Génération aléatoire d'automates et analyse d'algorithmes de minimisation

Julien David

Université Paris Est

THÈSE

pour obtenir le grade de

Docteur de l'Université Paris Est

Spécialité : **Informatique**

préparée au laboratoire **d'Informatique Gaspard Monge**

dans le cadre de l'École Doctorale **MSTIC**

présentée et soutenue publiquement
par

Julien David

le mardi 28 septembre

Titre:

Génération aléatoire d'automates et analyse d'algorithmes de minimisation

Directrice de thèse: **Frédérique Bassino**
Co-directeur de thèse: **Cyril Nicaud**

Jury

Mme. Frédérique Bassino,	Directrice de thèse
M. Jean Berstel,	Examinateur
M. Jean-Marc Champarnaud,	Examinateur
M. Alain Denise,	Examinateur
M. Philippe Duchon,	Rapporteur
M. Massimiliano Goldwurm,	Rapporteur
M. Cyril Nicaud,	Co-directeur de thèse
Mme. Michèle Soria,	Examinatrice

"Mais c'est ça qui est la révolution, la vraie, l'unique !
C'est avec ça, et non avec les bombes stupides,
qu'on révolutionne le monde !
Ce n'est pas en détruisant, c'est en créant ***un générateur aléatoire****,*
que vous venez de faire acte de révolutionnaire !...
Et que de fois je vous l'ai dit,
l'analyse en moyenne *seule est révolutionnaire,*
la seule qui, par-dessus les pauvres événements politiques,
l'agitation vaine des sectaires et des ambitieux,
travaille à l'humanité de demain,
en prépare la vérité, la justice, la paix !...
Ah ! mon cher enfant, si vous voulez bouleverser le monde
en essayant d'y mettre un peu plus de bonheur,
vous n'avez qu'à rester dans votre laboratoire,
car le bonheur humain ne peut naître
que de votre fourneau de savant."

Émile Zola, dans *Paris*.
(à quelque mots près...)

Remerciements

Au cours de ma thèse, j'ai pu rencontrer au bas mot une centaine de doctorants et d'enseignants-chercheurs de différents domaines, autant de gens brillants et cultivés avec lesquels j'ai eu grand plaisir à discuter. Je ne serai probablement pas exhaustif dans mes remerciements, j'espère que les absents m'en excuseront, car cela n'empêche en rien ma gratitude. De plus, cet exercice d'affichage public de mes bons sentiments s'apparentant pour moi à une forme de torture (auquel cependant je me plie volontiers, non pas par atavisme, mais parce qu'il me paraît indispensable), j'espère que le lecteur comprendra que c'est par pudeur que je garde (parfois) une certaine sobriété.

Je tiens à remercier

- Frédérique Bassino et Cyril Nicaud qui ont dirigé cette thèse. Ils ont su prendre le temps de suivre chacun de mes pas, depuis mon arrivée en Master recherche à l'université de Marne-la-Vallée il y a cinq ans. Dès lors, j'ai pu bénéficier de leur expérience et de leur pédagogie. Je les remercie de m'avoir proposé ce sujet passionnant, pour leur patience et pour la liberté qu'ils ont su me laisser tout au long de cette thèse. J'espère travailler souvent avec eux dans l'avenir et rester encore longtemps le testeur officiel des pâtisseries de Frédérique.
- Philippe Duchon et Massimiliano Goldwurm, qui m'ont fait l'honneur d'être rapporteur de cette thèse,
- Jean Berstel, Jean-Marc Champarnaud, Alain Denise et Michèle Soria qui m'ont fait l'honneur de prendre part à mon jury,
- Philippe Duchon une fois de plus, dont les explications m'ont permis d'avancer dans l'étude de la complexité moyenne de l'algorithme de Moore,
- Nicolas Bedon et Guillaume Blin pour leurs conseils en tous genres,
- Étienne Duris, Gilles Roussel, Giuseppina Rindone, Sylvain Lombardy et Marc Zipstein dont les portes étaient toujours ouvertes pour discuter des TDs associés à leur cours. Enseigner à leur côté fût aussi agréable qu'instructif.
- Jean-Christophe Novelli et Stéphane Vialette, dont les cours destinés aux doctorants et aux étudiants en Master recherche ont été particulièrement enrichissant,
- Carine Pivoteau sans qui je n'aurais jamais pu obtenir l'article de Moore,
- Hayat Cheballah pour ses conseils en tous genres et relectures d'articles,
- Yann Ponty et Julie Bernauer pour leurs conseils et leur aide dans la recherche de postdoc,
- Florian Sikora et Frédéric Fauberteau que j'ai souvent dérangé en plein travail lorsque j'avais besoin de faire une pause dans le mien,
- Florian Sikora (qui doit sûrement s'arracher les cheveux en voyant la longueur

de ces remerciements), Frédéric Fauberteau, Omar Aït Mous, Elsa Tolone, Laurent Brau, Nathalie Aubrun, Samuele Giraudo, Myriam Rakho, Hieu Dinh et Rosa Cetro pour la bonne ambiance qui règne entre les doctorants de l'institut Gaspard Monge,
- Catherine Sauvaget, Benjamin Raynal et Cédric Curcio qui m'ont supporté pendant le master recherche et qui sont ensuite partis en thèse de leur côté,
- Tous les jeunes chercheurs du groupe ALEA pour toutes les soirées passées ensemble, en particulier Cédric Saule avec qui j'ai refait le monde plus d'une fois, Annelyse Thévenin à qui je pardonne de m'avoir donné mes premiers cheveux blancs en même temps qu'une de mes plus belles frayeurs, Jérémie Lumbroso et Alice Jacquot à qui je pardonne les réactions singulières après ingestion d'une bouillabaisse et de quelques verres de vins,
- Sophie Toulouse, Sébastien Guérif et tous les membres du LIPN qui ont su m'accueillir "chaleureusement" dans leur laboratoire lors de mes nombreuses visites,
- Gabrielle Brossard et Line Fonfrède, deux murs porteurs du laboratoire Gaspard Monge, pour leur gentillesse et leur efficacité,
- Les 4 Jeudis pour toutes ces soirées passées à comparer l'herbe chez les thésards du domaine d'à côté,
- Jean-Jacques Bourdin et Vincent Boyer de l'université Paris 8, ainsi que Vincent Victorin, sans qui je n'aurais pas fait de recherche en informatique,
- Sébastien Tennenbaum, François Robelet, Daniel Boissin, Nataniel Obin, Nicolas Zdebski, Nicolas Louise, Élodie Pelletier, la famille Marquis au grand complet et bien sûr ma famille pour leur soutien moral,
- Clotilde Gonthier pour sa présence, sa compréhension, ses coassements et son soutien.

Table des matières

Introduction

La thématique principale de cette thèse se situe à la frontière entre la combinatoire et la théorie des automates. Plus précisément, on se place dans un contexte où les automates sont associés à une loi de probabilités, afin d'effectuer deux types de travaux :

- la génération aléatoire d'automates,
- l'analyse en moyenne d'algorithmes sur ces automates.

La génération aléatoire permet de mener une étude expérimentale sur les propriétés de l'objet engendré et des algorithmes qui les manipulent. Il s'agit également d'un outil de recherche, qui permet de faciliter l'étude théorique du comportement moyen des algorithmes.
L'analyse en moyenne des algorithmes s'inscrit dans la suite des travaux fondateurs de Donald Knuth [47–49]. Le schéma classique en analyse d'algorithmes consiste à étudier le pire des cas, qui n'est souvent pas représentatif du comportement de l'algorithme en pratique. D'un point de vue théorique, on peut définir ce qui se produit "souvent" en se donnant une loi de probabilités sur les entrées de l'algorithme. L'analyse en moyenne consiste alors à estimer des ressources utilisées en faisant l'hypothèse que les entrées de l'algorithme sont distribuées selon cette loi de probabilités.

Plus particulièrement, j'ai travaillé sur des algorithmes de génération aléatoire d'automates déterministes accessibles (complets ou non). Ces algorithmes produisent des automates à n états en temps moyen $\mathcal{O}(n^{3/2})$ et requièrent très peu d'espace mémoire. Ils utilisent de la combinatoire bijective qui permet d'obtenir des automates par transformation d'objets plus simples, ainsi qu'un procédé générique permettant d'engendrer des structures combinatoires spécifiables : les générateurs de Boltzmann. J'ai ensuite implanté ces méthodes dans deux logiciels : `REGAL` et `PREGA`.

Je me suis intéressé à l'analyse en moyenne des algorithmes de minimisation d'automates et j'ai obtenu des résultats qui montrent que le cas moyen des algorithmes de Moore et Hopcroft est bien meilleur que le pire des cas. Plus précisément, ces deux algorithmes minimisent des automates déterministes à n états en temps moyen $\mathcal{O}(n \log\log n)$.

Dans la suite de l'introduction, on présente plus en détails le contexte et les contributions présentés dans cette thèse.

Un *algorithme* est une méthode définie en un nombre fini d'instructions, permettant de résoudre un problème donné. En informatique théorique, l'*algorithmique* est un terme qui regroupe à la fois la conception et l'analyse des algorithmes.

L'*analyse des algorithmes* peut être effectuée par une approche expérimentale ou une approche théorique, les deux étant complémentaires du point de vue informatif. On s'intéressera à la *complexité d'un algorithme*, c'est-à-dire à l'estimation

des ressources utilisées en fonction de la taille de l'entrée, indépendamment de l'ordinateur ou du langage de programmation utilisés. Il existe deux grandes approches :

- La plus répandue est l'analyse du pire des cas. Elle consiste à caractériser l'exécution la plus coûteuse de l'algorithme pour une taille d'instance donnée, puis à estimer ce coût. Ce type d'analyse est généralement plus simple que l'analyse en moyenne présentée dans ce qui suit et permet d'obtenir une hiérarchie sur l'efficacité des méthodes, qui fait pleinement sens. En revanche, le pire des cas est parfois très éloigné du comportement de l'algorithme observé en pratique.
- L'analyse en moyenne permet d'obtenir une information à mi-chemin entre le pire des cas et le cas pratique. Le coût moyen de l'algorithme est la moyenne des coûts de toutes ses instances. En l'absence d'informations précises sur les données de l'algorithme, on effectue généralement le calcul pour la distribution uniforme. Un résultat de ce type est bien plus compliqué à obtenir qu'une estimation dans le pire des cas.

Un *objet combinatoire* est un objet pour lequel on peut définir une notion de taille et tel qu'il existe un nombre fini d'éléments d'une même taille.

La *génération aléatoire* joue un rôle central dans l'étude des objets combinatoires et des algorithmes qui s'y appliquent. Lorsqu'un sociologue effectue une étude quantitative sur les comportements ou les caractéristiques d'une population, il a conscience que ses statistiques ne sont en aucun cas basées sur une étude exhaustive. Pour que ses résultats prennent sens, la portion sélectionnée doit donc être représentative, en terme de proportion, de la population sur laquelle l'étude est effectuée. Afin que les minorités apparaissent dans les estimations, les statistiques doivent être calculées sur le plus grand nombre d'individus possible. On retrouve de pareilles analogies en biologie ou en chimie. En informatique théorique, la qualité d'une étude expérimentale dépend des mêmes conditions. En revanche, il existe deux façons de produire des statistiques : à partir d'une base de données ou à partir d'un générateur aléatoire. Dans cette thèse, on s'intéressera exclusivement au second cas. En effet, tout comme dans les autres disciplines, une étude exhaustive est souvent impraticable en raison du trop grand nombre d'objets à étudier. Supposons donc que l'on connaisse une méthode capable d'engendrer un objet combinatoire selon une loi de probabilité donnée. Élaborer une telle méthode nécessite de savoir énumérer, ou de savoir estimer asymptotiquement le nombre d'objets d'une taille fixée. On est alors en mesure d'effectuer un ensemble de tests, puis d'émettre des conjectures sur les propriétés moyennes des objets et sur le comportement moyen des algorithmes qui leur sont appliqués. On pourra également vérifier si la loi de probabilités choisie modélise convenablement un contexte donné en comparant les données aléatoires avec des données réelles. Dans cette thèse, on s'intéressera uniquement à la distribution uniforme, c'est-à-dire au cas où tous les objets de mêmes tailles ont la même probabilité d'être engendrés. Les conjectures établies grâce aux générateurs sont motivantes pour démarrer une étude théorique.

Parmi les différents buts d'une étude théorique d'un ensemble d'objets combinatoires, on a recensé les suivants :

- énumérer les objets possédant une propriété donnée. En plus d'être intéressant en soit, un tel résultat peut permettre la conception de nouveaux générateurs.
- obtenir la complexité moyenne des algorithmes associés.

La Figure 1 résume cette approche.

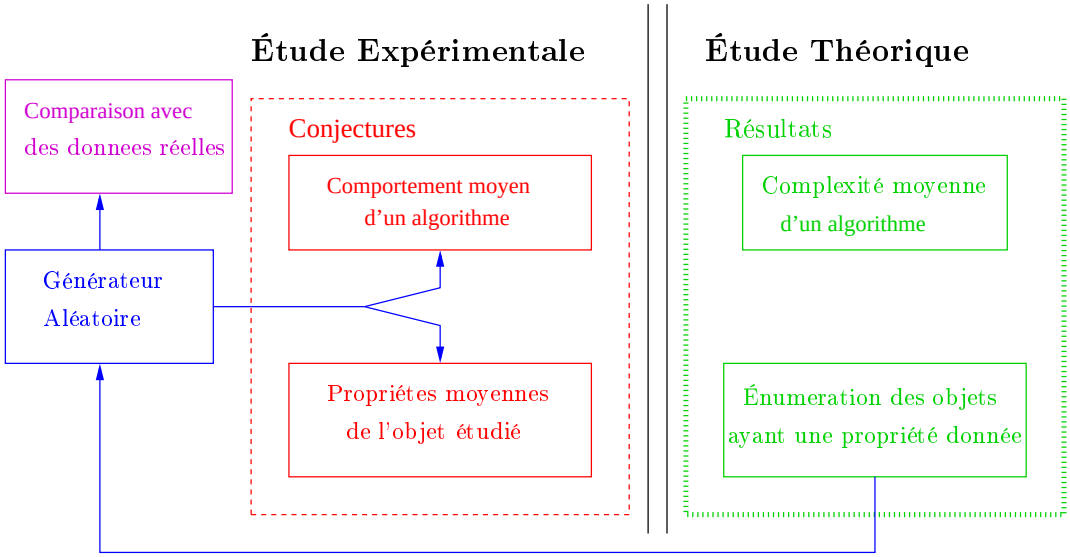


FIG. 1 – Utilisation d'un générateur aléatoire

Les objets étudiés dans cette thèse sont les *automates finis*. Il s'agit d'une représentation à la fois graphique et compacte d'un langage rationnel, dont on donnera la définition formelle ultérieurement. On peut voir les automates comme un graphe dont les arêtes sont étiquetées par des lettres. La terminologie est toutefois différente : on parle d'*états* plutôt que de *sommets* et de *transitions* plutôt que d'*arêtes*. On distingue deux sous-ensembles particuliers dans l'ensemble des états de l'automate : les états *terminaux* et les états *initiaux*.
La Figure 2 montre l'exemple d'un automate associé à l'ensemble de mots {*algorithme*,-*automate*, *autoroute*}. Les capacités de modélisation d'un automate en font un outil fondamental en linguistique, en bioinformatique, ou en électronique. L'*automate minimal* est l'unique automate déterministe associé à un langage, pour lequel le nombre d'états est minimal. Leur rôle est central en théorie des automates car ils permettent une représentation canonique des langages rationnels pour un coût minimal en mémoire. La plupart des algorithmes de minimisation calculent l'automate minimal à partir d'un automate déterministe reconnaissant le même langage.

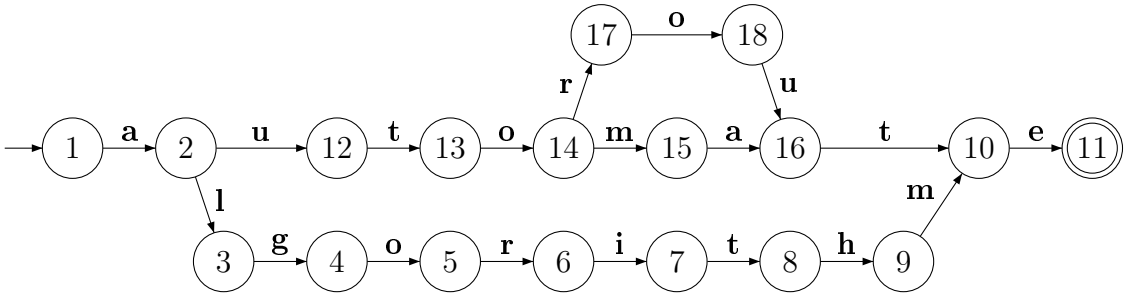


FIG. 2 – Un automate. L'état 1 est *initial* et l'état 11 est *terminal*.

* * *

Contributions et plan détaillé Les contributions sont rassemblées dans les chapitres 4, 5, 7, 8, 9, 10.

Première partie : on commence par présenter un ensemble de notions fondamentales pour la bonne compréhension des résultats présentés par la suite.

Dans le chapitre 1, on présente les deux grandes thématiques de cette thèse : la *génération aléatoire* et l'*analyse d'algorithme*.

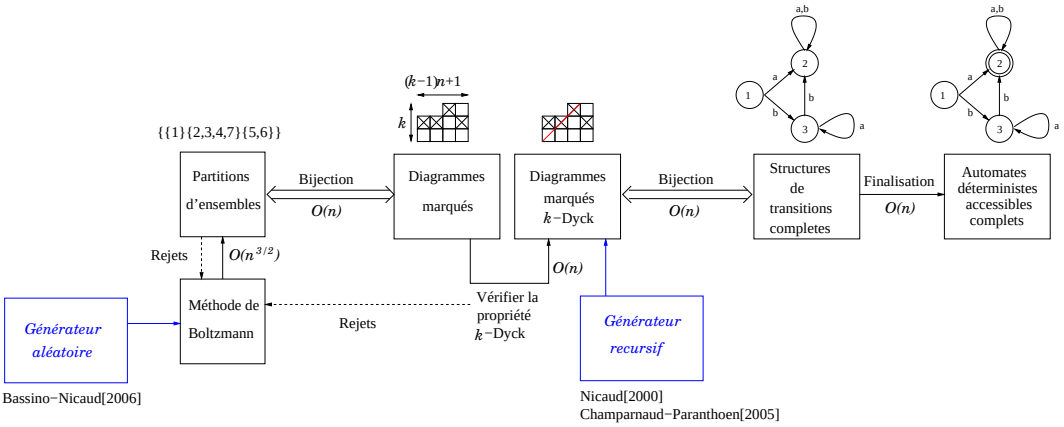
- La conception de générateurs aléatoires s'appuie soit sur des méthodes dites *ad hoc*, c'est-à-dire conçues pour résoudre un unique problème, soit sur des méthodes génériques. En effet, certains objets combinatoires possèdent une spécification récursive. La méthode récursive [36, 62] utilise cette description pour calculer le nombre total d'objets d'une taille donnée et permet d'obtenir des générateurs exhaustifs ou aléatoires. La méthode de Boltzmann [31] de génération aléatoire, évite le lourd précalcul consistant à énumérer toutes les solutions. Pour cela, on néglige la contrainte d'engendrer uniquement les objets d'une taille fixée. Très efficace pour faire de la génération en taille approchée, elle nécessite une phase de rejet qui peut être coûteuse dans le cas de la génération en taille exacte. Comme elle nécessite peu de mémoire, elle permet d'engendrer des structures de très grande taille.
- L'analyse théorique d'un algorithme a pour objectif d'identifier et d'estimer les différents coûts nécessaires pour résoudre un problème. On s'intéresse ici au temps d'exécution. On étudie l'évolution de ce temps en fonction de la taille des instances : on parle de *complexité en temps*. On présentera quatre approches différentes, lesquelles seront toutes envisagées dans cette thèse : la *complexité dans le pire des cas*, la *complexité moyenne*, la *complexité générique* et la *complexité amortie*.

Dans le chapitre 2, on présente les objets pour lesquels on va concevoir des générateurs aléatoires et effectuer une analyse des algorithmes qui leur sont appliqués : les automates. On s'intéressera aux différentes propriétés des automates, lesquelles vont nous permettre de caractériser des ensembles pour lesquels on concevra des générateurs, et auxquels on restreindra l'analyse des algorithmes. En particulier, les automates *accessibles*, sont des automates pour lesquels il est possible d'atteindre n'importe quel état de l'automate en partant d'un état initial et en suivant une suite de transitions. Cette propriété pourtant simple par sa définition va poser de grandes difficultés combinatoires. Les automates que l'on étudie sont également *déterministes*, c'est-à-dire qu'il existe au plus une transition étiquetée par une même lettre partant d'un même état. Enfin, on introduit la notion de *structure de transitions*, primordiale pour la bonne compréhension des chapitres suivants : il s'agit d'un automate sans états terminaux.

Deuxième partie : la génération aléatoire d'automates déterministes accessibles.

Dans le chapitre 3, on présente les algorithmes existants qui permettent d'engendrer des automates déterministes accessibles complets à n états sur un alphabet

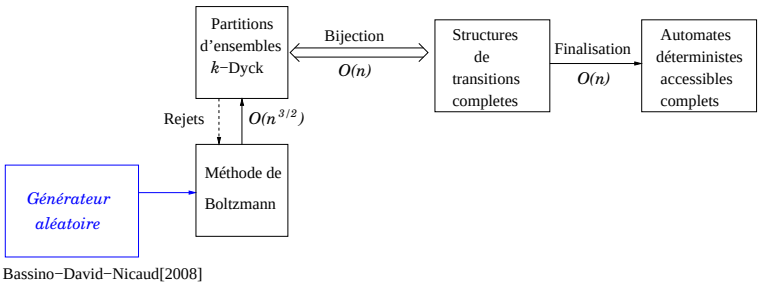
à k lettres, lesquels algorithmes sont basés sur les méthodes génériques de génération aléatoire associées à des bijections efficaces permettant de se ramener à l'étude d'objets combinatoires plus simples. Un automate pouvant être décomposé en une structure de transitions et un ensemble d'états terminaux, il est possible engendrer indépendamment ces deux composantes. La première méthode de génération [25,60] présentée utilise une bijection entre les structures de transitions sous-jacentes aux automates et un ensemble de diagrammes marqués particuliers, définis pour l'occasion et plus simples à manipuler que les initiaux. A ce stade, il est possible de définir un générateur exhaustif d'automates déterministes accessibles et complets, ainsi qu'un générateur aléatoire nécessitant de lourds précalculs. Dans les deux cas, il est impossible en pratique d'engendrer des automates de grandes tailles. La seconde méthode [10] utilise une bijection entre des diagrammes marqués et des partition d'un ensemble de taille $kn+1$ en n sous-ensembles. Le générateur utilise la méthode de Boltzmann pour engendrer les partitions d'ensembles, que l'on transforme en structures de transitions par deux transformations successives. Cette amélioration permet d'engendrer efficacement des automates de grandes tailles. On présente également une estimation du nombres d'automates déterministes accessibles complets à n états sur un alphabet à k lettres. La Figure ci-dessous résume ce paragraphe.



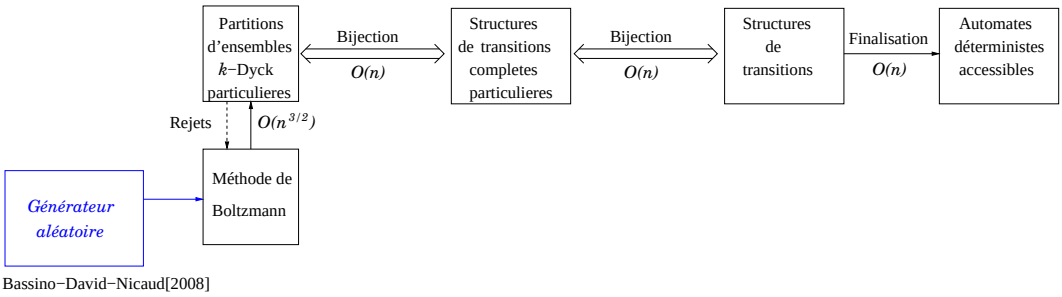
Dans le chapitre 4, on présente deux nouveaux générateurs d'automates, dont la conception repose sur des méthodes de combinatoire bijective.

Le premier est une simplification de l'algorithme de génération aléatoire d'automates déterministes accessibles complets [10] mentionnés au chapitre précédent. L'idée est la suivante : on remplace les deux transformations successives par une transformation unique. Pour cela, on décrit une bijection entre un ensemble particulier de partitions d'un ensemble de taille $kn+1$ et l'ensemble des structures de transitions.
Le générateur simplifié est décrit dans la Figure ci-dessous.

On présente ensuite un générateur aléatoire d'automates déterministes accessibles (possiblement incomplets). Pour cela, on décrit une nouvelle bijection entre un ensemble $\mathcal{E}_n$ de structures de transitions complètes particulières à $n+1$ états et l'ensemble $\mathcal{I}_n$ des structures de transitions possiblement incomplètes à n états. Puis, on caractérise un ensemble $\mathcal{F}_n$ de partitions d'ensembles particulières en bijection avec l'ensemble $\mathcal{E}_n$ (cette transformation n'apparaît pas dans la littérature). On



utilise ce résultat pour estimer asymptotiquement le nombre d'automates déterministes accessibles possiblement incomplets et on montre qu'il est asymptotiquement proportionnel au nombre d'automates déterministes accessibles et complets. Enfin, on décrit l'algorithme de complexité moyenne $\mathcal{O}(n^{3/2})$ permettant d'engendrer aléatoirement des automates déterministes accessibles.



Hormis la bijection entre $\mathcal{E}_n$ et $\mathcal{F}_n$ qui est remplacée par un résultat équivalent, l'ensemble de ce chapitre apparaît dans les articles [6, 7], écrits en collaboration avec Frédérique Bassino et Cyril Nicaud.

Dans le chapitre 5, on présente les logiciels `REGAL` et `PREGA`, les résultats expérimentaux qu'ils ont permis d'obtenir et à partir desquels on émet un ensemble de conjectures sur les propriétés moyennes des automates.

`REGAL`, pour `Random and Exhaustive Generators for Automata Library` est une bibliothèque développée en *C++*, permettant aux programmeurs d'utiliser des générateurs aléatoires et exhaustifs d'automates.

`PREGA`, pour `Program to Randomly and Exhaustivly Generate Automata`, a été conçus pour des utilisateurs qui ne programment pas en *C++*. Pour obtenir un résultat sous forme graphique, il suffit de décrire l'ensemble des tests à effectuer à l'aide d'un fichier *XML*. Parmi les générateurs fournis par `REGAL` et `PREGA`, on trouve naturellement ceux décrits dans le chapitre 4, auxquels s'ajoutent des générateurs efficaces d'automates fortement connexes, minimaux ou émondés, ainsi que des générateurs de transducteurs. On montre également que la complexité amortie du générateur exhaustif d'automates déterministes accessibles et complets est asymptotiquement proportionnelle au nombre d'automates à engendrer.
On pourra noter que `REGAL` est présenté dans [5], écrits en collaboration avec Frédérique Bassino et Cyril Nicaud, et [29]. Les sources peuvent être téléchargées à l'adresse suivante : `http ://regal.univ-mlv.fr`

Troisième partie : Analyse en moyenne des algorithmes de minimisation.

Dans le chapitre 6, on présente un ensemble d'algorithmes de minimisation. La plupart des algorithmes sont basés sur le calcul d'une relation d'équivalence entre les états de l'automate, appelés *relation d'équivalence de Myhill-Nerode* [58]. On utilisera cette relation pour définir formellement l'automate minimal. On accorde une attention particulière aux algorithmes de Moore [57] et de Hopcroft [43], dont on fait l'analyse en moyenne par la suite. La complexité dans le pire cas du premier est égale à $\Theta(n^2)$, celle du deuxième est égale à $\Theta(n \log n)$, ce qui est le meilleur résultat connu à ce jour. L'algorithme de Moore calcule la relation d'équivalence de Myhill-Nerode par raffinements successifs d'une partition de l'ensemble des états. Chaque raffinement de partitions a un coût égal à $\Theta(n)$ et il y a au plus n raffinements, d'où la complexité dans le pire cas. Étant donné qu'un pré-traitement sur l'automate d'entrée peut-être effectué en temps linéaire, on s'intéresse avant tout au coût des deux algorithmes appliqués aux automates déterministes accessibles et complets. Dans le reste du chapitre, on fait un rapide survol d'autres algorithmes de minimisation, pour la plupart récents.

Dans les chapitres 7, 8 et 9, on étudie exclusivement l'algorithme de Moore.

