الجمـــهوريـــة الجـــزائريـــة الديـمقـراطيـــــــة الشـــعبيـــــة
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعـلـيـم الـعـالـــي و الـبـحـث الـعـلـمي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
جـامـعـة عمّـــار ثـلـيـــجـــي بالأغـــواط
UNIVERSITE AMAR TELIDJI LAGHOUAT
كـلـيـــــة الـعـلـــــوم
**FACULTE DES SCIENCES**


DEPARTEMENT DE MATHEMATIQUES ET INFORMATIQUE

## *Mémoire de MASTER*

**Domain :** Mathématiques et Informatique
**Filière :** Informatiques
**Option :** Systèmes d'Information et de Décision

**Par** :
BEGOUGA Bilal

**THEME**

# Réalisation d'un générateur aléatoire d'automates d'arbres finis

*Soutenu publiquement devant le jury composé de:*

| | | |
|---|---|---|
| *Mr. ALLAOUI Taher* | *M.A.(A)* | *Président* |
| *Mme.CHERROUNE Hadda* | *M.A.(A)* | *Examinateur* |
| *Mme.KERROUCHE Badra* | *M.A.(A)* | *Examinateur* |
| *Mr. GUELLOUMA Younes* | *M.A.(A)* | *Encadreur* |

***Année Universitaire 2015/2016***

# Résumé

Les travaux contenus dans ce document se situent à la croisée de deux domaines de l'informatique théorique : la génération aléatoire et la théorie des automates à états finis. Les automates à états finis sont les machines les plus simples de la théorie des langages. Les autres machines utilisées dans ce domaine dérivent en général de leur structure. Les automates à états finis sont habituellement dédiés au test d'appartenance d'un mot à un langage, et de ce fait à la recherche de motif dans un texte ou à la création de structures d'indexation. C'est pourquoi ils se trouvent être utilisés sous leur forme simple ou sous des formes dérivées par l'ensemble des logiciels de manipulation de texte voire de signaux. On trouve parmi ces logiciels, les dictionnaires, les logiciels de reconnaissance vocale, de recherche d'un mot dans un texte, de compression de données, de compilation, on trouve aussi des applications à la bioinformatique ou à l'intelligence artificielle.

D'un point de vue théorique, la génération aléatoire est intimement liée aux problémes de dénombrement. C'est le pendant algorithmique des problémes mathématiques de dénombrement.Ceux-ci décrivent les algorithmes de génération et d'énumération des structures combinatoires "de base". D'une part ces observations permettent d'établir plus facilement des propositions autour des comportements de ces propriétés, d'autre part ces comportements justifient (et valident) les tests que l'on désire réaliser sur un logiciel. Les objets que l'on génére aléatoirement peuvent en effet être asymptotiquement complètement différents de ceux dont on désire disposer pour réaliser des tests. On conçoit sans difficulté que si les automates non-déterministes sont quasiment tous déterministes (ce qui est faux) il est inutile de les générer pour tester les algorithmes de conversion permettant d'obtenir un automate non déterministe.

**Mots clés** : la théorie des automates à états finis, Les automates à états finis, la théorie des langages,les automates déterministes, les automates non-déterministes, langages rationnels.

# Abstract

The work contained in this document stands at the crossroads of two theoretical fields of Computer Science: the Random Generation and the Theory of Finite-State Automata. Finite-state automata are the simplest machines used in the Theory of Languages. Other machines used in this field derive from their structure. Finite-state automata are usually devoted to the test of a word's belongingness to a language. They are thus used in their simple form or in one of their derived form by most softwares that treat texts. We can find among these softwares: dictionaries, speech recognition softwares, pattern matching softwares, spelling checker softwares, data compression, we also can find applications to the bio-informatic or to artificial intelligence.

From a theoretical point of view, generation of combinatorial structures is linked to the problems of enumeration. It is the algorithmic answer of the ma- thematical problem of enumeration . They give generation and random generation algorithms of the basic combinatorial structures. In one hand, these observations can lead to propositions around these properties. On the other hand, the behaviors of these properties justify (and validate) tests or benchmarks that are performed on softwares. Clearly, objects that one can generate can be asymptotically different from those that he needs to check or to test the speed of a software. We conceive without difficulties that if almost all nondeterministic automata are deterministic (which is false), it is useless to generate them in order to test the algorithm that converts a nondeterministic automaton.

**Keywords:** the Random Generation, Finite-state automata, Theory of Languages,deterministic

finite-state automata,Non-deterministic finite-state automata,Regular expressions .

# Remerciements

En premier lieu, nous remercions Dieu le tout puissant dont la sagesse et son savoir sont infinis. Il nous a donné santé, connaissance, force, courage et patience pour mener à bien notre travail et nous a guidés pour l'accomplissement de ce mémoire.

Nous tenons à exprimer notre profonde gratitude et nos sincères remerciements à Mr.GUELLOUMA Younes Pour ses dévouements, ses conseils qui nous ont éclairé dans l'élaboration de ce travail. Nous tenons également à remercier tous les enseignants qui nous ont encadrés tout au long de notre cursus universitaire. Tous les membres de nos familles et nos ami(e)s.

Nous tenons à remercier aussi les membres du jury d'avoir accepté de juger notre travail, et tout l'encadrement du Département d'informatique.

# Dédicaces

En premier, je dédie ce travail a mes très cher parent, ceux qui ont toujours été la pour moi quelque soit les circonstances, à ceux qui ont fait de moi ce dont je suis, à ceux qui n'ont jamais failli à leur devoir, et à qui j'espère rendre un jour le millième de ce qu'ils ont fait pour moi et mes sœurs, à la meilleure mère et le meilleur père sur terre, merci maman , merci papa et qu' Allah vous garde pour nous

Mes sœurs, mes cousins et cousines.

Mes chers grands-parents, et mes oncles et mes tantes, et a toutes les familles Begouga et Dziri.

Mes amis : Yacine, Mostafa, Youssef, Djamel et Midou , Riyad.

A mes collègues : toutes les filles et les garçons de ma promotion .

A tous ceux que je n'ai pas cités.

# Table des matières

**Résumé** i

**Abstract** iii

**Remerciements** v

**Dédicaces** vii

**Table des matières** ix

**Table des figures** xiii

**Liste des tableaux** xv

**Liste des algorithmes** xv

**Introduction** 1

**1 Automates, langages** 3

1.1 La hiérarchie de Chomsky . . . . . 4

1.1.1 Chomsky . . . . . 4

1.1.2 Grammaires de type 0 . . . . . 4

1.1.3 Grammaires contextuelles (type 1) . . . . . 4

1.1.4 Grammaires hors-contexte (type 2) . . . . . 4

1.1.5 Grammaires régulières (type 3) . . . . . 5

1.2 Langages et grammaires hors-contexte . . . . . 5

1.3 Définition des automates d'arbres . . . . . 10
1.3.1 Les arbres . . . . . 10
1.4 Arbre Abstraite . . . . . 13
1.5 Termes et les arbres . . . . . 14
1.6 Automates d'arbres rangés à états finis descendant . . . . . 17
1.7 Automates d'arbres rangés à états finis ascendant . . . . . 18
1.7.1 Reconnaissance d'un arbre . . . . . 19
1.7.2 Langages reconnaissables . . . . . 21
1.8 Conclusion . . . . . 22

**2 Generation aléatoire d'automate d'arbre 23**
2.1 Introduction . . . . . 24
2.2 Algorithmes et méthodes de génération aléatoire . . . . . 27
2.2.1 Génération, énumération . . . . . 27
2.2.2 Dénombrement . . . . . 29
2.2.3 Objets étiquetés, objets non étiquetés . . . . . 30
2.2.4 Notion de graphe aléatoire . . . . . 31
2.2.5 Méthode de génération aléatoire des arbres binaires . . . . . 32
2.2.6 Génération aléatoire et dénombrement d'automates à état finis 34
2.3 Les travaux connexes . . . . . 34
2.4 Génération aléatoire d' automates d'arbres à états finis non déterministe . . . . . 36
2.4.1 Introduction . . . . . 36
2.4.2 L'approche de Andreas Maletti et al . . . . . 37
2.4.3 Analyse analytique . . . . . 38
2.4.4 Analyse empirique . . . . . 40
2.5 Conclusion . . . . . 41

**3 Lazy générateur aléatoire 43**
3.1 générer puis valider . . . . . 44
3.1.1 Description générale . . . . . 44
3.1.2 Comment générer . . . . . 47
3.1.3 Comment Valider . . . . . 48

3.2 Psuedo Code . . . . . 48
3.3 Tests . . . . . 51
3.4 Analyse . . . . . 53
3.5 Conclusion . . . . . 56

**Conclusion et perspectives** **57**

**Bibliographie** **59**

# Table des figures

1.3.1 Exemple d'un arbre . . . . . 11
1.4.1 exemple d'arbre . . . . . 14
1.7.1 Exemple d'AAFA . . . . . 20
1.7.2 Reconnaissance d'un arbre par un AAFA . . . . . 20

2.4.1 Graphiques traçant la taille moyenne de l'determinization determinized FTA obtenue par plus de la densité pour FTA de taille 8 et 12. Réduction réduit la taille moyenne, mais ne déplace pas le pic. 39

3.1.1 Diagramme de notre modéle . . . . . 45
3.1.2 Diagramme de sequence expliquant le deroulement du générateur . 46
3.2.1 Resultat obtenu . . . . . 50
3.3.1 Capture d'écran de l'interface Générateur aléatoire . . . . . 51
3.3.2 Capture d'écran de resultat obtenue . . . . . 52
3.4.1 Capture de courbe 1| $\Sigma$ |= 50 . . . . . 53
3.4.2 Capture de courbe 2 | $\Sigma$ |= 150 . . . . . 54
3.4.3 Capture de courbe 1| $\Sigma$ |= 500 . . . . . 55

# Liste des tableaux

2.1 Densité attendue $d_2$et de densité observée $d'_2$ . . . . . 38
2.2 Ratio de garniture FTA dans une série d'accords de libre-échange généré aléatoirement paramétré par densité $d_2$ (rangées) et numéro $n$ (colonnes) des Etats. Les inscriptions en blanc indiquent qu'aucun de ces expériences a été effectuée. . . . . 41

# Liste des algorithmes

2.1 Genération aléatoire des permutation . . . . . 28
2.2 Générateur aléatoires des graphes avec un degré séquence . . . . . 36

3.1 Générateur aléatoire d'automates . . . . . 49

# Introduction

## Cadre général et objectifs

Le langage XML (eXtended Markup Language) est un format général de documents orienté texte. Il s'est imposé comme un standard incontournable de l'informatique. Il est aussi bien utilisé pour le stockage de documents que pour la transmission de données entre applications. Sa simplicité, sa flexibilité et ses possibilités d'extension ont permis de l'adapter à de multiples domaines allant des données géographiques au dessin vectoriel en passant par les échanges commerciaux. De nombreuses technologies se sont développées autour de XML et enrichissent ainsi son environnement.

Le langage XML dérive de SGML (Standard Generalized Markup Language) et de HTML (HyperText Markup Language). Comme ces derniers, il s'agit d'un langage orienté texte et formé de balises qui permettent d'organiser les données de manière structurée.

On a une bagage théorique pour plusieurs domaine a savoir des traitements automatique de langage naturelle, le Bioinformatique, Indexation...etc, et on à un nombre Important d'application pour les mots par exemple la recherche dans le net on utilise des Robots, les GREP...etc, on a une nouvelle tendance vers les arbres de prefixes et XML, et il existe une théorie pour les arbre [hors contexte] pas exploitée car on a des algorithmes complex et probléme NP_complet et enfin le probléme de décidabilité

Les automates reconnaissent les ensembles finis de mots. Pour cette raison, ils apparaissent naturellement dans divers domaines d'application comme le traitement automatique des langues. Nous nous sommes intéressés à la génération aléatoire de ces objets, en respectant la distribution uniforme pour un nombre d'états fixé.

Avoir un générateur aléatoire est un outil pratique à la fois pour tester empiriquement des algorithmes et pour guider le chercheur dans son étude des grands objets aléatoires.On a trouvé qu'il y a deux Solution : Corpus d'automates d'arbre et Generation aléatoire, C'est surtout cette derniére méthode qui s'avére ètre efficace qui on a un grand manque dans ce domaine et on a un seul travail de valeur de [10]qu'on va appuyée sur ca travaille .

## Contributions

On utilise les modèles des automates d'arbre qui sont des modèle de calcul à états finis généralisant les automates classique à la reconnaissance de termes et sont des modèles syntaxiques permettant de définir la strecture d'un ensemble d'arbres. Ils caractérisent les règles d'écriture d'un langage, le contribution se présente de deux principales étapes l'un est consiste à générer des modéles d'automates d'arbre pour la classification en utilisant le principe d'indication des arbres et les graphes , et l'autre consiste à simplifier les modéles en utilisant les propriétés et les algorithme de simplification d'automates . l'utilisation des automates ainsi que leurs propriétés permet d'éliminer la nécessité d'un échantillon supplémentaire pour l'élagage car la simplification se fait sur la strecture même de modéle généré.

## Organisation de la thèse

Ce mémoire est dans le cadre du projet de cooperation Algeria-SouthAfrica code[A/AS-2013002], Le présent mémoire est organisé comme suit. Dans le chapitre 1, nous allons vous présenter quelques concepts de base liés aux automates d'arbres et des notations. Le chapitre 2 est consacré à présenté queleques traveaux sur la génération aléatoire, le chapitre 3 nous allons pésenter notre travaille le générateur aléatoire pour les automates d'arbre non-deterministe. Nous terminons par une conclusion générale et queleque perspectives.

# Chapitre 1

# Automates, langages

Les arbres sont des représentations naturelles pour de nombreuses objet dans des applications réelles. Ainsi Des automates finis pour arbres constituent la base théorique pour beaucoup domaines tels que les langues de schemata XML, naturel langues protransformation, vérié techniques de cations, l'analyse du programme. Ces applications nécessitent efficace algorithmes pour les manipulations de base de l'arbre de automates finis (FTA) tels que déterminisation,inférence, et la minimisation. En outre, le développement rapide de ces applications domaines et en particulier les représentations des arbres provoque des problèmes de complexité en raison de l'explosion des données et l'augmentation arbre taille des automates. Pour résoudre ce problème, il est nécessaire d'utiliser des techniques de minimisation puis développer l'efficacité et force d'algorithmes pour cela. Les automates d'arbres finis sont une generalisation naturelle des automates de mots . Alors q'un automate de n'accepte q'un seul mot , un automate d'arbre accepte un arbre (terme).Les arbres apparaissent dans plusieurs domaines de l'informatique et d'ingénierie et ils sont utilisés dans des applications comme pour la manipulation de données au format XML, traitement de langage et la virification formelle . Les automates seront utilisé comme modèle formels pour les arbres de décision ou les graphes d'indiction pour realiser trois objectifs primodiaux : 1) l'utilisation d'un cadre formel de réprésentation , 2) la simplification des modéles générés en utilisant les propriétés des automates , 3) la reduction de temps de génération des régles par réécriture qui devient moins important que le temps que nécessiterait la

génération classique effectuée par un parcours de l'arbre ou de graphe . [3]

## 1.1 La hiérarchie de Chomsky

la hiérarchie de Chomsky est une classification des langages formels et des grammaires formelles, décrite par Noam Chomsky en 1956

Les classes de langages $L_0, L_1, L_2, L_3$(chaque langage étant un ensemble de mots) de la hiérarchie sont strictement imbriquées :. $L_0 \subseteq L_1 \subseteq L_2 \subseteq L_3 \subseteq U$ , $U$ Ici est l'univers de tous les langages..

### 1.1.1 Chomsky

identifie une hiérarchie de familles de grammaires de complexité croissante, chaque famille correspondant à une contrainte particulière sur la forme des règles de récriture. Cette hiérarchie, à laquelle correspond une hiérarchie de langages .

### 1.1.2 Grammaires de type 0

On appelle grammaire de type 0 une grammaire syntagmatique dans laquelle la forme des production est non-contrainte .

### 1.1.3 Grammaires contextuelles (type 1)

est une grammaire formelle dans laquelle les substitutions d'un symbole non terminal sont soumises à la présence d'un contexte gauche et d'un contexte droit. Elles sont plus générales que les grammaires algébriques. Les langages formels engendrés par les grammaires contextuelles sont les langages contextuels. Ils sont reconnus par les automates linéairement bornés.

### 1.1.4 Grammaires hors-contexte (type 2)

grammaire hors-contexte est une grammaire formelle dans laquelle chaque règle de production est de la forme $X \to \alpha$, où $X$ est un symbole non terminal et est une chaîne composée de terminaux et/ou de non-terminaux. Le terme « non

contextuel » provient du fait qu'un non terminal $X$ peut être remplacé par $\alpha$ , sans tenir compte du contexte où il apparaît. Un langage formel est non contextuel (ou hors contexte)s'il existe une grammaire non contextuelle qui l'engendre.

