n^3. La proposition 2.2 permet d'en déduire immédiatement que la méthode 1 est de complexité exponentielle dans ce cas.

Rappelons qu'un facteur gauche de Dyck est un mot de $\{x, \bar{x}\}^*$ dont les préfixes ont un nombre de $\bar{x}$ qui ne dépasse jamais le nombre de x. Un *mélange* de deux facteurs gauches

de Dyck est un mot $w \in \{x, \bar{x}, y, \bar{y}\}^*$ tel que w privé des lettres x et $\bar{x}$ est un facteur gauche de Dyck sur l'alphabet $\{y, \bar{y}\}$, et w privé des lettres y et $\bar{y}$ est un facteur gauche de Dyck sur l'alphabet $\{x, \bar{x}\}$. Un tel mot code un chemin du plan comportant des pas élémentaires Nord, Est, Sud et Ouest, débutant à l'origine et se développant exclusivement dans le premier quadrant du plan. On montre que $l_n \sim \frac{2}{\pi} n^{-1} 4^n$, donc la méthode 0 s'effectue en temps moyen $C_0(n) \sim \frac{2}{\pi} n^2$, et la méthode 1 en temps $C_1(n) \sim n \log n$, occasionnant un gain sensible.

Remerciements

Merci à Jean Berstel et Jean-Guy Penaud pour leurs précieux conseils et suggestions lors de la réalisation de ce travail.

Bibliographie

[1] D. B. Arnold, M. R. Sleep, *Uniform random generation of balanced parenthesis strings*, ACM Trans. Programming Languages and Systems 2 (1980) 122-128.

[2] E. Barcucci, R. Pinzani, R. Sprugnoli, *Génération aléatoire des animaux dirigés*, actes de l'Atelier Franco-Québécois de Combinatoire, Bordeaux 1991, eds. J. Labelle et J. G. Penaud, publi LaCIM 10, Université du Québec à Montréal (1992).

[3] E. Barcucci, R. Pinzani, R. Sprugnoli, *Génération aléatoire de chemins sous-diagonaux*, actes du $4^{ème}$ Colloque Séries Formelles et Combinatoire Algébrique, Montréal 1992, eds. P. Leroux et C. Reutenauer, publi LaCIM 11, Université du Québec à Montréal (1992).

[4] J. Berstel, D. Perrin, *Theory of Codes*, Academic Press Inc. (1985).

[5] S. Dulucq, D. Gouyou-Beauchamp, *Sur les facteurs de suites de Sturm*, Theoretical Computer Science 71 (1990) 381-400.

[6] Ph. Flajolet, B. Salvy, P. Zimmermann, *Automatic average-case analysis of algorithms*, Theoretical Computer Science 79 (1991) 37-109.

[7] Ph. Flajolet, P. Zimmermann, B. Van Cutsem, *A Calculus for the Random Generation of Combinatorial Structures*, prépublication, Janvier 1993.

[8] T. Hickey, J. Cohen, *Uniform Random Generation of Strings in a Context-Free Language*, SIAM. J. Comput 4 (1983) 645-655.

[9] F. Mignosi, *Sturmian words and ambiguous context-free languages*, Internat. J. Found. Comput. Sci 1 (1990) 3 309-323.

[10] A. Nijenhuis, H.S. Wilf, *Combinatorial algorithms*, Academic Press Inc. (1975).

[11] J. G. Penaud, *Une nouvelle bijection pour les animaux dirigés*, actes du $22^{ème}$ Séminaire Lotharingien de Combinatoire, publi IRMA, Université Strasbourg I (1989).

[12] J. L. Rémy, *Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire*, R. AI. R. O Informatique Théorique, 19, 2 (1985) 179-195.

[13] M. P. Schützenberger, *Context-free languages and pushdown automata*, Information and Control 6 (1963) 246-261.

[14] L. Wacrenier, *Génération aléatoire et uniforme de mots d'un langage*, rapport de stage effectué au LRI, Paris (1992).

Permutations à motifs exclus et cartes planaires non séparables

S. Dulucq, S. Gire et J. West *

LaBRI, Université Bordeaux I
URA CNRS 1304

De nombreux auteurs se sont intéressés à l'énumération des permutations dont certains motifs (sous-suites) sont exclus. En particulier, Knuth [7], considérant l'ensemble des permutations triables par passage dans une pile, appelé parfois permutations de Catalan, a montré qu'elles correspondaient à l'ensemble des permutations ne possédant pas de sous-suite de type 231 noté $S_n(231)$. Des travaux plus récents [12, 13] ont eu pour but d'énumérer les permutations ne possédant pas divers types de motifs.

West [16] s'est intéressé au problème des permutations triables par deux passages dans une pile en imposant une règle du jeu du type "tour de Hanoï" pour les empilements, problème différent de la généralisation considérée par Knuth [7] qui n'imposait aucun type de contrainte. West a caractérisé ces permutations en termes de motifs exclus; les motifs interdits sont 2341 et 3241, ce dernier étant autorisé dans le cas où il est lui même sous-suite du motif 35241 dans la permutation. L'ensemble $S_n(2341, 3\bar{5}241)$ désigne ces permutations sur $[n]$.
West a conjecturé que le nombre de ces permutations est $2(3n)!/((n+1)!(2n+1)!)$, conjecture démontrée par Zeilberger [18] de manière analytique (preuve faisant intervenir une équation en trois variables de degrés respectifs 6, 8 et 9 résolue en utilisant le logiciel de calcul formel Maple). Zeilberger [18] souligne le fait qu'il avait vainement cherché une preuve combinatoire de ce résultat.

En effet, il apparait que ces nombres énumèrent les cartes planaires pointées non séparables suivant le nombre d'arêtes (travaux de Tutte [14]).

L'objectif du présent travail est de relier combinatoirement cette famille de cartes à une famille de permutations à motifs exclus, ceci constituant une étape dans la preuve combinatoire de la conjecture de West que nous donnons dans [5].

La première partie de ce travail est consacrée à l'obtention d'un arbre de génération des cartes planaires pointées non séparables, arbre que nous caractérisons par un jeu de règles permettant de le construire, ces règles étant liées à certains paramètres sur les cartes (nombre de sommets, degré du sommet pointé, degré de la face pointée).

Dans la deuxième partie, nous montrons que cet arbre est exactement l'arbre de génération de la famille de permutations $S_n(2413, 41\bar{3}52)$ pour lequel les règles de construction s'interprètent suivant des paramètres sur ces permutations (nombre de descentes, nombre d'éléments saillants à droite, nombre d'éléments saillants à gauche).

Nous déduisons de ces faits que les permutations de $S_n(2413, 41\bar{3}52)$ sont énumérées par $2(3n)!/((n+1)!(2n+1)!)$ et obtenons plusieurs formules donnant les distributions de ces permutations suivant divers paramètres. Elles sont déduites des formules donnant le nombre de cartes planaires pointées non séparables à n arêtes suivant le nombre de sommets (travaux de Brown et Tutte [2]), ou relativement au degré de la face pointée (travaux de Brown [1]).

*E-mail : name@labri.u-bordeaux.fr