Le chapitre 7 contient des résultats qui ne sont pas issus de l'analyse en moyenne. On commence par une réflexion sur l'implantation efficace de l'algorithme de Moore, qui n'apparaît pas dans la littérature.
On montre ensuite que pour minimiser un automate, le nombre de raffinements de partitions est égal au nombre de raffinements de partitions nécessaires pour minimiser l'automate minimal associé. Ce résultat nous amène à étudier le nombre minimum de raffinements de partitions effectués par l'algorithme pour un automate minimal à n états. On montre que ce nombre est égal à $\Omega(\log \log n)$ lorsque l'alphabet contient au moins deux lettres et $\Omega(\log n)$ lorsque l'alphabet est unaire. Dans le meilleur des cas, la complexité de l'algorithme appliqué à un automate minimal est donc $\Omega(n \log \log n)$ lorsque l'alphabet contient au moins 2 lettres. Ce résultat apparaît dans un article [8], écrit en collaboration avec Frédérique Bassino et Cyril Nicaud, puis soumis en 2009 à *Algorithmica*.

Dans le chapitre 8, on étudie la complexité moyenne de l'algorithme de Moore en utilisant uniquement les propriétés des ensembles d'états terminaux. En effet, on a vu qu'un automate pouvait être décomposé en une structure de transitions et un ensemble d'états terminaux. On présente les résultats suivants :

- On montre que pour une structure de transitions fixée à n états, le nombre d'ensembles d'états terminaux, pour lesquels l'automate obtenu est minimisé en plus de ℓ raffinements de partitions, est inférieur ou égal à $n^4 2^{n-\ell}$. Pour $\ell = \lceil 5 \log_2 n \rceil$, la proportion d'automates d'automates minimisés en plus de ℓ itérations est donc inférieure ou égale à $\frac{1}{n}$. On obtient ainsi une majoration de la complexité moyenne de l'algorithme, pour toute une famille de distributions, puisque l'on peut choisir n'importe quelle loi de probabilités sur l'ensemble des structures de transitions. En particulier, pour la distribution uniforme sur l'ensemble des automates déterministes accessibles et complets la complexité

moyenne de l'algorithme de Moore est $\mathcal{O}(n \log n)$.
- le résultat est encore valable dans les cas suivants : pour la distribution uniforme sur l'ensemble des automates possiblement incomplets, sur l'ensemble des automates émondés, ou si l'on attribue à chaque état une probabilité $x \in]0,1[$ d'être terminal.
- Pour la distribution uniforme sur l'ensemble des automates unaires, c'est-à-dire pour lesquels l'alphabet ne contient qu'une seule lettre, la borne supérieure est optimale : la complexité moyenne de l'algorithme de Moore est $\Theta(n \log n)$. Autrement dit, en l'absence d'information sur la distribution de probabilités sur l'ensemble des structures de transitions, cette majoration de la complexité moyenne de l'algorithme de Moore est optimale.
- On montre en revanche que cette méthode est inefficace dans le cas où l'on fixe le nombre d'états terminaux dans l'automate, indépendamment de sa taille. Pour la distribution uniforme sur l'ensemble des automates unaires ne contenant qu'une seul état, la complexité moyenne de l'algorithme de Moore est $\Theta(n^2)$.

Ces résultats ont été présentés dans [8, 9], les articles sont écrits en collaboration avec Frédérique Bassino et Cyril Nicaud.

Dans le chapitre 9, on étudie la complexité moyenne de l'algorithme de Moore en utilisant simultanément les propriétés des structures de transitions et des ensembles d'états terminaux. L'énumération des automates accessibles ayant des propriétés données étant un problème ouvert, on étudie la distribution uniforme sur l'ensemble des automates déterministes complets, pour un alphabet de taille fixée et supérieure ou égale à 2. On montre ainsi que la complexité moyenne est $\mathcal{O}(n \log \log n)$. L'idée est la suivante : pour la plupart des structures de transitions, le nombre d'automates associés à une structure et minimisés en plus de $\lceil \log_k \log_2 n^3 + 2 \rceil$ raffinements de partitions, est égal à $\mathcal{O}(\frac{2^n \log^2 n}{n})$. De plus, l'ensemble des automates associés aux structures de transitions restantes apporte une contribution négligeable à la moyenne. Cette partie est présenté dans l'article [30]. Il s'agit du résultat principal de cette thèse. Tout comme dans le chapitre précédent, on étend le résultat au cas où l'on attribue à chaque état la probabilité $x \in]0,1[$ d'être terminal. On conclut le chapitre en montrant que la complexité générique de l'algorithme de Moore est $\mathcal{O}(n \log \log n)$.

Dans le chapitre 10, on effectue une étude expérimentale et théorique de l'algorithme de Hopcroft. L'étude de la complexité est plus difficile que pour l'algorithme de Moore : l'algorithme de Hopcroft offre des choix dans l'implantation : il en existe un grand nombre possible pour un même automate. De plus, le raffinement de la partition est effectué par une méthode très différente de celle utilisée dans l'algorithme de Moore. C'est de cette différence que l'algorithme de Hopcroft tire son efficacité. On commence par donner une nouvelle description de l'algorithme, pour laquelle on peut dire, à un instant donné, que la partition est au moins aussi fine que celle obtenue par l'algorithme de Moore en un même nombre d'itérations. On utilise cette analogie pour montrer que la complexité moyenne de l'algorithme de Hopcroft, pour cette description, est $\mathcal{O}(n \log \log n)$. On conclut le chapitre par une étude expérimentale des implantations habituelles de l'algorithme. Ces résultats seront présentés dans une version longue de [30], en cours de préparation.

Première partie

Notions Fondamentales

Chapitre 1

Analyse combinatoire

Dans ce chapitre, on présente les deux thématiques étudiées dans cette thèse : la génération aléatoire d'objets combinatoires et l'analyse des algorithmes qui s'y appliquent. On s'intéressera aux motivations de ces axes de recherche, ainsi qu'aux principes, avantages et défauts de leurs méthodes classiques.

Dans la section 1, on présente trois familles de générateurs aléatoires : la première est celle des générateurs *ad hoc*, qui sont des générateurs conçus spécifiquement pour un objet donné. Ils sont généralement les plus performants. Les deux autres familles reposent sur des méthodes de spécification automatique de l'objet étudié : la méthode récursive, qui permet d'obtenir des générateurs exhaustifs ou aléatoires mais qui nécessite de lourds précalculs et beaucoup de mémoire et la méthode de Boltzmann, qui permet d'obtenir des générateurs aléatoires très efficaces mais où la taille de l'objet engendré n'est pas fixée. On ne détaille que très peu le fonctionnement de ces méthodes, mais on renvoie le lecteur intéressé vers les ouvrages de référence. Un ou plusieurs exemples seront donnés pour chaque type de générateurs. Par la suite, on fera référence à chacun de ces exemples.

Dans la section 2, on présente les différentes façon d'analyser un algorithme :

- l'étude du pire cas, le type d'analyse le plus répandu, auquel on fera constamment référence par la suite.
- l'étude de la complexité en moyenne, qui constitue le thème de la partie *III* de cette thèse.
- l'étude de la complexité amortie, auquel on aura recours dans le chapitre 5 de la partie *II* (Voir page 60).
- l'étude de la complexité générique, qui sera brièvement mentionnée dans le chapitre 9 de la partie *III* (Voir page 118).

1 Génération aléatoire

Lorsque l'on souhaite étudier les propriétés d'un objet combinatoire, ou l'un des algorithmes qui s'y appliquent, l'idéal serait de faire une étude expérimentale à l'aide d'un générateur exhaustif. On pourrait, par exemple, connaître exactement le nombre d'objets possédant un ensemble de propriétés, la quantité d'objets pour lesquels un algorithme s'exécute avec un coût fixé. En pratique, le nombre d'objets de taille n croît souvent très rapidement et une étude exhaustive devient rapidement impossible. La génération aléatoire offre un compromis : elle va permettre de tester

les propriétés moyennes des objets de plus grande taille, ainsi que le coût moyen des algorithmes. Les générateurs sont des outils de recherche très utiles : ils permettent d'établir des conjectures, ou d'en réfuter. Par exemple, dans les chapitres 3 et 4, on présente un ensemble de générateurs d'automates déterministes accessibles complets. Dans le chapitre 5, on présente des résultats expérimentaux obtenus grâce à ces générateurs, à partir desquels on émet des conjectures sur les propriétés moyennes des automates. Dans la partie *III*, on présente une étude théorique de la complexité moyenne des algorithmes de minimisation d'automates, qui a été à l'origine motivée par des résultats expérimentaux.

Dans cette section, on présente trois types de méthodes pour construire un générateur aléatoire. Dans le chapitre 4, on n'applique pas précisément l'une de ces méthodes, car les générateurs que l'on présente reposent sur des méthodes de combinatoire bijective. En revanche, chacun des algorithmes que l'on présente sera utilisé par la suite.

1.1 Méthodes *ad hoc*

Un algorithme *ad hoc* est, comme sa locution latine l'indique, un algorithme conçu sur mesure pour résoudre un problème précis. Sa fonction spécifique permet souvent d'obtenir un gain d'efficacité sur les méthodes génériques que l'on présente par la suite. En revanche, une méthode donnée n'est pas réutilisable pour d'autres objets et si les algorithmes *ad hoc* abondent pour certains objets combinatoires simples, d'autres structures plus complexes nécessitent à ce jour l'emploi de méthodes automatiques.

Les deux algorithmes que l'on décrit ici sont implantés dans les logiciels *REGAL* et *PREGA* (chapitre 5). On y fera donc référence par la suite.

Les Permutations

Une permutation de taille n est une suite d'entiers compris entre 1 et n, où chaque valeur apparaît exactement une fois. Le nombre de permutations de taille n est $n!$. L'algorithme de génération aléatoire que l'on présente ci-dessous est appelé *mélange de Knuth* [48].

Algorithme 1 : Permutation aléatoire

Données : un tableau P d'entiers de longueur n
pour tous les $i \in \{1, \ldots, n\}$ **faire**
  $P[i] = i + 1$
pour tous les $i \in \{1, \ldots, n\}$ **faire**
  On tire aléatoirement un entier j entre i et n
  On échange les valeurs de $P[i]$ et $P[j]$
Résultat : P

Le principe est le suivant : on tire le premier élément de la suite uniformément dans l'ensemble $\{1, \ldots, n\}$, puis on l'extrait de cet ensemble. Le deuxième élément sera extrait de l'ensemble de taille $n-1$, et ainsi de suite. Le i-ème élément est donc tiré uniformément parmi un ensemble de taille $n-i+1$. La probabilité d'engendrer

une permutation donnée de taille n est donc :

$$\mathbb{P}(x^n) = \prod_{i=1}^{n} \frac{1}{n-i+1} = \frac{1}{n!}.$$

Le générateur est donc bien uniforme.

Les arbres binaires

Un arbre binaire de taille n est un arbre possédant n nœuds internes dont le degré sortant est 2 et $n+1$ feuilles dont le degré sortant est 0. L'algorithme que l'on décrit ici est connu sous le nom d'algorithme de Rémy [64]. Il s'agit d'un algorithme incrémental, c'est-à-dire qu'on engendre un arbre de taille n à partir d'un arbre de taille $n-1$. Pour cela, on sélectionne un des nœuds de l'arbre (interne ou externe), que l'on note s , et on insère un nœud interne à la place. On choisit ensuite de placer s comme fils gauche ou comme fils droit du nouveau nœud.

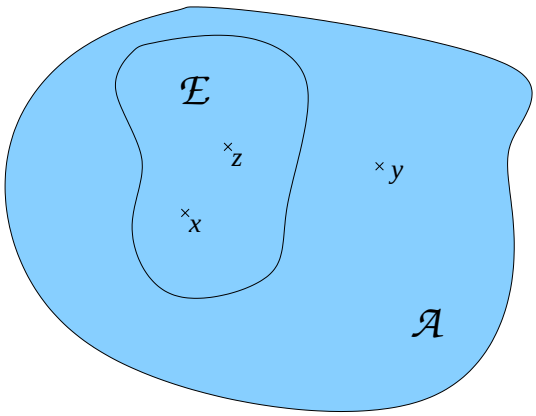
Algorithme 2 : *ArbreBinaireAleatoire*(n)

si $n = 1$ **alors**
  Résultat : 
ArbreBinaireAleatoire$(n-1)$
$s =$ Valeur aléatoire entre 1 et $2n-1$
direction = Valeur aléatoire entre 0 et 1
si *direction* $= 0$ **alors**
  Remplacer s par 
sinon
  Remplacer s par 

1.2 Algorithmes avec rejets

Il arrive parfois que les méthodes classiques soient inefficaces pour l'analyse d'un ensemble E d'objets combinatoires que l'on souhaiterait engendrer avec une loi de probabilités p_E. Une solution est toutefois possible si l'on possède un générateur aléatoire pour un ensemble A, tel que $E \subset A$ et tel que si l'on conditionne la loi de probabilités p_A afin que les objets choisis soient dans l'ensemble E, la loi obtenue est p_E. On peut alors utiliser une *méthode par rejet*. L'idée est la suivante : on engendre un objet de l'ensemble A et on teste si cet objet appartient également à l'ensemble E. Si ce n'est pas le cas, on rejette l'objet engendré et on recommence l'opération. La Figure 1.1 illustre le cas de la génération aléatoire uniforme.

L'efficacité d'une telle méthode dépends de plusieurs facteurs :
- l'efficacité du générateur sur l'ensemble A,
- la proportion d'objets de E dans A : plus celle ci est faible, et plus l'algorithme devra effectuer de rejets avant d'obtenir un résultat satisfaisant,
- le coût de la fonction qui teste l'appartenance d'un objet à l'ensemble E.



Si l'on possède un générateur uniforme sur l'ensemble A, on peut alors, avec la même probabilité engendrer les objets x, y, et z. Si l'on se restreint aux objets appartenant uniquement au sous-ensemble E, la probabilité d'engendrer x reste égale à celle d'engendrer z. On obtient donc un générateur uniforme sur l'ensemble E.

FIG. 1.1 – Méthode par rejet pour la génération uniforme

1.3 Méthode récursive

La méthode récursive fut introduite par Nijenhuis et Wilf [62] et systématisée par Flajolet, Van Cutsem et Zimmermann [36]. Le but avoué de cette méthode est d'automatiser la génération aléatoire en utilisant la décomposition combinatoire de l'objet. En effet, nombre d'objets combinatoires peuvent être décrits en utilisant les spécifications suivantes : ε désigne un élément de taille 0, z un élément de taille 1. On construit ensuite l'objet à l'aide des opérateurs union (noté $+$), produit (noté $\times$), séquence, ensemble, multiensemble, cycle. Une fois la spécification donnée, il est possible de construire automatiquement le générateur aléatoire. On donne une idée de ce principe avec l'exemple d'un générateur aléatoire d'arbres binaires.

Les arbres binaires

On rappelle qu'un arbre binaire de taille n est un arbre contenant n nœuds internes et $n+1$ feuilles. Il peut être décrit de la façon suivante : il s'agit soit d'une feuille, soit d'un nœud auquel sont reliés deux arbres binaires. La taille de l'arbre étant donné par son nombre de nœuds interne, sa spécification est :

$$\mathcal{B} = \varepsilon + z \times \mathcal{B} \times \mathcal{B}$$

auquel on associe la série génératrice ordinaire :

$$B(x) = 1 + xB(x)^2$$

À l'aide d'un dictionnaire que l'on ne décrit pas ici, on déduit automatiquement de la série génératrice que le nombre d'arbres de taille n est donné par :

$$B_n = \sum_{k=0}^{n} B_k B_{n-k-1}$$

et que la probabilité que le sous-arbre gauche d'un arbre de taille n soit de taille k est égale à :

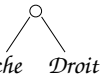
$$\frac{B_k B_{n-k-1}}{B_n}$$

La méthode récursive consiste à :

- précalculer tous les coefficients B_i pour tout $i \in \{1, \ldots, n\}$. On peut effectuer ce calcul récursivement.
- construire l'objet aléatoire récursivement, en tirant aléatoirement la taille des sous-arbres en utilisant la loi de probabilités donnée ci-dessus.

On remarque que le générateur décrit ici engendre des arbres non-étiquetés. Il suffit d'ajouter le tirage d'une permutation aléatoire pour obtenir un générateur d'arbres étiquetés. Le précalcul est très coûteux en temps et en espace mémoire. On préférera donc l'algorithme de Rémy à la méthode récursive dans ce cas. Cependant, il n'existe pas toujours d'algorithmes *ad hoc* pour engendrer des objets aléatoirement.

Algorithme 3 : $ArbreBinaireAleatoireRecursif(n)$

begin
  si $n = 0$ **alors**
    Résultat : $\square$
  $k = DeterminerTailleSousArbreGauche(n)$
  $Gauche = ArbreBinaireAleatoireRecursif(k)$
  $Droit = ArbreBinaireAleatoireRecursif(n - k - 1)$
  Résultat : 

end

Algorithme 4 : $DeterminerTailleSousArbreGauche(n)$

Données : Les cardinaux B_i, pour tout i de 1 à n
$alea$ = Valeur aléatoire en 0 et 1
$k = 0$
$somme = \frac{B_{n-1}}{B_n}$
tant que $somme < alea$ **faire**
  $k = k + 1$
  $somme = somme + \frac{B_k B_{n-k-1}}{B_n}$
Résultat : k

1.4 Méthode de Boltzmann

La méthode de Boltzmann est issue d'un travail de Duchon, Flajolet, Louchard et Schaeffer [31]. Elle permet de construire de façon systématique des générateurs aléatoires pour des classes d'objets étiquetés (une version pour les classes d'objets non-étiquetés est présentée dans [32] et pour les classes d'objets colorés dans [17]). Cette méthode n'engendre pas d'objets de taille fixe : elle dépend d'un paramètre réel $x > 0$ et, pour tout entier n, la valeur de x peut être choisie de telle sorte que la taille des éléments engendrés soit comprise entre $(1 - \varepsilon)n$ et $(1 + \varepsilon)n$ avec une forte probabilité. L'idée est la suivante : soit une classe d'objets combinatoires C. On note $C(z)$ la série génératrice exponentielle correspondante. Un générateur de Boltzmann engendre un objet c de la classe C avec la probabilité suivante :

$$\mathbb{P}_x(c) = \frac{1}{C(x)} \frac{x^{|c|}}{|c|!}$$

où x est le paramètre réel qui permet de "viser" la taille de l'objet c. Les générateurs de Boltzmann garantissent donc que deux éléments de même taille ont la même probabilité d'être engendrés.

L'espérance de la taille d'un élément engendré est donnée par :

$$\mathbb{E}_x(taille) = x\frac{C'(x)}{C(x)}.$$

Choisir le paramètre x de façon à ce que les éléments engendrés soient de taille moyenne n, revient à résoudre l'équation :

$$x\frac{C'(x)}{C(x)} = n$$

Pour obtenir un générateur en taille exacte, on utilise ensuite un algorithme par rejet.

Partitions d'un ensemble

Soit k et n deux entiers positifs. On donne ici l'exemple d'un générateur de Boltzmann permettant d'engendrer des partitions d'un ensemble de taille kn en n sous-ensembles non-vides. Ce générateur sera repris tout au long de la partie II.

La série génératrice exponentielle d'un ensemble non vide est $e^z - 1$ et celle des partitions d'un ensemble en n sous-ensembles non-vides est $P_n(z) = \frac{(e^z-1)^n}{n!}$, c'est-à-dire le produit de n éléments, où le rapport $\frac{1}{n!}$ permet de supprimer la notion d'ordre entre les n sous-ensembles. On a :

$$\mathbb{E}_x(\text{taille de la partition}) = x\frac{P'_n(x)}{P_n(x)} = nx\frac{e^x}{e^x-1}.$$

On souhaite obtenir des partitions d'un ensemble dont la taille moyenne est kn. On résout donc l'équation :

$$x\frac{e^x}{e^x-1} = k.$$

Le nombre de sous-ensembles n'apparaît pas dans l'équation : on tire aléatoirement la taille des n sous-ensembles dont l'espérance est égale à k. La probabilité de tirer un ensemble de taille k, pour $k > 0$, est égale à :

$$\frac{1}{e^x-1}\frac{x^k}{k!}$$

On reconnaît la loi de Poisson non nulle, c'est-à-dire la Loi de Poisson restreinte aux valeurs strictement positives (voir [31] pour plus de précisions). On décrit à présent l'algorithme de génération aléatoire de partitions d'un ensemble utilisant la méthode de Boltzmann.

2 Analyse d'algorithmes

Un algorithme est une méthode permettant de calculer une ou plusieurs solutions à un type de problème donné, en un nombre fini d'étapes. Il existe un large éventail

Algorithme 5 : Générateur de Boltzmann pour les partitions d'un ensemble de taille kn en n sous-ensembles non vides

```
Données : un tableau T
1  begin
2      Calcul du paramètre x
3      tant que la somme des T[i] est différente de kn faire
4          pour i de 1 à n faire
5              T[i]= valeur aléatoire tirée avec une loi de Poisson non nulle de
               paramètre x
6      Permutation = permutationAleatoire(kn)
7      i = 0
8      pour j < n faire
9          pour l de 1 à T[j] faire
10             i = i + 1
11             Partition[Permutation[i]] = j
12 end
Résultat : Partition
```

de façons d'analyser un algorithme. De manière générale, on identifie les différents types de ressources employées pour résoudre un problème, et on tente d'évaluer l'ordre de grandeur des quantités de ressources utilisées. Les types de ressources les plus étudiés sont le temps et l'espace mémoire, mais on pourrait imaginer une analyse du coût monétaire des solutions proposées, de la quantité d'énergie nécessaire pour effectuer le calcul, etc.

Dans le cas présent, on s'intéressera exclusivement à la *complexité en temps* des algorithmes, c'est-à-dire l'estimation du temps de calcul, indépendamment de l'ordinateur ou du langage de programmation utilisé. Dans cette section, on commence par rappeler un ensemble de notations usuelles et puisqu'il existe différents types d'étude de la complexité d'un algorithme, on en présentera plusieurs variantes, dont chacune sera évoquée par la suite.

2.1 Comparaisons de fonctions

Soient f et g deux fonctions dépendant d'un même paramètre n. On définit trois notations permettant de comparer les comportements asymptotiques des deux fonctions.

$$f(n) = \mathcal{O}(g(n)) \iff \exists c \in \mathbb{R}^+,\ \exists n_0 \in \mathbb{N},\ \text{tel que } \forall n \geq n_0,\ f(n) \leq cg(n)$$

$$f(n) = \Omega(g(n)) \iff \exists c \in \mathbb{R}^+,\ \exists n_0 \in \mathbb{N},\ \text{tel que } \forall n \geq n_0,\ f(n) \geq cg(n)$$

$$f(n) = \Theta(g(n)) \iff f(n) = \mathcal{O}(g(n)) \text{ et } f(n) = \Omega(g(n))$$

2.2 Étude de la complexité dans le pire cas.

Il s'agit de la méthode la plus répandue pour évaluer l'efficacité d'un algorithme. Un ouvrage de référence est [1]. Le principe est le suivant : on évalue l'efficacité d'un algorithme en fonction d'une estimation du temps nécessaire pour la pire exécution possible. Cette étude comporte deux parties :

- Trouver une fonction f de paramètre n, où n est la taille des objets sur lesquels on applique l'algorithme, telle que pour toutes les entrées possibles de l'algorithme, le temps d'exécution est égal à $\mathcal{O}(f(n))$. On parle alors de borne supérieure du temps d'exécution de l'algorithme.
- Montrer que cette borne est bien atteinte, c'est-à-dire qu'il existe bien une entrée pour laquelle le temps d'exécution est $\Omega(f(n))$. En effet, il arrive que la borne supérieure soit une majoration grossière du temps d'exécution dans le pire des cas. Par suite, le résultat peut encore être raffiné. De plus, selon le contexte, un algorithme conçu pour s'appliquer sur un ensemble d'objets E, peut ne pas avoir le même pire des cas lorsque l'on restreint l'ensemble des entrées à un sous-ensemble $E' \subset E$.

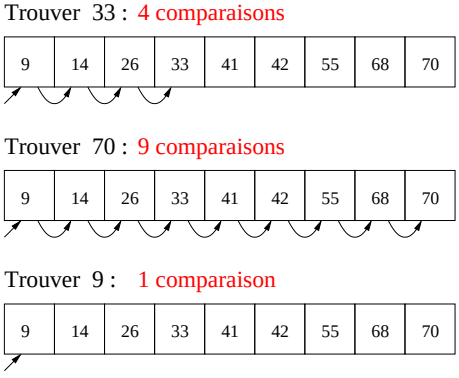
Il est également possible d'effectuer une étude similaire pour le meilleur cas. Cette démarche est utile pour mieux comprendre l'algorithme et nécessaire pour une analyse en moyenne, mais on ne lui accordera pas la même importance qu'au pire des cas pour déterminer l'efficacité de l'algorithme.

Exemple : la recherche dans une liste triée. On présente deux algorithmes permettant de chercher une valeur x dans un tableau trié de taille n. Le premier parcoure naïvement le tableau de gauche à droite jusqu'à trouver la bonne valeur (ou jusqu'à la fin du tableau, si x ne s'y trouve pas). Le second effectue une recherche dite *dichotomique*. Elle utilise à son avantage le contexte de la liste triée de la façon suivante : si la valeur x est plus grande que celle contenue dans une case c, alors x ne se trouve pas à gauche de c dans le tableau et si la valeur x est plus grande, alors elle ne se trouve pas à droite de c.

Algorithme 6 : Naïf

Données : Un tableau trié T et une valeur x

```
1 begin
2     pour i entre 1 à n faire
3         si T[i] = x alors
4             Renvoyer la position i;
5     Renvoyer faux;
6 end
```



Pour la recherche naïve, dans le meilleur des cas, la valeur recherchée est dans

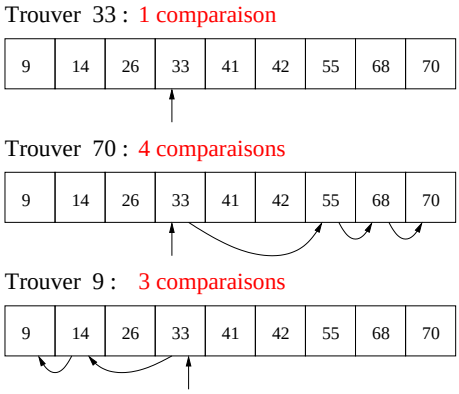
la première case : une seule comparaison est effectuée. Cependant, il faut au moins une comparaison pour le vérifier. La complexité dans le meilleur cas est donc $\Theta(1)$.

Dans le pire des cas, la valeur est dans la dernière case du tableau, ou n'y est pas du tout. On effectue alors autant de comparaisons qu'il y a de valeurs dans le tableau. L'un des exemples ci-dessus montre que ce pire cas est atteint. La complexité dans le pire des cas est donc $\Theta(n)$.

Algorithme 7 : Dichotomie

Données : Un tableau trié T et une valeur x

```
1  begin
2      debut = 1, fin = n;
3      tant que debut < fin faire
4          milieu = ⌊(debut+fin)/2⌋;
5          si T[milieu] = x alors
6              Renvoyer la position
               milieu;
7          si T[milieu] < x alors
8              fin = milieu − 1;
9          si T[milieu] > x alors
10             debut = milieu + 1;
11     si T[debut] = x alors
12         Renvoyer la position milieu;
13     sinon
14         Renvoyer faux;
15 end
```



Pour la recherche dichotomique, tout comme pour l'algorithme naïf, la complexité dans le meilleur cas est $\Theta(1)$. Ce cas se produit lorsque la valeur recherchée est au milieu du tableau.

A chaque étape de l'algorithme, on divise le nombre de positions possibles dans le tableau par 2. La complexité pire cas est donc $\mathcal{O}(\log n)$. L'exemple ci-dessus consistant à trouver 70 atteint le pire cas. La complexité pire cas est donc $\Theta(\log n)$.

L'étude de la complexité dans le pire des cas a permis d'établir une hiérarchie entre les deux algorithmes : l'algorithme effectuant une recherche dichotomique est plus efficace dans le pire cas que celui effectuant une recherche naïve.

Remarque 1. *L'étude d'un algorithme dans le pire des cas a l'avantage d'être en général plus simple car moins fine que les autres types d'analyse. En revanche il existe des algorithmes pour lesquels le pire cas et le meilleur cas se produisent rarement et ne reflètent pas le temps d'exécution en pratique.*

2.3 Analyse en moyenne

L'analyse en moyenne apporte un complément d'information sur le comportement de l'algorithme. Bien que beaucoup moins populaire que l'étude du pire cas,

son utilisation est recommandée par Donald Knuth [47–49] et il existe de nombreux ouvrages sur le sujet [33–35, 67].

Soit E_n l'ensemble des entrées de taille n d'un algorithme. Soit $e \in E_n$ un objet de taille n, on note $c(e)$ le coût de l'exécution de l'algorithme prenant e comme entrée. Pour une distribution de probabilités donnée sur E_n, on note $\mathbb{P}(e)$ la probabilité de tirer un objet e. Le coût moyen de l'algorithme est égal à :

$$\sum_{e \in \mathcal{E}_n} \mathbb{P}(e) \times c(e).$$

On s'intéressera principalement à la complexité en moyenne d'un algorithme lorsque celle-ci diffère de la complexité dans le pire des cas.