### 1.1.5 Grammaires régulières (type 3)

Une grammaire régulière, rationnelle ou à états finie permet de décrire un langage régulier. Bien que les expressions rationnelles soient le modèle de définition le plus courant d'un langage régulier, en particulier dans le domaine des langages de programmation, il ne s'en trouve pas moins que, techniquement, la reconnaissance d'un mot d'un langage régulier s'effectue par la traduction d'expressions rationnelles en grammaire régulière

## 1.2 Langages et grammaires hors-contexte

Les langages hors-contexte (angl. Context-Free Languages, CFL) ont de nombreuses applications dans l'étude des langages naturels, en compilation (syntaxe de langages de programmation), analyse de programmes, etc ...

#### Exemples

Langages des parenthèses, des expressions arithmétiques, des palindromes, $a^n b^n \mid n \geq 0$, ....

#### Grammaire hors-contexte

Une grammaire hors-contexte (CGF) $G = V, \Sigma, R, S$ si consiste d'un ensemble(fini) $V$ de symboles non-terminaux (ou variables), d'un alphabet (fini) $\Sigma$ de symboles terminaux, d'un ensemble de règles$R \subseteq V \times (V \bigcup \Sigma)*$ et d'une variable initiale (axiome) $S \in V$

#### Exemple

<phrase>→<GN><GV>
<GN>→<nom-complexe>|<nom-complexe><CdN>

<nom-complexe>→<article><nom>
<CdN>→<adjectif>|...
<GV>→<verbe><GN>
<article>→le|la|mon|ma...
<nom>→fille |garçon|danse
<adjectif>→aîné(e)|...
<verbe>→aime|...

### Dérivation

L'application d'une règle $A \to v$ de $R$ au mot $uAw \in (V \bigcup \Sigma)*$ produit le mot $uvw$ (on note $uAw \Rightarrow uvw$).On écrit $u = *vsil'$ on peut dériver le mot $u$ du mot $v$,c-a-d. si on a soit $u = v$,ou s'il existent $u_1, ..., u_n \in (V \bigcup \Sigma)*$tels que $u \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_n \Rightarrow v$. Le langage engendré par $G$ est défini par $L(G) = w \in \Sigma * | S ==\Rightarrow w$

### Exemple (dérivation)

$< phrase >\Rightarrow< GN >< GV >\to< nom - complexe >< CdN >< GV >\Rightarrow$
$< article >< nom >< CdN >< GV >\Rightarrow ma < nom >< CdN >< GV >\Rightarrow$
$ma\text{fille} < CdN >< GV >\Rightarrow ma\text{fille} < adjectif >< GV >\Rightarrow$
$ma\text{fille}a\hat{i}n\acute{e}(e) < GV > * ==\Rightarrow ma\text{fille}a\hat{i}n\acute{e}(e)aimeladanse$

### Dérivation gauche

Une dérivation gauche est une dérivation où on remplace toujours la variable la plus à gauche (si possible), voir exemple.

Un arbre de dérivation pour un mot $w \in \Sigma*$ à partir d'une variable $B \in V$ est un arbre ordonné étiqueté, dont les nœuds internes sont étiquetés pardes variables dans$V$,et les feuilles pardes terminaux (dans $\Sigma$) ou pour tout nœud $v$ étiqueté par $A \in V$, si $A_1, ..., A_n \in V \bigcup \Sigma$ sont les étiquettes des enfants $v_1, ..., v_n$ de $v$, alors $A \to A_1, ..., A_n$ est une règle de $G$. La racine est étiquetée par $B$,et $w$ est la frontière de l'arbre.

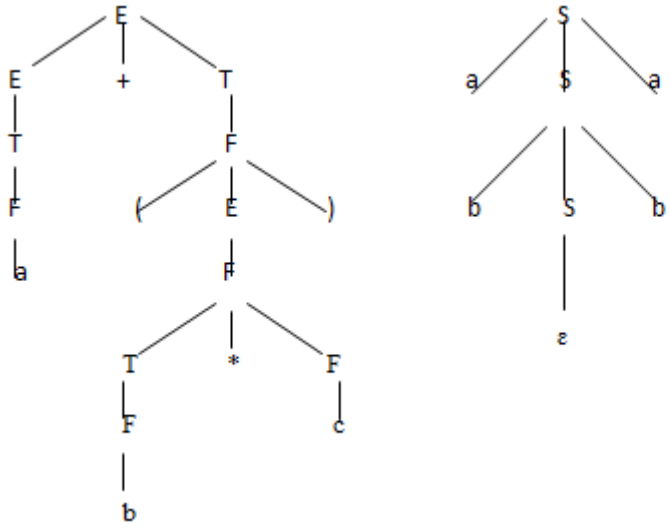
### Exemples - grammaires

- Une CFG qui engendre des expressions arithmétiques utilisant +, * et les variables a, b, c (langage non-régulier !)
  $E \to E + T \mid TT \to T * F \mid F$
  $F \to (E) \mid a \mid b \mid c$
- Une CFG qui engendre le langage $a^n b^n \mid n \geq 0$ :
  $S \to aSb$
- Une CFG qui engendre les palindromes sur {a, b} :
  $S \to aSa \mid bSb$

### Définition

Un langage de mots s'appelle hors-contexte (angl. context-free language, CFL), s'il est engendré par une grammaire hors-contexte.

### Exemples - arbres de dérivation







### REG ⊆CFL

Soit $L \subseteq \Sigma*$ accepté par un NFA $A = \{Q, \Sigma, \delta, q0, Fi\}$. Alors L est engendré par la CFG $G = \{Q, \Sigma, R, q0i\}$, où R contient toutes les règles de la forme $p \to aq$ si $q \in \delta(p, a)$, ainsi que $q \to$ si $q \in F$.

### Ambiguïté

Une CFG $G$ telle qu'il existe un mot dans $L(G)$ qui possède au moins 2 arbres de dérivation à partir de l'axiome, s'appelle ambiguë. Un langage hors-contexte $L$ est ambigu, si toute CFG qui l'engendre est ambiguë.

### Exemple

La CFG $G = hE, a, b, c, R, Ei$ où R contient les règles $E \to E{+}E|E{*}E|(E)|a|b|c$ est ambiguë. Le langage des expressions arithmétiques ne l'est pas.

### Opérations de clôture

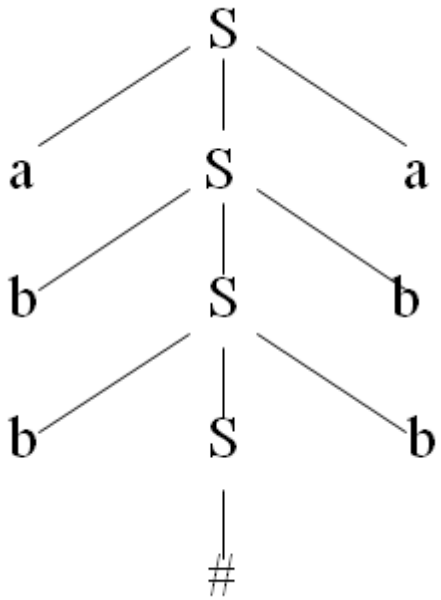
La famille des langages hors-contexte est fermée par les opérations rationnelles (union, produit, itération) et par intersection avec les réguliers. Elle n'est pas fermée ni par complémentaire, ni par intersection.

### Lemme de l'étoile pour les CFL

Soit $L \in \sum *$ un langage hors-contexte. Alors il existe un entier $N > 0$ tel que pour tout mot $z \in L$ de longueur supérieure à $N$, il existe une décomposition $z = uvwxy$ avec les propriétés suivantes :

1. $\mid vwx \mid = N$
2. $vx =!\varepsilon$
3. Pour tout $k \geq 0$, le mot $uv^k wx^k y$ appartient à $L$.

### Exemple




### Applications

Pour toute CFG G il existe un entier $N > 0$ (qui dépend de$G$ ) t.q. $L(G)$ est infini ssi il existe un mot $w \in L(G)$ de longueur$|w| > N$ . Le lemme de l'étoile permet de démontrer qu'un langage n'est pas hors-contexte.

L n'est pas CFL si :

Quelque soit $N > 0$ il existe $z_N \in L(G)$ de longueur supérieure à $N$ t.q. quelque soit la décomposition $z_N = uvwxy$ satisfaisant $|vwx| \leq N$ et $vx$ non-vide, il existe $k \geq 0$ t.q. $uv^k wx^k y \notin L$.

- Exemple 1 : $a^n b^n c^n \mid n \geq 0$ n'est pas CFL. Par contre, $L1 = a^n b^n c^m \mid m, n \geq 0$ et $L2 = a^n b^m c^n \mid m, n \geq 0$ sont CFL - donc $L1 \cap L2$ ne l'est pas.
- Exemple 2 :$L = w\#|w \in a, b*$ n'est pas CFL. Par contre $L^{co}$ est CFL (pas immédiat à voir...).

**Théorème**

Pour tout langage hors-contexte il existe un automate à pile (à un état) qui le reconnaît (avec pile vide). Réciproquement, les langages reconnus par les automates à pile sont des langages hors-contexte.

Dans les fin des années 60, une tendance vers les automates d'arbres qui sont une généralisation de celle des mots a commencée a être discuter. Le problème dans la relation entre les langages d'arbres et les langages Hors contexte on effet langage d'arbre INCLUS dans Langage Hors Contexte. Ces instances d'un langage hors contexte peuvent être vue via leur arbre de dérivation comme des Langages d'arbres. Mais dans la dérivation Le nouveau sur les arbres (XML, compit) ont commencé a voir le jour. Contrarient au langage hors contexte + Grammaire +Automate à pile. Les algorithmes sur les automates d'arbre sont d'une extrême puissance ce qui implique qu'on utilise les langages d'arbres + les Automates d'arbres. Au lieu d'explorer les langages hors contextes (utilisation de piles) on utilise les automates d'arbres .

## 1.3 Définition des automates d'arbres

Définissons maintenant les automates d'arbres, dont le langage est un ensemble (possiblement infini) de termes. Dans la modélisation d'un système informatique par un automate d'arbres, un terme permet de représenter une configuration (ou état) du système, et un automate d'arbres permet de représenter un ensemble de configurations (possiblement infini).

### 1.3.1 Les arbres

On a deux definition Informel pour les arbres :

**Def 1**

En informatique, un arbre est une structure de données récursive générale, représentant un arbre au sens mathématique. C'est un cas particulier de graphe qui n'a qu'une seule source et aucun cycle. Un arbre est une structure qui peut se

définir de manière récursive : un arbre est un arbre qui possède des liens ou des pointeurs vers d'autres arbres. Cette définition plutôt étrange au premier abord résume bien la démarche qui sera utilisé pour réaliser cette structure de données.

### Def 2

Arbre : un arbre est une structure de données composée d'un ensemble de nœuds. Chaque nœud contient l'information spécifie que de l'application et des pointeurs vers d'autres nœuds (d'autres sous-arbres).

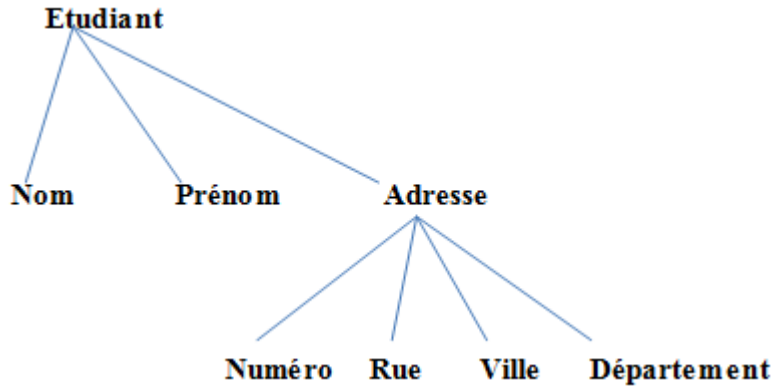
Spécification :ensemble d'éléments appelés nœuds liés par une relation de parenté établissant une hiérarchie entre les nœuds

Repesentation graphique

Terminologie associée

- Liens de parenté : père, fils, frère
- Branche
- Nœud, nœud racine, nœud feuille
- Etiquette d'un nœud
- Sous-arbre

FIGURE 1.3.1 : Exemple d'un arbre




L'exemple peut se noter :(étudiant (nom, prénom, adresse (numéro, rue, ville, département))).

### Feuilles

les nœuds ne pointant vers aucun autre nœud sont appelés feuilles (nom, prénom, numéro, rue, ville, département sont des feuilles).

### Racine

il existe un nœud au niveau 1 qui n'est pointé par aucun autre nœud :c'est la racine de l'arbre (étudiant est la racine de l'arbre).

### Niveau

Le niveau de la racine est 1. Les autres nœuds ont un niveau qui est augmenté de un par rapport au nœud dont ils dépendent.

### Hauteur (profondeur) d'un arbre

c'est le niveau maximum atteint (par la branche la plus longue). La hauteur du nœud étudiant est de 3.

### Arbre ordonné

si l'ordre des sous-arbres est significatif, on dit que l'arbre est ordonné (arbre généalogique par exemple).

### Degré d'un nœud

on appelle degré d'un nœud, le nombre de successeurs de ce nœud.

### Degré d'un arbre

si N est le degré maximum des nœuds de l'arbre, l'arbre est dit n-aire. Sur l'exemple, adresse a un degré 4. L'arbre est un arbre 4-aire.

### Taille

c'est le nombre total de nœuds de l'arbre. La taille est de 8 sur l'exemple.

#### Arbre binaire complet

c'est un arbre binaire de taille $2^k - 1$ (k étant le niveau des feuilles).

#### Arbre binaire parfaitement équilibré :

un arbre binaire est parfaitement équilibré si pour chaque nœud, les nombres de nœuds des sous-arbres gauche et droit diffèrent au plus d'un.

#### Arbre binaire équilibré

un arbre binaire est équilibré si pour chaque nœud, les hauteurs des sous-arbres gauche et droit diffèrent au plus d'un.

#### Un arbre formelement

En maths : cas particulier de graphe non orienté, connexe et acyclique. Définition formelle :

Un noeud $N$ unique est un arbre dont $N$ est la racine

Si N est un noeud et $A1, A2, ..., Ak$ sont des arbres dont les racines sont $N1, N2, ..., Nk$ alors la structure telle que N est le noeud père de $N1, N2, ..., Nk$ est aussi un arbre.

Un arbre sans noeud est un arbre vide

## 1.4 Arbre Abstraite

Nous désignons par $N$ l'ensemble des entiers positifs. Nous décrire l'ensemble de chaînes finie sur $N$ par $N*$. La chaîne vide est indiquée par $\varepsilon$.

Un classement alphabet est un couple ($\mathcal{F}$, arité) où $\mathcal{F}$ est un ensemble fini et arité est un mappage à partir de $\mathcal{F}$ en $N$.

L'arité d'un symbole $f \in \mathcal{F}$ est arité($f$). L'ensemble de symboles de arité $p$ est indiquée par $\mathcal{F}_p$. Éléments d'arité $0, 1, ...p$ sont respectivement appelées constantes, l'opérateur unaire, . . . , p-ary symboles. Nous supposons que $\mathcal{F}$ contient au moins une constante. Dans les exemples, nous utilisons des parenthèses et des virgules pour une courte déclaration de symboles avec arité. Par exemple, $f(,)$ est une courte déclaration pour un symbole binaire $f$.

La série $T(\mathcal{F})$ des termes sur l'alphabet classé $\mathcal{F}$ est défini par :
$\mathcal{F}_0 \subseteq T(\mathcal{F})$ et
Si $p \geq 1, f \in \mathcal{F}_p$et $t_1, \ldots, t_p \in T(\mathcal{F})$, Alors $f(t_1, \ldots, t_p) \in T(\mathcal{F})$
Le terme $T(\mathcal{F})$ est appelé terme de masse.

**Exemple**

Soit $\mathcal{F} = \{cons(,), nil, a\}$,Ici cons est un symbole binaire, $nil$ et $a$ sont constant, le terme $cons(x, y)$ est lineaire, le terme $cons(x, cons(x, nil))$ est non linéaire, le terme $cons(a, cons(a, nil))$ est un terme de masse. Ces termes peuvent être représentés sous forme graphique. Par exemple, le terme $cons(a, cons(a, nil))$ Est représenté par :

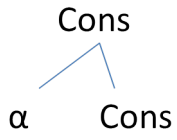



FIGURE 1.4.1 : exemple d'arbre

## 1.5 Termes et les arbres

Une arborescence ordonné finie $t$ sur un ensemble d'étiquettes $E$ est une application d'un préfixe- jeu fermé $Pos(t) \subseteq N*$ en $E$, Ainsi le terme $t \in T(\mathcal{F})$ Peut être considéré comme un arbre fini ordonné classé , dont les feuilles sont étiquetées avec des variables ou constantes noeuds internes et symboles sont étiquetées avec des symboles de positif, à l' extérieur arité degré égal à la arité de l'étiquette , c-a-d le terme $t \in T(\mathcal{F})$ Peut également être définie comme une fonction partielle $t : N* \to \mathcal{F}$ avec le domaine $Pos(t)$ qui satisfait les propriétés suivantes :

1. $Pos(t)$ est non-vide et préfix fermé .
2. $\forall p \in Pos(t)$, if $t(p) \in \mathcal{F}_n, n \geq 1$,Alors $j \mid p_j \in Pos(t) = \{1, \ldots, n\}$.
3. $\forall p \in Pos(t)$, if $t(p) \in \mathcal{F}_0$,Alors $j \mid p_j \in Pos(t) = 0$.