Remarque 2. *La distribution de probabilités sur l'ensemble des entrées pouvant varier du tout au tout selon le contexte, les résultats d'une analyse peuvent devenir caducs lorsque ce dernier change. De plus, il est difficile d'une part de modéliser une situation donnée, c'est-à-dire d'obtenir la distribution de probabilités dans un contexte précis, et d'autre part d'analyser la complexité moyenne d'un algorithme dans le cas non-uniforme. L'analyse en moyenne sera donc souvent réalisée pour la distribution uniforme. Dans le chapitres 7, on prendra soin d'analyser l'algorithme de Moore pour différentes distributions de probabilités.*

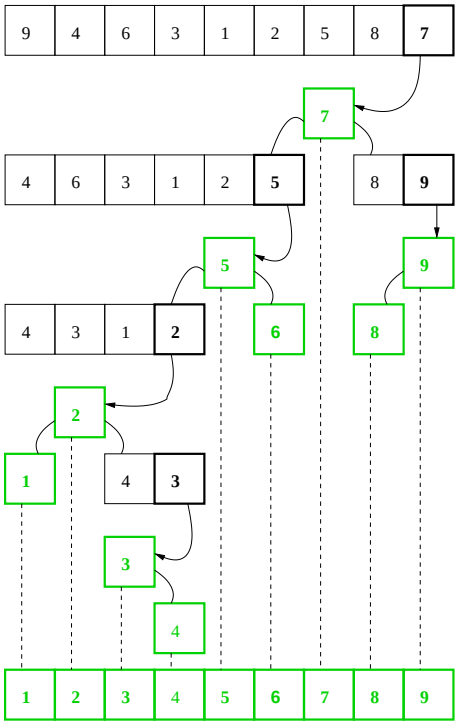
Remarque 3. *Il est tout à fait possible que le coût moyen d'un algorithme ne soit pas du tout représentatif du temps d'exécution pratique, quand bien même la distribution de probabilité serait réaliste. Supposons par exemple qu'avec probabilité $\frac{1}{n}$, le coût de l'algorithme soit $\Theta(2^n)$, et que pour tout autre exécution, le coût soit $\Theta(n)$. La complexité moyenne de l'algorithme sera $\Theta\left(\frac{2^n}{n}\right)$. Pourtant le temps d'exécution de l'algorithme est presque sûrement $\Theta(n)$. On verra que le problème ne se pose pas pour les algorithmes qu'on analyse ici.*

Exemple : Quicksort. L'algorithme *Quicksort* [42] est un algorithme de tri efficace, dont la complexité dans le pire cas est quadratique, alors que la complexité moyenne est $\Theta(n \log n)$. Les études sur la complexité de cet algorithme étant abondantes [33, 34, 42, 49, 68], on donnera simplement au lecteur une vague idée de son fonctionnement et de son analyse.

L'algorithme consiste à choisir un pivot, c'est-à-dire une valeur quelconque dans le tableau, puis à partitionner ce dernier en plaçant d'un côté les valeurs inférieures au pivot et de l'autre les valeurs supérieures ou égales. La complexité de cette opération est linéaire en la taille du tableau à trier. Dans la Figure 2.3, la partition du tableau est faite au cours des lignes 3 à 7. Le pivot choisi dans cette description est toujours la dernière case. On aurait cependant pu prendre n'importe quelle valeur. Une fois le partitionnement terminé, le pivot est à la "bonne place" et on relance l'algorithme avec les deux sous-tableaux restants. La complexité de l'algorithme dépend donc du nombre et de la taille des sous-tableaux que l'algorithme va traiter. Dans le pire des cas, le pivot choisi est toujours la valeur minimale ou maximale du tableau courant : le calcul se poursuit sur un seul sous-tableau, contenant toutes les valeurs sauf le pivot. La complexité de l'algorithme, dans un tel cas, est quadratique. En revanche, pour la distribution uniforme sur toutes les permutations possibles, le pivot permet d'obtenir deux sous-tableaux de tailles asymptotiquement égales.

Algorithme 8 : *Quicksort*$(Tableau\ T,\ debut,\ fin)$

```
1  begin
2      si debut < fin alors
3          i = debut;
4          pour j de debut à fin − 1
           faire
5              si T[j] < T[fin] alors
6                  Echanger T[j] et
                   T[i];
7                  j = j + 1;
8          Echanger T[j] et T[dernier];
9          Quicksort(T, j + 1, fin);
10         Quicksort(T, debut, j − 1);
11 end
```



Tout comme dans la recherche dichotomique, on va réappliquer l'algorithme sur des sous-tableaux de taille $\frac{n}{c}$, où c est une constante. La complexité moyenne est alors $\Theta(n \log n)$. Il est de plus possible de mélanger les valeurs d'un tableau, en temps linéaire, afin d'obtenir cette distribution uniforme.

2.4 Autres types d'analyse

Comme on l'a vu dans les remarques 1, 2 et 3, connaître la complexité d'un algorithme dans le pire cas et dans le cas moyen n'est pas suffisant pour connaître l'efficacité d'un algorithme en pratique. On décrit brièvement d'autre types d'analyse qui, en s'ajoutant aux précédentes, permettent d'accéder à une meilleure compréhension de la méthode étudiée.

Complexité générique. La notion de complexité générique est définie de façon précise dans [46]. Elle est basée sur les notions d'ensembles **négligeables** et d'ensembles **génériques**. Soit E l'ensemble des instances d'un problème. On détermine un paramètre de taille sur $\mathcal{E}$: soit E_n l'ensemble des instances de taille n et $B_n = \bigcup_{i \leq n} E_i$ l'ensemble des instances de tailles au plus n. Soit X un ensemble d'instances. Pour une distribution de probabilités fixée, on dit que X est **négligeable** si et seulement si :

$$\text{pour } e \in B_n,\ \lim_{n \to +\infty} \mathbb{P}(e \in X) = 0,$$

ce qui, pour la distribution uniforme, est équivalent à

$$\lim_{n \to +\infty} \frac{|X \cap B_n|}{|B_n|} = 0,$$

et que X est **générique** si et seulement si :

$$\text{pour } e \in B_n,\ \lim_{n \to +\infty} \mathbb{P}(e \in X) = 1,$$

ce qui, pour la distribution uniforme, est équivalent à

$$\lim_{n\to+\infty} \frac{|X \cap B_n|}{|B_n|} = 1,$$

La complexité générique d'un algorithme correspond à la complexité dans le pire cas d'un ensemble générique d'instances de l'algorithme. On préférera par exemple connaître la complexité générique à la complexité moyenne dans le cas évoqué dans la remarque 3.

Remarque 4. *Dans le cas particulier ou l'on a :*

$$\textit{pour } e \in B_n,\ \lim_{n\to+\infty} \mathbb{P}(e \in B_{n-1}) = 0,$$

pour montrer qu'un ensemble X est générique, il suffit de montre que

$$\textit{pour } e \in E_n,\ \lim_{n\to+\infty} \mathbb{P}(e \in X) = 1.$$

De façon générale, si on a un ensemble X tel que :

$$\textit{pour } e \in E_n,\ \lim_{n\to+\infty} \mathbb{P}(e \in X) = 0.$$

alors X est négligeable.

Remarque 5. *Si un ensemble X est négligeable, alors l'ensemble $E\backslash X$ est générique. Si un ensemble X est générique, alors l'ensemble $E \setminus X$ est négligeable.*

Complexité amortie. La complexité amortie, contrairement aux définitions précédentes, ne s'intéresse pas aux ressources utilisées par un algorithme pour une entrée donnée, mais à la somme des ressources utilisées par une suite d'exécutions de cet algorithme.

Prenons l'exemple de la saisie d'un texte dans une chaîne de caractères. A chaque fois qu'un utilisateur rentre un caractère, ce dernier est ajouté dans un tableau. Si la quantité de mémoire allouée pour le tableau est suffisante, on ajoute simplement le caractère et le coût de l'opération est $\theta(1)$. En revanche, si le tableau est plein, il faut au préalable réallouer un tableau dans la mémoire puis recopier les valeurs de l'ancien dans le nouveau. Dans les implantations classiques, la taille du nouveau tableau est égale au double de taille de l'ancien. La complexité de l'opération est alors $\mathcal{O}(n)$, où n est la taille de l'ancien tableau. Somme toute, la complexité dans le pire cas de l'ajout d'un caractère dans un tableau est $\mathcal{O}(n)$.

Soit m le nombre de caractères à ajouter. Le nombre d'ajouts dont le coût est $\mathcal{O}(1)$ est égal à $m - \log(m)$. La complexité amortie est donc majorée par :

$$\frac{m - \log(m) + \sum_{i=\{2,4,\ldots,\log(m)\}} i}{m} = \mathcal{O}(1)$$

On dit alors que la complexité amortie de l'opération d'ajout de caractères est $\mathcal{O}(1)$ et que la complexité de l'ajout d'une chaîne de caractère est $\mathcal{O}(n)$.

Chapitre 2

Langages et automates

Dans ce chapitre, on présente uniquement les notions de théorie des automates qu'on utilise par la suite. Le lecteur souhaitant approfondir les notions évoquées dans ce chapitre pourra consulter les ouvrages [14, 66]. Le lecteur intéressé par l'histoire des sciences pourra consulter l'article de Dominique Perrin [63], qui décrit les origines de la théorie de automates.

1 Langages

1.1 Alphabet et Mot

Un *alphabet* est un ensemble, le plus souvent fini, dont les éléments sont appelés des *lettres*. Un *mot* est une suite finie de lettres. Ainsi pour un alphabet A, un mot w *défini* sur A pourra donc s'écrire :

$$w = a_1 a_2 \dots a_n,$$

où tous les a_i sont des lettres de A. On note A^* l'ensemble des mots définis sur un alphabet A.

La *longueur d'un mot* w se note $|w|$: il s'agit du nombre de lettres qui constituent le mot. On nomme *mot vide* le mot qui ne contient aucune lettre ou, dit autrement, qui est de longueur 0. Dans la littérature, il peut être noté ε ou 1. Nous utiliserons ici la première notation, par atavisme. L'ensemble des mots de longueur n sur l'alphabet A est noté A^n.

Le produit de *concaténation* de deux mots $u = a_1 a_2 \dots a_n$ et $v = b_1 b_2 \dots b_m$ désigne le mot uv, obtenu en écrivant les deux mots l'un à la suite de l'autre. Ainsi, on aura :

$$uv = a_1 a_2 .. a_n b_1 b_2 .. b_m \text{ et } |uv| = |u| + |v|$$

1.2 Langages formels et langages rationnels

Un langage formel est un sous-ensemble de A^*. On définit trois opérations sur les langages formels. Soient X et Y deux langages :

– leur *union* est l'ensemble :

$$X \cup Y = \{z \mid z \in X \text{ ou } z \in Y\},$$

que l'on note également $X + Y$,

– leur *produit de concaténation* est l'ensemble :

$$XY = \{xy \mid x \in X,\ y \in Y\},$$

– l'étoile de X est le langage :

$$X^* = \{x_1 x_2 \dots x_n \mid \forall n \geq 0,\ x_1, x_2, \dots, x_n \in X\}.$$

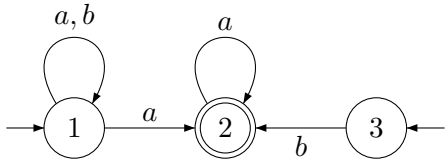
L'ensemble des *langages rationnels* est la plus petite famille de langages formels contenant les langages finis à être stable pour ces trois opérations. Ses éléments sont obtenus à partir des langages finis et d'un nombre fini d'utilisation des trois opérations précédentes.

2 Automates finis

Les automates permettent de modéliser l'évolution d'un système dans le temps. Ces automates étant finis, on admettra que le temps est discret, c'est-à-dire que chaque événement permettant de modifier l'état du système peut se décomposer en une suite finie de sous-événements, lesquels se réalisent dans une même unité de temps. La modélisation du système ne permet qu'un nombre fini de configurations, dont le comportement peut être décrit avec une mémoire finie.

2.1 Définition

Un automate fini est défini par un quintuplet $\mathcal{A} =< Q, A, E, I, F >$, où Q représente l'ensemble des états de l'automate, A est l'alphabet sur lequel l'automate est défini, $E \subset Q \times A \times Q$ est un ensemble de *transitions*, $I \subseteq Q$ est un sous-ensemble d'états dits *initiaux* et $F \subseteq Q$ est un sous-ensemble d'états dits *terminaux*.



Représentation sous forme de graphe

	a	b	I	F
1	$\{1,2\}$	$\{2\}$	1	0
2	$\emptyset$	$\{2\}$	0	1
3	$\{2\}$	$\emptyset$	1	0

Représentation sous forme de table

FIG. 2.1 – Automate $\mathcal{A} =< Q, A, E, I, F >$ tel que $Q = \{1, 2, 3\}$, $A = \{a, b\}$, $I = \{1, 3\}$, $F = \{2\}$, $E = \{(1, a, 1), (1, b, 1), (1, a, 2), (2, a, 2), (3, b, 2)\}$

Un *chemin* est une suite finie de transitions consécutives. Un *chemin réussi* est un chemin dont l'état de départ est un état initial et l'état d'arrivée est un état terminal. Un mot est *accepté* ou reconnu par un automate s'il étiquette un chemin réussi de cet automate.

Le *langage reconnu* par un automate est l'ensemble des mots reconnus par cet automate.

Théorème 6 (Kleene). *L'ensemble des langages rationnels est l'ensemble des langages reconnaissables par un automate fini.*

2.2 Propriétés des automates finis

La suite de ce chapitre présente un ensemble de propriétés associées aux automates finis. On utilisera l'ensemble de ces notions par la suite.

Automates équivalents

On dit de deux automates qu'ils sont équivalents s'ils reconnaissent le même langage.

Automates isomorphes

On dit de deux automates $\mathcal{A} =< Q_{\mathcal{A}}, A, E_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}} >$ et $\mathcal{B} < Q_{\mathcal{B}}, A, E_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}} >$ qu'ils sont isomorphes s'il existe une bijection $\varphi : Q_{\mathcal{A}} \mapsto Q_{\mathcal{B}}$ telle que :

- $\varphi(I_{\mathcal{A}}) = I_{\mathcal{B}}$,
- $\varphi(F_{\mathcal{A}}) = F_{\mathcal{B}}$,
- pour tout état $p \in Q_{\mathcal{A}}$ et toute lettre $a \in A$, on a $(p, a, q) \in E_{\mathcal{A}} \iff (\varphi p, a, \varphi q) \in E_{\mathcal{B}}$.

En d'autres termes, deux automates isomorphes ne diffèrent que par l'étiquetage de leur états.

FIG. 2.2 – Deux automates isomorphes

Automates déterministes

Un automate est déterministe si

- il possède un unique état initial,
- pour tout état $q \in Q$, et toute lettre $a \in A$, il existe au plus une transition sortant de l'état p étiquetée par la lettre a.

On peut ainsi définir la *fonction de transition* $\cdot$ de l'automate par :

$$Q \times A \mapsto Q$$

Un automate déterministe est défini comme un quintuplet $< Q, A, q_0, F, \cdot >$, où q_0 est l'unique état initial. On peut étendre cette fonction de transition aux mots. Ainsi pour tout état $q \in Q$, on a

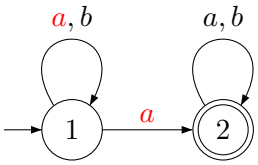
$$q \cdot \varepsilon = q$$

On étend ensuite la fonction de façon récursive avec, pour tout mot $w \in A^*$, et toute lettre $a \in A$:

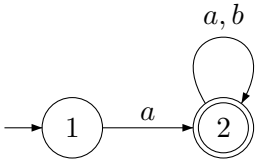
$$q \cdot wa = (q \cdot w) \cdot a$$

En résumé, on peut dire que $q \cdot w = p$, s'il existe un chemin étiqueté par le mot w partant de q et se terminant en p.

Théorème 7 (Myhill)**.** *Pour tout automate fini non déterministe, il existe un automate fini déterministe reconnaissant le même langage.*



Automate non-déterministe



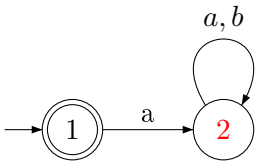
Automate déterministe

Automates accessibles, co-accessibles, émondés

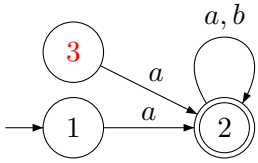
Un état p de l'automate est dit accessible s'il existe un chemin partant d'un état initial de l'automate passant par p. De la même façon, on dira que p est co-accessible s'il existe un chemin dans l'automate partant de p et passant par un état terminal.

Un automate est *accessible* si tout ses états sont accessibles et *co-accessible* si tous ses états sont co-accessibles.

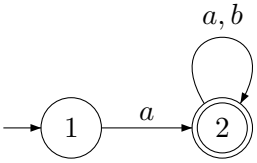
Enfin, un automate est émondé s'il est à la fois accessible et co-accessible.



Automate accessible et non-coaccessible



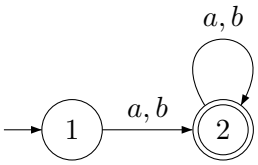
Automate non-accessible et co-accessible



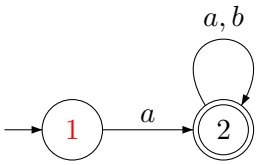
Automate émondé

Automates complets et incomplets

Un automate est *complet* si, pour tout état q de l'automate et pour toute lettre a de l'alphabet, la transition sortant de p étiquetée par a est définie. Si l'automate est déterministe, cela signifie simplement que la fonction de transition est totale.



Automate complet



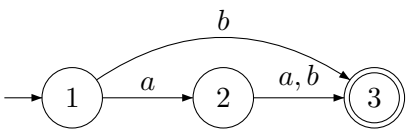
Automate incomplet

Automates acyclique

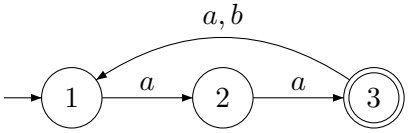
Un automate est acyclique si, pour tout état p de l'automate, il n'existe pas de mot $w \in A^*$, tel que $p \cdot w = p$.

Automates fortement connexes

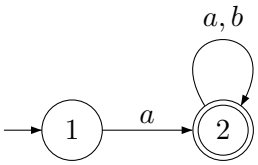
Un automate est *fortement connexe* si pour toute paire d'états (p, q) de l'automate, il existe un chemin partant de p et terminant passant par q.



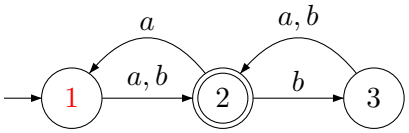
Automate acyclique



Automate cyclique



Automate non fortement connexe



Automate fortement connexe

Automates minimaux

La notion d'automate minimal est approfondie dans le Chapitre 6. Cet automate est une représentation canonique d'un langage rationnel, puisqu'il s'agit de l'unique, à isomorphisme près, automate déterministe accessible complet à posséder le nombre minimal d'états et qui reconnaît un langage donné.

Récapitulatif

La Figure 2.3 donne un aperçu des relations d'ordre entre ensembles d'automates déterministes.

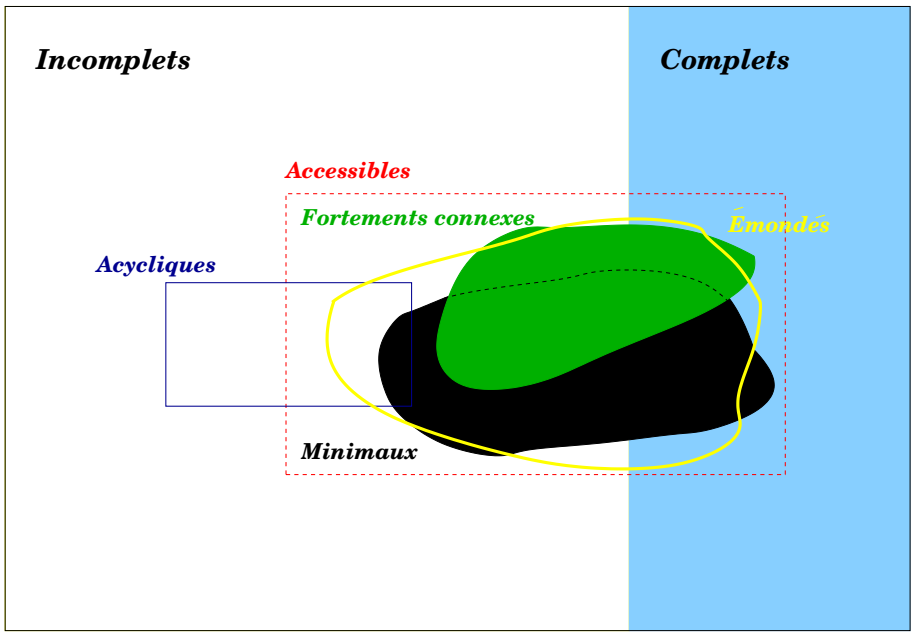


FIG. 2.3 – Récapitulatif des sous-ensembles de l'ensemble des automates déterministes possiblement incomplets.

3 Structures de transitions

La *structure de transitions* d'un automate $\mathcal{A} =< Q, A, E, I, F >$ est le quadruplet $\tau =< Q, A, E, I >$. Cela revient à ignorer les états terminaux. Cette notion est introduite dans [60] afin de faciliter l'énumération puis la génération d'automates déterministes accessibles complets sur un alphabet de taille 2. Un automate peut ainsi être désigné par un couple (τ, F) et le nombre d'automates associés à une structure de transitions donnée est 2^n, le nombre d'ensembles d'états terminaux.

Notons que cette notion est à distinguer de celle de *semi-automate*, pour laquelle on ignore l'ensemble des états initiaux en plus de celui des terminaux : la combinatoire n'est pas la même.

Dans les Chapitres 3 et 4, les structures de transitions que l'on utilise sont déterministes et accessibles. Dans le Chapitre 7, la condition d'accessibilité n'est pas toujours garantie. On précisera alors dans quel cas on se trouve. Les structures de transitions sont toujours déterministes, tout comme les automates que l'on étudie. Ce sont donc des quadruplets $\tau =< Q, A, \cdot, q_0 >$. Le lecteur pourra considérer que cette caractéristique est implicite.

On note $\mathcal{C}_n$ l'ensemble des structures de transitions déterministes **accessibles complètes non-isomorphes**. Dans le chapitre 3, page 44, on donne l'équivalent asymptotique de $|\mathcal{C}_n|$, établi par Korshunov dans [53] et reformulé par Bassino et Nicaud dans [10].

On note $\mathcal{I}_n$ l'ensemble des structures de transitions déterministes **accessibles non-isomorphes**. Dans le chapitre 4, page 54, on donne l'équivalent asymptotique de $|\mathcal{I}_n|$.

On note $\mathcal{T}_n$ l'ensemble des structures de transitions déterministes **complètes**. Si l'alphabet contient k lettres et les automates n états, on a alors n^{kn} structures de transitions possibles et $2^n n^{kn}$ automates associés.

Deuxième partie

Génération aléatoire et exhaustive d'automates finis

Chapitre 3

État de l'art

Ce chapitre contient une présentation des méthodes de générations aléatoires et d'énumération d'automates déterministes accessibles complets non-isomorphes.

On commence par donner au lecteur néophyte une intuition de la difficulté de ce problème ainsi que de son intérêt, en le comparant à un problème plus simple.

Les automates déterministes complets, si l'on s'autorise à avoir des automates isomorphes et non nécessairement accessible, sont à la fois simples à énumérer et à engendrer : le nombre de structures de transitions est égal au produit du nombre d'états d'arrivées possibles pour chaque transition. Si l'alphabet contient k lettres et les automates n états, on a alors n^{kn} structures de transitions possibles et $2^n n^{kn}$ automates. Pour engendrer un automate aléatoirement, il suffit, pour toutes les transitions de l'automate, de choisir uniformément parmi $\{1, \ldots, n\}$ l'état d'arrivée.

La difficulté apparaît lorsque l'on ajoute la condition d'accessibilité : pour toute transition de l'automate, l'ensemble des états d'arrivée possible dépend de l'état d'arrivée des autres transitions de l'automate. Les méthodes que l'on présente reposent sur deux bijections (section 2) :

- une bijection entre l'ensemble des structures de transitions déterministes accessibles complètes non-isomorphes et un ensemble de diagrammes,
- une bijection entre un ensemble de diagrammes, moins contraint que le premier, et un ensemble de partitions d'ensemble.

Les premiers algorithmes proposés pour la génération aléatoire d'automates déterministes utilisaient la méthode récursive présentée dans le chapitre 1. Plus récent, l'algorithme de Bassino-Nicaud (section 4) utilise la méthode de Boltzmann (également présentée dans le chapitre 1) pour engendrer des partitions d'ensembles, puis utilise les bijections mentionnées ci-dessus.

On commence par donner la définition des objets pour lesquels les bijections ont été établies (section 1).

1 Définitions

1.1 Automate de Base

Un *chemin simple* dans un automate déterministe accessible complet $\mathcal{A} =< Q, A, \cdot, q_0, F >$ est un chemin étiqueté par un mot u tel qu'il n'existe pas deux préfixes distincts de u, notés u_o et u_1 tels que $q_0 \cdot u_0 = q_0 \cdot u_1$. Dit autrement, il

s'agit d'un chemin partant de l'état initial et qui ne passe jamais deux fois par le même état. On définit l'application w de Q dans A^* de la manière suivante : pour tout état $q \in Q$ par :

$$w(q) = min_{lex}\{u \in A^* \mid q_0.u = q \text{ et } u \text{ est un chemin simple}\},$$

où $\min_{lex}$ désigne le minimum selon l'ordre lexicographique.
Le mot $w(q)$ existe toujours car l'automate est accessible.

On définit l'*automate de base* l'automate $\mathcal{A} =< Q, A, \cdot, q_0, F >$ où tous les états de Q ont été étiquetés par des mots (on a donc $Q \subset A^*$) en utilisant la règle suivante : pour tout état $u \in Q$, $w(u) = u$. Deux automates de bases distincts ne peuvent donc pas être isomorphes. Les techniques de génération aléatoire d'automates déterministes accessibles complets non-isomorphes vont donc s'intéresser aux automates de base. On notera *structure de transitions de base* la structure de transitions d'un automate de base.

On définit *l'arbre couvrant* d'un automate de base, comme la structure de transitions suivante :

- ses états sont ceux de l'automate de base,
- pour tout état $w(q)$, il existe une transition sortante étiquetée par une lettre a dont l'état d'arrivée est $w(p)$ si et seulement si $w(p) = w(q) \cdot a$.

La Figure ci-dessous illustre ces définitions.

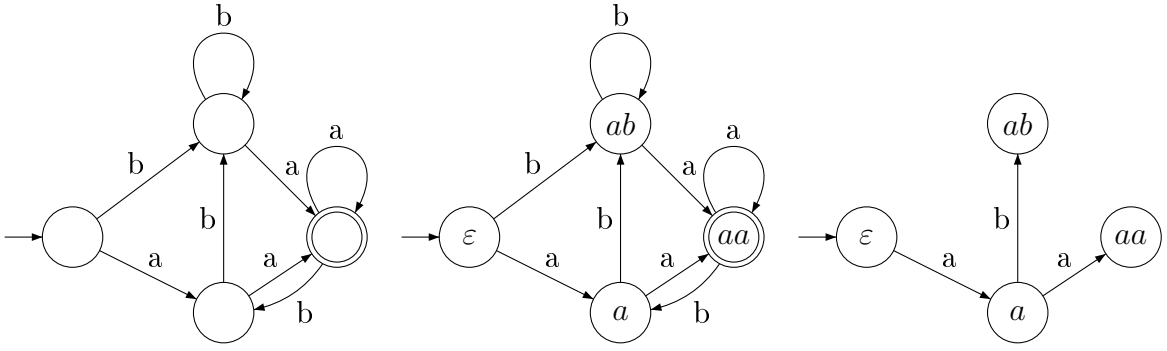


FIG. 3.1 – Un automate, son automate de base et son arbre couvrant

1.2 Diagrammes marqués et Diagrammes k-Dyck

Un *diagramme* de longueur m et de hauteur n est une suite $(x_1, \ldots, x_m)$ d'entiers croissants non négatifs pour laquelle $x_m = n$, que l'on représente habituellement par des diagrammes de boîtes, comme sur la Figure 3.2.

Un *diagramme k-Dyck* de taille n est un diagramme de longueur $(k-1)n+1$ et de hauteur n tel que :

$$\text{pour tout } i \leq (k-1)n,\ x_i \geq \lceil i/(k-1) \rceil.$$

Un *diagramme marqué* est une paire de suites $((x_1, \ldots, x_m), (y_1, \ldots, y_m))$ où $(x_1, \ldots, x_m)$ est un diagramme et pour tout $i \in [\![1..m]\!]$, la y_i-ème boîte de la colonne i du diagramme est marqué (voir Figure 3.2). En conséquence, partant

d'un diagramme, le nombre de diagrammes marqués qu'il est possible d'engendrer est :

$$\prod_{i=1}^{m} x_i.$$

Enfin, un *diagramme k-Dyck marqué* de taille n est un diagramme marqué dont la suite $(x_1, \ldots, x_{(k-1)n+1})$ est un diagramme k-Dyck de taille n.

Les diagrammes k-Dyck ont été introduits dans [60] et les diagrammes marqués dans [10], comme on le verra dans la section 2.

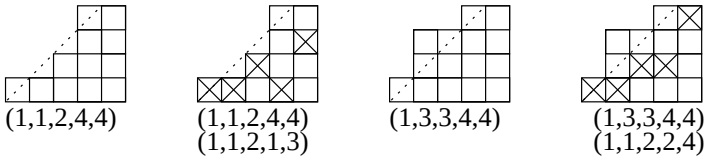


FIG. 3.2 – Un diagramme de longueur 5 et de hauteur 4, un diagramme marqué, un diagramme 2-Dyck et un diagramme marqué 2-Dyck.

1.3 Partitions d'un ensemble

On note $\mathcal{P}_{n,m}$ l'ensemble de toutes les partitions de l'ensemble $\{1, \ldots, n\}$ en m sous-ensembles non-vides. Le cardinal de l'ensemble $\mathcal{P}_{n,m}$ est égal à $\left\{{n \atop m}\right\}$, les nombres de Stirling de seconde espèce, dont on rappelle la définition ci-dessous.

On rappelle que la fonction W de Lambert [27] est l'inverse de la fonction $x \mapsto xe^x$. Sa branche principale W_0 est à valeur réelle pour x inclus dans $[-e^{-1}, +\infty[$ et est l'unique branche qui est analytique en 0. Son développement en série est :

$$W_0(z) = \sum_{n=1}^{\infty} \frac{(-n)^{n-1}}{n!} z^n = z - z^2 + \mathcal{O}\big(z^3\big) \tag{3.1}$$

Les nombres de Stirling de seconde espèce peuvent être estimés asymptotiquement par la méthode du col :

Théorème 8 (Good [38]). *Lorsque n et m tendent tous deux vers l'infini avec $n = \Theta(m)$, on a :*

$$\left\{{m \atop n}\right\} \sim \frac{m!(e^\rho - 1)^n}{n!\rho^m \sqrt{2\pi m(1 - \frac{m}{n}e^{-\rho})}}$$

où $\rho = W_0(-\frac{m}{n}e^{-\frac{m}{n}}) + \frac{m}{n}$ est l'unique racine positive de l'équation $n\rho = m(1 - e^{-\rho})$. En particulier, si $m = kn$, où k est un entier fixé, on a :

$$\left\{{kn \atop n}\right\} \sim \frac{(kn)!}{n!} \frac{(e^{\zeta_k} - 1)^n}{\zeta_k^{kn} \sqrt{2\pi kn(1 - ke^{-\zeta_k})}} \sim \alpha_k \beta_k^n n^{(k-1)n - 1/2}$$

avec $\alpha_k = (2\pi(\zeta_k - (k-1)))^{-\frac{1}{2}}$ et $\beta_k = \frac{(k\zeta_k)^k}{e^{k-1}(e^{\zeta_k} - 1)}$.

2 Bijections

La génération et l'énumération des automates déterministes accessibles utilise un ensemble de bijections qui ramène l'étude de ces objets complexes à des objets combinatoires mieux connus. On explique aussi brièvement comment ces bijections sont utilisées par leur auteurs.

2.1 Structures de transitions et diagrammes k-Dyck marqués

Dans [60], Nicaud introduit les diagrammes 2-Dyck, en bijection avec les arbres couvrants des automates de base de même taille sur un alphabet à 2 lettres. Le nombre d'automates de base associés à un même arbre couvrant est égal au produit de la hauteur des colonnes du diagramme 2-Dyck associé. On peut donc engendrer les diagrammes 2-Dyck à l'aide de la méthode récursive, où le poids de chaque diagramme est égal au nombre d'automates associés.

Cette méthode fut ensuite généralisée à tout alphabet fini contenant au moins deux lettres par Champarnaud et Paranthoën dans [25].

Dans [10], Bassino et Nicaud reformulent la bijection en introduisant la notion de diagramme marqué et donnent un algorithme de complexité linéaire permettant de passer d'un diagramme à une structure de transition de base.

Théorème 9. *[10,25,60] L'ensemble $\mathcal{D}_n$ des structures de transitions de base déterministes accessibles complètes à n états sur un alphabet à k lettres est en bijection avec l'ensemble $\mathcal{B}_n$ des diagrammes k-Dyck marqués de taille n. Cette transformation et son inverse peuvent être calculées en temps linéaire.*

Démonstration. Pour $n \geq 1$, soit $\mathcal{D} = (A, Q, \cdot, \varepsilon) \in \mathcal{C}_n$ une structure de transition de base, déterministe, accessible, complète, définie sur un alphabet à k lettres. La structure de transitions $\mathcal{D}$ étant complète, elle contient kn transitions de la forme (u, α), avec $u \in Q$ et $\alpha \in A$. On partitionne l'ensemble de ces transitions en fonction de leur appartenance à l'arbre couvrant. En utilisant les propriétés de l'étiquetage des états de $\mathcal{D}$, on peut décrire la partition de cet ensemble comme suit : pour tout état $u \in Q$,

- Si $u\alpha \in Q$ alors $u \cdot \alpha = u\alpha$ et (u, α) est une transition de l'arbre couvrant.
- Si $u\alpha \notin Q$ alors $u \cdot \alpha <_{lex} u\alpha$, et (u, α) est une *transition manquante*. Elle n'appartient pas à l'arbre couvrant.

Il existe $n-1$ transitions dans l'arbre couvrant et $(k-1)n+1$ transitions manquantes.

Soit ν l'unique bijection croissante de l'ensemble Q, ordonné dans l'ordre lexicographique, vers $\{1, \ldots, n\}$, c'est-à-dire pour laquelle $\nu(q)$ est le nombre d'éléments de Q inférieurs ou égaux à q selon l'ordre lexicographique. A chaque transition manquante on associe une paire d'entiers $t = (x_t, y_t)$ définie par :

$$\begin{cases} x_t &= |\{u \in Q \mid u <_{lex} q\alpha\}| \\ y_t &= \nu(q \cdot \alpha). \end{cases}$$

On ordonne ensuite les transitions manquantes de $\mathcal{D}$ selon la relation : $(u, \alpha) < (v, \beta)$ si et seulement si $u\alpha <_{lex} v\beta$. La bijection Ψ entre l'ensemble $\mathcal{C}_n$ et l'ensemble des diagrammes k-Dyck marqués de taille n peut alors être définie comme suit : soit

$(t_1, \ldots, t_{(k-1)n+1})$ la suite des paires d'entiers associées aux transitions manquantes de $\mathcal{D}$, on a :

$$\Psi(\mathcal{D}) = ((x_{t_1}, \ldots, x_{t_{(k-1)n+1}}), (y_{t_1}, \ldots, y_{t_{(k-1)n+1}})).$$

L'application Ψ est une bijection (voir [25, 60] pour plus de détails).
La suite $(x_{t_1}, \ldots, x_{t_{(k-1)n+1}})$ représente l'arbre couvrant de $\mathcal{D}$ et définit l'étiquetage des états.
La suite $(y_{t_1}, \ldots, y_{t_{(k-1)n+1}})$ contient toutes les informations nécessaires sur les transitions manquantes :

$$u \cdot \alpha = \nu^{-1}(y_{(u,\alpha)}).$$

Ce qui conclut la preuve. □

Dans le Chapitre 5, on utilise cette bijection pour implanter un générateur exhaustif. A ce stade, la méthode récursive nécessitant de très lourds précalculs, le générateur aléatoire est limité à des automates composés d'une centaine d'états. En effet, comme on le verra dans la section 3, le nombre de structures de transitions croît de manière exponentielle. En plus de demander beaucoup de temps, les précalculs ont un coût en espace mémoire qui devient rapidement prohibitif.

2.2 Diagrammes marqués et partitions d'un ensemble

Théorème 10. *[10] L'ensemble $\mathcal{P}_{kn+1,n}$ des partitions d'un ensemble de taille $kn+1$ en n sous-ensembles non-vides est en bijection avec l'ensemble $\mathcal{B}_n$ des diagrammes marqués de taille n. Cette transformation et son inverse peuvent être calculées en temps linéaire.*

Démonstration. On présente la construction bijective d'un diagramme à partir d'une partition $\mathcal{P}_{kn+1,n}$. On considère que les sous-ensembles de la partition sont triés en fonction de leur plus petit élément. On peut donc indexer les sous-ensembles en fonction de leur ordre d'apparition dans la partition. Soit $\sigma : \mathbb{N} \mapsto \mathbb{N}$ une application qui, à tout entier $i \in \{1, \ldots, kn+1\}$, associe l'indice du sous ensemble de la partition où se trouve i. On commence par transformer la partition d'un ensemble en une paire de suites $((x_1, \ldots, x_{kn+1}), (y_1, \ldots, y_{kn+1})$ que l'on transformera ensuite en diagramme marqué de taille n. Pour tout entier $i \in \{1, \ldots, kn+1\}$, on a

$$\begin{cases} x_i & = max\{\sigma(j) \mid j \leq i\} \\ y_i & = \sigma(i) \end{cases}$$

Pour obtenir un diagramme de taille n, on supprime toutes les entrées de type (x_i, y_i) pour lesquelles :

$$\forall j \in \{1, \ldots, i-1\},\ x_j < x_i$$

Dit autrement, on supprime les colonnes dont les indices correspondent aux plus petites valeurs de chaque sous-ensemble de la partition. La Figure 3.3 illustre cette transformation. □

La mise en évidence de cette bijection a une double utilité. En effet, elle va permettre :

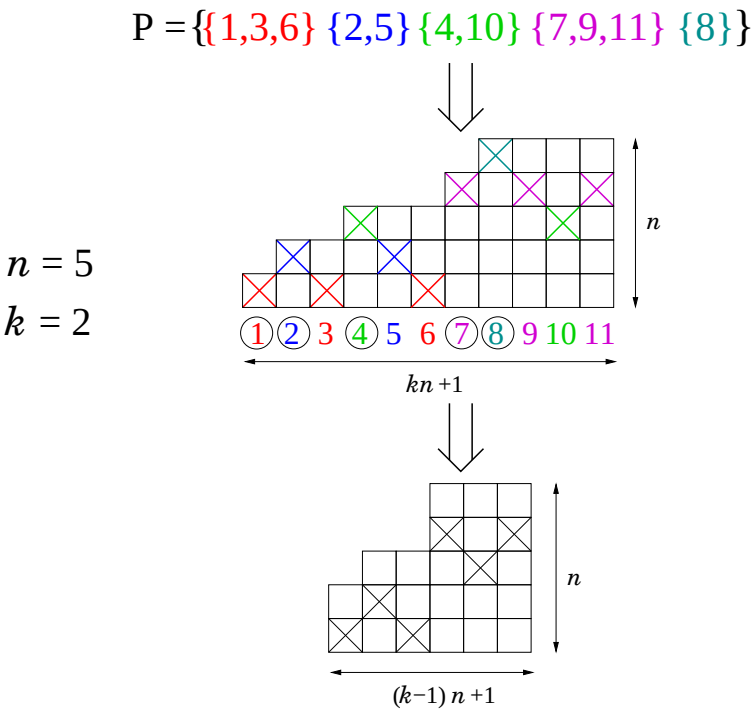


FIG. 3.3 – Passage d'une partition d'un ensemble de $kn+1$ élément en n sous-ensembles non-vides à un diagramme marqué de taille n.

- d'évaluer asymptotiquement le nombre d'automates à n états sur un alphabet à k lettres en fonction du nombre de partitions d'un ensemble de taille kn en n sous-ensembles non-vides (section 3),
- d'engendrer uniformément des structures de transitions en passant par les partitions d'un ensemble, avec lesquels la méthode de Boltzmann s'applique de façon élégante (section 4).

3 Énumération des automates déterministes accessibles complets

Dans [53], Korshunov donne un équivalent asymptotique du nombre de structures de transitions déterministes accessibles complètes à n états sur un alphabet à k lettres. Ce résultat fut reformulé dans [10] à l'aide des nombres de Stirling de seconde espèce.

Théorème 11 (Korshunov [52, 53]). *Le nombre $|\mathcal{C}_n|$ de structures de transitions déterministes accessibles complètes à n états sur un alphabet à k lettres est asymptotiquement égale à :*

$$|\mathcal{C}_n| \sim E_k\, n\, \left\{ {kn \atop n} \right\} \quad où \quad E_k = \frac{1+\sum_{r=1}^{\infty} \frac{1}{r} \binom{kr}{r-1} \left(e^{k-1}\beta_k\right)^{-r}}{1+\sum_{r=1}^{\infty} \binom{kr}{r} \left(e^{k-1}\beta_k\right)^{-r}}, \quad \beta_k = \frac{(k\zeta_k)^k}{e^{k-1}(e^{\zeta_k}-1)}$$

et ζ_k est la racine positive de $\rho = k(1-e^{-\rho})$.

4 Algorithme de Bassino-Nicaud

Dans [10], Frédérique Bassino et Cyril Nicaud utilisent les théorèmes 9 et 10 afin d'obtenir un générateur aléatoire uniforme d'automates déterministes accessibles complets non-isomorphes, à partir des partitions d'un ensemble de taille $kn+1$ en n sous-ensembles non-vides. Dans cet algorithme, on engendre une partition d'un ensemble de taille kn en n sous-ensembles non-vides avec la méthode de Boltzmann, puis on ajoute le $(kn+1)$-ème élément de la partition. Du fait de ce découpage, le générateur n'est pas uniforme sur $\mathcal{P}_{kn+1,n}$, car le $(kn+1)$-ème élément ne peut être isolé dans une part, mais reste uniforme sur les partitions k-Dyck. De plus, ce découpage permet de faciliter le calcul du paramètre x du générateur de Boltzmann, en le rendant uniquement dépendant du paramètre k (voir page 24).

Algorithme 9 : Générateur aléatoire d'automates

Données : le nombre d'états n, la taille de l'alphabet k
begin
  $Diagramme = \emptyset$
  tant que *le Diagramme n'est pas k-Dyck* **faire**
    $Partition = PartitionEnsembleAleatoire(kn, n)$
    $Partition[kn+1] =$ Valeur aléatoire entre 1 et n.
    $Diagramme = PartitionVersDiagrammes(Partition)$
  $Structure = DiagrammesVersStructureDeTransitions(Diagramme)$
  pour q *état de* $\mathcal{A}$ **faire** Choisir uniformément si q est final ou non
end

Théorème 12. *Soit k un entier fixé. La complexité moyenne du générateur aléatoire d'automates déterministes accessibles complets à n états sur un alphabet à k lettres est $\Theta(n\sqrt{n})$.*

Démonstration. D'après les théorèmes 9 et 10, la transformation d'une partition d'un ensemble en diagramme marqué, puis en structure de transitions, peut être effectuée en temps linéaire. La génération aléatoire de partitions d'un ensemble en taille approchée présentée dans le chapitre 1, page 24, et le tirage aléatoire de l'ensemble des état terminaux se font également en temps linéaire. L'algorithme passe par deux phases de rejets potentiels.

1. Si la partition engendrée n'est pas de la taille souhaitée. On a vu dans le chapitre 1, page 24, que la taille de chaque sous-ensemble est tirée avec une loi de Poisson, de telle sorte que l'espérance soit k. L'écart-type est donc égal à $\sqrt{k}$. Les n variables aléatoires étant indépendantes, l'écart-type de leur somme est égal à $\sqrt{kn}$. De plus, la loi de Poisson est unimodale et la somme de variables aléatoires indépendantes tirées avec une loi de Poisson l'est également. Par suite, le nombre moyen de rejets est égal à $\Theta(\sqrt{n})$.
2. Si le diagramme obtenu après transformation de la partition d'un ensemble n'est pas un diagramme k-Dyck. Dans [10], les auteurs montrent que le nombre de rejets est une constante dépendant de k.

Dans les deux cas, afin de garantir l'uniformité, on recommence la génération dès le début. Le générateur aléatoire a bien une complexité moyenne de $\Theta(n\sqrt{n})$. □

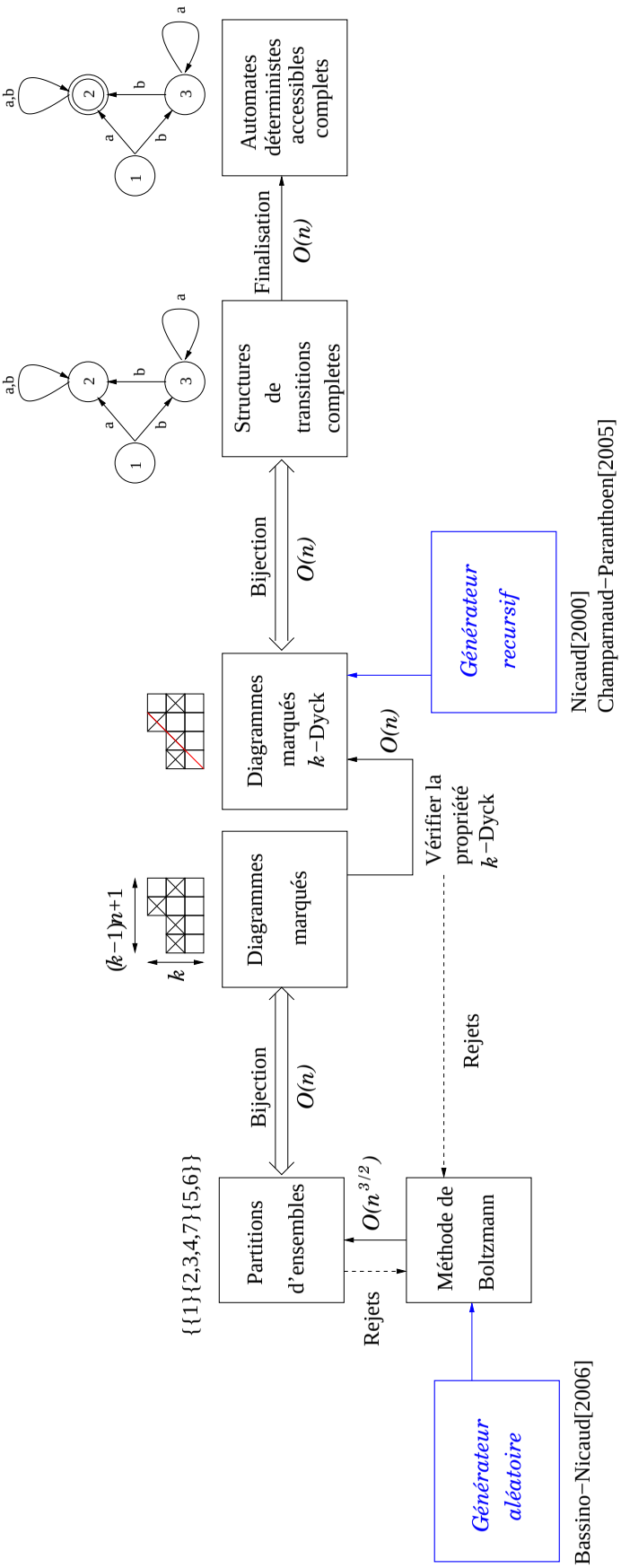


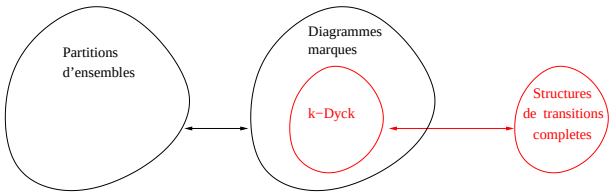
FIG. 3.4 – Méthodes de génération aléatoire d'automates déterministes accessibles complets

Chapitre 4

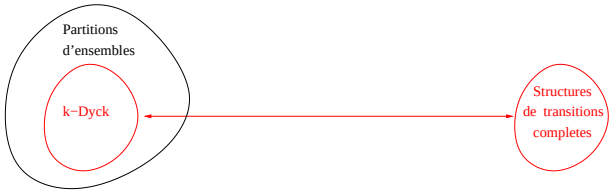
Génération aléatoire d'automates déterministes accessibles

Dans ce chapitre, on s'intéresse à la génération aléatoire des automates déterministes accessibles. Comme on a pu le mentionner dans le chapitre précédent, la difficulté de ce problème provient de la notion d'accessibilité. Les premiers générateurs [25, 60] utilisaient la méthode récursive, dont le coût en temps et en mémoire pose problème. Ici, on utilise la méthode de Boltzmann, qui nous permet d'obtenir des générateurs aléatoires de complexité moyenne $\mathcal{O}(n^{3/2})$ et nécessite peu d'espace mémoire. Ce chapitre est organisé de la façon suivante :

- on présente une reformulation de l'algorithme de Bassino-Nicaud (section 1), dans laquelle on supprime le passage par les diagrammes marqués, en établissant une bijection directe entre les structures de transitions déterministes accessibles complètes et des partitions particulières d'un ensemble. En d'autres termes, on propose de remplacer les deux transformations successives données ci-dessous :

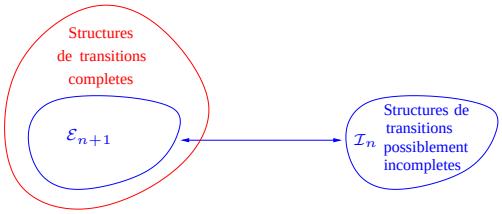


  par l'unique transformation suivante :

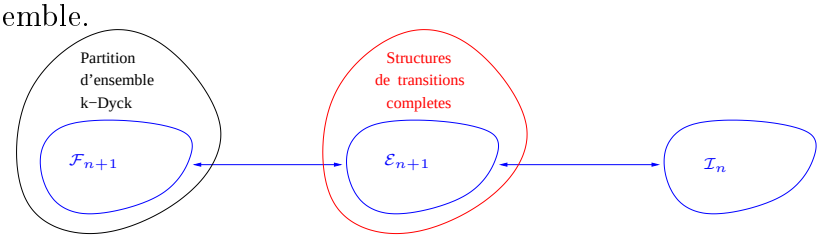


  On obtient ainsi un générateur aléatoire plus rapide. On commencera par donner la bijection, avant de décrire l'algorithme permettant de passer directement d'une structure de transitions à une partition d'un ensemble et l'algorithme permettant la transformation inverse.
- Dans la section 2.1, on s'intéresse à la génération aléatoire d'automates qui ne sont plus nécessairement complets. Dans ce but, on propose une méthode pour compléter un automate qui est compatible avec la transformation qui vient d'être évoquée. Il s'agit une bijection entre un sous-ensemble $\mathcal{E}_{n+1}$ de structures de transitions complètes particulières à $n+1$ états et l'ensemble

$\mathcal{I}_n$ des structures de transitions déterministes accessibles de taille n (lesquels peuvent donc être complets ou incomplets). La transformation que l'on utilise ici n'est pas la méthode dite de *complétion*, couramment utilisée en théorie des automates.



– Dans la section 2.2, pour traiter ces cas d'automate éventuellement incomplet, on généralise la bijection entre structure de transitions et partitions d'un ensemble. Plus précisément, on présente une bijection entre l'ensemble $\mathcal{E}_n$ des structures de transitions et un sous-ensemble $\mathcal{F}_n$ de l'ensemble des partitions d'un ensemble.



On en déduit une estimation asymptotique du nombre d'automates déterministes accessibles (section 1). On montre qu'il existe une proportion constante d'automates complets et d'automates incomplets parmi les déterministes accessibles. Autrement dit, le nombre d'automates incomplets est asymptotiquement proportionnel au nombre d'automates complets.

– On utilise l'ensemble des résultats précédents pour construire un algorithme de génération aléatoire d'automates déterministes accessibles possiblement incomplets (section 2). La complexité moyenne de ce générateur, basé sur la méthode de Boltzmann, est $\mathcal{O}(n\sqrt{n})$. De plus il nécessite très peu d'espace mémoire.

Un équivalent de l'ensemble de ces résultats a été publié dans [6, 7]. La bijection présentée dans la section 2.2 est inédite, bien qu'induite par les résultats donnés dans [6].

1 Reformulation de l'algorithme classique

On présente ici une transformation bijective entre un ensemble particulier de partitions d'un ensemble et les structures de transitions déterministes accessibles complètes. On a vu dans le chapitre précédent que les partitions d'un ensemble sont en bijection avec les diagrammes marqués, et qu'un sous-ensemble de ces diagrammes marqués, respectant la condition k-Dyck, est en bijection avec les structures de transitions. On commence par caractériser l'ensemble des partitions d'un ensemble qui sont en bijection avec les diagrammes marqués k-Dyck : *les partitions k-Dyck d'un ensemble*.

Partition k-Dyck d'un ensemble. Soit un entier $k \geq 2$. Une partition d'un ensemble $P = \{P_1, \ldots, P_n\}$ de $\mathcal{P}_{kn+1,n}$, où les parties P_i sont triées en fonction de

leur plus petit élément, est une *partition k-Dyck d'un ensemble* si elle satisfait la condition suivante :

$$\text{pour tout } j \in \{1, \ldots, n\},\ \min\{i \mid i \in \mathcal{P}_j\} \leq k(j-1)+1$$

Par exemple, pour l'ensemble $\{1, \ldots, 13\}$ et $n = 4$, la partition d'un ensemble

$$P = \{\{1, 11, 13\}, \{2, 3, 6, 9\}, \{4, 8, 10\}, \{5, 7, 12\}\}$$

est une partition d'un ensemble 3-Dyck. En revanche,

$$P = \{\{1, 2, 3, 4, 11\}, \{5, 9, 13\}, \{6, 8, 10\}, \{7, 12\}\}$$

n'en est pas une. En effet, pour $j = 2$, $\min\{i \mid i \in P_j\} = 5$ et $k(j-1)+1 = 4$.

On présente maintenant la nouvelle bijection qui permet de simplifier, d'un point de vue algorithmique, le générateur aléatoire d'automates déterministes accessibles complets proposé dans le chapitre précédent.

Bijection. Soit $\mathcal{D} = (A, Q, \cdot, \varepsilon)$ une structure de transitions accessible et complète à n états sur un alphabet à k lettres ordonnées $A = \{a_1, \ldots, a_k\}$. On ajoute formellement la transition $(\emptyset, \varepsilon)$, qui peut être vue comme la flèche entrante marquant l'état initial dans la représentation graphique de $\mathcal{D}$. Par convention, $\emptyset \cdot \varepsilon = \varepsilon$, l'état initial de $\mathcal{D}$.
Soit $\mathcal{T}_\mathcal{D}$ l'ensemble des transitions de $\mathcal{D}$ qui inclut la transition qui vient d'être ajoutée. Par construction $|\mathcal{T}_\mathcal{D}| = kn + 1$. On numérote de 1 à $kn + 1$ les transitions $t = (u, a)$ de $\mathcal{T}_\mathcal{D}$. le numéro $n(t) \in \{1, \ldots, kn + 1\}$ en utilisant un parcours en profondeur de $\mathcal{D}$ et en respectant l'ordre lexicographique :

$$n((\emptyset, \varepsilon)) = 1$$

et pour tout $t = (u, a)$ et $t' = (u', a')$ dans $\mathcal{T}_\mathcal{D} \setminus \{(\emptyset, \varepsilon)\}$, on a

$$n(t) < n(t') \text{ si et seulement si } ua <_{lex} u'a'.$$

Le parcours effectué pour numéroter les transitions est identique a celui utilisé pour construire l'automate de base, défini page 39. La Figure 1 illustre un exemple de ce type d'étiquetage.

Soit $P_\mathcal{D} = \{P_1, \ldots, P_n\}$ la partition d'un ensemble de $\mathcal{P}_{kn+1,n}$ telle que pour tous i et j de $\{1, \ldots, kn + 1\}$, $i = n((u, a))$ et $j = n((u', a'))$ sont dans la même partie de $\mathcal{P}_\mathcal{D}$ si et seulement si $u \cdot a = u' \cdot a'$. En d'autres termes, i et j sont dans la même partie lorsque les deux transitions associées à ces nombres aboutissent dans le même état de $\mathcal{D}$.

On note χ l'application de l'ensemble de structures de transitions à n états dans $\mathcal{P}_{kn+1,n}$, où $\chi(\mathcal{D}) = P_\mathcal{D}$ est la partition définie ci-dessus.