Nous confondons les termes et des arbres, qui est par "arbre" nous toujours synonyme d'un arbre classé satisfaisante commandés finie 1. , 2. et 3.

Le lecteur devrait noter que finie avec délimitée rang ordonné arbres $k$ - c'est-à-dire il y a une limite k sur le degrés de noeuds internes - Peuvent être codées dans un ordre finis pour classait les arbres : la marque $e \in E$ est associée à $k$ symboles $(e, 1)$ d'arité $1, \ldots, (e, k)$ d'arité $k$.

Chaque élément de $Pos(t)$ est appelé *position*, *une position de frontière* est une position $p$ tel que $\forall j \in N, pj \notin Pos(t)$. L'ensemble des positions de frontière est notée $\mathcal{F}Pos(t)$.Chaque position $p$ dans $t$ est appelé *position variable* . L'ensemble des positions variables de $p$ est notée $\mathcal{VP}os(t)$, on note par $\mathcal{H}ead(t)$ *le symbole de la racine* de $t$ qui est défini par $\mathcal{H}ead(t) = t(\varepsilon)$.

### Sous-termes

Le sous-terme $t \mid_p$ de terme $t \in T(\mathcal{F})$ en position $p$ se définit de la façon suivante :

- $Pos(t \mid_p) = \{j \mid pj \in Pos(t)\}$,
- $\forall q \in Pos(t \mid_p), t \mid_p (q) = t(pq)$.

  On note par $t[u]_p$ L'expression obtenue on remplace dans $t$ le sous-terme $t \mid_p$par $u$.

  On note par $\trianglerighteq$ *La commande subterm* c'est-à-dire nous écrivons $\trianglerighteq t'$ si $t'$ est un sous-terme de $t$. On note $t \trianglerighteq t'$ si $t \trianglerighteq t'$ et $t \neq t$.

  L'ensemble des termes $F$ est dit *être fermé* si elle est fermée sous le subterm commander , c'est à dire. $\forall t \in F\ (t \trianglerighteq t' \Rightarrow t' \in F)$.

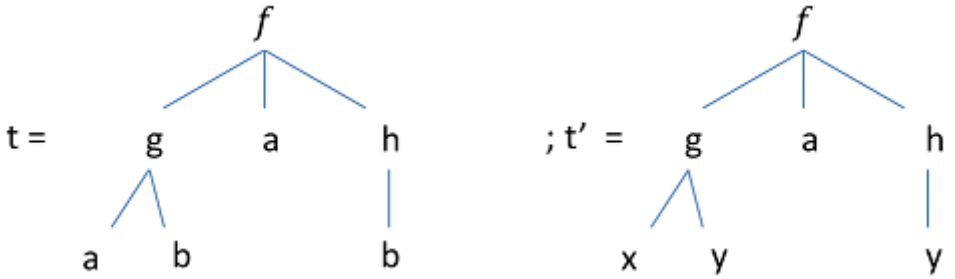
## Fonctions sur les termes

La taille d'un terme $t$ , notée par $\parallel t \parallel$ et la hauteur de $t$, indiquée par $\mathcal{H}eight(t)$ sont définies par induction par :

- $\mathcal{H}eight(t) = 0$, $\parallel t \parallel = 0$
- $\mathcal{H}eight(t) = 1$, $\parallel t \parallel = 1$ Si $t \in \mathcal{F}_0$,
- $\mathcal{H}eight(t) = 1 + max(\{\mathcal{H}eight(t_i) \mid i \in \{1, \ldots, n\}\})$, $\parallel t \parallel = 1 + \sum_{i \in \{1,\ldots,n\}} \parallel t_i \parallel$ Si $\mathcal{H}ead(t) \in \mathcal{F}_n$.

### Exemple

Soit $F = \{f(,,), g(,), h(), a, b\}$. considere les termes suivantes :




Le symbole de racine de $t$ est $f$ ; l'ensemble des postes frontieres de $t$ est $11, 12, 2, 31$ ; la série de

variable positions de $t'$ est $11, 12, 31, 32$ ; $t \mid_3 = h(b)$; $t[a]_3 = f(g(a, b)a, a)$;

$\mathcal{H}eight(t) = 3$ ; $\mathcal{H}eight(t') = 2$ ; $\| t \| = 7$ ; $\| t' \| = 4$.

## Substitutions

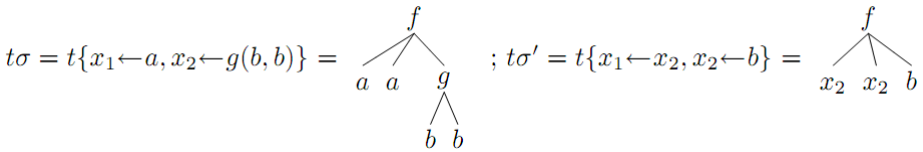
Une substitution (respectivement une base de substitution) $\sigma$ est une application de $\mathcal{X}$ dans $T(\mathcal{F})$ où il n'y a que des nombreuses variables finis non representé avec eux-mêmes,Des Substitution peuvent être étendus comme suit :

$\forall f \in \mathcal{F}_n, \forall t_1, \ldots, t_n \in T'(\mathcal{F})\ \sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), ...; \sigma(t_n))$.

On considére une substitution et son extension à $T(\mathcal{F})$. Des substitutions seront souvent utilisés dans la notation postfixée :$t\sigma$ est le résultat de l'application $\sigma$ au terme $t$.

### Exemple

Soit $F = \{f(,,), g(,), a, b\}$, On considére le terme $t = f(x_1, x_1, x_2)$, et soit le subestitution $\sigma = \{x_1 \leftarrow a, x_2 \leftarrow g(b, b)\}$, et le substitution $\sigma' = \{x_1 \leftarrow x_2, x_2 \leftarrow b\}$ Alors :



## Contextes

Le terme linéaire $C \in T(\mathcal{F})$ est appelé *contexte et l'expression* $C[t_1, ..., t_n]$ *pour* $t_1, ..., t_n \in T(\mathcal{F})$ désigne le terme $T(\mathcal{F})$ obtenu à partir de $C$ en remplaçant la variable $x_i$ par $t_i$ pour chacun $1 \leq i \leq n$ , c'est $C[t_1, ..., t_n] = C\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$. On note par $\mathcal{C}^n(\mathcal{F})$ l'ensemble de contextes plus de $(x_1, \ldots, x_n)$.

On note par $\mathcal{C}(\mathcal{F})$ l'ensemble de contextes contenant une seule variable. Un contexte est trivial si elle est réduite à une variable. On donne le contexte $C \in \mathcal{C}(\mathcal{F})$, On note par $C^0$ c'est le contexte trivial,$C^1$ est egal a $C$ et pour $n > 1$, $C^n = C^{n-1}[C]$ est le contexte dans $\mathcal{C}(\mathcal{F})$.

## 1.6 Automates d'arbres rangés à états finis descendant

Un automate d'arbres rangés à états finis descendant (AAFD) a travers un alphabet rangé $\Sigma$ est un quadruplet $\mathcal{D} = (Q, \Sigma, Q_i, \Delta)$ tel que:

1. $Q$ est un ensemble de symboles appelés *états*,
2. $Q_i \subseteq Q$ est un ensemble d'états initiaux
3. $\Delta \subseteq Q \times \cup_{n \geq 0} \Sigma_n \times Q^n$ est l'ensemble des transitions.

Quelques références [1] utilisent le formalisme des systèmes de règles de réécriture. $\Delta$ est alors définie comme suit :

$$q \to f(q_1, \ldots, q_n) \text{ tel que } f \in \Sigma_n, q_1, \ldots, q_n \in Q$$

On définit la fonction de transition $\delta$ comme suit : $\delta : Q \times T_\Sigma \to T(\Sigma)$ defined for $t = f(t_1, \ldots, t_n)$ as:

$$\delta(q, t) = \{f(\delta(q_1, t_1), \ldots, \delta(q_n, t_n)) \mid \exists (q, f, q_1, \ldots, q_n) \in \Delta\} \tag{1.6.1}$$

Un arbre $t$ est dit*reconnu* par un l'AAFD $\mathcal{D}$ si $\exists q \in Q_i \mid \delta(q, t) = t$.

Soit l'automate suivant :

$\mathcal{D} = (\{q_f, q_g, q_a\}, \{f(,), g(), a\}, \{q_f\}, \{(q_f, f, q_g, q_g), (q_g, g, q_a), (q_a, a)\})$ un AAFD et un arbre $t = f(g(a), g(a))$. On a :

$$\begin{aligned} \delta(q_f, t) &= f(\delta(q_g, g(a)), \delta(q_g, g(a))) \text{ on a } (q_f, f, q_g, q_g) \in \Delta \\ &= f(g(\delta(q_a, a)), g(\delta(q_a, a))) \text{ on a } (q_g, g, q_a) \in \Delta \\ &= f(g(a), g(a)) = t \end{aligned}$$

Alors $t$ est reconnu par $\mathcal{D}$.

Le problème des automates descendent réside dans sa version déterministe. Un AAFD est déterministe si et seulement si :

1. $|Q_i| = 1$
2. $\forall q \in Q, |\{(q, f, q_1, \ldots, q_n) \mid q_1, q_n \in Q, f \in \Sigma_n\}| \leq 1$

En effet, la version déterministe d'un AAFD non déterministe n'existe pas toujours. Voici un exemple de référence [1].

Soit $t = f(a, b)$ et $t' = f(b, a)$ deux arbres définis sur $\Sigma = \{f(,), a, n\}$. L'automate non déterministe $\mathcal{D} = (\{q_f, q_a, q_b\}, \Sigma, \{q_f\}, \{(q_f, f, q_a, q_b), (q_f, f, q_b, q_a), (q_a, a), (q_b, b)\})$ reconnait $t, t'$.

Maintenant supposons que $\mathcal{D}'$ est un automate déterministe qui reconnait $t, t'$. Par définition on a un seul état initial $q_i$ alors $|\delta(q_i, t)| = \delta(q_i, t')| = 1$. Supposons qu'il existe $q', q''$ tel que $\delta(q_i, t) = f(\delta(q', a), \delta(q'', b))$. pour reconnaitre $t'$ il faut que $(q', a) \in \Delta$. Mais comme $mathcalD$ est déterministe alors il faut que $(q', b) \in \Delta$. Ceci est considéré comme une contradiction.

## 1.7 Automates d'arbres rangés à états finis ascendant

Un automate d'arbres rangés à états finis ascendant AAFA à travers un alphabet rangé $\Sigma$ est le quadruplet $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ où:

1. $Q$ est un ensemble de symboles appelés *états*,
2. $Q_f \subseteq Q$ est un ensemble détts finaux,
3. $\Delta \subseteq \bigcup_{n \geq 0} \Sigma_n \times Q^{n+1}$, $n \in N$ est un ensemble fini de transitions.

Un AAFA n'a pas d'états initiaux. En effet, le calcul des arbres se fait à partir de ses feuilles. Pour simplifier la notation, on écrit seulement AAF pour indiquer un AAFA.

(Taille de l'automate) Soit $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ un AAF. La taille d'une transition $\rho = (f, q_1, \ldots, q_n, q), f \in \Sigma,\ q, q_1, \ldots, q_n \in Q$ is $|\rho| = n + 1$. On peut définir la taille d'un automate à partir des tailles de ses transitions.

$$|\mathcal{A}| \ = \ \sum_{\rho \in \Delta} |\rho| \tag{1.7.1}$$

## 1.7.1 Reconnaissance d'un arbre

La première opération sur les $AAF$ est la reconnaissance d'un ou de plusieurs arbres. En effet, Un automate d'arbre est conçu pour lire des arbres et donc décider s'ils sont reconnaissables ou pas .

La fonction de sortie $\delta : T(\Sigma) \to 2^Q$ d'un AAF $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ est définie pour un arbre $t = f(t_1, \ldots, t_n)$ comme:

$$\delta(t) = \{q \mid \exists (f, q_1, \ldots, q_n, q) \in \Delta \text{ and } q_i \in \delta(t_i) \text{ for } i = 1, \ldots, n\} \tag{1.7.2}$$

On dit qu'un arbre $t$ est reconnu par un automate $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ si et seulement si

$$\delta(q) \cap Q_f \neq \emptyset$$

Soit l'automate d'arbres suivant.

$\mathcal{A} = (\{q_a, q_g\}, \{a, g(,)\}, \{q_g\}, \{(a, q_a), (a, q_g), (g, q_a, q_g)\})$ (Dans la figure Figure 1.7.1 les petits nœuds présentent une lettre de $\Sigma$, pour chaque transition, la position des états est présentée par des flèches numérotées).

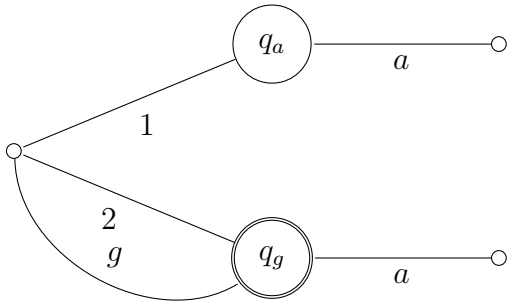



FIGURE 1.7.1 : Exemple d'AAFA

L'arbre $t = g(a, g(a, a))$ est reconnu par l'automate $\mathcal{A}$ (voir les étapes dans la figure Figure 1.7.2) mais l'arbre $t' = g(g(a, a), g(a, a))$ ne l'est pas (voir les étapes dans a figure Figure 1.7.2).

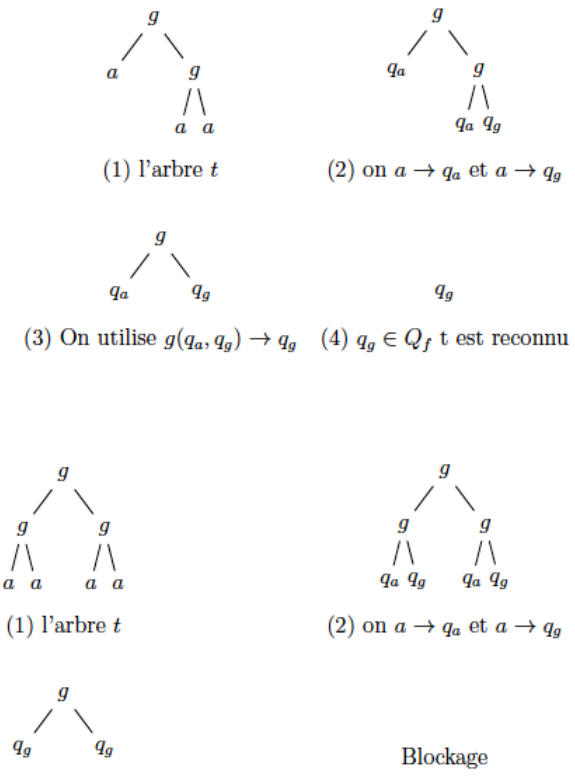



FIGURE 1.7.2 : Reconnaissance d'un arbre par un AAFA

### 1.7.2 Langages reconnaissables

Exactement comme les langages sur les mots, les automates d'arbres définissent une nouvelle classe de langages appelés *langages reconnaissables*. Cette classe est d'une extrême importance car elle permet de reconnaitre un ensemble infini d'arbres par une structure compacte et finie qui est l'automate d'arbre. Mais avant d'introduire cette classe, on définit le calcul des arbres par un automate.

Maintenant on définit ce que c'est qu'un langage reconnaissable .

Un langage $L \in T(\Sigma)$ est reconnaissable si et seulement s'il existe un AAFA $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ tel que $\forall t \in L, \delta(t) \in Q_f$

D'après cette définition, l'ensemble des arbres reconnus par un AAFA forment un langage reconnaissable.

Soit $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ un AAFA, le langage accepté par $\mathcal{A}$ est:

$$L(\mathcal{A}) = \{t \mid \delta(t) \in Q_f\} \tag{1.7.3}$$

Un langage accepté (bas) $L(q)$ d'un état $q \in Q$ est $L(q) = \{t \in T(\Sigma) \mid \delta(t) = q\}$.

Un langage résiduel (haut) $L^{\uparrow}(q)$ d'un état $q \in Q$ est $L^{\uparrow}(q) = \{t(s \leftarrow \Box) \mid t \in T(\Sigma), s \in L(q) \text{ et } \delta(t) \in Q_f\}$.

La classe des langages reconnaissables $Rec(\Sigma) \subseteq 2^{T(\Sigma)}$ est donc définie à travers un alphabet $\Sigma$ comme suit.

Un langage $L$ appartient à $Rec(\Sigma)$ si et seulement s'il est reconnaissable.

#### Automates unaires

Un automate $A =< Q, \Sigma, \delta, I, F >$ est unaire si $|\Sigma| = 1$.

Soit $A$ un automate déterministe unaire. Cet automate possède la structure de transitions.