Théorème 13. *Pour tout $n \geq 1$ et $k \geq 2$, l'application χ qui transforme $\mathcal{D}$ en $P_\mathcal{D}$ est une bijection de l'ensemble des structures de transitions d'automates de base à n états sur un alphabet à k lettres dans l'ensemble des partitions k-Dyck de $\mathcal{P}_{kn+1,n}$.*

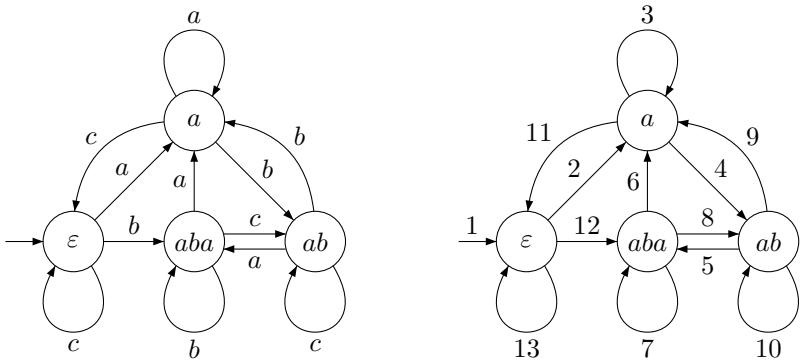


FIG. 4.1 – À gauche : une structure de transitions $\mathcal{D}$. À droite : les transitions numérotées en effectuant un parcours en profondeur et en respectant l'ordre lexicographique. La partition d'un ensemble $P_{\mathcal{D}} = \{\{1, 11, 13\}, \{2, 3, 6, 9\}, \{4, 8, 10\}, \{5, 7, 12\}\}$ est obtenue en regroupant les valeurs associées aux transitions dont l'état d'arrivée est le même.

Démonstration. Soit $P = \{P_1, \ldots, P_m\}$ une partition d'un ensemble dans $\mathcal{P}_{n,m}$ et $\ell \in \{1, \ldots, n\}$. La *$\ell$-ème sous-partition* de P, notée $P^{(\ell)}$, est la partition d'un ensemble de $\{1, \ldots, \ell\}$ dont les éléments sont les sous-ensembles non vides $P_i \cap \{1, \ldots, \ell\}$ avec $i \in \{1, \ldots, m\}$. Ainsi donc, la partition d'un ensemble $P^{(\ell)}$ contient au plus m sous-ensembles non vides.

Une construction permettant de transformer une partition d'un ensemble $P = \{P_1, \ldots, P_n\}$ de $\mathcal{P}_{kn+1,n}$ en diagramme marqué de largeur $kn + 1$ et de hauteur n est donnée dans la preuve du théorème 10, page 43. Si l'on admet que les ensembles P'_i sont triées en fonction de leur plus petit élément, le diagramme marqué $B = ((x_1, \ldots, x_{kn+1}), (y_1, \ldots, y_{kn+1}))$ est tel que, pour tout $i \in \{1, \ldots, kn + 1\}$,

- x_i est égal au nombre de sous-ensembles dans la sous-partition $P^{(i)}$,
- y_i est l'indice du sous-ensemble de P auquel i appartient.

Un diagramme marqué k-Dyck est obtenu en enlevant de B les colonnes correspondant aux plus petits éléments de chaque partie de P. Cette construction étant une bijection de l'ensemble de partitions k-Dyck de $\mathcal{P}_{kn+1,n}$ dans les diagrammes marqués k-Dyck de taille n, on utilise le théorème 9 (page 42), selon lequel il existe une bijection entre les diagrammes et les structures de transitions, pour conclure que les deux ensembles considérés dans l'énoncé du théorème 13 sont de même cardinalité.

On montre à présent que χ, l'application de l'ensemble des structures de transitions à n états dans $\mathcal{P}_{kn+1,n}$, est une injection. On suppose que $\mathcal{D} = (A, Q, \cdot, \varepsilon)$ et $\mathcal{D}' = (A, Q', *, \varepsilon)$ sont les structures de transitions de deux automates de base déterministes accessibles complets distincts à n états. Soit $e = (p, a)$ la première transition rencontrée dans un parcours en profondeur de $\mathcal{D}$ et $\mathcal{D}'$ telle que $p \cdot a \neq p * a$, c'est-à-dire que l'état d'arrivée de la transition partant de p et étiquetée par a n'est pas le même dans $\mathcal{D}$ et $\mathcal{D}'$. On remarque que les états dont l'étiquette est inférieure ou égal à p dans l'ordre lexicographique sont exactement les mêmes dans $\mathcal{D}$ et $\mathcal{D}'$. Soient $q = p \cdot a$ et $q' = p * a$. On suppose par symétrie que q est strictement inférieur à q' dans l'ordre lexicographique. Ainsi, q est inférieur ou égal à p et appartient à Q et à Q'. En conséquence, $n(e)$ est dans le même sous-ensemble de $P_{\mathcal{D}}$ que la première arête aboutissant dans q, ce qui n'est pas le cas dans $P_{\mathcal{D}'}$. L'application χ est donc une injection.

Pour conclure, on prouve que l'image par χ d'une structure de transitions complète à n états sur un alphabet à k lettres est une partition k-Dyck de $\mathcal{P}_{kn+1,n}$. En raisonnant par l'absurde, on suppose qu'il existe une structure de transitions $\mathcal{D}$ d'un automate de base pour lequel ce n'est pas le cas. Soit $P_{\mathcal{D}} = \{P_1, \ldots, P_n\}$, tel que les sous-ensembles P_i sont triés selon leur plus petit élément. Soit ℓ le plus petit entier positif, appartenant à $\{2, \ldots, m\}$, tel que le plus petit élément de P_ℓ est strictement supérieur à $k(\ell-1)+1$. Ainsi, les états d'arrivée des $k(\ell-1)+1$ premières transitions de $\mathcal{D}$ sont inclus dans $\{1, \ldots, \ell-1\}$, formant une structure de transition complète à $\ell - 1$ états. La structure de transitions $\mathcal{D}$ n'est donc pas accessible, ce qui est une contradiction. Ainsi, χ est également une surjection dans le sous-ensemble de $\mathcal{P}_{kn+1,n}$ constitué des partitions k-Dyck d'un ensemble. □

On donne à présent l'algorithme permettant de transformer une partition k-Dyck d'un ensemble en une structure de transitions en partition k-Dyck d'un ensemble.

1.1 Des partitions d'un ensemble aux structures de transitions

L'algorithme ci-dessous est une amélioration de l'algorithme 9, page 45. Sa complexité est linéaire. Partant d'une partition d'un ensemble, au lieu de calculer le diagramme marqué puis la structure de transitions associée, on calcule directement la structure de transitions. Au cours de l'algorithme, on considère que les sous-ensembles $P = \{P_1, \ldots, P_n\}$ de la partition sont triés selon leur plus petit élément. L'algorithme est basé sur le fait que les valeurs numériques des éléments de P correspondent aux transitions de la structure obtenue en sortie en fonction de l'ordre dans lequel elles sont rencontrées dans un parcours en profondeur et que, lorsque $i \in P_j$, l'état d'arrivée de la i-ème transition est j.

Algorithme 10 : DesPartitionsAuxStructuresDeTransitions(P)

Données : Une partition k-Dyck d'un ensemble $P = \{P_1, \ldots, P_n\}$ de $\mathcal{P}_{kn+1,n}$

```
1  S = pile vide
2  Créer l'état initial q_0 = 1
3  nouvelEtat = 2 // il s'agit de l'indice du prochain nouvel état
4  pour chaque a ∈ A dans l'ordre alphabétique inverse faire
5      Ajouter (q_0, a) dans S
6  pour tous les i ∈ {2, .., kn + 1} faire
7      Extraire une paire (p, a) de S
8      q = j, avec i ∈ P_j
9      si q = nouvelEtat alors
10         Créer l'état q
11         nouvelEtat = nouvelEtat + 1
12         pour chaque b ∈ A dans l'ordre lexicographique inverse faire
13             Ajouter (q, b) dans S
14     Ajouter une transition de p vers q étiquetée par a
```

2 Génération aléatoire d'automates possiblement incomplets

On note $\mathcal{I}_n$ l'ensemble des structures de transitions déterministes accessibles à n états sur un alphabet à k lettres, c'est-à-dire l'ensemble des structures de transitions pouvant être complètes ou incomplètes.

Dans cette section, on présente un générateur aléatoire uniforme sur $\mathcal{I}_n$, basé sur la méthode de Boltzmann et sur deux transformations combinatoires des structures de transitions possiblement incomplètes.

La première est une bijection non classique entre un sous-ensemble $\mathcal{E}_{n+1}$ de structures de transitions complètes à $n+1$ états et l'ensemble $\mathcal{I}_n$, qui satisfait la propriété suivante : les arbres couvrants de deux structures de transitions sont égaux si et seulement si les deux arbres couvrants des structures après transformation le sont également.

La seconde lie les structures de transitions possiblement incomplètes à une sous-classe de partitions k-Dyck d'un ensemble. On utilisera cette dernière pour énumérer les structures de transitions dans la section 2.3.

2.1 Des structures de transitions aux structures de transitions complètes

En théorie des automates, un automate incomplet est habituellement transformé en un automate complet reconnaissant le même langage par la méthode dite de *complétion*, qui consiste à ajouter un état puits, lequel devient l'état d'arrivée de toutes les transitions jusqu'alors indéfinies. Cette transformation n'est pas adaptée au cas présent. En effet, si deux automates incomplets ont le même arbre couvrant, l'ajout de l'état puits peut changer la donne, comme on peut le voir sur la Figure. 4.2.

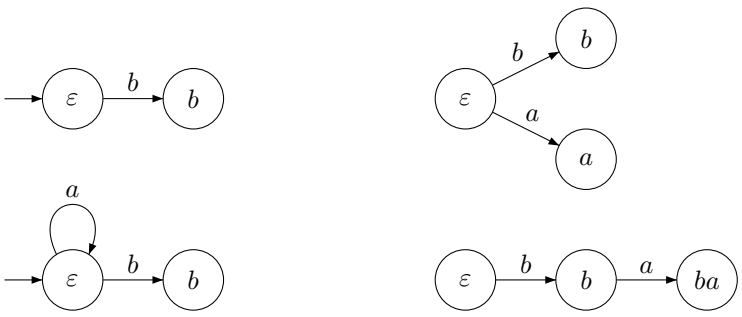


FIG. 4.2 – Sur la gauche, les deux structures de transitions partagent le même arbre couvrant. Sur la droite, on peut voir leur arbre couvrant respectif, après l'ajout d'un état puits suivant la méthode de complétion usuelle.

On présente donc une autre transformation, notée ϕ, que l'on définit de la façon suivante : à toute structure de transitions $\mathcal{I} \in \mathcal{I}_n$, avec $\mathcal{I} = (A, Q, \cdot, \varepsilon)$, on associe une structure de transitions de base complète (les états sont donc étiquetés par des mots, voir définition page 39) : $\phi(\mathcal{I}) = (A, Q', *, \varepsilon)$ appartenant à l'ensemble $\mathcal{C}_{n+1}$. L'ensemble Q' des états de $\phi(\mathcal{I})$ est défini comme suit :

$$Q' = \{\varepsilon\} \cup a_k Q \quad \text{où} \quad a_k = \max_{lex}\{\alpha \in A\}$$

Les transitions de $\phi(\mathcal{I})$ sont définies par :

$$\begin{cases} \varepsilon * \alpha = \varepsilon & \text{si } \alpha \neq a_k \\ \varepsilon * a_k = a_k & \\ q' * \alpha = a_k(q \cdot \alpha) & \text{si } \exists q \in A^*,\ q' = a_k q \text{ et } q \cdot \alpha \neq \emptyset \\ q' * \alpha = \varepsilon & \text{si } \exists q \in A^*,\ q' = a_k q \text{ et } q \cdot \alpha = \emptyset. \end{cases}$$

La construction se déroule de la façon suivante :

- on ajoute un nouvel état, qui devient l'état initial ε de $\phi(\mathcal{I})$ et une transition $\varepsilon * a_k = q_0$, étiquetée par la plus grande lettre de l'alphabet et où q_0 est l'état initial de $\mathcal{I}$,
- on réétiquette la structure de transitions afin d'obtenir la structure de transitions d'un automate de base,
- on change les transitions indéfinies $q \cdot \alpha = \emptyset$ en $q * \alpha = \varepsilon$.

On remarque que ϕ ne préserve pas le langage reconnu par l'automate. En revanche, ϕ préserve la propriété d'accessibilité : si dans un automate $\mathcal{I}$, tous les états sont accessibles à partir de l'état initial q_0, la fonction ϕ rajoutant une transition de ε vers q_0, alors tous les états de $\phi(\mathcal{I})$ sont accessibles à partir de ε.

Lemme 14. *On note $\mathcal{E}_n$ le sous-ensemble des structures de transitions de $\mathcal{C}_n$ tel que $\varepsilon \cdot \alpha = \varepsilon$ pour $\alpha \in A \setminus \{\max_{lex}\{\alpha \in A\}\}$. La fonction ϕ est une bijection de $\mathcal{I}_n$ dans $\mathcal{E}_{n+1}$.*

D'après la définition de ϕ, $\mathcal{E}_{n+1} = \phi(\mathcal{I}_n)$. L'inverse de ϕ est obtenu en retirant l'état initial, en faisant de l'état a_k l'état initial, et en réétiquetant les états. La Figure 4.3 contient un exemple de transformation.

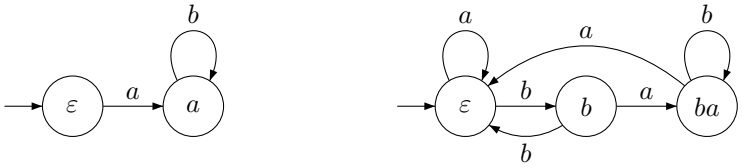


FIG. 4.3 – A gauche une structure de transitions de base incomplète de taille 2. A droite, la structure de transitions de taille 3 obtenue à l'aide de la transformation ϕ.

2.2 Des partitions k-Dyck d'un ensemble associés aux éléments de $\mathcal{E}_n$

On renvoie le lecteur à la définition de l'application χ qui à toute structure de transitions de base à n états sur un alphabet de taille k associe une partition d'un ensemble de taille $kn+1$ en n sous-ensembles non-vides. L'image de $\mathcal{E}_n$ par χ est facile à caractériser.

Théorème 15. *Soit $\mathcal{F}_n \subset \mathcal{P}_{kn+1,n}$ l'ensemble des partitions k-Dyck d'un ensemble de taille $kn+1$ telles que pour tout $i \in \{1, \ldots, k\}$, $i \in P_1$. Pour tout $n \geq 2$, χ est une bijection entre $\mathcal{E}_n$ et $\mathcal{F}_n$.*

Démonstration. Soient $A = \{a_1 < \ldots < a_k\}$ et $\mathcal{I} = (A, Q, \cdot, \varepsilon) \in \mathcal{C}_n$. On a $\mathcal{I} \in \mathcal{E}_n$ si et seulement si

$$\text{pour tout } i \in \{1, \ldots, k-1\},\ \varepsilon \cdot a_i = \varepsilon.$$

On rappelle que la partition d'un ensemble $\chi(\mathcal{I})$ est telle que pour tous i et j de $\{1, \ldots, kn+1\}$, i et j sont dans la même partie si et seulement si $u.a = u'.a'$, où $n(u, a) = i$ et $n(u', a') = j$. On rappelle également qu'une transition $(\emptyset, \varepsilon)$ est ajoutée aux structures de transitions, de sorte que :

$$n((\emptyset, \varepsilon)) = 1.$$

Les sous-ensembles de la partition étant par définition ordonnés en fonction de leur plus petit élément, on a $1 \in P_1$. De plus, ε est le seul mot de Q qui ne commence pas par a_k. Ainsi, les $k-1$ premières transitions indéfinies de $\mathcal{I}$ sont (ε, a_1), ..., (ε, a_{k-1}). Par suite, $\mathcal{I} \in \mathcal{E}_n$ si et seulement si

$$\text{pour toute transition } t \text{ telle que } n(t) \leq k,\ \text{on a } n((\varepsilon, a_i)) \in P_1.$$

et $\chi(I) \in \mathcal{F}_n$. Ceci conclut la preuve. □

2.3 Énumeration des automates déterministes accessibles

L'énumération des structures de transitions possiblement incomplètes utilise le lemme suivant :

Lemme 16. *Pour tout entier fixé $k \geq 2$, lorsque n tend vers l'infini, on a*

$$\left\{ {kn+1 \atop n+1} \right\} \sim e^{\zeta_k} \left\{ {kn \atop n} \right\} \quad \textit{avec} \quad \zeta_k = W_0(-ke^{-k}) + k.$$

Démonstration. La preuve suivante est basée sur la comparaison des estimations de $\left\{ {kn \atop n} \right\}$ et $\left\{ {kn+1 \atop n+1} \right\}$. On rappelle la formule de Stirling :

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

et on adapte l'énoncé du théorème 8 :

$$\left\{ {kn \atop n} \right\} \sim \frac{(kn)!}{n!} \frac{(e^{\zeta_k} - 1)^n}{\zeta_k^{kn} \sqrt{2\pi kn(1 - ke^{-\zeta_k})}} \sim \alpha_k \beta_k^n n^{(k-1)n - 1/2}$$

avec $\alpha_k = (2\pi(\zeta_k - (k-1)))^{-\frac{1}{2}}$ et $\beta_k = \frac{(k\zeta_k)^k}{e^{k-1}(e^{\zeta_k} - 1)}$.
On note f la fonction $f(x) = W_0(-xe^{-x}) + x$. Pour utiliser le théorème 8 avec $\left\{ {kn+1 \atop n+1} \right\}$, on doit calculer $\rho_{n,k} = f\left(\frac{kn+1}{n+1}\right) = f(k - \frac{k-1}{n+1})$. Comme la fonction f est analytique, on peut calculer son développement de Taylor :

$$\rho_{n,k} = f\left(k - \frac{k-1}{n+1}\right) = f(k) - \frac{k-1}{n+1} f'(k) + \mathcal{O}\left(\frac{1}{n^2}\right) = \zeta_k - (k-1) f'(k) \frac{1}{n} + \mathcal{O}\left(\frac{1}{n^2}\right).$$

D'après le théorème 8 on a :

$$\left\{ {kn+1 \atop n+1} \right\} \sim \frac{(kn+1)!(e^{\rho_{n,k}} - 1)^{n+1}}{(n+1)! \rho_{n,k}^{kn+1} \sqrt{2\pi(kn+1)(1 - \frac{kn+1}{n+1} e^{-\rho_{n,k}})}}.$$

En utilisant les approximations classiques et la formule de Stirling, on obtient :

$$\frac{(kn+1)!}{(n+1)!} \sim e^{-(k-1)n} k^{3/2} k^{kn} n^{(k-1)n}$$

$$\sqrt{2\pi(kn+1)(1-\frac{kn+1}{n+1}e^{-\rho_{n,k}})} \sim \sqrt{2\pi kn(1-ke^{-\zeta_k})}$$

$$(e^{\rho_{n,k}}-1)^{n+1} \sim (e^{\zeta_k}-1)^{n+1} e^{-\frac{(k-1)f'(k)e^{\zeta_k}}{e^{\zeta_k}-1}}$$

$$\rho_{n,k}^{kn+1} \sim \zeta_k^{kn+1} e^{-\frac{k(k-1)f'(k)}{\zeta_k}}$$

De plus, comme ζ_k satisfait $\zeta_k = k(1-e^{-\zeta_k})$, $e^{\zeta_k}/(e^{\zeta_k}-1) = k/\zeta_k$. On obtient finalement

$$\left\{ {kn+1 \atop n+1} \right\} \sim \alpha'_k \beta_k^n n^{(k-1)n-1/2}$$

où $\alpha'_k = \frac{e^{\zeta_k}}{\sqrt{2\pi(\zeta_k-(k-1))}}$. Ainsi $\left\{ {kn+1 \atop n+1} \right\} \sim e^{\zeta_k} \left\{ {kn \atop n} \right\}$, ce qui conclut la preuve. □

Le théorème 17 montre qu'il y a $\Theta(n\, 2^n \left\{ {kn \atop n} \right\})$ automates de base déterministes et accessibles à n états sur un alphabet de taille k.

Théorème 17. *Le nombre $|\mathcal{I}_n|$ de structures de transitions de base déterministes et accessibles à n états sur un alphabet à k lettres est $\Theta(n \left\{ {kn \atop n} \right\})$.*

Démonstration. Tout d'abord, on a $|\mathcal{C}_n| \leq |\mathcal{I}_n|$ car $\mathcal{C}_n \subset \mathcal{I}_n$. Le théorème 11 nous donne une borne inférieure. On cherche donc à majorer $|\mathcal{I}_n|$.

On a montré qu'il existe une bijection en deux étapes entre l'ensemble $\mathcal{I}_n$ et l'ensemble $\mathcal{F}_{n+1}$ des partitions k-Dyck d'un ensemble telles que pour tout entier $i \in \{1, \ldots, k\}$, $i \in P_1$.

Il existe une bijection entre l'ensemble $\mathcal{F}_{n+1}$, c'est-à-dire l'ensemble des partitions d'un ensemble $kn+k+1$ en $n+1$ sous-ensembles non-vides, telles que pour tout $i \in \{1, \ldots, k\}$, $i \in P_1$, et l'ensemble des partitions d'un ensemble $kn+2$ en $n+1$ sous-ensembles non-vides : il suffit de supprimer de la partition les valeurs $\{2, \ldots, k\}$ de la partition et de remplacer toute valeur $j > k$ de la partition par $j-(k-1)$.

On en déduit que $|\mathcal{I}_n| \leq \left\{ {kn+2 \atop n+1} \right\}$. On utilise le lemme 16 et la relation de récurrence sur les nombres de Stirling de seconde espèce :

$$\left\{ {kn+2 \atop n+1} \right\} = \left\{ {kn+1 \atop n} \right\} + (n+1) \left\{ {kn+1 \atop n+1} \right\}$$

$$\left\{ {kn+2 \atop n+1} \right\} = \left\{ {kn+1 \atop n} \right\} + (n+1) e^{\zeta_k} \left\{ {kn \atop n} \right\}$$

par le même raisonnement on obtient $\left\{ {kn+1 \atop n} \right\} = \Theta(n \left\{ {kn \atop n} \right\})$. Ceci conclut la preuve. □

Corollaire 18. *Lorsque n tend vers l'infini, on a $|\mathcal{C}_n| = \Theta(|\mathcal{I}_n \setminus \mathcal{C}_n|) = \Theta(|\mathcal{I}_n|)$.*

2.4 Engendrer les partitions d'un ensemble

Afin d'engendrer uniformément des partitions d'un ensemble à $kn+1$ éléments en $n+1$ sous-ensembles non-vides, on commence par considérer l'ensemble des partitions

d'un ensemble en $n+1$ sous-ensembles non-vides. La série génératrice exponentielle de cet ensemble est

$$P_{n+1}(z) = \frac{(e^z-1)^{n+1}}{(n+1)!}.$$

En utilisant une construction à la Boltzmann, on engendre la taille de chacun des $n+1$ sous-ensembles, en supposant que cette valeur suit une loi de Poisson non nulle $\text{Pois}_{\geq 1}$ de paramètre x. La taille moyenne de la partition obtenue est :

$$\mathbb{E}_x(\text{taille de la partition}) = x\frac{P'_{n+1}(x)}{P_{n+1}(x)} = (n+1)x\frac{e^x}{e^x-1}.$$

Puisque l'on veut une partition d'un ensemble à $kn+1$ éléments, la valeur du paramètre x_n est choisie de telle façon que :

$$(n+1)x_n\frac{e^{x_n}}{e^{x_n}-1} = kn+1,$$

c'est-à-dire, $x_n = \rho_{n,k}$ (voir la preuve du lemme 16). Lorsque le paramètre de Boltzmann x_n est égal à $\rho_{n,k}$, la probabilité qu'une partition d'un ensemble aléatoire soit de taille $kn+1$ est [31] :

$$\mathbb{P}_{\rho_{n,k}}(N = nk+1) = \frac{\rho_{n,k}^{kn+1}[z^{kn+1}]P_{n+1}(z)}{P_{n+1}(\rho_{n,k})} = \frac{\left\{{kn+1 \atop n+1}\right\}\rho_{n,k}^{kn+1}}{(kn+1)!}\frac{(n+1)!}{(e^{\rho_{n,k}}-1)^{n+1}}.$$

Cette valeur peut-être estimée asymptotiquement en utilisant la même méthode que dans la preuve du lemme 16 :

$$\mathbb{P}_{\rho_{n,k}}(N = nk+1) \sim \frac{\alpha_k}{\sqrt{kn}}. \tag{4.1}$$

2.5 Générateur aléatoire d'automates possiblement incomplets

L'algorithme que l'on présente ici utilise l'algorithme 10 afin d'engendrer uniformément des automates déterministes accessibles à n états sur un alphabet à k lettres. Le principe est le suivant :

- on engendre uniformément un partition de $\{1, k+1, k+2, \ldots, k(n+1)\}$, un ensemble de $kn+1$ éléments, en $n+1$ sous-ensemble non-vides,
- on ajoute les valeurs $\{2, \ldots, k\}$ dans le premier sous-ensemble,
- on ajoute la valeur $k(n+1)+1$ dans un des $n+1$ sous-ensembles non-vides, choisi uniformément. Tout comme pour l'algorithme 9, page 45, du fait que cette dernière valeur soit tirée séparement, le générateur aléatoire n'est plus uniforme sur $\mathcal{P}_{kn+1,n+1}$ mais l'est encore sur les partitions k-Dyck.

Le détail de l'algorithme est donné ci-dessous.

Algorithme 11 : StructureDeTransitionAleatoire(n)

Données : La taille n de l'automate

1 Calculer $\rho_{n,k}$ en utilisant l'Équation (3.1) p.41.
2 **répéter**
3 | **répéter**
4 | | **pour** $i \in \{1, \dots, n+1\}$ **faire** $\lambda_i = \text{Pois}_{\geq 1}(\rho_{n,k})$
5 | **jusqu'à** $\lambda_1 + \cdots + \lambda_{n+1} = kn + 1$
6 | $P = \{\{1, \dots, \lambda_1\}, \{\lambda_1 + 1, \dots, \lambda_1 + \lambda_2 + 1\}, \dots, \{\lambda_1 + \dots \lambda_{n+1}, \dots, kn + 1\}$
7 | σ =PermutationAleatoire($\{1, \dots, kn+1\} \mapsto \{1, k+1, \dots, k(n+1)\}$)
8 | Remplacer chaque $i \in \{1, \dots, kn+1\}$ de P par $\sigma(i)$
9 | **pour chaque** $i \in \{2 \dots k\}$ **faire** Ajouter i dans P_1
10 | Choisir uniformément $j \in \{1, \dots, n+1\}$ et ajouter $k(n+1)+1$ dans P_j
11 **jusqu'à** *ce que P soit une partition k-Dyck d'un ensemble*
12 $\mathcal{A}'$ =DesPartitionsAuxStructuresDeTransitions(P)
13 Calculer $\mathcal{A}$ à partir de $\mathcal{A}'$ en supprimant l'état initial, les transitions entrantes ou sortantes du premier ensemble, et en désignant le deuxième état comme initial.
14 **pour** q *état de* $\mathcal{A}$ **faire** Choisir uniformément si q est final ou non

Théorème 19. *La complexité moyenne du générateur aléatoire est* $\mathcal{O}(n^{3/2})$.

Proof: Toutes les étapes peuvent être exécutées en temps linéaire, si le nombre de rejets n'est pas pris en compte. Dans un algorithme avec rejet, si la propriété est vraie avec probabilité p, le nombre moyen de rejet est $1/p$. Ainsi, en utilisant le théorème 17 et le lemme 16, on en déduit que le nombre moyen de rejets effectués à l'Étape 5 est borné par une constante. L'Équation (4.1) p. 56 nous dit qu'en moyenne, il y a $\mathcal{O}(\sqrt{n})$ rejets à la ligne 11. □

Chapitre 5

Logiciels et expériences

Dans ce chapitre, on présente une bibliothèque en *C++* nommée `REGAL`, ainsi que son programme associé, `PREGA`, développé dans le cadre de cette thèse et tous deux sous licence *GNU*. En utilisant les générateurs décrits dans les chapitres précédents, ils vont permettre à leur utilisateur d'engendrer aléatoirement et exhaustivement des automates déterministes aux diverses propriétés, dans le but d'étudier l'objet en lui-même mais aussi les algorithmes qui s'y appliquent.

Dans la section 1, on présente l'ensemble des générateurs et des outils d'analyses proposés par `REGAL`. Cette partie a fait l'objet de publications à *CIAA'07* [5] et *MAJECSTIC'07* [29]. Outre le programme `PREGA`, il existe des outils de manipulations d'automates [28, 54] qui utilisent `REGAL`. On donnera la complexité amortie du générateur exhaustif d'automates déterministes accessibles complets.

Dans la section 2, on donne une description de l'utilisation de `PREGA` et quelques exemples d'automates engendrés par le programme.

Dans la section 3, on présente un ensemble de résultats expérimentaux obtenus à l'aide de ces outils, lesquels nous permettent d'établir des conjectures crédibles.

1 Description de la bibliothèque `REGAL`

Le code source de *REGAL* (environ 5 000 lignes) ainsi que sa documentation sont disponibles à l'adresse *http ://regal.univ-mlv.fr*.

Le choix du langage de programmation a été guidé par deux objectifs : le langage devait être générique pour permettre à l'utilisateur de décrire sa propre implantation d'automates et l'exécution du code devait être rapide pour permettre la génération en grand nombre d'automates de grande taille. Parmi les langages répondant à ces besoins, le *C++* est le plus utilisé de tous et il existe d'autres logiciels manipulant les automates programmés dans le même langage, comme `VAUCANSON` [26, 55]. C'est pourquoi nous avons opté pour ce langage.

Automates. Bien que les algorithmes de `REGAL` utilisent des entiers pour décrire les étiquettes des états et des transitions, ces dernières ont été codées comme des types paramétrés, c'est-à-dire qu'ils sont entièrement génériques. La traduction d'un type quelconque à un entier a été automatisée (en utilisant les *Maps* du *C++*). L'utilisateur peut donc facilement définir ses propres étiquettes.

Les automates de `REGAL` ont été implantés avec des tables de transitions : les premiers algorithmes étant destinés aux automates déterministes complets, ce mode de représentation semblait le plus approprié. Si toutefois un utilisateur souhaite définir son propre type d'automate, la classe qu'il définit devra hériter de la classe abstraite qui définit l'objet automate, laquelle contient un ensemble de méthodes que l'utilisateur devra redéfinir, afin de pouvoir utiliser les algorithmes de génération d'automates offerts par `REGAL`. Ce faisant, la bibliothèque `REGAL` a pu être utilisée pour engendrer directement des automates de type `VAUCANSON`,

Outils d'analyse :

La bibliothèque est équipée de méthodes qui permettent de faciliter la mise en place de tests par l'utilisateur. On recense les outils suivants :

- Les moments : une classe calcule à la volée la moyenne, la variance et l'écart-type des valeurs qui sont passées en paramètre.
- Affichage : la mise en page de données expérimentales, permettant de les afficher avec l'outil *gnuplot* sous forme de courbes ou d'histogramme est automatisée.
- Conversion : il est possible d'enregistrer les automates aux formats *GasteX*, *DoT* ou *XML*.

Il ne s'agit certes pas d'outils complexes, mais il est à espérer que leurs présence dans la bibliothèque en facilite la prise en main par ses utilisateurs.

1.1 Générateurs exhaustifs

On utilise une méthode standard [62] pour engendrer tous les automates déterministes accessibles et complets de taille n donnée. On définit un ordre total sur leurs représentations en diagrammes k-Dyck marqués (voir théorème 9) et on fournit 3 méthodes pour calculer le minimum (`first`), le successeur d'un diagramme donné (`next`) et pour vérifier si le maximum est atteint (`last`) ou non (voir la Figure 5.1). Le but de ces trois méthodes est d'imiter les itérateurs du `C++`, afin de les rendre faciles à utiliser pour l'utilisateur. Tous les ensembles possibles d'états terminaux sont également engendrés.
Soient $D = (\mathbf{x}, \mathbf{y})$ et $D' = (\mathbf{x}', \mathbf{y}')$ deux diagrammes k-Dyck de taille n marqués. On dit que D est plus petit que D', soit lorsque le vecteur $\mathbf{x}$ est strictement inférieur à $\mathbf{x}'$ dans l'ordre lexicographique, soit lorsque $\mathbf{x} = \mathbf{x}'$ et le vecteur $\mathbf{y}$ est plus petit que $\mathbf{y}'$.
Le plus petit diagramme k-Dyck de taille n marqué pour cet ordre est

$$D_{\min} = ((x_1, \ldots, x_{(k-1)n+1}), (1, \ldots, 1)),$$

où $x_i = \lceil i/(k-1) \rceil$, et le plus grand est

$$D_{\max} = ((n, \ldots, n), (n, \ldots, n)).$$

Soit $D = (\mathbf{x}, \mathbf{y})$ un diagramme k-Dyck marqué qui n'est pas égal à D_{max} :

(a) Si $\mathbf{x} \neq \mathbf{y}$, le successeur de D est obtenu en changeant y_i en $y_i + 1$, où i est le plus grand indice tel que $y_i < x_i$.

(b) Si $\mathbf{x} = \mathbf{y}$, soit i le plus grand indice tel que $x_i < n$. Le successeur de D est $D' = ((x_1, \ldots, x_{i-1}, x_i + 1, x'_{i+1}, \ldots, x'_{(k-1)n+1}), (1, \ldots, 1))$ où pour tout $j \geq i + 1$, $x'_j = \max(x_i + 1, \lceil j/(k-1) \rceil)$.

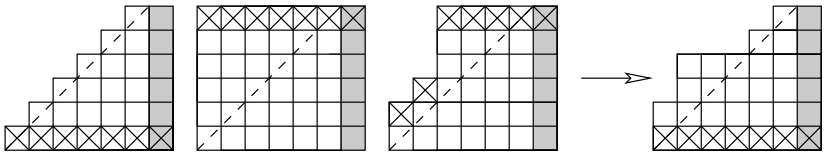


FIG. 5.1 – $D_{\min}$,$D_{\max}$ et `next` appliqué au cas (b) avec $i = 2$

Complexité amortie. En utilisant cet ordre sur l'ensemble des diagrammes k-Dyck marqués de taille fixée, les trois algorithmes `first`, `next` et `last` ont une complexité linéaire dans le pire cas.

Lemme 20. *La complexité amortie de la fonction* `next` *est* $\mathcal{O}(1)$.

Démonstration. Soit $\mathcal{O}(\kappa)$ le coût de la fonction `next`, où κ est le nombre de colonnes que la fonction va modifier. La fréquence à laquelle `next` aura à modifier κ colonnes est

$$\frac{1}{\prod_{i=kn+1-\kappa}^{kn+1} y_i},$$

où y_i est la hauteur de la i-ième colonne. Cette valeur est largement inférieure à $\frac{1}{\kappa}$, la complexité amortie de la fonction `next` est donc $\mathcal{O}(1)$ et la complexité de la génération exhaustive de tous les automates de l'ensemble $\mathcal{C}_n$ est $O(|\mathcal{C}_n|)$. □

1.2 Générateurs aléatoires

On présente dans cette sous-section l'ensemble des générateurs implantés dans *REGAL*. On l'a déjà expliqué plusieurs fois précédement : pour engendrer certains automates aléatoires, il est possible d'engendrer séparement la structure de transitions et l'ensemble des états terminaux. On présente donc dans un premier temps l'ensemble des générateurs de structures de transitions, puis celui des états terminaux, et enfin les générateurs obtenus après la réunion des deux, et l'utilisation de la méthode du rejet.

Structures de transitions

- Automates déterministes complets : pour la distribution uniforme parmi les déterministes complets, où l'on néglige la question des automates isomorphes ou accessibles, engendrer une structure de transitions aléatoire correspond à tirer k applications aléatoires de $\{1, \ldots, n\}$ dans $\{1, \ldots, n\}$. C'est-à-dire, pour toutes les transitions de l'automate, tirer l'état d'arrivée avec une loi uniforme parmi $\{1, \ldots, n\}$.
- Automates déterministes accessibles complets, incomplets, possiblement incomplets : on renvoir le lecteur aux chapitres 3 et 4.
- Transducteurs déterministes accessibles (in)complets : engendrer des transducteur lorsque l'automate sous-jacent est complet ne présente aucune difficulté : il suffit, pour chaque transition, de choisir une valeur de sortie aléatoirement.

Le cas où l'automate est incomplet est plus complexe car la méthode précédente ne garantit plus l'uniformité. *REGAL* implante la méthode décrite dans [40]. Notons qu'il est possible d'associer à chaque lettre de l'alphabet d'entrée un sous-ensemble de l'alphabet de sortie, afin d'imposer des restrictions.

– Tries : un trie est un automate dont la structure de transition est un arbre. On engendre cet arbre aléatoirement à l'aide de l'algorithme de Remy [64] (voir section 1.1, page 20).

Ensembles d'états terminaux On a envisagé deux distributions pour les ensembles d'états terminaux.

– Distribution de Bernoulli : on fixe une probabilité $p \in]0 \dots 1[$. Pour tout état de l'automate, on décide ensuite si celui ci est terminal en tirant uniformément une valeur x entre 0 et 1. Si $x \leq p$ alors l'état est terminal, sinon il est non-terminal. En utilisant cette méthode, le nombre d'états terminaux croît de façon linéaire avec n, puisque le nombre moyen d'états terminaux sera pn. Cette méthode permet d'obtenir des générateurs uniformes en prennant $p = \frac{1}{2}$, mais ne fournira que très rarement des automates de grandes tailles avec peu d'états terminaux.
– Fixer le nombre d'états terminaux : cette méthode permet d'obtenir des automates de grandes tailles avec très peu d'états terminaux. L'utilisateur fixe la taille de l'ensemble F des états terminaux, qui seront choisis uniformément parmi les états de Q.

On verra par la suite que l'utilisation de ces deux distributions permet d'obtenir des résultats très différents dans l'étude expérimentale du comportement moyen des algorithmes appliqués aux automates.

Générateurs efficaces par la méthode du rejet :

Les générateurs que l'on obtient ici en utilisant la méthode du rejet peuvent être obtenus indifférement en partant de générateurs d'automates déterministes accessibles complets, incomplets, ou possiblement incomplets.

– Automates fortements connexes : selon un résultat énoncé par Korshunov [52], il existe une proportion constante d'automates fortement connexes parmi les automates déterministes accessibles complets (79% pour un alphabet de taille 2, 94% pour un alphabet de taille 3, ...). D'après le corollaire du théorème 17, page 55, on a un résultat équivalent sur les automates possiblement incomplets.
– Automates émondés : la proportion d'automates ne contenant aucun état terminal étant négligeable, la proportion d'automates émondés est une conséquence directe du résultat précédent : un automate fortement connexe contenant au moins un état final est émondé.
– Automates minimaux : ici les automates sont bien sûr complets. Comme on le verra dans la section 3, il existe expérimentalement une forte proportion d'automates minimaux parmi les automates déterministes accesssibles complets. Partant de ce générateur, on a donc obtenu un générateur aléatoire d'automates minimaux.

2 Description du programme PREGA

Le logiciel PREGA fut développé pour permettre à des utilisateurs réticents à manipuler du *C++* d'effectuer des tests sur des automates aléatoires.

2.1 L'interface utilisateur

Une fois le programme installé, l'utilisateur décrit l'ensemble des automates qu'il souhaite engendrer à l'aide d'un fichier *XML* (un exemple est fourni avec le logiciel). Il doit alors remplir les différents champs :

- Le champ *generator* : nécessite de définir un champ *automata* et un *type*, qui peut être égal à *random* ou *exhaustive*, selon que l'on souhaite un générateur aléatoire ou exhaustif. Notons que seuls les automates peuvent être engendrés exhaustivement, non les transducteurs.
- Le champ *automata* : nécessite de définir l'ensemble des champs suivants :
  - *number* : le nombre d'automates à engendrer (dans le cas d'un générateur aléatoire).
  - *size* : la taille des automates à engendrer.
  - *type* : il est possible de choisir entre *dfa* et *transductor*. Le premier renvoie des automates déterministes accessibles, le second des transducteurs.
  - *alphabet* : selon que l'on engendre un automate ou un transducteur, il faudra distinguer les alphabets de type *input* et *output*. Dans le cas des transducteurs, un champ *restriction* permet d'associer une lettre de l'alphabet d'entrée à un sous-ensemble de lettres de l'alphabet de sortie.
  - Les champs optionnels *complete*, *minimal* et *strongly_ connected* : il est possible de donner les trois valeurs *yes*, *no* ou *indifferent*, selon que l'on veut imposer que les automates vérifient ou non une de ces propriétés, ou ne rien imposer. Si les champs ne sont pas spécifiés, les valeurs par défaut sont respectivement *yes*, *indifferent* et *indifferent*.
  - *output* : permet de spécifier le format d'encodage des automates engendrés. L'utilisateur peut choisir entre *dot*, *gastex*, ou *xml*. Le format *dot* pourra ensuite être utilisé par programmes afin d'obtenir des affichages différents (*dot*, *neato*, *twopi*, *circo*, *fdp*, ...)

2.2 Exemples d'automates engendrés par PREGA

On donne ici deux exemples : un petit transducteur à 10 états et un automate à 100 états. De manière générale, il est beaucoup plus rapide d'engendrer un automate à l'aide de PREGA, que de l'afficher avec un rendu graphique appréciable. Les algorithmes qui tentent de minimiser le nombre de transitions qui se croisent n'arrivent pas à afficher des automates aléatoires de plus d'une centaine d'états. On a tenté d'afficher des grands automates à l'aide des outils utilisés par les biologistes pour leur réseaux d'interaction, mais le résultat n'est pas satisfaisant.

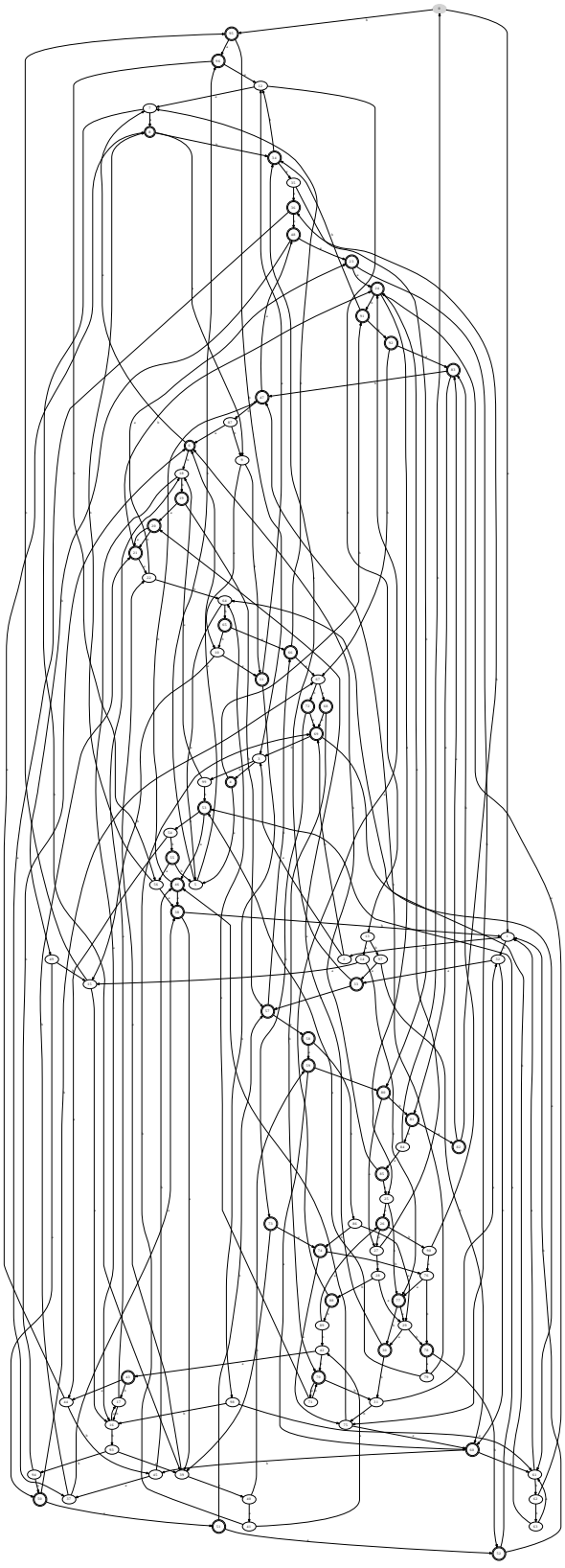


FIG. 5.2 – Automate engendré avec *PREGA*, de taille 100 sur l'alphabet $\{a, b\}$.

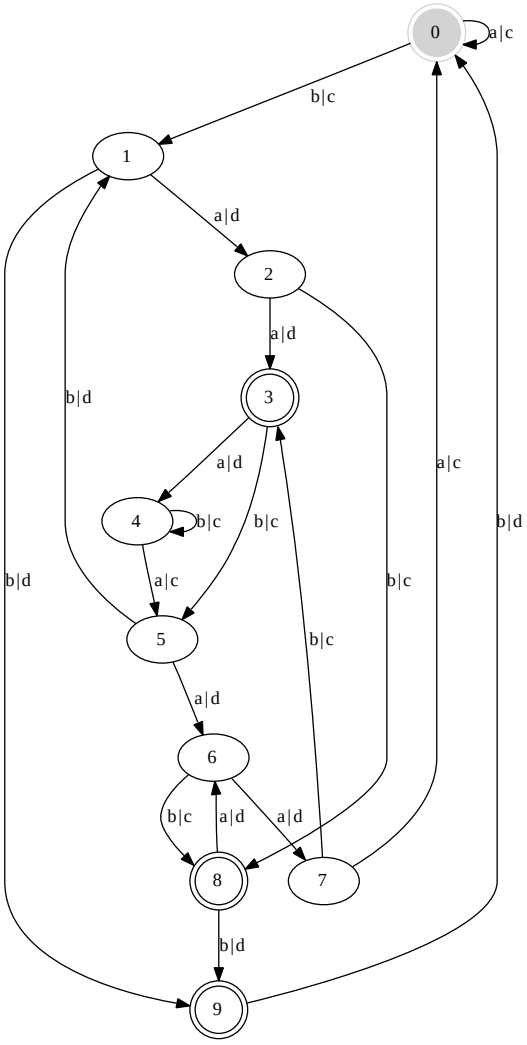


FIG. 5.3 – Transducteur engendré avec *PREGA*, de taille 10. Alphabet d'entrée $\{a, b\}$, alphabet de sortie : $\{c, d\}$. L'état initial est dessiné en gris.

3 Résultats expérimentaux

Cette section est dédiée à la présentation de résultats expérimentaux obtenus avec `REGAL`. Pour chaque test, les valeurs sont calculées sur la moyenne obtenue sur 10 000 automates.

3.1 Sur l'efficacité de la bibliothèque

L'utilisation de la méthode de Boltzmann permet d'éviter de trop lourds précalculs. On peut ainsi engendrer aléatoirement des automates de grandes tailles. L'utilisation de la méthode récursive limitait la génération à des automates de quelques centaines d'états. Les tableaux ci-dessous contiennent les temps moyens nécessaires à la génération des automates de différentes tailles.

n	100	500	1000	5000	10000
$k=2$	0.0014	0.010	0.035	0.37	1.1
$k=3$	0.0017	0.014	0.045	0.51	1.4
$k=4$	0.0021	0.020	0.054	0.55	1.8

TAB. 5.1 – Mesure du temps moyen (en secondes) nécessaire pour engendrer un automate déterministe accessible complet à n états sur un alphabet à k lettres

n	100	500	1000	5000	10000
$k=2$	0.0014	0.011	0.035	0.38	1.25
$k=3$	0.0018	0.015	0.045	0.52	1.3
$k=4$	0.0022	0.018	0.055	0.60	1.6

TAB. 5.2 – Mesure du temps moyen (en secondes) nécessaire pour engendrer un automate déterministe accessible possiblement incomplet à n états sur un alphabet à k lettres

3.2 Expériences à l'aide de générateurs exhaustifs

Automate minimal. Avec le générateur exhaustif, on peut calculer le nombre exact d'automates minimaux sur un alphabet à k lettres. Mais comme le nombre d'automates déterministes accessibles et complets de taille n est asymptotiquement proportionnel à $n\,2^n\,\left\{{kn \atop n}\right\}$ [10,52] où $\left\{{kn \atop n}\right\}$ est le nombre de Stirling de seconde espèce (soit le nombre de façons de partitionner un ensemble à kn éléments en n sous-ensembles non-vides), le nombre exact d'automates minimaux ne peut être calculé que pour de petites valeurs de n.

Nombre d'états	2	3	4	5	6	7
Automates minimaux	24	1 028	56 014	3 705 306	286 717 796	25 493 886 852

3.3 Expériences à l'aide de générateurs aléatoires

Automates minimaux.

Taille	10	100	500	1 000	5 000	10 000
$k=2$ (%)	81.33	85.06	85.32	85.09	85.32	85,24
$k=3$ (%)	96.92	99.78	99.99	99.98	99.99	99.99
$k=4$ (%)	99.53	99.99	99.99	99.99	99.99	99.99

Expérimentalement, il semblerait donc que la proportion d'automates minimaux tend vers une constante assez élevée d'environ 85%. Grâce à cela, nous avons pu créer un générateur d'automates minimaux de taille n : on génère un automate déterministe accessible complet de taille n, on le minimise avec l'algorithme de Moore. Si après minimisation, la taille de l'automate a varié, on rejette l'automate et on recommence. En pratique, ce générateur se révèle efficace.

Conjecture 21. *Soit un entier $k \geq 2$. Pour la distribution uniforme sur les automates déterministes accessibles complets à n états sur un alphabet à k lettres, il existe une proportion contante d'automates minimaux.*

Sur un alphabet à deux lettres nous avons testé de combien la taille d'un automate aléatoire est réduite par la minimisation. Le tableau suivant montre des résultats obtenus avec des automates de taille $10,000$.

Réduction de la taille	0	1	2	3	4	5
Proportion d'automates (%)	85.03	13.75	1.10	0.11	0.01	0

Automates émondés. On rappelle qu'un automate émondé est un automate à la fois accessible et co-accessible. Un automate co-accessible, est un automate dans lequel, pour tout état $q \in Q$, il existe un chemin qui mène vers un état final. Ainsi, l'automate ne contient pas d'état puits. Il serait donc intéressant d'étudier la proportion d'automates émondés parmi l'ensemble des automates accessibles.

Nombre d'états	10	100	1 000	5 000	10 000
Émondés (%)	96.42	99.70	99.92	99.98	100.00

Cette forte proportion d'automates co-accessibles nous a permis de créer un générateur aléatoire d'automates co-accessibles, basé sur le principe du rejet des automates n'ayant pas la bonne propriété.

Automates fortement connexes. Korshunov a estimé asymptotiquement la proportion d'automates fortement connexes parmi les automates déterministes accessibles complets [52] [53], c'est-à-dire les automates dans lesquels, pour tous états p et q il existe un chemin de p à q. Pour un alphabet de taille 2, la proportion constante d'automates fortement connexes donnée par Korshunov est d'environ 79.6 %.

Nombre d'états	10	100	500	1 000	5 000	10 000
$k = 2$ (%)	69.6	78.85	79.27	79.67	79.90	79.77
$k = 3$ (%)	91.63	94.34	94.37	94.06	93.08	94.32
$k = 4$ (%)	97.54	97.97	98.14	98.09	97.99	98.17

Les résultats donnés par `REGAL` montrent une rapide convergence vers l'estimation de Korshunov. Les tests ont été effectués avec 10 000 automates pour chaque taille. Comme pour les automates minimaux ou co-accessibles, le fait qu'une proportion constante d'automates déterministes accessibles complets soient fortement connexes permet d'obtenir un générateur efficace en utilisant un algorithme avec rejet.

Diamètre moyen. L'expérience suivante porte sur le diamètre moyen d'un automate déterministe accessible complet à n états sur un alphabet à 2 lettres. Le diamètre d'un automate est la longueur maximum parmi l'ensemble des plus courts chemins simples entre deux états. entre deux états. La Figure 5.4 présente le résultat de l'expérience.

Conjecture 22. *Soit un entier $k \geq 2$. Pour la distribution uniforme sur l'ensemble des automates déterministes accessibles complets, le diametre moyen d'un automate est $\mathcal{O}(\log n)$.*

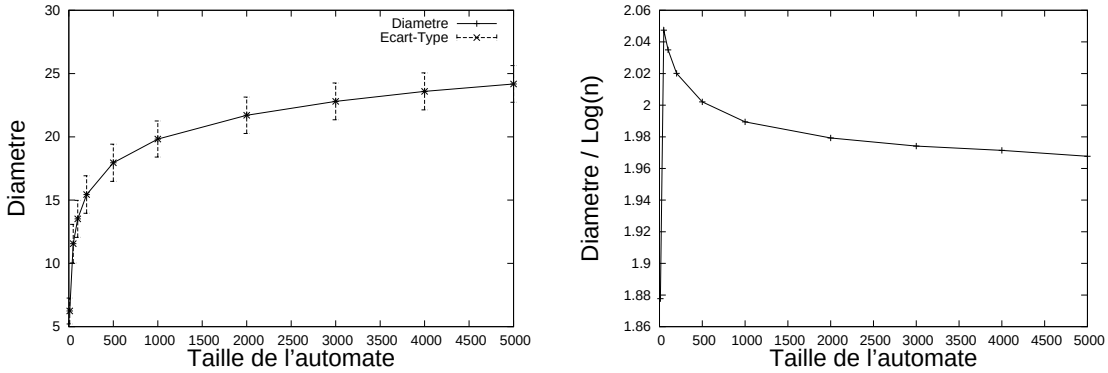


FIG. 5.4 – A gauche, la courbe illustre la mesure effectuée avec REGAL du diamètre moyen d'un automate déterministe accessible complet. A droite, les mêmes valeurs, divisées par $\log_2(n)$, où n est la taille de l'automate.

Applications aléatoires : taille moyenne de la partie accessible. Pour cette expérience, on a engendré aléatoirement et uniformément des automates déterministes complets, et on a calculé le pourcentage moyen d'états accessibles dans l'automate. Les résultats sont surprenants :

Nombre d'états	10	100	500	1 000	5 000	10 000
k=2 (%)	77.626	79.511	79.677	79.663	79.693	79.682
k=3 (%)	93.301	93.988	94.027	94.034	94.044	94.053
k=4 (%)	97.628	97.983	98.003	98.017	98.015	98.017

Il s'agit des mêmes valeurs que celles de la proportion d'automates fortement connexes parmi les accessibles ! La convergence semble même être plus rapide dans le cas présent.

Conjecture 23. *Soit un entier $k \geq 2$. Pour la distribution uniforme sur les automates déterministes complets à n états sur un alphabet à k lettres, il existe une constante $c_k \in]0,1[$ dépendante de k telle que la taille moyenne de la partie d'accessible soit $c_k n$.*

Automates possiblement incomplets. On effectue un ensemble de tests en engendrant 10 000 automates possiblement incomplets à 10 000 états.

- Pour $k = 2$, 80.1% des automates engendrés sont incomplets. Pour $k = 3$, cette proportion s'élève à 94.1%.
- Pour $k = 2$, 95.4% des automates sont minimaux.
- For $k = 2$, 79.0% sont fortement connexes, c'est-à-dire une proportion similaire au cas où les automates sont complets.
- Pour $k = 2$, un automate a en moyenne $1,6$ transitions indéfinies.

Bien que les automates soient majoritairement incomplets, le nombre de transitions indéfinies est en moyenne relativement faible. Il n'est pas possible d'obtenir un générateur efficace d'automates dont le nombre de transitions indéfinies est fixé, en utilisant simplement la méthode du rejet. La création d'un tel générateur reste donc un problème ouvert.

Troisième partie

Analyse en moyenne d'algorithmes de minimisation

Chapitre 6

Les algorithmes de minimisation

Ce chapitre présente un ensemble d'algorithmes de minimisation d'automates. Une taxonomie a déjà été faite sur le sujet par Bruce Watson en 1994 [70]. Le but n'est donc pas d'être exhaustif, mais plutôt de présenter les algorithmes dont on fera l'analyse par la suite et de citer un ensemble de travaux ultérieurs à l'article de Watson. On supposera toujours (sauf mention explicite) que les automates sont déterministes, accessibles et complets : il est de toute façon possible de calculer la partie accessible ou de compléter un automate avant de lui appliquer un algorithme de minimisation.

1 Équivalence de Myhill-Nerode

Avant toute chose, on rappelle la définition du crochet d'Iverson, dont l'utilisation est recommandée par Donald Knuth [50] : $[\![Cond]\!]$ vaut 1 si la condition $Cond$ est satisfaite et 0 sinon.