Cet automate est décomposable de la manière suivante :

– l'ensemble des états Q est décomposé en deux parties : la queue décrite par les états $\tilde{A}_1, ..., \tilde{A}_k$ et la boucle décrite par les états $\tilde{A}_{k+1}, ..., \tilde{A}_n$

– la fonction $\delta$ est définie par $\delta(q) = q + 1 \forall q \in \hat{A}_1, ..., \tilde{A}_{n-1}$, et $\delta(n) = k$

C'est la représentation classique des automates déterministes unaires sous forme

de poële à frire. Ces structures de transitions se générent sans difficulté pour un automate de taille n. La seule connaissance de $k$ la taille de la queue permet de construire la structure de transitions d'un automate déterministe unaire.

### Automates à groupe

Un automate $A =< Q, \Sigma, \delta, I, F >$ est réversible si pour tout état $q \in Q$ et tout symbole $x \in \Sigma$, il existe, au plus, une transition sortant de $q$ et étiquetée par $x$ et il existe au plus une transition entrant en $q$ et étiquetée par $x$. Si de plus l'ensemble des états initiaux de $A$ est réduit à un singleton et que la fonction de transition est complète, alors l'automate est un automate à groupe. La fonction de transition des automates à groupe peut être décrite à l'aide de permutations. Une permutation est utilisée pour chaque symbole de l'alphabet sur lequel est défini l'automate.

## 1.8 Conclusion

Ce chapitre a pour but d'exposer les bases de la théorie des automates à états finis et celles de la théorie des langages, tout en fournissant une bibliographie propre aux poursuites de travaux sur la génération aléatoire et sur l'étude de la concision des automates, sans toutefois négliger d'exposer les corrélations qui existent entre les définitions et entre les théorémes fondamentaux qui sont présentés.

Parmi les définitions originales de ce chapitre, Nous définissons par ailleurs la notion de classe d'automate d'un langage L comme étant l'ensemble fini ou infini des automates qui reconnaissent L. Cette notion de classe d'automate d'un langage se doit d'être mise en paralléle avec celle de classe d'isomorphisme d'un automate.

# Chapitre 2

# Generation aléatoire d'automate d'arbre

Un automate d'arbre est un graphe, les graphe aléatoire ont été introduit pour la premiére fois en 1959 par Erdös et Rényi.Briévement, un graphe aléatoire de taille $n$ est un graphe à n sommets dont on a choisi aléatoirement les arétes.

Le livre fondateur sur les algorithmes de génération de structures combinatoires est le livre de Nijenhuis et Wilf . Dans celui-ci, Nijenhuis et Wilf établissent les bases des algorithmes de génération de structures combina- toires. L'objet de son contenu est de fournir la liste exhaustive des algorithmes de génération d'objets combinatoires simples, "les blocs de base permettant de construire des structures plus complexes". On trouvera dans [4] un exposé plus récent du contenu de ce livre. Le dénombrement de structures combinatoires est un probléme différent que l'on a abordé depuis longtemps. En particulier au travers des probabilités .

Nous présentons, dans un premier temps, les méthodes de génération aléatoire. Nous exposons, dans cette partie, les techniques qui viennent de l'énumération combinatoire, puis suivent les autres techniques utilisées lorsque les premiéres ne peuvent être abordées. Comme les arbres et les graphes sont les objets les plus proches des automates, nous présentons en surface les méthodes utilisées pour les générer aléatoirement. Finalement, nous introduisons un ensemble de résultats obtenus en génération et en énumération des automates à états finis

## 2.1 Introduction

Le modèles aléatoires des Graphes et ces algorithmes sont l'un des sujets centraux de la théorie des graphes et en informatique théorique. Récemment, ils ont également suscité beaucoup d'attention pour la modélisation de réseaux réels tels que l'Internet, le World Wide Web et de réseaux sociaux et biologiques. En particulier, la génération de graphiques aléatoires est un outil important pour la détection de certains motifs dans les réseaux biologiques et pour simuler un large éventail de protocoles réseau sur la topologie d'Internet. Dans ces demandes nous sont habituellement intéressés dans les réseaux avec de simples graphiques sous-jacent, c'est-à-dire lorsque aucune auto-edge boucles ou plusieurs bordures sont autorisés.

Malheureusement, il y a un grand écart entre la théorie et la pratique pour ce problème. Lorsque le degré séquence est non-réguliers, il n'y a pas d'algorithmes efficaces connus pour la génération de graphiques aléatoires. Les algorithmes polynomiaux existants avoir un grand temps d'exécution, ce qui les rend très difficiles à utiliser pour les réseaux du monde réel qui ont des dizaines de milliers de noeuds . D'autre part, il n'y a aucune analyse rigoureuse de l'heuristique simple utilisée dans la pratique. En fait, il est connu que certains de ces heuristiques sont incorrects .

Nous donnons un algorithme de pratique pour l'échantillonnage uniformément à l'ensemble des graphes simples avec compte tenu de degré $d_1$.....$d_n$ Supposons que$\sum_{i=1}^{n} d_i = 2m$ Notre algorithme est comme suit : Démarrer avec un graphique vide et séquentiellement ajouter entre paires de bords non-sommets adjacents. À chaque étape de l'algorithme, la probabilité qu'une arête est ajouté entre deux sommets $i$ et $j$ est proportionnelle à $\hat{d}_i\hat{d}_j$ $(1 - d_i d_j/4m)$ Où $\hat{d}_i$ et $\hat{d}_j$ désignent les autres degrés de sommets $i$ et $j$ .

Nous montrons que pour n'importe quel $e > 0$ Lorsque le degré maximum $d_{max}$ Est de $O(m^{1/4-e})$, Notre algorithme trouve un Asymptotiquement échantillon uniforme avec un temps de fonctionnement de $O(md_{max})$. De plus pour $d = O(n^{1/2-e})$, Notre algorithme Génère une asymptotiquement uniforme d-graphique ordinaire. Nos résultats sont l'amélioration des limites de Kim et vu et Steger et Wormald Pour les graphiques ordinaire et leur extension à de séquences général menant à

un grade.

Afin d'expliquer l'intuition qui sous-tend notre algorithme et la preuve, nous pouvons définir le modèle (appariement) configurationnelle pour les graphes de diplôme général des séquences. Dans le modèle configurationnelle, nous représentons tous les vertex $i$ de degré $d_i$ avec $d_i$ mini-sommets.Une parfaite adéquation entre le mini-sommets correspondrait à un multi-graphique avec le degré Séquence $d_1, d_2, ..., d_n$ . Il est facile de voir que même lorsque les diplômes sont de plus en plus comme $logn$, la probabilité qu'une correspondance parfaite choisie uniformément correspond à un graphe simple est très petit.

Une façon naturelle pour augmenter la probabilité de succès est de générer l'appariement de manière séquentielle en ajoutant les nouvelles paires uniformément au hasard en évitant les choix non valide qui créent des boucles ou plusieurs bordures dans le graphique. Nous allons montrer que pour les non-régulière séquences, l'ci-dessus degré algorithme a une exponentielle de partialité. Plus précisément, la probabilité de générer un graphe G par cet algorithme est proportionnelle à exp $(\sum_{i\sim Gj} d_i d_j/4m)$.C'est principalement parce que le couplage des mini-sommets appartenant à haut degré sommets réduit le nombre de choix valide dans les futures étapes et, par conséquent, crée un biais en faveur des graphiques avec plus de ces bords. Notre algorithme équilibre cet effet en distribuant la partialité terme dans le probabilités des paires.

La partie principale de notre preuve découle très proches des estimations de la probabilité de chaque choix valable dans chaque étape. Cela est nécessaire car l'erreur finale dans la sortie de la distribution de notre algorithme sera le produit de ces erreurs. En utilisant la concentration polynomiale inégalités Kim-Vu est essentielle pour notre analyse. En outre, pour d-graphes ordinaire lorsque $d = \Omega(n^{1/3})$ Nous utilisons une simple martingale exposition où l'inégalité de queue polynomiale d'inégalité semble être non applicable. C'est parce que le rapprochement de la variable aléatoire d'intérêt avec un polynôme de variables aléatoires indépendantes provoque la variance de croître de façon exponentielle.

Nous avons également prouver pour degré général de séquences avec $d_{max} = O(m^{1/4-e})$ La probabilité qu'un procès de notre algorithme produit un graphe simple est très proche de 1. Nous notons que notre preuve ne repose pas sur la et Wormald McKay existants des estimations du nombre de graphiques simples avec

un degré séquence

En fait, notre analyse donne un autre (et peut-être plus simples) preuve de McKay's formule.

Notre algorithme et son analyse donnent plus de précisions sur les modèles de graphes aléatoires moderne comme le modèle configurationnelle ou graphes aléatoires avec un degré attendu séquence . Ces modèles sont considérés comme graphiques aléatoires approximative d'un degré et plus propice à la séquence estimation graphique comme le diamètre ou invariants les tailles de composants connectés. Il est facile de voir que, dans ces modèles, la probabilité d'avoir un bord entre les sommets $i$ et $j$ du graphique est proportionnelle à $d_i d_j$ . Toutefois, on peut utiliser notre analyse ou de McKay's formule à voir que dans un graphe simple aléatoire ce probabilité est proportionnelle à $d_i d_j(1-d_i d_j/2m)$, Des exemples simples montrent que cette différence peut reporter sur d'autres quantités telles que le nombre prévu de voisins d'un vertex qui ont fixé une certaine. D'autre part, nous nous attendons à ce que, en ajoutant le terme de correction et à l'aide de la concentration de ce document, résultat il est possible d'atteindre une meilleure approximation pour ce qui de coincer les théorèmes similaires à peut-être appliquée

Nous avons également utilisé des idées semblables pour générer des graphes aléatoires avec grande circonférence . Ces graphiques sont utilisées pour la conception hautes performances Parity-Check à densité faible (LDPC) codes. Une des méthodes pour construire ces codes est de générer une valeur aléatoire bipartites avec un graphique Degré optimisé séquence . Le rendement de Le code correspondant dans le régime à faible bruit est directement liée à la taille de la circonférence du graphique. Deux des heuristiques communes pour trouver de bons codes sont comme suit : (i) générer de nombreuses copies aléatoire et choisir celui avec le plus haut la circonférence ; (ii) augmenter progressivement le graphique tout en évitant les boucles courtes. Alors que l'ancien implique un ralentissement dans la exponentielle target circonférence, ce dernier induit un biais incontrôlées et systématique dans le graphe de la distribution. En utilisant les mêmes principes, nous pouvons concevoir un algorithme plus efficace que séquentiellement ajoute les bords en évitant de petits cycles. Semblable à notre analyse ici, nous pouvons calculer la partialité de l' algorithme suggéré et distribuer en conséquence pour atteindre la génération uniforme ainsi que faible probabilité d'échec.

## 2.2 Algorithmes et méthodes de génération aléatoire

Nous résumons dans ce qui suit les principes des algorithmes de génération de structures combinatoires. On pourra trouver une illustration des démarches présentées dans le livre de Kreher et Stinson [4] qui montre avec beaucoup de pédagogie ce domaine tout en illustrant les limites des méthodes. à l'exception de la notion de propriété de graphe exposée chez Diestel[2] pour aborder les graphes aléatoires, les méthodes décrites peuvent être retrouvées dans les notes de cours de Denise et Gouyou-Beauchamps . On considére dans la suite que est une classe d'objets de taille c. Lorsqu'une famille posséde des restrictions, on met ces derniéres en indice. Par exemple, $E(n)$ représente la classe des permutations de taille$n$, et $p(n)$ est le nombre de ses éléments

### 2.2.1 Génération, énumération

Dans le cadre de la génération d'objets combinatoires on cherche à générer les sous-ensembles, les permutations, les partitions d'un ensemble, ou encore les arbres de taille donnée. Il existe trois types d'algorithme d'énumération et de génération de structures combinatoires : les algorithmes de génération aléatoire qui permettent de construire les éléments d'une classe d'objets avec une distribution uniforme, les algorithmes d'énumération qui permettent de lister l'ensemble des objets d'une classe selon un ordre, et les algorithmes de génération qui per- mettent de construire les objets 'a partir d'un indice dans la liste. Pour le premier type d'algorithme, un exemple type venant du folklore est l'algorithme de génération aléatoire des permutations. Cet algorithme est donné ci-dessous. L'essence de ce type d'algorithme est de conserver la propriété algo- rithmique de terminaison.

**Algorithme 2.1** Genération aléatoire des permutation

Ranper(n)
> Entrée : n entier positif
> Sortie : P un tableau contenant une permutation aléatoire de taille n
> Auxiliaire : P tableau ; i,l : indices 1
Début
2 pour i ← 1 à n faire
3 P[i] ← i
4 fin pour
5 pour i ← 1 à n faire
6 $l \leftarrow i + rand(A0, n - i + 1A)$
7 echange$(P[i], P[l])$
8 fin pour
9 Retourner P
10 Fin

Considérons une classe d'objets $O = o_1, o_2, ..., o_{c(n)}$ de taille $c(n)$, tel qu'il existe une relation d'ordre $<$ sur ses objets. L'existence de la relation d'ordre $>$ permet d'associer à la classe d'objet la liste de ses objets ordonnés par la relation $O = (o_1, o_2, ..., o_{c(n)})$ avec $o_i < o_{i+1}$,$\forall i \in A_1, c(n) - A_1$. L'ordre total sur les objets de permet de construire deux types d'algorithme : les fonctions d'énumération Suivant et Precedent, et les fonctions de génération Rang. Les fonctions Suivant et Precedent permettent d'obtenir à partir d'un objet o de le précédent ou le suivant de o dans la liste $L$. On considére évidemment que Suivant$(o_{c(n)} = o_1$ et que Précedent $o_1 = o_{c(n)}$, en d'autres termes le suivant (resp. précédant) du dernier (resp. premier) élément de $L$ est le premier (resp. dernier) élément de cette liste.Les fonctions Rang , et DeRang permettent à partir d'un objet $o$ d'obtenir respectivement la position de $o$ dans $L$, ou d'obtenir l'objet de rang $k, (k \in A_1, c(n)A_1)$ de la liste $L$. Par abus de notation, nous nommerons de maniére identique les fonctions et les algorithmes définissant ces procédés d'énumération ou de génération.

Ces fonctions permettent évidemment de produire des algorithmes de génération aléatoire. Dans le cas de Rang et DeRang on obtient directement un algorithme de génération des objets de à partir d'un générateur de nombres aléatoires et de la méthode DeRang. Remarquons que ces deux fonctions permettent de construire les deux algorithmes Suivant et Précedent tout en conservant des algorithmes de complexité similaire.On a en effet pour tout objet o d'une classe d'objets ,

$Suivant(o) = DeRang(Rang(o) + 1)$ et $Precedent(o) = DeRang(Rang(o) - 1)$. L'inverse ici n'est pas trivial.On ne peut obtenir directement d'algorithme Rang et DeRang à partir des algorithmes Suivant et Precedent sans accroıtre la complexité de l'algorithme de $c(n)$. En effet, une définition de Rang à l'aide de la fonction Suivant est :

$Rang(o) = minx \mid Suivant(o_1) = o$

On peut toutefois à l'aide d'un algorithme Suivant ou Precedent consituer la liste compléte des objets d'une classe d'objets. On obtient ainsi un algorithme de génération aléatoire dont la complexité peut être exponentielle en espace : il faut sauvegarder tous les objets ; mais dont la complexité en temps est instantanée : il suffit de tirer aléatoirement un terme de la liste. On s'intéresse dans le cadre de la combinatoire énumérative à énumérer selon des ordres particuliers les objets combinatoires. On utilise naturellement l'ordre lexicographique, et les ordres avec changements minimaux au travers des codes de Gray.[9]

### 2.2.2 Dénombrement

Le dénombrement consiste à chercher le nombre de structures différentes d'un type particulier. De part le fait qu'un algorithme de génération ou d'énumération d'une classe d'objets permet de construire tous les objets de cette classe, celui- ci permet toujours d'obtenir une méthode de dénombrement de cette classe. L'inverse n'est évidemment pas possible. La formule du binôme de Newton, par exemple, donne le nombre de sous-ensembles de taille k d'un ensemble de taille n. Cependant, cette formule, comme elle n'implique aucun ordre, ne propose pas d'algorithme de génération.

Les problémes de dénombrement de structures sont bien différents des problémes de génération. Les problémes de dénombrement sont des problémes mathématiques alors que les problémes de génération sont des problémes algorithmiques. Il est agréable dans le cas du dénombrement de disposer de formules fermées ou d'approximations, plutôt que de formules récursives, alors que pour des objectifs algorithmiques des formules récursives peuvent suffire malgré les risques d'explosion combinatoire.[9]

### 2.2.3 Objets étiquetés, objets non étiquetés

Les listes, les ensembles et les multi-ensembles sont les structures simples utilisées en combinatoire. Les arbres et les graphes sont des structures composées d'atomes : les nœuds des arbres, les sommets des graphes. On dit de ces structures qu'elles sont des structures atomiques. ces nœuds ou ces sommets peuvent être étiquetés par des entiers. La création d'un étiquetage sur une structure atomique fait apparaıtre des problémes d'isomorphisme de structures : on peut étiqueter une même structure de différentes maniéres .Dans le cas des algorithmes de génération et d'énumération, un seul candidat de chaque classe d'isomorphisme de la classe d'objets que l'on étudie doit être construit. A l'opposé, un algorithme de génération aléatoire autorise que plusieurs candidats d'une même classe d'isomorphisme soient générés, sous la condition que le nombre de candidats reste identique pour chacune des classes.Chaque classe d'isomorphisme est ainsi générée avec une distribution uniforme. C'est le cas, par exemple, des algorithmes de génération aléatoire des arbres binaires non étiquetés aux arcs.