Relation d'équivalence régulière à droite :

Soit $\mathcal{A} = (A, Q, \cdot, q_0, F)$ un automate. Une relation d'équivalence $\equiv$ définie sur l'ensemble des états Q d'un automate déterministe est *régulière à droite* si

$$\text{pour tout mot } u \in A^* \text{ et pour tous } p, q \in Q, \quad p \equiv q \Rightarrow p \cdot u \equiv q \cdot u.$$

Relation d'équivalence de Myhill-Nerode :

Soit $\mathcal{A} =< Q, A, \cdot, q_0, F >$ un automate. Pour tout état q, on pose :

$$L_q = \{w \mid q \cdot w \in F\},$$

autrement dit L_q est l'ensemble des mots qui étiquettent un chemin partant de q et qui se terminent dans un état final ou encore, le langage reconnu par l'automate en prenant q comme état initial. On remarque donc que L_{q_0} est le langage reconnu par l'automate $\mathcal{A}$. On définit à présent la relation d'équivalence $\sim$ appelée *équivalence de Myhill-Nerode* [58], comme suit :

$$\text{pour tous états } p, q \in Q,\ p \sim q \Longleftrightarrow L_p = L_q.$$

Autrement dit, deux états p et q sont *équivalents*, lorsque l'ensemble des mots étiquetant un chemin qui part de p et aboutit dans un état final est égal à l'ensemble des mots étiquetant un chemin qui part de q et aboutit à un état final :

$$\text{pour tous états } p, q \in Q,\ p \sim q \iff \text{pour tout mot } w, [\![p \cdot w \in F]\!] = [\![q \cdot w \in F]\!].$$

On note $\sim$ la *partition de Myhill-Nerode*, où deux états sont dans le même sous-ensemble si et seulement si ils sont équivalents. La Figure 6.1 contient un exemple où les états 2 et 5 sont équivalents. Il est possible de fusionner deux états équivalents, sans modifier le langage reconnu par l'automate.

On étend à présent cette définition en prenant en compte la longueur des mots. Pour tout état q et tout entier i, on pose :

$$L_{q,i} = \{w \mid q \cdot w \in F \text{ et } |w| \leq i\},$$

Pour tout entier positif i, on dit de deux états $p, q \in Q$ de l'automate qu'ils sont *i-équivalents*, relation que l'on note $p \sim_i q$, si et seulement si pour tout mot u de longueur inférieure ou égale à i, on a $[\![p \cdot u \in F]\!] = [\![q \cdot u \in F]\!]$ ou, dit autrement :

$$\text{pour tous états } p, q \in Q, p \sim_i q \iff L_{p,i} = L_{q,i}.$$

À partir de cette notion, on définit la partition associée à la relation d'équivalence $\sim_i$ de l'ensemble Q des états de l'automate comme suit : deux états p et q sont dans le même sous-ensemble de la partition $\sim_i$ si et seulement si $p \sim_i q$.

Remarque 24. *Sachant que l'on a :*

$$p \sim_0 q \iff [\![p \in F]\!] = [\![q \in F]\!],$$

le calcul de la relation d'équivalence $\sim_0$ d'un automate revient à partitionner l'ensemble des états en deux sous-ensembles F et $Q \setminus F$.

On récapitule les propriétés de l'équivalence de Myhill-Nerode dans la proposition suivante.

Proposition 25. *Soit $\mathcal{A} = (A, Q, \cdot, q_0, F)$ un automate déterministe à n états. Il vérifie les propriétés suivantes :*

1. *Pour tout $i \in \mathbb{N}$, la relation d'équivalence $\sim_{i+1}$ est un raffinement de la relation d'équivalence $\sim_i$, ce qui signifie que pour tous $p, q \in Q$, si $p \sim_{i+1} q$ alors $p \sim_i q$.*
2. *Pour tout $i \in \mathbb{N}$ et pour tous $p, q \in Q$, $p \sim_{i+1} q$ si et seulement si $p \sim_i q$ et que pour tout $a \in A$, on a $p \cdot a \sim_i q \cdot a$.*
3. *Si, pour $i \in \mathbb{N}$, la relation d'équivalence $\sim_{i+1}$ est égale à la relation d'équivalence $\sim_i$ alors pour tout $j \geq i$, la relation d'équivalence $\sim_j$ est égale à la relation d'équivalence de Myhill-Nerode.*
4. *La relation d'équivalence $\sim_{n-2}$ est égale à la relation d'équivalence de Myhill-Nerode.*
5. *L'équivalence de Myhill-Nerode est régulière à droite.*

2 Automate quotient

Soit $\mathcal{A} = (A, Q, \cdot, q_0, F)$ un automate et $\equiv$ une relation d'équivalence sur l'ensemble des états Q qui est régulière à droite. L'*automate quotient* de $\mathcal{A}$ par $\equiv$ est l'automate

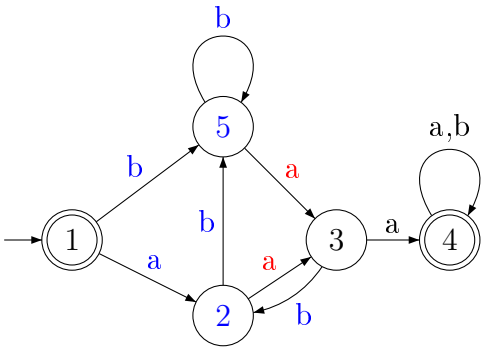
$$(\mathcal{A}/\equiv) = (A, Q/\equiv, *, [q_0], \{[f], f \in F\})$$

où $Q/\equiv$ est l'ensemble des classes d'équivalence, l'état $[q] \in Q/\equiv$ est la classe d'équivalence de $q \in Q$ et la fonction de transition $*$ est définie pour toute lettre $a \in A$ et tout état $q \in Q$ par $[q] * a = [q \cdot a]$. La validité de cette définition repose sur la régularité à droite de la relation d'équivalence $\equiv$.

Construction :

La plupart des algorithmes que nous allons présenter dans ce qui suit calculent une partition de l'ensemble des états de l'automate correspondant à l'équivalence de Myhill-Nerode. La construction de l'automate minimal se fait de la façon suivante (voir Exemple 6.1) : on crée autant d'états qu'il y a de sous-ensembles dans la partition. On considère que ces derniers sont triés en fonction de leur plus petit élément, dans l'ordre croissant. L'état i de l'automate quotient correspond au i-ème sous-ensemble de la partition. Pour tout état p de l'automate d'origine, on note $\pi[p]$ l'indice du sous-ensemble où l'état p se trouve dans la partition. Pour toute lettre a de l'alphabet, on ajoute une transition $(\pi[p], a, \pi[p \cdot a])$.

Automate $\mathcal{A}$:



Partition de Myhill-Nerode :

$$\begin{array}{rccccccc} \textit{Partition} = & \{ & \{1\} & \{2,5\} & \{3\} & \{4\} & \} \\ \pi = & [& 1 & 2 & 3 & 4 &] \end{array}$$

Automate Quotient $\mathcal{A}/\sim$:

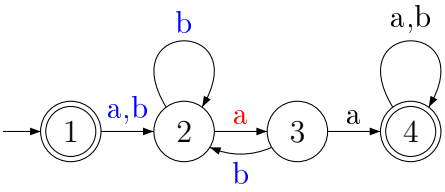


FIG. 6.1 – Exemple de construction de l'automate quotient

3 Automate minimal

On commence par donner une définition de l'automate minimal avant d'expliquer les raisons de son importance.

Théorème 26. *[57] Pour tout automate déterministe accessible complet $\mathcal{A}$, l'automate quotient $\mathcal{A}/\sim$ est l'unique automate déterministe complet ayant un nombre d'état minimal qui reconnaît le même langage que $\mathcal{A}$. On l'appelle l'automate minimal de $L(\mathcal{A})$.*

L'automate est unique à isomorphisme près, c'est-à-dire à l'étiquetage près de ses états. Le théorème 26 montre que l'automate minimal est une notion fondamentale en théorie des langages :

- c'est la représentation la moins coûteuse en terme d'espace mémoire d'un langage régulier par un automate déterministe,
- le fait qu'il soit unique définit une bijection entre langages réguliers et automates minimaux,
- il s'agit d'un automate canonique qui va permettre de tester l'équivalence de deux automates ou l'égalité de deux langages rationnels. Dans le premier cas, il suffit de calculer les automates minimaux à l'aide de l'une des méthodes que l'on va présenter dans le reste de ce chapitre, dans le second, il faut au préalable transformer les langages en automates (algorithme de Thompson, algorithme de Glushkov [13, 37, 61]).

4 Algorithme de Moore

En 1956, Edward F. Moore décrit le premier algorithme de minimisation [57], sans doute l'un des premiers algorithmes de la théorie des automates. Dans la littérature, il arrive que la méthode soit attribuée à Huffman [45]. L'algorithme calcule l'équivalence de Myhill-Nerode que l'on vient de définir, en se basant principalement sur les propriétés (2) et (3) de la proposition 25. Dans un souci de clarté, on commence par donner une description grossière de l'algorithme, que l'on précise par la suite.

Description grossière :

L'algorithme cherche à calculer la relation d'équivalence $\sim$, afin de l'utiliser pour construire l'automate quotient en utilisant la méthode décrite dans la section 2. D'après la Propriété (3) de la proposition 25, on a :

$$\forall i \in \mathbb{N},\ \sim = \sim_i \Longleftrightarrow \sim_i = \sim_{i+1}$$

De plus la Propriété (2) indique qu'il est possible que calculer une relation d'équivalence $\sim_i$ en raffinant la relation d'équivalence $\sim_{i-1}$ selon la règle suivante :

$$p \sim_{i+1} q \Leftrightarrow \begin{cases} p \sim_i q \\ p \cdot a \sim_i q \cdot a \quad \forall a \in A \end{cases}$$

Le calcul peut donc se faire par itérations successives en partant de la relation d'équivalence $\sim_0$ (voir remarque 24). Seules deux partitions on besoin d'être stockées simultanément : celle que l'on s'apprête à calculer ($\sim_i$) et celle que l'on vient de calculer ($\sim_{i-1}$). Dans la description qui suit, elles sont stockées sous la forme de tableaux π et π', qui à chaque état associent l'indice de sa classe d'équivalence. L'algorithme 12 contient donc cette vision simpliste de l'algorithme.

Remarque 27. *Dans cette description, on peut voir que la complexité de l'algorithme va dépendre de deux paramètres :*

1. *le coût de l'opération* `raffiner` π `en` π'*, que l'on appellera par la suite* `itération` *de l'algorithme de Moore.*
2. *le nombre de fois que l'on exécute cette opération.*

Algorithme 12 : Algorithme de Moore

Données : Deux tableaux π et π'

```
1 begin
2 |  pour tous les p ∈ {1, ··· , n} faire
3 |  |  π'[p] = [[p ∈ F]]
4 |  tant que π ≠ π' faire
5 |  |  π = π'
6 |  |  raffiner π en π'
7 |  fin
8 end
```

Plus en détails On va à présent décrire une méthode efficace pour implanter l'algorithme. Une description complète est donnée dans la Figure 6.2 et un exemple de l'exécution de l'algorithme est donnée dans la Figure 6.4. Tout d'abord, les lignes 1 à 4 correspondent aux cas particuliers où $F = \emptyset$ ou $F = Q$: il n'est pas nécessaire de calculer $\sim_1$. L'automate minimal est dans ce cas de taille 1 et $*$ est la fonction de transition telle que $1 * a = 1$ pour tout $a \in A$.

Afin de calculer efficacement π' à partir de π, on utilise la méthode suivante :

- Pour chaque état p, on calcule une signature $s[p]$ composé de sa classe d'équivalence $\pi[p]$, ainsi que de celle de ces successeurs, $\pi[p \cdot a]$, pour toute lettre a de l'alphabet. Ainsi, $p \sim_{i+1} q$ si et seulement si $s[p] = s[q]$. On va ensuite minimiser le nombre de comparaison de signatures à effectuer.
- On trie les états selon leurs signatures, à l'aide d'un tri lexicographique. Cela revient à faire $k + 1$ tri par bac successifs (chaque bac correspondant à une classe d'équivalence) des états de l'automate. Le coût total du tri est $\mathcal{O}(kn)$ en temps et $\mathcal{O}(n)$ en espace.
- On parcourt la liste triée et on compare deux à deux les signatures des états.

Le coût d'une itération est donc $\mathcal{O}(kn)$.

Le résultat qui suit est une description plus précise du pire cas, qu'on utilisera pour l'analyse en moyenne.

Pour tout entier $n \geq 1$ et tout $m \in \{0, \cdots, n-2\}$, on note $\mathcal{A}_n^m$ l'ensemble des automates déterministes à n états où m est le plus petit entier pour lequel la relation d'équivalence $\sim_m$ est égale à l'équivalence de Myhill-Nerode. On notera aussi $\mathcal{M}oore(\mathcal{A})$ le nombre d'itérations effectuées par l'algorithme lorsqu'on l'applique sur l'automate $\mathcal{A}$.

Lemme 28. *Pour tout automate $\mathcal{A}$ de $\mathcal{A}_n^m$, les propriétés suivantes sont satisfaites :*

- *le nombre d'itérations $\mathcal{M}oore(\mathcal{A})$ effectuées par l'algorithme est égal à 0 si $L(\mathcal{A}) = \emptyset$ ou $L(\mathcal{A}) = A^*$ et est égal à $m + 1$ autrement.*
- *$\mathcal{M}oore(\mathcal{A})$ est toujours inférieur ou égal à $n - 1$.*
- *la complexité dans le pire cas $\mathcal{W}(n, m)$ de l'algorithme de Moore est égale à $\Theta((m + 1)n)$, où le Θ est uniforme pour $m \in \{0, \ldots, n-2\}$, ou de façon équivalente il existe deux nombres réels positifs C_1 et C_2 indépendants de n et m tel que $C_1(m + 1)n \leq \mathcal{W}(n, m) \leq C_2(m + 1)n$.*

Démonstration. La boucle principale est itérée exactement $m+1$ fois lorsque l'ensemble F d'états terminaux n'est ni vide, ni égal à $\{1, \ldots, n\}$. De plus, on sait de la

Algorithme 13 : Algorithme de Moore

```
1  si F = ∅ alors
       Résultat : (A, {1}, ∗, 1, ∅)
2  si F = {1, ··· , n} alors
       Résultat : (A, {1}, ∗, 1, {1})
3  pour tous les p ∈ {1, ··· , n} faire
4      π'[p] = [[p ∈ F]]
5  π = indefini
6  tant que π ≠ π' faire
7      π = π'
8      pour tous les p ∈ {1, ··· , n} faire
9          s[p] = (π[p], π[p · a_1], ··· , π[p · a_k])
10     Calculer la permutation σ qui trie les états en fonction de s[]
11     à l'aide d'un tri lexicographique
12     i = 0
13     π'[σ(1)] = i
14     pour tous les p ∈ {2, ··· , n} faire
15         si s[p] ≠ s[p − 1] alors  i = i + 1
16         π'[σ(p)] = i
   Résultat : le quotient de A par π
```

FIG. 6.2 – Description de l'algorithme de Moore

propriété (4) de la proposition 25 que l'entier m est inférieur ou égal à $n-2$. Si F est vide ou égal à $\{1,\ldots,n\}$, alors on a $m=0$ et le temps nécessaire pour déterminer la taille de F est $\Theta(n)$.

L'initialisation et la construction de l'automate minimal à partir de la relation d'équivalence de Myhill-Nerode sont toutes deux faites en temps $\Theta(n)$. La complexité de chaque itération de la boucle principale est $\Theta(n)$. Plus précisément, le coût d'une itération est inclus dans $[C_1 n, C_2 n]$. Les constantes C_1 et C_2 ne dépendent pas de m, ce qui prouve l'uniformité pour les bornes inférieure et supérieure. □

Notons que le lemme 28 prouve que la complexité de l'algorithme de Moore dans le pire cas est $\mathcal{O}(n^2)$, puisqu'il n'y a pas plus de $n-1$ itérations. Il est relativement facile de voir que cette borne est atteinte et que la complexité est $\Theta(n^2)$. La Figure 6.3 illustre un cas d'automate minimisé en un nombre maximum d'opérations.

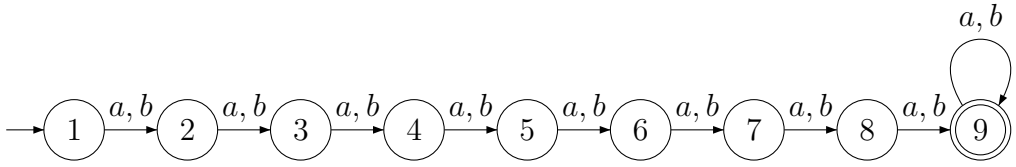


FIG. 6.3 – Automate pour lequel le pire cas de l'algorithme de Moore est atteint.

Avantages de l'algorithme :

- Hormis le tri lexicographique, nécessaire pour garantir sa complexité quadratique, l'algorithme de Moore demeure relativement simple à implanter.
- À la fin d'une itération i, il est possible de dire de deux états distincts p et q s'ils sont ou non i-équivalents. On verra que ce n'est pas le cas pour beaucoup d'algorithmes.
- De plus, l'étude de la complexité de l'algorithme peut se faire en deux parties distinctes :

  1. le coût d'une itération,
  2. le nombre d'itérations.

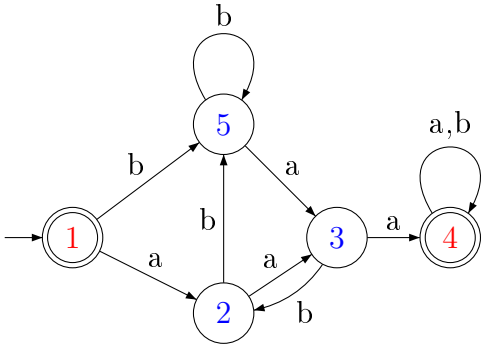
  Dans le Chapitre 7 , l'étude de la complexité moyenne de l'algorithme se concentre donc sur l'étude du nombre moyen d'itérations.
- Les exemples d'automates pour lesquels la complexité dans le pire cas ou la complexité dans le meilleur cas est bien atteinte sont abondants et intuitifs.

Inconvénient majeur de l'algorithme :

- Les calculs inutiles : à chaque itération, la classe d'équivalence de l'état d'arrivée de chaque transition est calculée. Or, si celle-ci n'a pas été modifiée lors de l'itération précédente, elle n'aura pas d'impact sur le raffinement de partition en cours. Dans l'exemple de la Figure 6.3, à chaque itération, l'algorithme traite toutes les transitions de l'automate. Or, l'analyse d'une seule transition par itération, correctement choisie, suffirait à calculer l'équivalence de Myhill-Nerode.

Exemple pour l'algorithme de Moore

Initialisation : Séparation des états terminaux et non terminaux

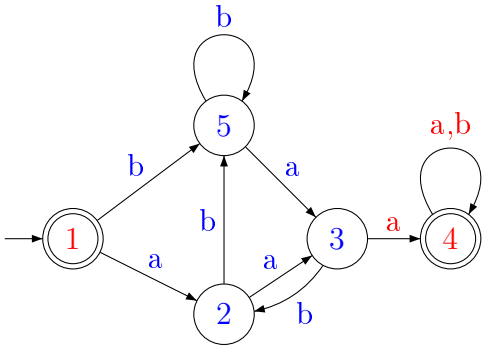


	π'
1	1
2	0
3	0
4	1
5	0

$\mathcal{P}_0 = \{\{1,4\}\{2,3,5\}\}$

1ère itération :

– Calcul de la classe d'équivalence de l'état d'arrivée de toutes les transitions :



	π	a	b
1	1	0	0
2	0	0	0
3	0	1	0
4	1	1	1
5	0	0	0

– Tri lexicographique des états :

Tri par b.
Entrée : $1, 2, 3, 4, 5$

Classe	Liste d'états
0	$1, 2, 3, 5$
1	4

Tri par a.
Entrée : $1, 2, 3, 5, 4$

Classe	Liste d'états
0	$1, 2, 5$
1	$3, 4$

Tri par π.
Entrée : $1, 2, 5, 3, 4$

Classe	Liste d'états
0	$2, 5, 3$
1	$1, 4$

– Calcul de π' en comparant deux à deux les éléments de $2, 5, 3, 1, 4$

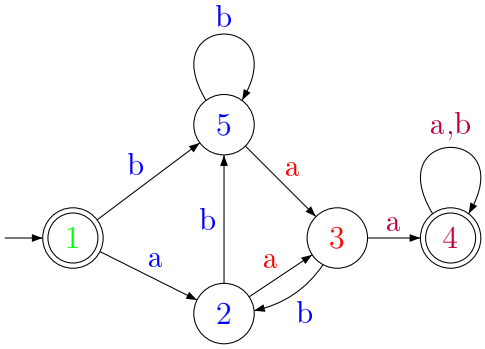
	π	a	b	π'
2	0	0	0	0
5	0	0	0	0
3	0	1	0	1
1	1	0	0	2
4	1	1	1	3

$\mathcal{P}_1 = \{1\}\{2,5\}\{3\}\{4\}$

$\mathcal{P}_0$ est différente de $\mathcal{P}_1$.
(Se vérifie en gardant un compteur sur le nombre de classes d'équivalence).
L'algorithme se poursuit.

2ème itération :

– Calcul de la classe d'équivalence de l'état d'arrivée de toutes les transitions :



	π	a	b
1	2	0	0
2	0	1	0
3	1	3	0
4	3	3	3
5	0	1	0

– Tri lexicographique des états :

Tri par b.
Entrée : $1, 2, 3, 4, 5$

Classe	Liste d'états
0	$1, 2, 3, 5$
1	–
2	–
3	4

Tri par a.
Entrée : $1, 2, 3, 5, 4$

Classe	Liste d'états
0	1
1	$2, 5$
2	–
3	$3, 4$

Tri par π.
Entrée : $1, 2, 5, 3, 4$

Classe	Liste d'états
0	$2, 5$
1	3
2	1
3	4

– Calcul de π' en comparant deux à deux les éléments de $2, 5, 3, 1, 4$

	π	a	b	π'
2	0	1	0	0
5	0	1	0	0
3	1	3	0	1
1	2	0	0	2
4	4	3	3	3

$\mathcal{P}_2 = \{1\}\{2, 5\}\{3\}\{4\}$

$\mathcal{P}_2$ est égale à $\sim_1$.
L'algorithme s'arrête.

Construction de l'automate minimal :

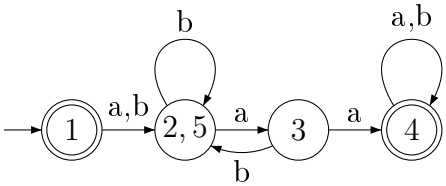


FIG. 6.4 – Exemple d'exécution de l'algorithme de Moore

5 Algorithme de Hopcroft

L'algorithme de Hopcroft [43], publié en 1971, est aussi basé sur le calcul de l'équivalence de Myhill-Nerode. On peut le voir comme une amélioration de l'algorithme de Moore, puisque celui-ci réduit les calculs inutiles mentionnés dans la section précédente. En effet, l'algorithme effectue un calcul sur une transition de l'automate si et seulement si l'état d'arrivée de cette dernière appartient à un sous-ensemble de la partition qui a subi une modification. La complexité est, grâce à cela, $\mathcal{O}(n \log n)$: il s'agit de l'algorithme le plus efficace dans le pire cas, parmi ceux capables de minimiser n'importe quel automate déterministe. On retrouve des utilisations de la méthode de partitionnement utilisée par l'algorithme de Hopcroft dans des algorithmes sur les graphes [20, 21].

Algorithme 14 : Hopcroft

Données : $Partition = \{F, \overline{F}\}$, $C = min\{F, \overline{F}\}$, $Structure = \emptyset$
pour chaque $a \in A$ **faire**
  Ajouter (C, a) dans $Structure$
tant que $Structure \neq \emptyset$ **faire**
  $(P, a) = Extraire(Structure)$
  pour chaque $B \in Partition$ *tel que* (P, a) $raffine$ B **faire**
    $B', B'' \leftarrow Raffiner(B, P, a)$
    Casser le bloc B *en* B' *et* B'' *dans la Partition*
    $MiseAJour(Structure, B, B', B'')$
Résultat : le quotient de $\mathcal{A}$ par $Partition$

Algorithme 15 : $MiseAJour(Structure, B, B', B'')$

Données : $C = min\{B', \overline{B''}\}$
pour chaque $b \in A$ **faire**
  si $(B, b) \in Structure$ **alors**
    Remplacer (B, b) par (B', b) et (B'', b) dans Structure
  sinon
    Ajouter (C, b) dans $Structure$

FIG. 6.5 – Description de l'algorithme de Hopcroft

La Figure 6.5 contient la description de l'algorithme de Hopcroft. Avant d'expliciter celle ci, on précise que :

- $min\{A, B\}$ désigne l'ensemble de plus petite taille parmi A et B et n'importe lequel des deux s'ils ont la même taille.
- un *bloc* (C, a) désigne l'ensemble des transitions étiquetées par la lettre a et dont la classe d'équivalence de l'état d'arrivée est désignée par C. La *Structure* utilisée dans la description permet de stocker les blocs. Elle est couramment appelée *waiting set* dans la littérature anglophone. C'est par le biais de cette *Structure* que l'algorithme sélectionne les transitions avec lesquelles il raffine la partition.

On détaille à présent l'Algorithme 14.

Initialisation. On partitionne l'ensemble des états de l'automate en deux sous-ensembles : les terminaux et les non terminaux. Il s'agit là du même précalcul que dans l'algorithme de Moore. Les blocs désignant les transitions entrant dans le plus petit des deux sous-ensembles sont ensuite ajoutés dans la *Structure*. Si les sous-ensembles sont de même taille, on en choisit un au hasard.

Boucle Principale. Tant que la *Structure* n'est pas vide, on extrait un bloc (P, a) de celle-ci (Ligne 4) et on le traite de la façon suivante :

- Une classe d'équivalence B est raffinée par (P, a) si et seulement si il existe une partition de B en deux sous-ensembles non vides B' et B'' tels que :

  $$\text{pour tout état } p \in B',\ p \cdot a \in P \text{ et pour tout état } q \in B'',\ q \cdot a \notin P$$

  En calculant l'automate inverse au début de l'algorithme, il est possible d'effectuer ce calcul en un temps proportionnel aux nombres de transitions désignées par (P, a).
- Ligne 7 : On remplace B par B' et $B'' = B \setminus B'$ dans la partition. Bien que l'on ne décrive pas ici l'ensemble des structures de données nécessaire pour effectuer efficacement cette opération, on remarque que le coût de celle-ci est proportionnel au nombre d'états dans B' (si on extrait de B tous les états de B', alors $B = B''$).
- Ligne 8 et Algorithme 15 : On met à jour la *Structure*. Pour toute lettre b de l'alphabet, si le bloc (B, b) appartenait déjà à la *Structure*, on le remplace par les blocs (B', b) et (B'', b). Dans le cas contraire, on ajoute uniquement le bloc (C, b), où C correspond au plus petit des deux ensembles B' et B''. À ce stade, on manipule les blocs, c'est-à-dire des paires $(entier, lettre)$, et non l'ensemble des transitions qu'ils représentent. Le coût de cette étape est donc $\mathcal{O}(k)$, où k est la taille de l'alphabet.

Le nombre d'opérations effectuées dans la boucle principale est proportionnel à la somme des cardinaux des ensembles de transitions associés aux blocs extraits de la Structure. Autrement dit, si $|(P, a)|$ est le nombre de transitions associées au bloc (P, a), alors le coût de la boucle principale est :

$$\sum_{(P,a)\ extrait\ de\ Structure} (c \times |(P, a)|),$$

où c est une constante multiplicative dépendant de l'implantation de l'algorithme.

Fin de l'algorithme. Une fois que la *Structure* est vide, la partition obtenue est celle définie par l'équivalence de Myhill-Nerode. On peut donc l'utiliser pour calculer l'automate minimal.

Inconvénients de l'algorithme. Partant d'une telle description, il n'est pas facile de se rendre compte que l'algorithme calcule bel et bien la partition de Myhill-Nerode. Ainsi, plusieurs preuves de sa validité ont été données [39, 43, 51]. De plus, trouver un ensemble d'automates pour lequel le pire cas est atteint n'est pas simple et le problème a fait l'objet de nombreuses études [15, 16, 22, 23]. Il semble donc que l'analyse de l'algorithme de Hopcroft soit plus complexe que celle de l'algorithme de

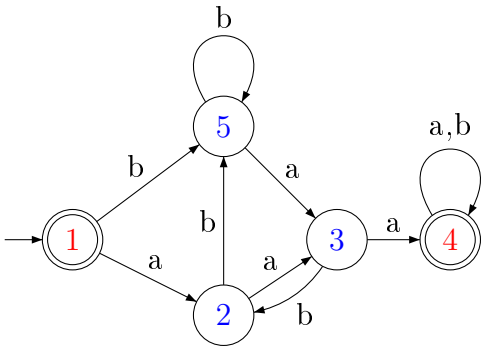
Moore. On peut imputer cela au fait que l'algorithme ne soit pas déterministe, mais aussi au fait que, contrairement à l'algorithme de Moore, on ne possède pas d'information sur les relations de i-équivalence entre les états, hormis lorsque l'algorithme s'arrête.

Le second inconvénient de l'algorithme de Hopcroft est son implantation, qui nécessite l'utilisation d'un ensemble de structures de données afin de garantir la complexité dans le pire cas (voir [56]), ce qui impose au programmeur d'avoir un minimum d'expérience.