Considérons un graphe $G =< V, E >$, ou un automate $M = (Q, \Sigma, F, P)$ , un arbre binaire, ou encore la représentation d'une permutation sous forme de graphes cycliques. D'un point de vue combinatoire, la taille de ces objets est le nombre de leurs atomes (leur nombre d'états, de sommets d'éléments ou de termes). L'étiquetage des atomes d'une structure composée est dit canonique si l'ensemble des étiquettes de ces atomes forme un segment initial de N. On dit aussi que cette structure est bien étiquetée. Par extension, une fonction f permettant d'étiqueter les atomes de ces structures est canonique sous la condition que l'étiquetage qu'elle engendre l'est.

Lorsque l'on aborde une classe d'objets composés on peut chercher à construire cette classe à partir d'une grammaire qui compose des objets de base de la combinatoire (permutation, partition, composition ...). Ce probléme de construction d'objets composés à partir de blocs de base à été résolu au travers des séries génératrices. Il existe une littérature importante sur les séries génératrices, citons en particulier le livre de Wilf , et les rapports de Flageolet et Sedje- wick . Les séries génératrices proposent une méthode systématique de dénombrement et de génération de classes d'objets étiquetés, lorsque l'on arrive à décrire ces objets

sous forme d'objets combinatoires simples. Flageolet et al. exposent dans [8] la méthode boustrophédonique, une méthode de génération aléatoire , qui peut être utilisée lorsque l'on dispose d'une méthode de dénombrement d'une classe par série génératrice. Les classes pouvant être décrites 'a l'aide de séries génératrices sont dites constructibles. . En particulier pour les automates déterministes , pour lesquels nous donnons une méthode combinatoire de génération sans établir la constructibilité de ces objets sous forme d'objets simples.

Le probléme de l'isomorphisme de structures peut être réglé à l'aide de fonctions d'étiquetage canonique à un isomorphisme prés. Une telle fonction étiquette les objets d'une classe d'isomorphisme de la même maniére. Chaque classe d'isomorphisme est représentée par un unique objet étiqueté. On peut cependant, par exemple dans le cas des arbres, trouver une fonction d'étiquetage des sommets, telle que les classes d'isomorphisme possédent un même nombre d'objets étiquetés par cette fonction. Dans le cas où la fonction d'étiquetage que l'on propose est canonique à un isomorphisme prés, le test d'isomorphisme des structures est immédiat.

### 2.2.4 Notion de graphe aléatoire

Les graphes orientés font partie des objets combinatoires les plus proches des automates non déterministes. Les graphes étiquetés et les automates se distinguent aisément. Il existe cependant des confusions classiques lorsque l'on parle de graphes étiquetés. Lorsque l'on parle de graphes étiquetés, on parle du triplet $G_{etiquete} =< V, \Sigma, E >$dont les arcs sont étiquetés par les symboles de l'alphabet $\Sigma$. Lorsque $G$ posséde un étiquetage canonique, on dira de $G$ que c'est un graphe étiqueté et étiqueté canoniquement aux sommets. Il existe des méthodes de génération aléatoire de graphes non orientés et non étiquetés possédant un nombre donné de sommets et d'arêtes . Cependant, de nombreuses familles de graphes ne peuvent être énumérées en temps et en espace raisonnable. On peut, par exemple, dans le cas des graphes planaires, énumérer l'ensemble des graphes et constituer la liste de ces graphes planaires. On obtient, par la suite, un graphe aléatoire en temps constant, mais l'espace nécessaire au stockage des graphes est exponentiel. Pour parvenir à obtenir une génération quasi uniforme en temps et en espace polynomial,

on simule une chaıne de Markov sur la classe que l'on étudie .

Ces notions d'algorithmes approchés nous permettent de marquer la distance qui existe entre les algorithmes combinatoires et les algorithmes probabilistes. Un algorithme probabiliste n'est pas, par définition, un algorithme puisqu'il se termine avec une probabilité donnée. De plus, il propose une méthode de génération aléatoire avec une uniformité approchée. Il existe donc une probabilité plus ou moins importante que cet algorithme ne se termine jamais. La notion d'algorithme probabiliste, ainsi que l'étude de sa complexité n à de sens que sous réserve de la connaissance de sa probabilité de terminaison comme dans le cas des algorithmes avec rejets.

Intuitivement, on est capable de générer un graphe étiqueté aux sommets $G =< V, E >$comme suit. Pour tout couple non orienté $e \in V \times V$ , on décide par certaines expérimentations si oui ou non e est une arête de $E$ ; ces expérimentations sont effectuées indépendamment, et pour chacune la probabilité de succés c'est- à-dire d'accepter $e$ comme une arête de $E$ est égale à un nombre fixé $p \in [0, 1]$. Cette probabilité p pouvant être une constante ou une fonction $p(n)$ du cardinal de $V$.

Pour retrouver nos notations, mais par abus de langage, nous dirons qu'un graphe $G$ est un graphe de $(n, p(n))$, s'il est de taille $n$ et qu'il a été généré avec la fonction de probabilité $p(n)$.

### 2.2.5 Méthode de génération aléatoire des arbres binaires

La génération d'arbres binaires est un probléme résolu de longue date. L'intérêt, aujourd'hui, des études se porte sur la complexité spatiale des algorithmes utilisés pour générer aléatoirement les arbres binaires. La discussion que nous présentons se base essentiellement sur les travaux de Mäkinen et al. qui comparent les algorithmes de génération aléatoire d'arbres binaires. On trouvera dans ces papiers les références permettant d'étudier les algorithmes de génération aléatoire et d'énumération des arbres binaires. Arnold et Sleep proposent une méthode incrémentale de construction d'arbres binaires à partir de mots de Dyck. Leur méthode ne se différencie que par le codage utilisé pour représenter les arbres binaires. Leur raisonnement se base sur la représentation des familles de Catalan sous forme de chemins de Dyck, pour lesquels on peut utiliser le principe de la réflection des

lignes polygonales . Cependant leur construction incrémentale peut se réaliser sur n'importe quelle autre famille de Catalan, en particulier sur les arbres, ou les mots de Dyck. Dans le cas des arbres, la distribution est uniforme si les arêtes sont étiquetées par les symboles d'un alphabet. Dans ce cas les étiquetages lexicographique et hiérarchique sont canoniques à un isomorphisme prés. A l'opposé, si ces arêtes ne sont pas étiquetées, les algorithmes peuvent générer deux arbres de chaque classe d'isomorphisme, la distribution reste uniforme, cependant les étiquetages lexicographique ou hiérarchique ne sont pas canoniques à un isomorphisme prés.

On désire générer aléatoirement un mot de Dyck m de longueur $2n$, ou un chemin de Dyck c de longueur $2n$, ou un arbre binaire ABR d'ordre $n$, à l'aide d'une méthode incrémentale. A une étape de cette construction, on dispose soit d'un préfixe w de longueur $r$ du mot Dyck que l'on génére possédant $k$ parenthéses ouvrantes qui ne sont pas fermées par une parenthése fermante, soit d'un arbre de taille $r$ possédant k possibilités d'ajout de nœuds ou de feuilles. Soit, enfin, d'un mot de Dyck de longueur r, et dont la hauteur courante est égale à $r$. Dans le cas des méthodes incrémentales, on observe que l'on a une équivalence entre l'ajout d'une parenthése fermante (resp. d'une parenthése ouvrante) à un mot de Dyck, l'ajout d'une arête de longueur 1 de pente croissante (resp. de pente décroissante) à une ligne polygonale, ou enfin l'ajout d'un nœud (resp. d'une feuille) à un arbre binaire.

Arnold et Sleep utilisent cette équivalence de construction qui existe entre certaines familles de Catalan. Ils calculent, en se basant sur les chemins de Dyck, la probabilité de construire, à chaque étape, une arête croissante ou une arête décroissante d'un chemin de Dyck de longueur $2n$. Ils en déduisent une méthode de construction des arbres binaires.

Une grande majorité d'algorithmes permettant de construire des arbres binaires sont conçus sur ce même calcul de probabilité : on utilise un codage des arbres binaires sous forme de mots que l'on génére aléatoirement. Les qualités de la méthode de génération sont basées d'une part sur l'espace mémoire nécessaire à la réalisation des calculs de probabilité, et d'autre part sur l'uniformité de la distribution obtenue due aux calculs d'arrondis des probabilités. Certaines méthodes different des autres. Par exemple, Atkinson et Stack, qui, plutôt que de construire

incrémentalement une séquence aléatoire codant un arbre binaire, décrivent un procédé de type diviser pour conquérir qui permet de construire un arbre en espace linéaire, et en utilisant une méthode ingénieuse de construction de mots de Dyck. Leur méthode posséde de plus l'avantage de ne pas utiliser de probabilité et ainsi de conserver une distribution uniforme. Cependant, les algorithmes de génération aléatoire d'arbres restent linéaires et conservent une complexité spatiale polynomiale.[9]

### 2.2.6 Génération aléatoire et dénombrement d'automates à état finis

Domaratzki, Kisman et Shallit présentent dans [6] la meilleure référence sur le dénombrement des automates à états finis possédant $n$ états. Les études données ne proposent pas de familles d'automates qui peuvent être aisément générées. La génération aléatoire des automates a été principalement étudiée par Nicaud et Harrison. Au travers de l'étude du comportement en moyenne des automates à états finis, Nicaud étudie des familles particuliéres d'automates à états finis. L'étude de leur comportement l'améne à décrire des méthodes de génération aléatoire. Nous décrivons, dans ce qui suit, l'ensemble de ces méthodes à l'exception de la méthode de génération aléatoire des automates déterministes, Nicaud étudie deux familles d'automates qui se décomposent facilement sous forme d'objets combinatoires simples : les automates déterministes unaires et les automates à groupe.

## 2.3 Les travaux connexes

Générer les graphes aléatoires avec un degré séquence a été longuement étudié. . Probablement la plus étudiée et la plus efficaces à ce problème a été le Monte Carlo par chaîne de Markov (MCMC) méthode. . et Sinclair a donné une Jerrum entièrement polynôme randomisés schéma d'approximation pour générer des graphiques avec diplôme " stable " séquence . Dans un papier différent et à l'aide de McKay's estimation , elles ont introduit une autre chaîne de Markov avec un meilleur temps d'exécution de $O(m^2n^2logm)$ Pour degré des séquences avec un maximum Degré de o($m^{1/4}$) Pour générer de façon aléatoire bipartites graphiques avec un degré

général de distribution, Bezàkovà et al. L'amélioration la plus connue en cours d'exécution temps et atteint le temps d'exécution de$O(n^4m^3d_{max}log^5(2n))$.

Le modèle configurationnelle est également étudié pour les graphiques ordinaire et, plus récemment, pour diplôme général des séquences . McKay et Wormald considéré comme une version modifiée de ce modèle dans lequel les boucles et plusieurs bordures peuvent être retirés par une séquence de commutateurs. Mais afin d'obtenir des échantillons uniforme à la fin on s'est servi d'un régime d'acceptation/de rejet au sein de la phase de commutation. Si $M = \sum_{i=1}^{n} d_i$ et $\mathrm{M}_2 = \sum_{i=1}^{n} d_i(d_i - 1)$ puis ils ont montré pour $d_{max}^3 = O(M^2/M_2)$ et $d_{max}^3 = o(M + M_2)$ La moyenne d'exécution de l'algorithme est $O(M + M_2^2)$ Cela conduit à $O(n^2d^4)$ Temps de marche moyenne pour D-graphes ordinaire avec $d = o(n^{1/3})$.

Les algorithmes séquentiels et en particulier l'importance de l'échantillonnage séquentiel (SIS) méthodes ont été largement utilisées en pratique pour ce et d'autres problèmes similaires . Chen et al. a utilisé la méthode SIS pour générer bipartites graphiques avec un degré l'ordre. Plus tard Diaconis Blitzstein a utilisé une approche similaire pour générer des graphiques de générale. Bezàkovà et al ont étudié l'algorithme de Chen et coll et a montré que leur procédure peut générer des tables de contingence avec probabilité loin de distribution uniforme pour certaines sommes de ligne et de colonne. Toutefois la simplicité de ces algorithmes et leurs excellentes performances dans certains cas, suggère une étude plus approfondie de la méthode SIS est nécessaire.

En fait, Leurs algorithme est inspiré par l'algorithme de Diaconis Blitzstein qui utilise l'importance de l'échantillonnage séquentiel. Ils proposent une petite modification de leur algorithme qui améliore son comportement asymptotique de façon substantielle.[7]

Supposons que nous sommes donné une séquence de n des entiers non négatifs $d_1, d_2, ..., d_n avec \sum_{i=1}^{n} d_i = 2m$ Et un ensemble de sommets $V = \{v_1, v_2, ..., v_n\}$Assumer la séquence de degrés a étant donné $d_1, d_2, ..., d_n$Est graphique. Ce n'est qu'il existe au moins un graphe simple avec ces degrés. Ils suggèrent l'algorithme suivant pour Échantillonner un élément de l'ensemble $\lambda(\bar{d})$ de toutes étiqueté de graphiques simples $G$avec $V(G) = V$ et séquence de degré $\bar{d} = (d_1, d_2, ..., d_n)$ [7]

**Algorithm 2.2** Générateur aléatoires des graphes avec un degré séquence

1. Commencer avec un ensemble vide $E$ de bords. Laissé également $\hat{d} = (\hat{d}_1, ..., \hat{d}_n)$ Être un n-uplet d'entiers et de l'initialiser par $\hat{d} = \bar{d}$.
2. Choisissez deux sommets $v_i, v_j \in V$ Avec probabilité proportionnelle à $\hat{d}_i d_j (1 - \hat{} \frac{d_i d_j}{4m})$ Parmi toutes les paires $i, j$ avec$i \neq j$ et $(v_i, v_j) \notin E$ . Ajouter $(v_i, v_j)$ à $E$ Et réduire chacun de $\hat{d}_i$, $\hat{d}_j$par 1.
3. Répéter l'étape (2) jusqu'à ce plus de chant peut être ajouté à E.
4. Si $|E| < 2m$ Rapport sur l'échec et redémarrez à partir de l'étape (1), sinon la sortie $G = (V, E)$.

## 2.4 Génération aléatoire d' automates d'arbres à états finis non déterministe

On a un seul travail dans ce domaine qui est fait par Andreas Maletti et al. et dans la suite en présentons leur travail sur les automates d'arbres à états finis non-déterministe :

### 2.4.1 Introduction

Algorithmes pour (non déterministe)les automates d'arbre à états finis (FTA) sont souvent mises à l'épreuve sur le hasard FTA, dans laquelle toutes les transitions internes sont équiprobables. Le run-time Résultats obtenus de cette manière sont généralement trop optimiste que la plupart de ces accords de libre-échange aléatoire généré sont négligeables en ce sens que le nombre de membres de l'équivalent d'un FTA déterministe minimale est extrêmement faible. Il est démontré que non triviaux sont obtenus seulement FTA aléatoire pour une bande étroite de probabilités de transition. En outre, une analyse analytique des rendements une formule de rapprocher la probabilité de transition que les rendements les plus aléatoire complexe, qui devrait être FTA utilisés dans des expériences.

Les automates d'arbre à états finis non déterministe (FTA) jouent un rôle important dans plusieurs domaines du traitement des langues naturelles. Par exemple,

l'analyseur de Berkeley utilise un (pondéré) FTA comme do Syntaxe axées sur la traduction automatique statistique. Trousses à outils pour ale permettent aux utilisateurs d'exécuter facilement des expériences. Cependant, les algorithmes comme determinization ne peuvent généralement pas être testé sur des exemples réels (tout comme l'ALE de l'analyseur de Berkeley) en raison de leur complexité. Dans de tels cas, les entrées sont souvent aléatoire ale, qui sont généralement créés par les densités de fixation, qui sont les probabilités qu'une éventuelle transition (d'un certain groupe) est en effet une transition de la générés.[10]

### 2.4.2 L'approche de Andreas Maletti et al

Dans la première, Il décrive comment Ils génèrent FTA. Il suit de près la génération aléatoire décrites dans [11], augmentée par les paramètres Densité $d_2$ et $d_0$ , qui est similaire à la configuration de [5].Les deux méthodes [5, 11][5, 11]] sont discutées dans [12], où ils sont appliqués aux automates de chaîne à états finis .Pour un événement $E$, soit $\pi(E)$ être la probabilité de $E$. pour conserver la présentation simple, nous supposons que le rang des symboles d'entrée s'alphabet est binaire (c'est-à-dire $\Sigma = \Sigma_2 \cup \Sigma_0$), Notez que toutes les langues d'arbres régulière peut être codé à l'aide d'un classement binaire alphabet. Afin de générer de façon aléatoire un FTA $M = (Q, \Sigma, F, P)$.Avec $n =\mid Q \mid$ membres et densités de transition nulle et binaires $d_2$ et $d_0$ , chaque transition (incl. l'état cible) est une variable aléatoire et chaque État est une variable aléatoire représentant s'il est définitif ou non. Plus précisément, nous utilisons l'approche suivante :

- $Q = \{1, ...., n\}$,
- $\pi(q \in F) = \frac{1}{2}$ Pour tous $q \in Q$ (c'est-à-dire, pour chaque état $q$ la probabilité qu'elle soit finale est $\frac{1}{2}$),
- $\pi(\alpha \to q \in P) = d_0$ pour tous les arités $\alpha \in \Sigma_0$et $q \in Q$, et
- $\pi(\sigma(q_1, q_2) \to q \in P) = d_2$ pour tous les symbols binaire $\sigma \in \Sigma_2$ et tous les états $q_1, q_2, q \in Q$.

Si les telles créé n'est pas de garniture FTA, alors nous recommencer et générer une nouvelle FTA. Ainsi, tous nos accords de libre-échange générée de façon aléatoire ont effectivement$n$ utile états.[10]

### 2.4.3 Analyse analytique

Il présentent une analyse analytique et densités de calcul, pour lesquelles Il attend le FTA devant être générée de façon aléatoire non-trivial. Plus précisément, Ils estiment pour lesquels des densités de la minimisation (et ultérieures determinization) renvoie le plus grand FTA canonique.Il est connu dans la génération aléatoire de chaîne à états finis automates que le plus grand déterministe determinization des automates sont obtenus pendant si chaque État $q \in Q$ se produit avec une probabilité$\frac{1}{2}$ dans la cible de transition de la transition, dans laquelle la source membres sont choisis au hasard de façon uniforme. Cette observation a été confirmé empiriquement plusieurs fois par des groupes de recherche indépendants [11, 5, 12]. En outre, tous les États ciblés de transition sont équiprobables pour l'entrée des États qui sont tirées au hasard de façon uniforme. Cette dernière observation supporte l' optimalité réclamation par les arguments de la théorie de l'information parce que si tous les États visés par une transition sont équiprobables, puis l'Entropie de la transition est maximale. Bien que ces faits appuient notre hypothèse (et les conclusions subséquentes), Ils comptent également sur une évaluation empirique dans la section suivante pour le valider.

TABLE 2.1 : Densité attendue $d_2$et de densité observée $d'_2$

| $n$ | $d_2$ | $d'_2$ | Conf.interval | $n$ | $d_2$ | $d'_2$ | Conf.interval |
|---|---|---|---|---|---|---|---|
| 2 | .6364 | .6264 | [.5769,.6804] | 8 | .0431 | .0408 | [.0317,.0526] |
| 3 | .2965 | .2570 | [.2091,.3159] | 9 | .0341 | .0342 | [.0272,.0430] |
| 4 | .1696 | .1334 | [.1024,.1737] | 10 | .0276 | .0282 | [.0231,.0343] |
| 5 | .1094 | .0855 | [.0642,.1138] | 11 | .0228 | .0251 | [.0208,.0303] |
| 6 | .0763 | .0635 | [.0475,.0848] | 12 | .0192 | .0212 | [.0182,.0248] |
| 7 | .0562 | .0501 | [.0380,.0662] | 13 | .0164 | .0189 | [.0162,.0219] |

Le déterminisation construit l'ensemble de l'État $\mathscr{P}(Q)$.Soit$\mathscr{P}(M) = (\mathscr{P}(Q), \Sigma, F', P')$ étre le deterministe FTA donné l'aléatoire FTA $M$ avec $n =\mid Q \mid$Selon leur intuition, la probabilité qu'un état$Q' \in \mathscr{P}(Q)$ est la cible de transition d'une transition donnée $\sigma(Q_1, Q_2)$, Où $Q_1$et $Q_2$ sont uniformément sélectionnés au hasard à partir de $\mathscr{P}(Q)$, devrait être$2^{-n}$[c-a-d$\pi(\sigma(Q_1, Q_2) \to Q' \in P')$].Ainsi, chaque État donné $q \in Q$ est dans l'État successeur (réel) $Q''$ de la transition donné $\sigma(Q_1, Q_2)$ avec la probalité $\frac{1}{2}$[c-a-d , $\pi(q \in Q'') = \frac{1}{2}$].Etant donné$\sigma \in \Sigma_2$et $q \in Q$, Soit

$\pi_{\sigma,q} = \pi(q \in Q'')$, Où $Q_1$et $Q_2$ sont uniformément sélectionnés au hasard à partir de $\mathscr{P}(Q)$et $Q'' = \bar{\sigma}(Q_1, Q_2)$, et aussi $\alpha \in \Sigma_0$et $q \in Q$, En fait $\pi_{\alpha,q} = \pi(q \in \bar{\alpha})$ donc on a $\pi_{\sigma,q} = \pi_{\sigma',q'}$ pour tous $\sigma, \sigma' \in \Sigma_2$et $q, q' \in Q$ parce que dans leur modèle de génération tout $\sigma$-transitions en $M$ avec $\sigma \in \Sigma_2$sont équiprobables.Ainsi, il suffit d'écrire $\pi_2$ au lieu de $\pi_{\sigma,q}$.La même propriété contient des symboles nulaires, donc nous écrivons $\pi_0$ pour $\pi_{\alpha,q}$. Soit $n$ le nombre d'états de l'original FTA au hasard, et que $d_2$et $d_0$ Soit le densités de transition de celui-ci. Pour$n = 1$ l'intuition présentée ne peut être satisfaite.

THEOREM. Soit $n > 1$. Si $d_2 = 4(1 - \sqrt[n^2]{5})$et $d_0 = \frac{1}{2}$ alors $\pi_2 = \pi_0 = \frac{1}{2}$.

LA PREUVE. Nous commençons avec $\pi_0$. Soit $\alpha \in \Sigma_0$et $q \in Q$. Alors $\pi(q \in \bar{\alpha}) = \pi(\alpha \to q \in P) = d_0 = \frac{1}{2}$comme demandé

Pour $\pi_2$soit$Q_1, Q_2 \in \mathscr{P}(Q)$ être choisi uniformément au hasard, $\sigma \in \Sigma_2$ et $q \in Q$. Alors

$\pi(q \in \bar{\alpha}(Q_1, Q_2)) = 1 - \pi(q \notin \bar{\alpha}(Q_1, Q_2))$

$\pi(q \in \bar{\alpha}(Q_1, Q_2)) = 1 - \prod(1 - \pi(q_1 \in Q_1)\pi(q_2 \in Q_2)\pi(\sigma(q_1, q_2) \to q \in P))$

$\pi(q \in \bar{\alpha}(Q_1, Q_2)) = 1 - (1 - \frac{d_2}{4})^{n^2} = 1 - (1 - 1 + \sqrt[n^2]{5})^{n^2} = 1 - (\sqrt[n^2]{5})^{n^2} = \frac{1}{2}$

Le tableau 2.1 Presente certain valeurs $d_2$ calculé avec la theoreme pour les tailles $n \in \{2, \ldots, 13\}$.[10]

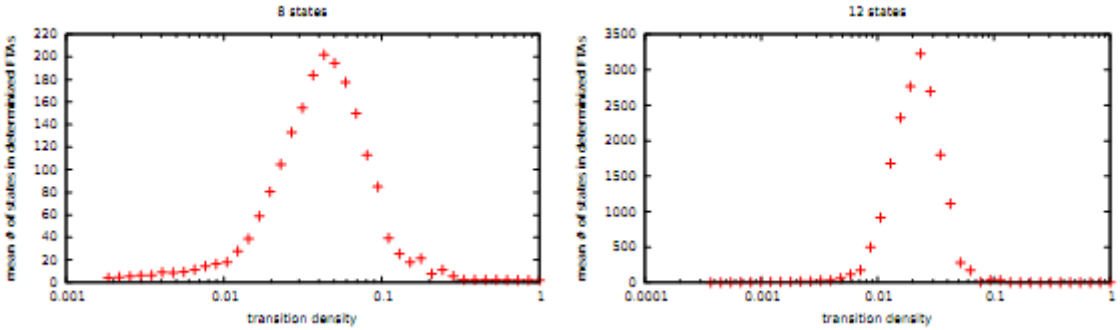



FIGURE 2.4.1 : Graphiques traçant la taille moyenne de l'determinization determinized FTA obtenue par plus de la densité pour FTA de taille 8 et 12. Réduction réduit la taille moyenne, mais ne déplace pas le pic.

### 2.4.4 Analyse empirique

Nous voulons confirmer que la densité calculée représente en effet la plus difficile des instances pour les contrôles aléatoires de FTA . construite Nous utilisons deux paramètres :

(A) $\Sigma = \{\alpha^{(0)}, \sigma^{(2)}\}$et

(B) $\Sigma = \{\alpha^{(0)}, \sigma^{(2)}, \delta^{(2)}\}$

Pour les deux paramètres (A) et (B) et la variable densité $d_2 = e^{\frac{x.log.D_n}{20}}$ et $d_0 = \frac{1}{2}$, Où $D_n = 4(1 - \sqrt[n^2]{5})$, Pour tous $0 \leq x \leq 40$ et les tailles $2 \leq n \leq 13$, Nous avons généré au moins 40 trim FTA. Le ratio de la garniture FTA de diverses densités et état tailles définies peuvent être trouvés dans le tableau 2.2, En règle générale, les grandes état fixe et des densités plus élevées augmente la chance d'obtenir un FTA de garniture. Le choix des densités nous fait garantit que suffisamment de nombreux points de données (en étapes également espacés sur une échelle logarithmique) existe sur les deux côtés de la densité qui est prédite pour générer des instances les plus difficiles. Nous allons discuter de la raison pour laquelle nous avons privilégié l' échelle logarithmique sur une échelle linéaire dans le paragraphe suivant.Ces FTA ont par la suite été, réduite et determinized les tailles de les FTA's canonique ont été enregistrés

Nos expériences confirment les prédictions théoriques. Un pic dans la taille moyenne de determinée FTA peuvent être observées lorsqu'il est prédit. Exemplaire pour le paramétres (A) et $n \in \{8, 12\}$ sont Présenté à la figure 2.5.1 sur une échelle logarithmique.Depuis ces graphiques semblent être des distributions log-normales. Nous avons calculé la moyenne et la variance de ces distributions log-normales, l'interprétation de la densité comme la variable aléatoire et le nombre d'états comme la fréquence. Les statistiques pertinentes sont présentées dans le tableau 2.1 pour le paramètre (A). Toutes les densités prédites sont dans l'intervalle de confiance pour le niveau de confiance $p > 95$ (c'est-à-dire la densité prédite se trouve à 1.96, $\sigma$ distance observée de la moyenne, où $\sigma$ est l' écart type).

Il est intéressant de noter que l'emplacement du pic n'est pas changer les paramètres (A) et (B). Cela signifie que la taille de l'alphabet de symboles binaires n'influence pas la dureté du problème, La seule différence est la taille de l'résultant determinized FTA, Qui est généralement plus grande dans le paramètre (B).

Aussi, la minimisation ne modifie pas l'emplacement du pic, ce qui signifie que les instances dur pour la determinisation sont aussi durement les instances de minimisation. En outre, nous avons également effectué des expériences qui a confirmé le résultat semblable de [12] pour chaîne à états finis utilise la chaîne d'automates automates toolkit.

TABLE 2.2 : Ratio de garniture FTA dans une série d'accords de libre-échange généré aléatoirement paramétré par densité $d_2$ (rangées) et numéro $n$ (colonnes) des Etats. Les inscriptions en blanc indiquent qu'aucun de ces expériences a été effectuée.

| $d_2 \backslash n$ | 2 | 4 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|
| 01 | | | | 7% | 11% | 18% | 27% | 38% | 50% | 64% |
| 05 | | | 68% | 82% | 92% | 98% | 99% | 100% | 100% | 100% |
| 10 | | 54% | 90% | 96% | 99% | 99% | 100% | 100% | 100% | 100% |
| 25 | | 83% | 97% | 98% | 99% | 100% | 100% | 100% | 100% | 100% |
| 50 | 47% | 88% | 96% | 98% | 100% | 100% | 100% | 100% | 100% | 100% |

## 2.5 Conclusion

Trois mots permettent de décrire les problémes de la génération aléatoire d'objets combinatoires : "génération", "énumération" et "dénombrement". Le début de la première section de ce chapitre est construit autour de la formalisation de ces trois notions : construire, construire selon un ordre et dénombrer, ainsi que sur les algorithmes de génération aléatoire que l'on peut en extraire. Les problémes d'isomorphismes de structures interviennent fréquemment lors de la création de mécanismes de génération. Les standards de la génération aléatoire d'objets combinatoires peuvent, dans de nombreux cas, ne pas être satisfaits à cause de l'inexistance de propriétés de composition structurels des objets que l'on désir générer. On utilise alors des méthodes approchées. On trouve parmi ces méthodes, les méthodes de génération avec rejets et les méthode simulant des chaınes de Markov, et on a parlé aussi de l'approche de Andreas Maletti et al et on a vu que ils ne sont pas implimenté leurs travaille dans la section suivante on va illustrer notre travaille et implimenté notre algorithme.

# Chapitre 3

# Lazy générateur aléatoire

Dans ce chapitre nous présentons un générateur aléatoire d'automate d'arbre fini ordonné rangé et non-déterminisme qu'on va appeler « Lazy random génération », ce générateur nous permet de tester la performance des algorithmes et à illustrer les résultats théoriques. Une bonne connaissance de l'espace des structures que nous voulons générer est indispensable à la conception d'un algorithme de génération. Il y a quelques études de la génération aléatoire équiprobables de Non déterministes n-automates d'État . Notre objectif est de faire une étude similaire concernant la génération aléatoire de Non-deterministeDutomates Finis(NFA). Nous considérons dans ce chapitre la méthode de génération de NFA aléatoire basé sur générateur aléatoire. Nous développons une analyse probabiliste de tables de transition non déterministe produite par cette méthode, qui met l'accent sur certaines propriétés de l'associé NFA. Nous avons en particulier mis l'accent sur le nombre de membres de l'AFD obtenus par sous-ensemble construction.

Dans la première partie on fait une description générale sur notre générateur et ces paramètres d'entrée, la seconde partie presente l'algorithme de base de ce générateur et ces paramètres de sortie et la dernière partie nous présentons les résultats obtenus à partir de « lazy random génération ». l'idée est la suivante :

## 3.1 générer puis valider

.On va générer certains nombres d'alphabets et affecter à chaque alphabet un nombre aléatoire c'est le nombre des fils il faut que l'un de ces alphabets égale à zéro pour qu'on dit que valider et on dit que c'est un automate valide car a un de ces alphabets contient zéro fils .

### 3.1.1 Description générale

On a trois classes « état », « Transition », « Alphabet », chaque État a une étiquète et les alphabets ont une étiquette et le Rang, Entre chaque État et transition une classe associative d'alphabet, c-à-d ont associé à chaque transition et état n alphabet, et l'association possède des propriétés (Étiquète, Rang), en effet, ses deux propriétés n'appartiennent ni à l'état, qui peut contient plusieurs Transition, ni aux Transition, qui peuvent avoir plusieurs État.

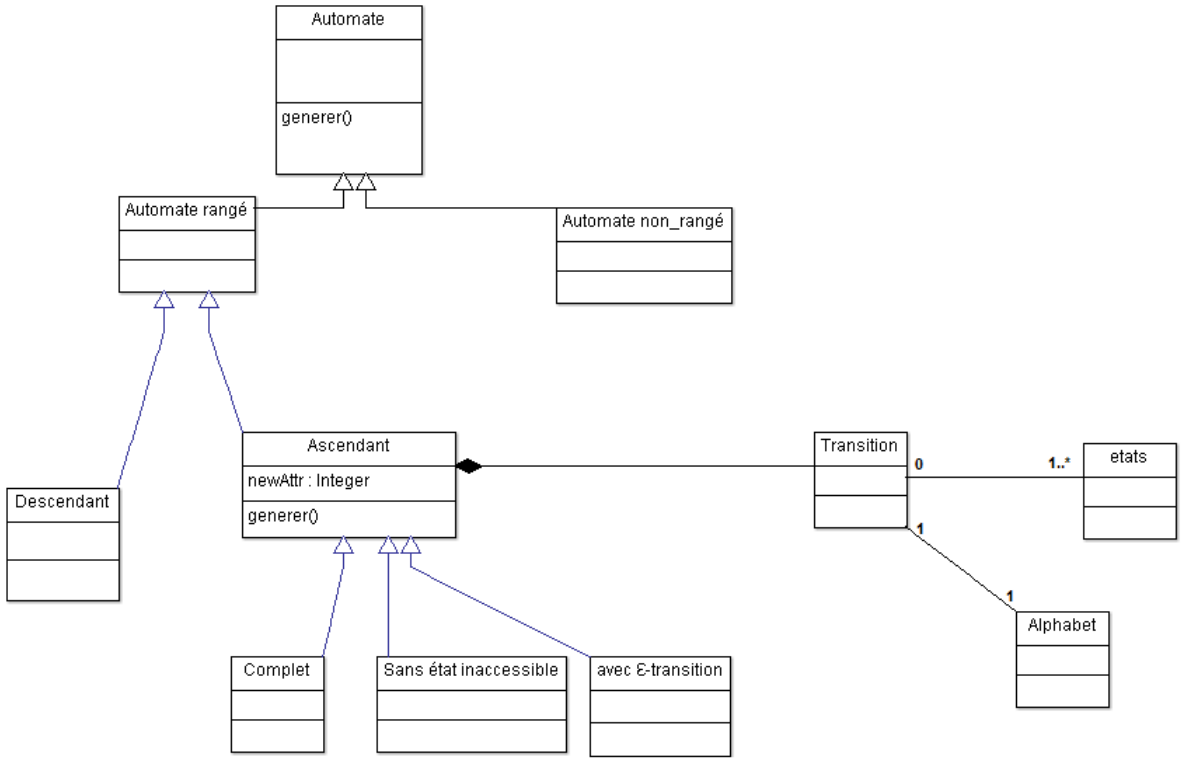



FIGURE 3.1.1 : Diagramme de notre modéle

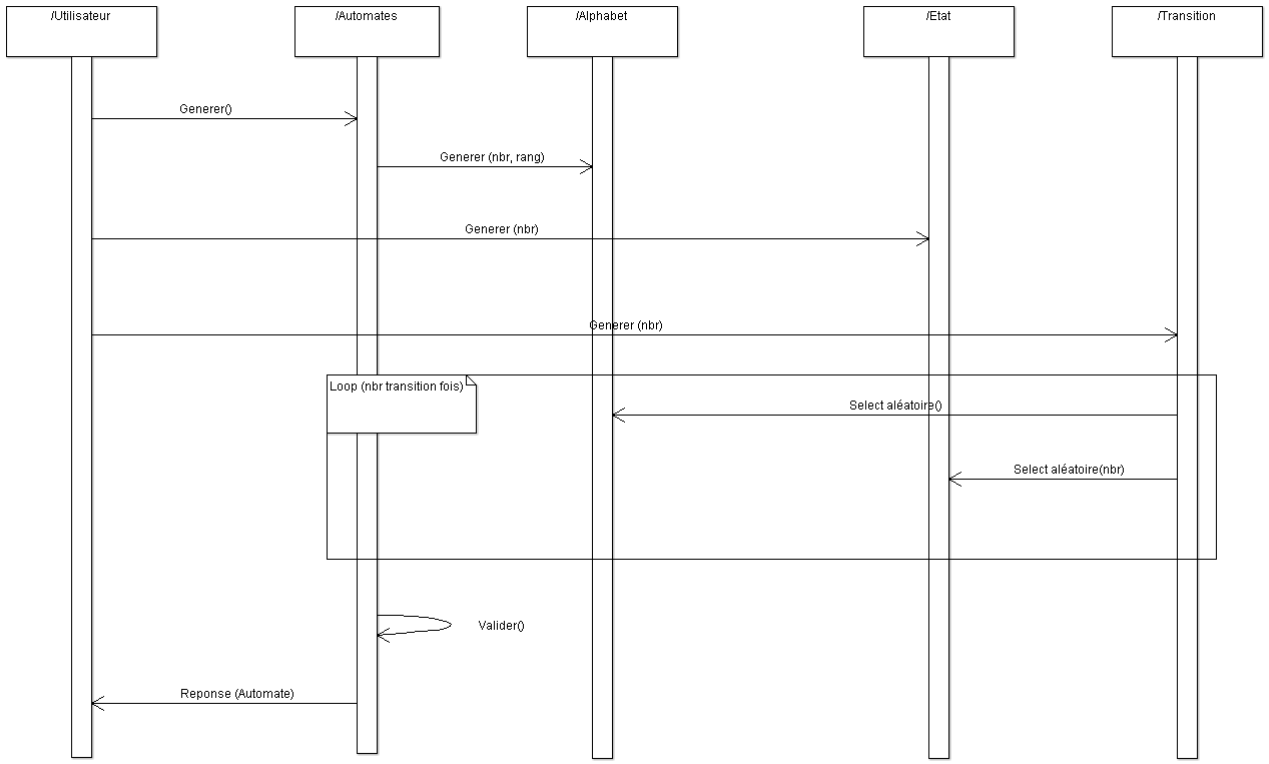



FIGURE 3.1.2 : Diagramme de sequence expliquant le deroulement du générateur

D'abord definir les paramétres :

### La taille de l'alphabet

C'est la taille et le nombre de alphabets générer c-à-d lorsque la taille de tableau égal à $n$ le générateur génère un alphabet et le numéroter jusqu'à$n$

### Le rang maximum

C'est le nombre maximum des fils qui contient l'alphabet, Si est égale à zéro alors on a une feuille .

#### Les états

C'est le nombre des états générer par le générateur pour effectuer à chaque alphabet un état et pour générer les états ont considéré après des états finaux

#### Les transition

La transition se définit comme un arc entre l'état sourcestate et l'état targetstate qui doivent être tous les deux identifiés en utilisant les noms des états (qui doivent être uniques dans la définition de la machine d'État), on définit une transition avec un alphabet et les États et état final

### 3.1.2 Comment générer

**Un générateur de nombres aléatoires** : est un dispositif capable de produire une séquence de nombres dont on ne peut pas « facilement » tirer des propriétés déterministes, de façon que cette séquence puisse être appelée « suite de nombres aléatoires ».

Des méthodes pour obtenir des nombres aléatoires existent depuis très longtemps et sont utilisées dans les jeux de hasard : dés, roulette, tirage au sort, mélange des cartes, etc. Elles peuvent toutefois souffrir (et souffrent généralement) de biais. Actuellement, les meilleures méthodes, censées produire des suites véritablement aléatoires, sont des méthodes physiques qui profitent du caractère aléatoire des phénomènes quantiques

Générer des chaines de caractères aléatoires est une chose souvent utilisée sur les sites internet. On peut l'utiliser par exemple pour générer des mots de passe.

Pour mon cas je voulais créer un automate d'arbre Non-deterministe pour créer des transitions et au lieu de créer les États et les alphabets manuellement(par l'utilisateur), je voulais les générer automatiquement pour que le valider.

Premièrement je génère les alphabets aléatoirement, je donne le nombre des variables à générer et le générateur génère le premier alphabet généré il sera choisis comme l'alphabet source et on met des nombres de zéro jusq'au $N$ ($N$ c'est le nombre des alphabets que je veux générer) après je génère les états à partir d'un nombre donné par l'utilisateur et pour chaque alphabet on génère un nombre

aléatoire affecté à chaque alphabet pour dire qu'on a une feuille ou bien on a $m$ alphabet au-dessous de l'alphabet existant.

### 3.1.3 Comment Valider

On dit qu'un automate d'arbre est valide :

Si on a l'un des alphabets générés à un rang maximum égale à zéro, nous on valide notre automate généré par affecter parmi les alphabets générés un rang qui égale à zéro.

Et si on a tous les états accessible c-ad on fait une matrice qui contient les états en ligne et en colonne, si on a un état accessible par un autre état on met un sinon zéro et on multiple notre matrice par le nombre de transition donné par l'utilisateur si on a une matrice qui contient des uns alors l'automate est valide sinon on élimine les états inaccessible et les transitions associées et on génère autrefois jusqu'à on obtient le nombre des transitions donné par l'utilisateur.

## 3.2 Psuedo Code

Maintenant nous donnons l'algorithme de génération aléatoire d'un automate qu'on a implémenté et on a créé pour les automates d'arbre non deterministes :

**Algorithme 3.1** Générateur aléatoire d'automates

>Entrée : la taille de l'alphabet $a$, le nbr d'etat $k$, le rang max $r$, nombre de transition $t$

**>Sortie** : Fichier contient des transitions

**Debut**

**pour** $i \leftarrow 1$ à $a$ faire
$T[i]$ =la valeur aléatoire entre $a$ et $z$
$P[i]$ =la valeur aléatoire entre 1 et $r$
**finpour**
**pour** $i \leftarrow 1$ à $k$ faire
$K[i]$ =l'état 'S'$i$
**finpour**
**pour** $i \leftarrow 1$ à $t$ faire
indice =générer aléatoirement un nombre entre 1 et $a$
**pour** $i \leftarrow 1$ à $P[i] + 1$ faire
TabEtat[i]= générer a chaque fois un ètat a partir de tableau des ètats
**finpour**
Enregistrer dans le fichier la transition ($T[i]$,TabEtat)
**finpour**

**Fin.**

Le fichier sorti contient $t$ nombre de transition qui sont valides et on va écrire ce fichier comme les fichiers de Timbuk comme on le voit dans la figure suivante :

```
Ops accept:0 aa0:2 aa1:2
Automaton 200
States  q0 q1 q2 q3 q4 q5 q6 q7 q8 q9 q10 q11 q12 q13 q14 q15
Final States q0
Transitions
aa0(q6,q6) -> q0
aa0(q8,q9) -> q0
aa0(q10,q15) -> q0
aa0(q6,q11) -> q1
aa0(q10,q12) -> q1
aa0(q15,q2) -> q1
aa0(q3,q9) -> q2
aa0(q4,q15) -> q2
aa0(q10,q15) -> q2
aa0(q11,q14) -> q2
aa0(q15,q1) -> q2
aa0(q15,q7) -> q2
aa0(q2,q1) -> q3
aa0(q2,q5) -> q3
aa0(q2,q10) -> q3
aa0(q5,q7) -> q3
aa0(q9,q12) -> q3
aa0(q2,q8) -> q4
aa0(q5,q3) -> q4
aa0(q9,q13) -> q4
aa0(q0,q15) -> q5
aa0(q6,q9) -> q5
aa0(q7,q11) -> q5
aa0(q8,q4) -> q5
aa0(q8,q12) -> q5
aa0(q12,q0) -> q5
aa0(q0,q6) -> q6
aa0(q5,q2) -> q6
aa0(q6,q0) -> q6
aa0(q5,q3) -> q7
```

FIGURE 3.2.1 : Resultat obtenu

## 3.3 Tests

Dans ce projet on a travaillé avec Net Beans, Netbeans est un environnement de développement intégré (EDI), placé en open source, Netbeans est un IDE qui supporte une large variété de langage de programmation et d'outils de collaboration.

Dans la figure suivante on voit l'interface de notre générateur :

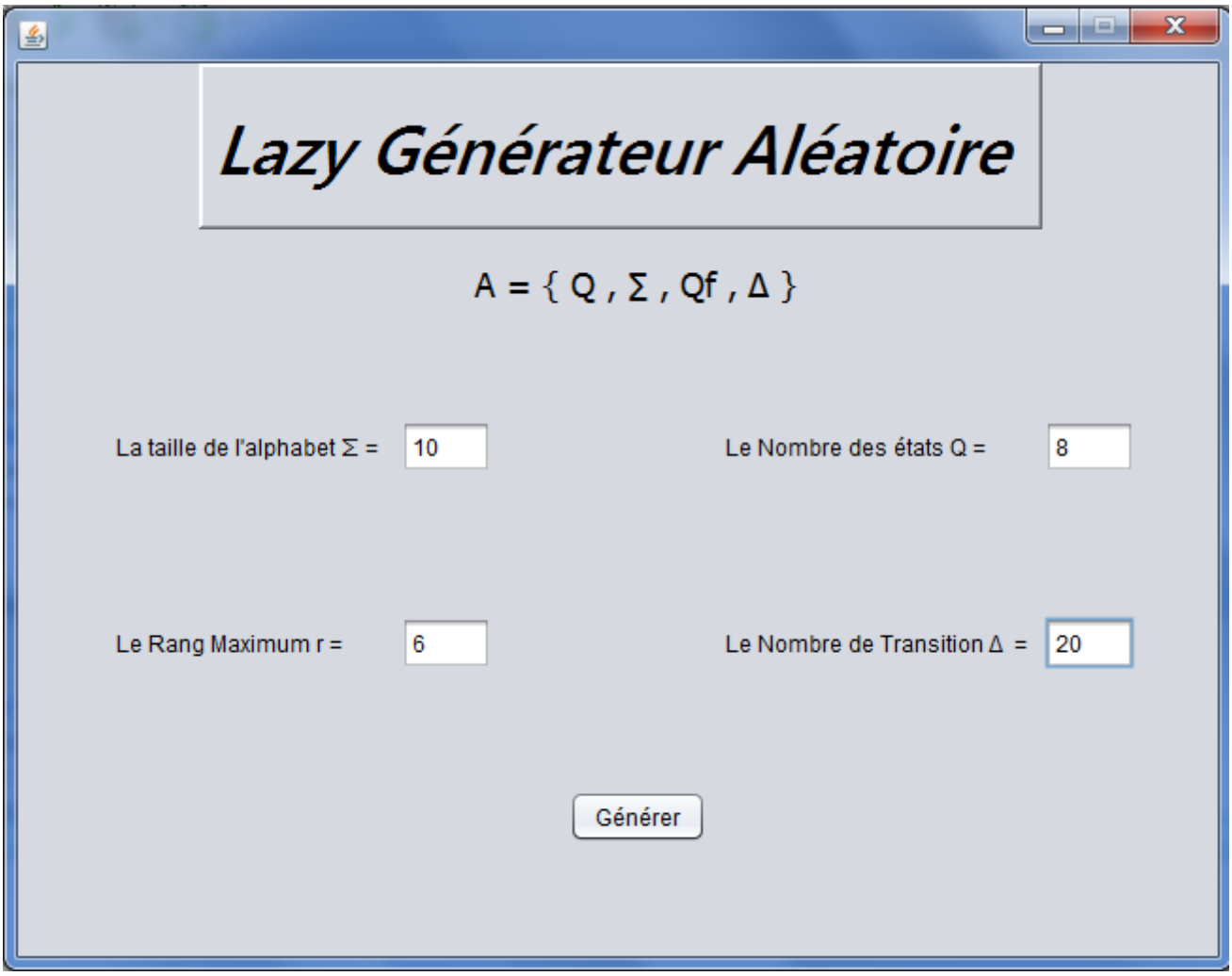



FIGURE 3.3.1 : Capture d'écran de l'interface Générateur aléatoire

Nous notons également qu'on a comme entrait la taille de l'alphabet qui va générer avec le générateur et le rang maximum, et on va générer aussi un nombre des états, et enfin le nombre de transition générer et stocker dans un fichier texte comme l'on voit dans la figure suivante le résultat de l'entrée dans la figure précédente :

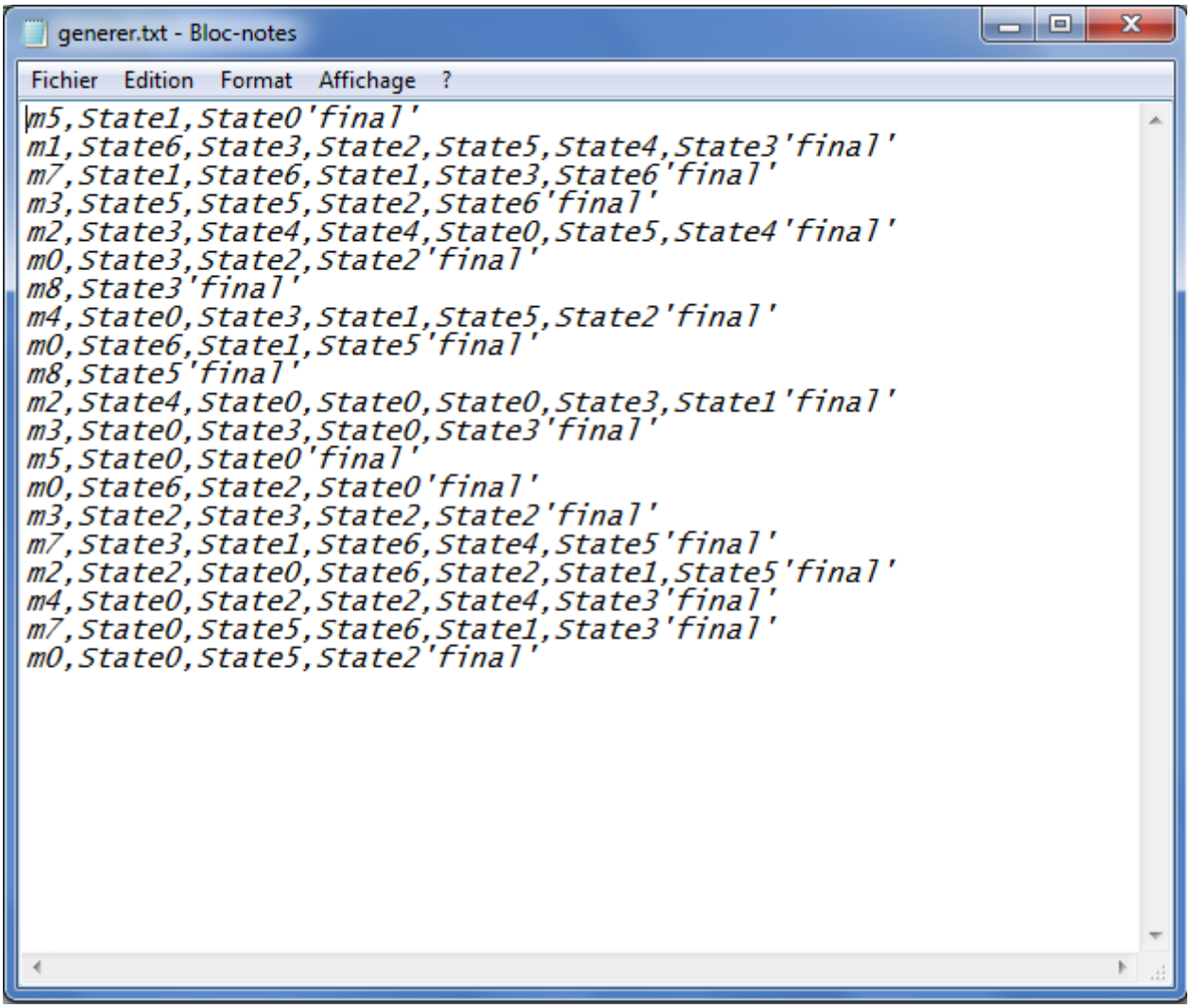



FIGURE 3.3.2 : Capture d'écran de resultat obtenue

on voit bien qu'on a une liste de transition qui commence avec un alphabet généré aléatoirement et se termine avec un état final.

## 3.4 Analyse

Dans cette section, nous présentons une courte analyse analytique et densités de calcul, pour lesquelles nous attendons les transitions doivent être générées de façon aléatoire non trivial.

Voici quelques graphes qui ont généré par notre générateur qui trace une courbe de nombre de transitions par rapport au temps :

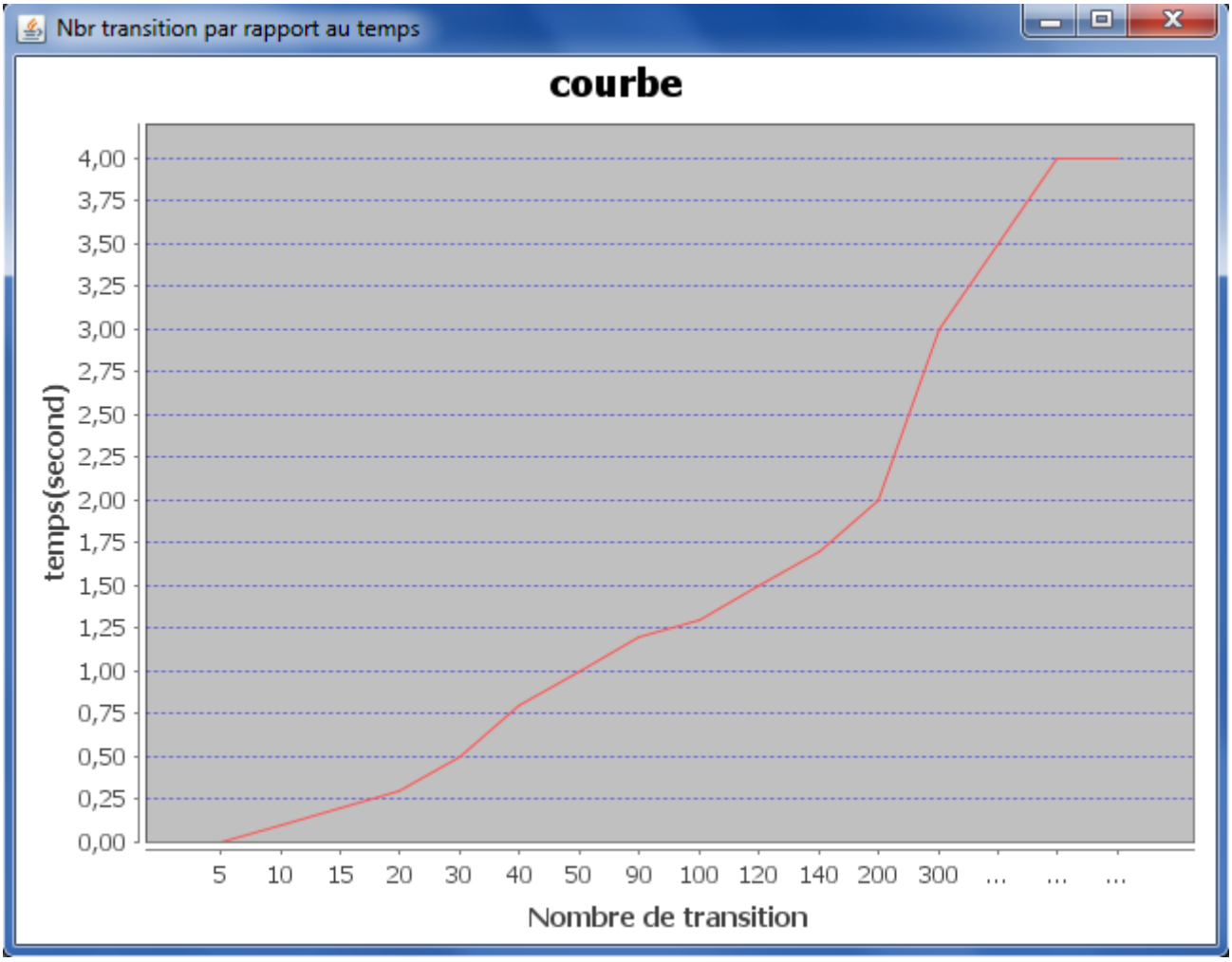



FIGURE 3.4.1 : Capture de courbe 1$|\ \Sigma\ |= 50$

Cette courbe est traçé avec une taille d'alphabet qui égale à 50 et les nombres de transition égale à 200 on voit que aprés quatre second on aura tous les transitions

et la courbe démare dans un nombre de transition égale à 5 par contre si on remarque dans la figure suivante le nombre de transition pour démarer la courbe est diminuer et aprés 4.5 Second on aura tous les transition généré.

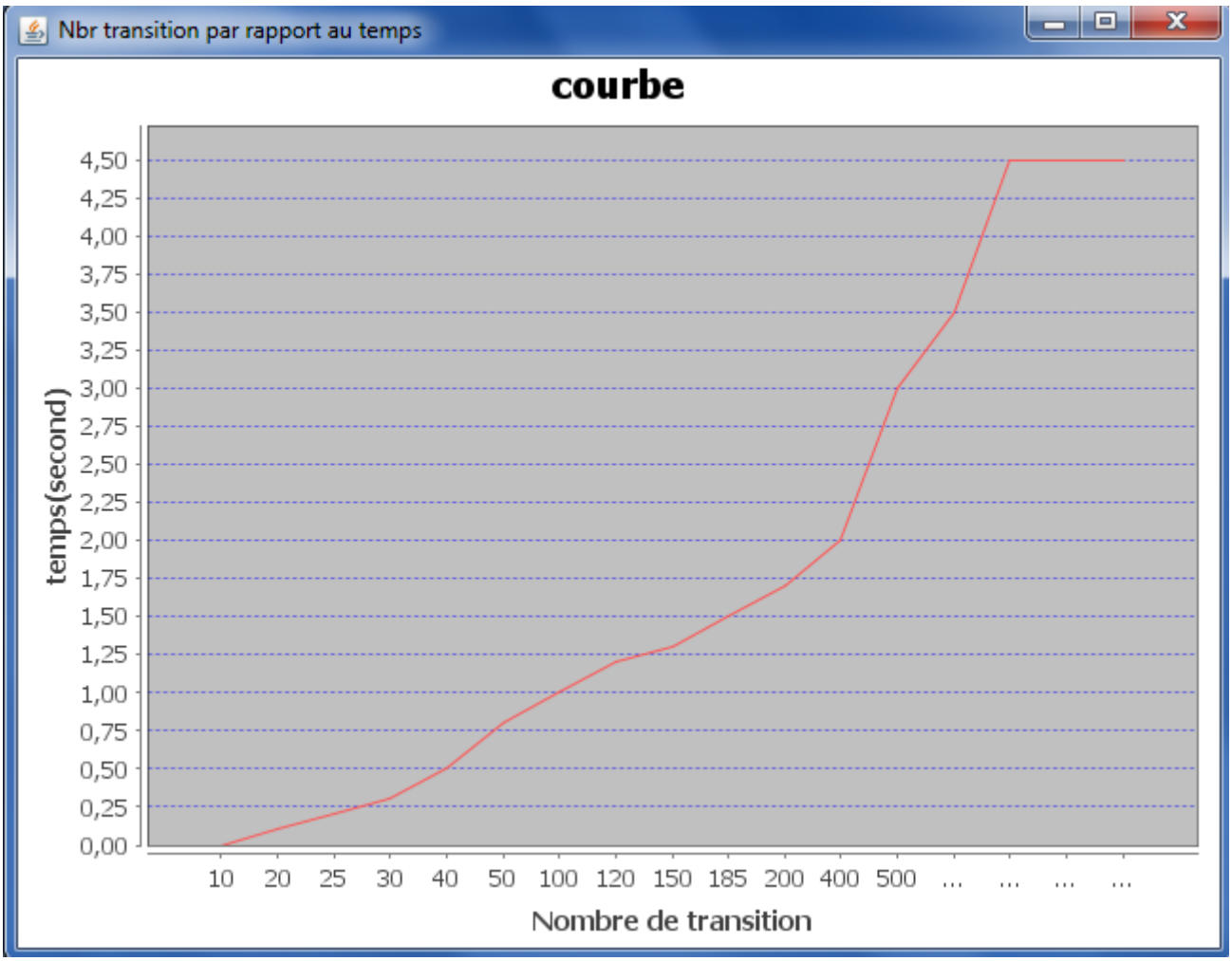



FIGURE 3.4.2 : Capture de courbe 2 | $\Sigma$ |= 150

Lorseque on augmente la taille de l'alphabet on augmente le temps pour génèré un certain nombre de transition, on voit dans la figure précédente que lorseque on augmente la taille de l'alphabet on augmente le nombre de transition dans un temps t.

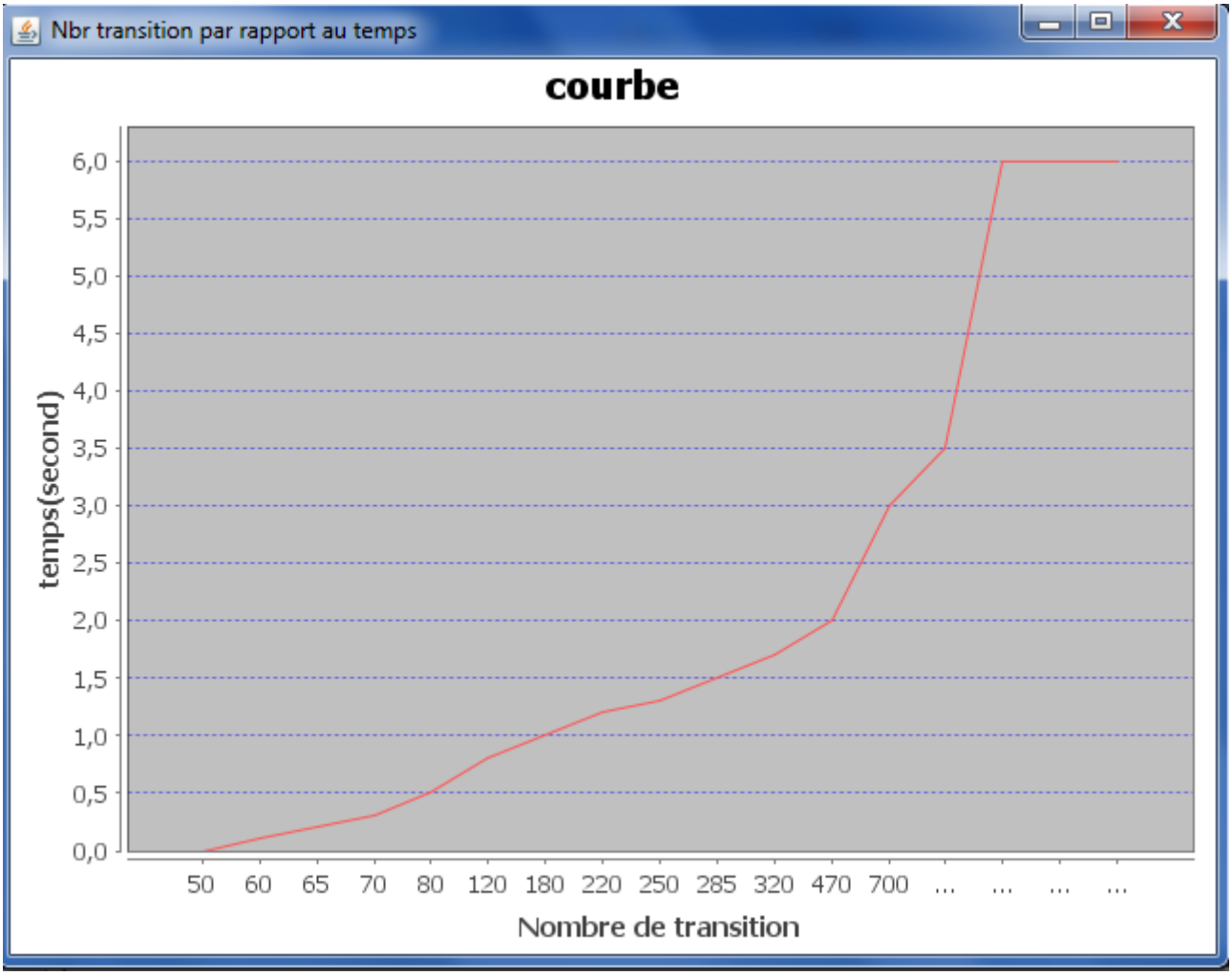



FIGURE 3.4.3 : Capture de courbe 1| $\Sigma$ |= 500

Dans la figure 3.4.3 on voit que la taille de l'alphabet fait l'ajustement exponentiel avec le nombre de transition dans un temps donnée car lorsequ'on augmente la taille de l'alphabet on voit changement de la courbe (l'augmentation) dans un temps donnée.

On remarque que dans les 3 graphes l'evolution en temps par rapport au nombre de transition est estimer entre le linéaire et le parabolique. Le compertement du générateur est acceptable si le nombre de transition est petit.

On remarque aussi que les courbes sont indépendants vis à vis du nombre de l'alphabet. cela est claire car la selection aléatoire d'un symbol de l'alphabet se

fait en temps constant (operation atomique).

On peut dire que malgré la nature du générateur qui génére avant de valider, les resultats obtenues sont trés satisfaisante.

## 3.5 Conclusion

Dans ce dernier chapitre, on a donné une description sur notre application « lazy générateur aléatoire » et on a donné comment il fonctionne et comment faire pour générer, on a donné aussi le résultat obtenu à partir de ce générateur et on a fait quelques analyses à partir des courbes obtenues.

# Conclusion et perspectives

Dans ce mémoire, nous avons étudies les générateurs aléatoire d'automate d'arbre finis, Nous avons proposé un nouveau générateur aléatoire appellé Lazy générateur aléatoire. Son principe est simple générer et aprés vérifier. nous avons entamé un étude exprimentale temporelle et spéciale, nous avons dévloppé un générateur aléatoire qui génére les transitions a partir des donnée que l'utilisateur va donnée(la taille de l'alphabet, le rang, nombre des états, nombre de transitions qu'on veut extraire), nous avons dessiner des courbes et faire une comparaison entre elles.

Comme perspectives nous espérons pouvoir faire des études comparative avec d'autre générateur comme le générateur de Malleti et al [10] pour pouvoir juger la performence de notre outil.

De plus nous espérons étendre le générateur aléatoire pour d'autre types d'automates a savoir les automates desscendants déterministe, non rangés, à un multiplicité transducteurs... etc .

# Bibliographie


[1] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. Available on : `http://www.grappa.univ-lille3.fr/tata`, 2008. release Novembre, 18th 2008.

[2] Reinhard Diestel. Graph theory. 2000.

[3] Younes Guellouma, Hadda Cherroun, and Djelloul Ziadi. Efficient data structure for deterministic tree automata minimization. page 10, 2014.

[4] D. L. Kreher and D. R. Stinson. Combinatorial algorithms : Generation enumeration and search. *CRC Press*, 1999.

[5] Ted Leslie. Efficient apprpaches to subset construction. Technical report, University of Waterloo,Canada, 1995.

[6] D. Kisman M. Domaratzki and J. Shallit. On the number of distinct languages accepted by finite automata with n states. *Journal of Automata*, 2002.

[7] Amin Saberi Mohsen Bayati, Jeong Han Kim. A sequential algorithm for generating random graphs. page 15, 2007.

[8] P. Zimmerman P. Flageolet and B. Van Custem. A calculus for the random generation. 1994.

[9] Thomas Paranthoën. *Génération aléatoire et structure des automates à états finis*. these de doctorat en informatique, Université de Rouen, mars 2004.

[10] Andreas Malletti Thomas Hanneforth. Random generation of nondeterministic finite-state tree automata. Novembre 2013.

[11] Lynette van Zijl. *Generalized Nondeterminism and the Succinct Representation of Regular Languages*. PhD thesis, Stellenbosch University, South Africa., 1997.

[12] Djelloul Ziadi, Thomas Paranthoën, and Jean-Marc Champarnaud and-Georges Hansel . Random generation models for nfas. 2004.