Avantages de l'algorithme. Non seulement la complexité dans pire cas de l'algorithme de Hopcroft est la meilleure connue à ce jour, mais pour certains automates minimisés en temps $\mathcal{O}(n^2)$ par l'algorithme de Moore, l'algorithme de Hopcroft s'exécute en temps linéaire. La Figure 6.3 page 76 en est un exemple : un bloc inséré dans la *Structure* contient toujours exactement une transition et le traitement d'un bloc peut soit ne pas modifier la partition, soit créer un sous-ensemble de taille 1. Comme on va le voir à présent, il existe des algorithmes spécialisés dont la complexité dans le pire des cas est linéaire. Dans le cas général, l'existence d'un tel algorithme reste un problème ouvert.

Exemple pour l'algorithme de Hopcroft File

Initialisation :

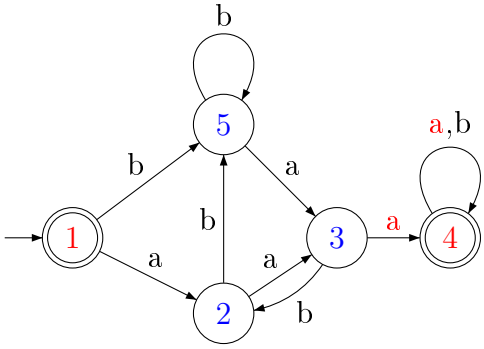


Partition = { $\{1,4\}$ $\{2,3,5\}$ }
Classe = [0 1]

Ajout ↓	
$(0,b)$	Extraction
$(0,a)$	↓

Exécution :

Raffinement par $(0,a)$:

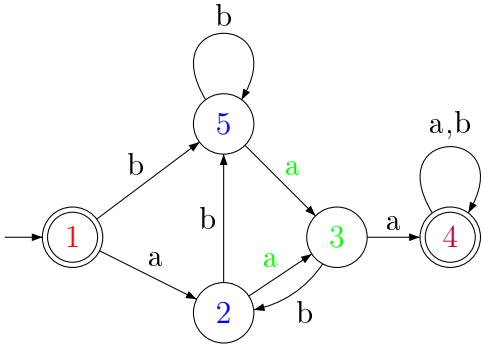


Partition = { $\{1\}$ $\{2,5\}$ $\{3\}$ $\{4\}$ }
Classe = [0 1 2 3]

$(3,b)$
$(3,a)$
$(2,b)$
$(2,a)$
$(0,b)$

Raffinement par $(0,b)$: la partition n'est pas modifiée, car aucune transition n'entre dans un état de classe d'équivalence 0.

Raffinement par $(2,a)$:



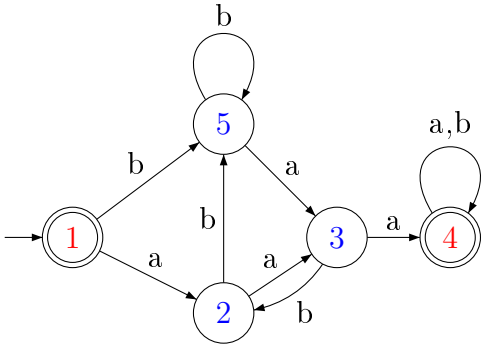
La partition n'est pas modifiée car 2 et 5 ont tous deux une transition étiquetée par a allant dans la classe 2.

$(3,b)$
$(3,a)$
$(2,b)$

Raffinement par $(2,b)$: la partition n'est pas modifiée
Raffinement par $(3,a)$: la partition n'est pas modifiée
Raffinement par $(3,b)$: la partition n'est pas modifiée
La pile est vide, l'algorithme s'arrête. On calcule l'automate minimal en utilisant la partition, de la même façon que l'algorithme de Moore.

Exemple pour l'algorithme de Hopcroft Pile

Initialisation :

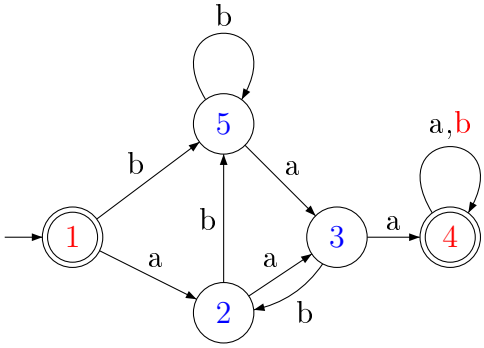


Partition = { $\{1,4\}$ $\{2,3,5\}$ }
Classe = [0 1]

Ajout ↓		Extraction ↑
	$(0,b)$ $(0,a)$	

Exécution :

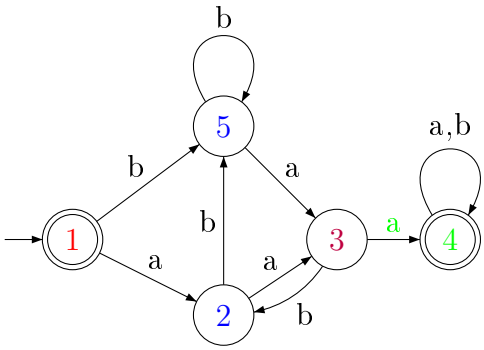
Raffinement par $(0,b)$



Partition = { $\{1\}$ $\{2,3,5\}$ $\{4\}$ }
Classe = [0 1 2]

$(2,b)$ $(2,a)$ $(0,a)$

Raffinement par $(2,b)$: la partition n'est pas modifiée
Raffinement par $(2,a)$:



Partition = { $\{1\}$ $\{2,5\}$ $\{3\}$ $\{4\}$ }
Classe = [0 1 3 2]

$(3,b)$ $(3,a)$ $(0,a)$

Raffinement par $(3,b)$: la partition n'est pas modifiée
Raffinement par $(3,a)$: la partition n'est pas modifiée
Raffinement par $(0,a)$: la partition n'est pas modifiée
La pile est vide, l'algorithme s'arrête. On calcule l'automate minimal en utilisant la partition, de la même façon que l'algorithme de Moore.

6 Rapide survol d'autres algorithmes

On présente ici un ensemble d'algorithmes pour lesquels une étude en moyenne à été effectuée, ou pour lesquels on peut montrer avec des arguments simples que la complexité moyenne est égale au pire des cas.

L'algorithme de Hopcroft-Ullman [18, 44] permet également de minimiser tous les automates déterministes. Il est inutile d'en faire une analyse en moyenne puisque sa complexité est $\Theta(n^2)$. En effet, l'algorithme construit une table contenant toutes les paires d'états (p, q) de l'automate, et calcule si ces derniers sont ou non équivalents :

- si $[\![p \in F]\!] \neq [\![q \in F]\!]$, on note dans la table que la paire d'états (p, q) a été testée et que $p \not\sim q$. En ayant au préalable calculé l'automate inverse, il est possible de parcourir l'ensemble des chemins menant à (p, q). Pour tout mot w et toute paire d'états (r, s) telle que $r \cdot w = p$ et $s \cdot w = q$, on marque alors la paire d'états (r, s) comme étant testée et $r \not\sim q$.
- si $[\![p \in F]\!] = [\![q \in F]\!]$, on note dans la table que la paire d'états a été testée.

L'algorithme s'arrête lorsque toutes les paires d'états ont été testées : les états équivalents sont ceux qui n'ont pas été marqués comme non-équivalents. L'avantage de cette méthode est qu'elle peut également être utilisée pour calculer l'équivalence de deux automates déterministes, sans passer par leur automate minimal : on construit pour cela une table contenant les états des deux automates. Les automates sont équivalents si les deux états initiaux le sont.

L'algorithme de Brzozowski [19] représente un cas très particulier parmi les algorithmes de minimisation : il permet de minimiser des automates non-déterministes et n'est pas basé sur le calcul de l'équivalence de Myhill-Nerode. L'idée est la suivante : on inverse l'automate, on le déterminise, on l'inverse de nouveau et on de le déterminise de nouveau. L'automate obtenu est l'automate minimal. L'algorithme a une complexité exponentielle dans le pire cas, due aux deux étapes de déterminisation. Dans [24], une variante de l'algorithme est donnée, où la deuxième étape de déterminisation est remplacée par une procédure spécifique, qui permet d'en améliorer les performances.
Dans [60], Cyril Nicaud montre que la complexité en moyenne de l'algorithme de Brzozowski, pour la distribution uniforme sur les automates à groupes, reste exponentielle.

On décrit à présent un ensemble d'algorithmes qui, en se restreignant à un sous-ensemble des automates déterministes, vont obtenir de meilleures performances que les algorithmes classiques. Aucun de ces algorithmes n'est mentionné dans l'article de Watson [70], pour la simple raison qu'ils ont pour la plupart été proposés ultérieurement.

Algorithme de Revuz [65] : les automates acycliques. On rappelle que dans un automate acyclique, il n'existe aucun chemin d'un état vers lui-même. L'algorithme de Dominique Revuz, en se restreignant au sous-ensemble des automates respectant cette propriété, a une complexité en temps linéaire. L'idée est la suivante :

on définit la hauteur d'un état comme étant la distance maximum entre cet état et un état final. On partitionne ensuite l'ensemble des états grâce à cette fonction. Pour tous les états p et q dont la distance maximale est 0, on a :

$$p \sim q \Longleftrightarrow [\![p \in F]\!] = [\![q \in F]\!].$$

On calcule le reste de la partition de manière récursive : pour toute paire d'états p et q de hauteur i,

$$p \sim q \Longleftrightarrow ([\![p \in F]\!] = [\![q \in F]\!]) \wedge (\forall a \in A,\ p \cdot a \sim q \cdot a)\,.$$

Les états $p \cdot a$ et $q \cdot a$ étant de hauteur strictement inférieure à i, ce calcul peut-être fait en temps constant. On a bien un algorithme de complexité linéaire. Si l'on suppose que les automates sont accessibles, chaque état est parcouru au moins une fois et la complexité moyenne, pour toute distribution, est $\Theta(n)$.
A noter qu'une extension de cet algorithme permet de minimiser les automates contenant au plus un cycle en temps linéaire [2].

Algorithme de Nicaud [59] : les automates unaires. Un automate unaire, c'est-à-dire pour lequel l'alphabet ne contient qu'une seule lettre, notée a, est un automate dont la structure de transitions est extrêmement contrainte : les automates accessibles, la transition sortant d'un état p va obligatoirement dans l'état $p+1$, à l'exception de celle partant du dernier état, qui est l'unique transition pour laquelle il existe n états d'arrivée possibles. Dans [59], Cyril Nicaud montre que le calcul de l'automate minimal peut être fait en se restreignant à des tests sur l'ensemble des états terminaux, que l'on ne décrira pas ici. L'algorithme parvient ainsi à calculer l'automate minimal en temps $\Theta(n)$.

Algorithme de Aït Mous-Bassino-Nicaud : l'automate A^*X, quand X est fini [3]. Cet article récent est une modification de l'algorithme de Brzozowski, qui permet calculer l'automate minimal de A^*X, où X est un ensemble fini de mots finis et pour lequel le nombre de mots est fixé. Ils construisent pour cela un automate co-déterministe et remplace la deuxième étape de déterminisation par un calcul équivalent utilisant la table de bords. La complexité de cet algorithme, qui peut être implanté avec des listes creuses, est linéaire en temps et quadratique en espace.

On clôt cette section en évoquant des algorithmes de minimisation restreints à des sous-ensembles d'automates :

- les algorithmes de minimisation des automates incomplets [12, 69], basés sur l'algorithme de Hopcroft, permettent de minimiser des automates en temps $\mathcal{O}(m \log n)$, où m est le nombre de transitions de l'automate.
- l'algorithme de Béal-Crochemore [11] permet de minimiser des automates locaux, c'est-à-dire des automates dans lesquels, pour tout état p, il existe un mot w tel que tout chemin étiqueté par w aboutit dans p. Son efficacité dépend du nombre d'états équivalents dans l'automate. Plus exactement, la complexité est linéaire si l'automate est déjà minimal et égal à $\mathcal{O}(m \log n)$, où m est le nombre de transitions de l'automate, dans le pire des cas. Une analyse plus fine du coût moyen de l'algorithme nécessite de connaître le nombre moyen d'états équivalents dans un automates local de taille n.

Chapitre 7

Étude de l'algorithme de Moore

Comme cela a été évoqué dans la section 4 du chapitre 6, l'analyse de la complexité de l'algorithme de Moore peut être décomposée en deux parties :

- celle du coût d'un raffinement de partition, dont on sait qu'il est égal à $\mathcal{O}(kn)$ si on utilise un tri lexicographique,
- celle du nombre de raffinements de partition, dont on sait qu'il est inférieur à $n-1$.

Ce chapitre contient un ensemble de résultats sur les performances de l'algorithme de Moore, mais qui ne font pas partie de son analyse en moyenne. Les résultats sont les suivants :

- Dans la section 1, on présente un ensemble de remarques sur l'implantation de l'algorithme qui permet, par des arguments simples, de réduire expérimentalement le coût d'un raffinement de partition. On présentera un résultat expérimental obtenu à l'aide de *REGAL*, comparant le temps d'exécution moyen de l'implantation optimisée avec l'implantation classique. Le résultat obtenu étant trop élémentaire pour être publié, n'apparaît pas dans la littérature. Le lecteur conviendra toutefois qu'il mérite d'être précisé ici.
- Dans la section 2, on montre dans un premier temps que le nombre de raffinements de partition effectués par l'algorithme pour un automate déterministe quelconque est égal au nombre de raffinements de partition que l'algorithme effectuerait pour son automate minimal. On étudie ensuite le comportement de l'algorithme appliqué à l'ensemble restreint des automates minimaux : on montre que la borne inférieure, c'est-à-dire la complexité de l'algorithme dans le meilleur cas, est alors égale à $\Omega(n \log\log n)$ lorsque l'alphabet contient au moins deux lettres et $\Omega(n \log n)$ lorsqu'il n'en contient qu'une. Ce résultat apparaît dans [8].

1 Une implantation efficace de l'algorithme

On décrit ici une optimisation de l'implantation classique de l'algorithme, basée sur un argument très simple, mais qui permet expérimentalement de gagner une constante sur le temps moyen de calcul de l'algorithme.

L'idée est la suivante : à une itération donnée de l'algorithme, certains états vont être isolés des autres, de telle sorte que l'ont peut garantir que leur classe d'équivalence ne sera plus jamais scindée par l'algorithme. On a pu recenser les cas

suivants :

1. La classe d'équivalence ne contient qu'un seul état.
2. La classe d'équivalence contient un ensemble d'états dont les transitions sortantes étiquetées par une même lettre ont le même état d'arrivée.

Dans le cas 1, il suffit de faire un test sur le cardinal des classes d'équivalences, lors de leur création. Ce calcul est simple et peu coûteux. De plus, tout état qui n'a pas d'équivalent dans l'automate sera dans le cas 1 à la fin de l'algorithme. Cette optimisation sera donc d'autant plus efficace si ces états sont isolés rapidement. Dans l'exemple classique où la complexité dans le pire cas de l'algorithme de Moore est atteinte (Figure 6.3), on isolera à chaque itération un nouvel état. Le coût de chaque itération sera donc proportionnel au nombre d'état restant et le temps de calcul au final divisé par 2.

Le cas 2 nécessite une phase de précalcul pour identifier les ensembles d'états concernés. Cependant, lorsque la taille de l'alphabet est supérieure ou égale à 3, on conjecture que la proportion d'automates déterministes accessibles complets pour lesquels il existe une paire d'états dans le cas 2, est négligeable. En revanche, il peut exister des contextes propices à ce type d'automates, pour lesquels un tel prétraitement est conseillé.

On a implanté l'optimisation donnée par le cas 1 et comparé son temps de calcul moyen avec l'implantation classique à l'aide de REGAL (Figure 7.1). L'optimisation semble être efficace en moyenne. Dans certains contextes l'optimisation n'aura cependant pas d'effet notable sur le temps de calcul. En effet, il existe des exécutions de l'algorithme de Moore pour lesquelles presque toutes les classes d'équivalences garderont un cardinal supérieur à 1 jusqu'aux dernières itérations de l'algorithme.

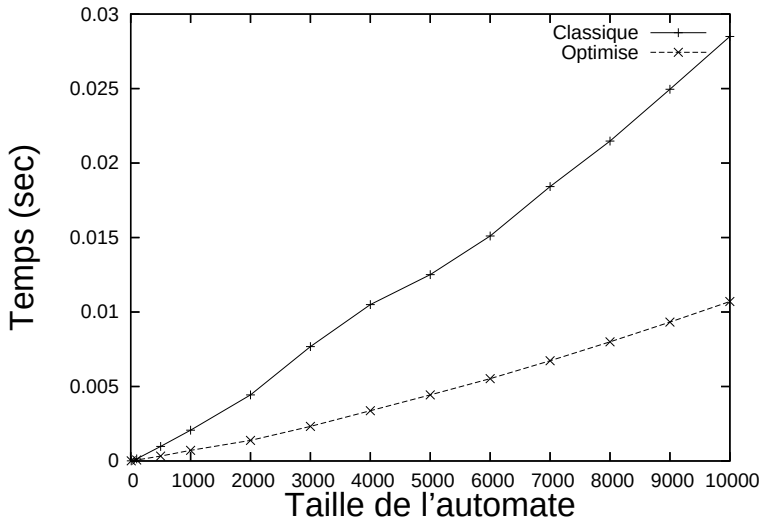


FIG. 7.1 – Pour chaque taille d'automate testée, 10.000 automates ont été engendrés aléatoirement à l'aide de REGAL, puis minimisés avec les deux versions de l'algorithme de Moore

2 Minimisation d'automates minimaux

On établit ici le résultat suivant : le nombre d'itérations effectuées par l'algorithme de Moore dépend uniquement du langage reconnu, et est donc le même pour tout automate déterministe et son automate minimal associé. Puis on établit une borne inférieure du nombre d'itérations effectuées par l'algorithme lorsqu'il est appliqué aux automates minimaux.

2.1 Une exécution dépendant uniquement du langage reconnu

Lemme 29. *Pour tout automate déterministe, le nombre d'itérations de l'algorithme de Moore est égal au nombre d'itérations effectuées pour l'automate minimal associé.*

Démonstration. Soient $\mathcal{A} = (A, Q, \cdot, q_o, F)$ un automate et $\mathcal{A}_{min} = (A, Q/\sim, *, [q_o], F')$ son automate minimal associé. Pour tout état $p \in Q$ de l'automate $\mathcal{A}$, on note $[p] \in Q/\sim$ son état associé dans l'automate minimal, tel que :

$$[p] = [q] \iff p \sim q$$

On rappelle que $\mathcal{M}oore(\mathcal{A})$ désigne le nombre d'itérations effectuées par l'algorithme en prenant l'automate $\mathcal{A}$ comme entrée. Si le langage reconnu est A^* ou $\emptyset$, alors d'après le lemme 28 :

$$\mathcal{M}oore(\mathcal{A}) = \mathcal{M}oore(\mathcal{A}_{min}) = 0.$$

Sinon, on note m le plus petit entier tel que $\sim_m = \sim$ pour l'automate $\mathcal{A}$, c'est-à-dire le plus petit entier pour lequel, si deux états ne sont pas équivalents, il existe un mot de longueur inférieure ou égale à cet entier qui permet de distinguer les deux états. D'après le lemme 28, $\mathcal{M}oore(\mathcal{A}) = m + 1$. On a

$$\begin{aligned}\mathcal{M}oore(\mathcal{A}) = m + 1 = \min\{i \in \mathbb{N} \mid \forall (p, q) \in Q^2 \text{ tels que } p \nsim q,\\ \exists u \in A^{\leq i}, [\![p \cdot u \in F]\!] \neq [\![q \cdot u \in F]\!]\} + 1\end{aligned}$$

De plus $p \nsim q$ si et seulement si $[p] \neq [q]$ et pour tout $p \in Q$ et tout mot $u \in A^*$, $[p \cdot u] = [[p] * u]$, donc

$$\begin{aligned}\mathcal{M}oore(\mathcal{A}) = \min\{i \in \mathbb{N} \mid \forall (p, q) \in Q^2 \text{ tels que } [p] \neq [q],\\ \exists u \in A^{\leq i}, [\![[p] * u \in F]\!] \neq [\![[q] * u \in F]\!]\} + 1\end{aligned}$$

La partie droite de l'égalité étant par définition égale à $\mathcal{M}oore(\mathcal{A}_{min})$, ceci conclut la preuve. □

2.2 Borne inférieure

Le lemme précédent nous amène à nous intéresser au comportement de l'algorithme appliqué aux automates minimaux. La complexité dans le pire cas est toujours égale à $\Theta(n^2)$ car l'automate de la Figure 6.3 page 76 est minimal. En revanche, puisque l'automate minimal garantit que tous les états seront isolés dans leur classe d'équivalence à la fin de l'algorithme, un nombre minimum d'itérations est requis pour pouvoir tous les distinguer.

Proposition 30. *L'algorithme de Moore, appliqué à un automate minimal, a une complexité de* $\Omega(n \log\log n)$ *lorsque l'alphabet est de taille* $k \geq 2$ *et de* $\Omega(n \log n)$ *lorsque l'alphabet ne contient qu'une seule lettre.*

Démonstration. L'algorithme s'arrête lorsque tous les états de l'automate minimal sont isolés dans un sous-ensemble de la partition, c'est-à-dire lorsque le nombre de sous-ensembles est égal au nombre d'états dans l'automate $\mathcal{A}$. Pour tout entier i et tout état p, on définit l'application $\phi_p^{(i)} : A^{\leq i} \to \{0,1\}$ par : $\phi_p^{(i)}(u) = [\![p \cdot u \in F]\!]$. Il existe k^j mots de longueur j. Le nombre de mots de longueur au plus i est donc :

$$\sum_{j=0}^{i} k^j = \begin{cases} \frac{k^{i+1}-1}{k-1} & k \geq 2 \\ i+1 & k = 1 \end{cases}$$

Il existe donc au plus $2^{\frac{k^{i+1}-1}{k-1}}$ (resp. 2^{i+1}) $\phi_p^{(i)}$ distincts, pour un alphabet de taille au moins 2 (resp. égal à un). Puisque $p \sim_i q$ si et seulement si $\phi_p^{(i)} = \phi_q^{(i)}$, il existe au plus $2^{\frac{k^{i+1}-1}{k-1}}$ (resp. 2^{i+1}) sous-ensembles distincts dans la partition à la i-ème itération de l'algorithme. L'algorithme s'arrêtant quand le nombre de sous ensemble est égal à n, on a :

$$\begin{cases} n \leq 2^{\frac{k^{\mathcal{M}oore(\mathcal{A})}-1}{k-1}} & k \geq 2 \\ n \leq 2^{\mathcal{M}oore(\mathcal{A})+1} & k = 1 \end{cases}$$

Ceci conclut la preuve puisque le coût d'une itération est $\Theta(kn)$. □

Chapitre 8

Complexité moyenne de l'algorithme de Moore : 1^{ere} partie

Comme nous l'avons vu dans le chapitre 6, l'algorithme de Moore calcule la relation d'équivalence de Myhill-Nerode sur les états d'un automate :

1. en partant d'une partition de l'ensemble des états de l'automate qui sépare les états terminaux des non terminaux.
2. en raffinant cette partition à l'aide d'informations liées à l'ensemble des états terminaux. En effet, on rappelle que pour tous états p et q, on a :

$$p \sim_i q \Longleftrightarrow \forall w \in A^{\leq i},\ [\![p \cdot w \in F]\!] = [\![q \cdot w \in F]\!]$$

Ainsi, dans ce chapitre, on étudie la complexité en moyenne de l'algorithme de Moore pour une structure de transitions fixée en prenant en compte uniquement les propriétés de l'ensemble des états terminaux.

On montre tout d'abord que pour une structure de transitions fixée et la distribution uniforme sur l'ensemble des états terminaux, le nombre moyen de raffinements de partitions effectués par l'algorithme de Moore est $\mathcal{O}(\log n)$. Cette borne étant satisfaite par toutes les structures de transitions déterministes, on en déduit que, quel que soit l'ensemble de structures de transitions et quelle que soit la distribution de probabilités que l'on fixe sur cet ensemble, si l'ensemble des états terminaux est choisi uniformément, alors la complexité en moyenne de l'algorithme de Moore est $\mathcal{O}(n \log n)$ (théorème 36). L'ensemble des automates déterministes accessibles complets est le plus intéressant à étudier puisqu'il est possible en un prétraitement linéaire de transformer tout automate déterministe en un automate accessible et complet. Le résultat énoncé précédemment implique que pour la distribution uniforme sur l'ensemble des automates déterministes accessibles et complets, la complexité en moyenne de l'algorithme de Moore est $\mathcal{O}(n \log n)$. Ce résultat a été publié dans [9].

Dans la section 5, on montre que le résultat est toujours valable si l'ensemble des états terminaux est choisi suivant une loi de Bernoulli.

Dans la section 6, on prouve que cette borne est optimale pour l'ensemble des automates unaires.

Dans les sections 7 et 8, on établit que le résultat est valable pour la distribution uniforme sur l'ensemble des automates possiblement incomplets et sur l'ensemble des automates émondés. En effet, la distribution sur l'ensemble des états terminaux est

légèrement différente de celles précédemment envisagées et la complexité en moyenne ne se déduit pas directement du théorème 36.

On clôture le chapitre en évoquant le cas où le nombre d'états terminaux est fixé. On montre que, pour la distribution uniforme sur l'ensemble des automates unaires possédant un seul état final, la complexité moyenne de l'algorithme de Moore est égal à $\Theta(n^2)$.

L'ensemble de ces résultats apparaît dans [8].

1 L'idée

Découpage

Commençons tout d'abord par quelques rappels. Le coût d'une itération étant borné par $\mathcal{O}(kn)$, l'étude de la complexité moyenne de l'algorithme de Moore peut se résumer à l'étude du nombre moyen d'itérations effectuées par l'algorithme, que l'on notera $\mathfrak{M}_n$. On rappelle que $\mathcal{M}oore(\mathcal{A})$ désigne le nombre d'itérations effectuées par l'algorithme de Moore en prenant l'automate $\mathcal{A}$ comme entrée. Soit $\mathbb{P}(\mathcal{A})$ la probabilité d'avoir un automate $\mathcal{A}$ parmi tous les automates de $\mathcal{A}_n$, on a :

$$\mathfrak{M}_n = \sum_{\mathcal{A}\in\mathcal{A}_n} \mathbb{P}(\mathcal{A}) \times \mathcal{M}oore(\mathcal{A}),$$

On rappelle que $\mathcal{A}_n^i$ désigne l'ensemble des automates déterministes à n états pour lesquels i est le plus petit entier tel que la partition $\sim_i$ est égale à l'équivalence de Myhill-Nerode. D'après le lemme 28, on a :

$$\forall i > 0,\ \mathcal{A}_n^i = \{\mathcal{A} \in \mathcal{A}_n \mid \mathcal{M}oore(\mathcal{A}) = i + 1\},$$

c'est-à-dire l'ensemble des automates à n états minimisés en exactement i itérations. D'après la définition l'algorithme, l'équivalence de Myhill-Nerode est atteinte après $n-2$ dans le pire des cas, pour tout automate $\mathcal{A}$. On a donc :

$$\mathfrak{M}_n = \sum_{i=1}^{n-2} \left(\sum_{\mathcal{A}\in\mathcal{A}_n^i} \mathbb{P}(\mathcal{A}) \times (i+1) \right)$$

Dans cette première analyse de la complexité moyenne de l'algorithme de Moore, on note $\gamma_n = \lceil 5 \log n \rceil$ et on exhibe différentes distributions de probabilité pour lesquelles l'ensemble des automates minimisés en plus de γ_n itérations apporte une contribution négligeable à la moyenne. Cela revient à décomposer le calcul comme suit :

$$\mathfrak{M}_n = \sum_{i=1}^{\gamma_n} \left(\sum_{\mathcal{A}\in\mathcal{A}_n^i} \mathbb{P}(\mathcal{A}) \times (i+1) \right) + \sum_{j=\gamma_n+1}^{n-2} \left(\sum_{\mathcal{A}\in\mathcal{A}_n^j} \mathbb{P}(\mathcal{A}) \times (j+1) \right)$$

La partie gauche de la somme étant égale à $\mathcal{O}(\log n)$, si l'on montre que :

$$\sum_{j=\gamma_n+1}^{n-2} \left(\sum_{\mathcal{A}\in\mathcal{A}_n^j} \mathbb{P}(\mathcal{A}) \times (j+1) \right) = \mathcal{O}(\log n),$$

alors on a bien $\mathfrak{M}_n = \mathcal{O}(\log n)$ et la complexité moyenne de l'algorithme est $\mathcal{O}(n \log n)$. Certaines distributions de probabilité pour lesquelles on a obtenu de résultat sont particulièrement intéressantes, comme la distribution uniforme sur l'ensemble des automates déterministes accessibles complets, sur l'ensemble des automates fortement connexes, ou encore sur l'ensemble des automates déterministes complets.

Plus précisément

On rappelle qu'un automate peut être vu comme un couple (τ, F), où τ est la structure de transitions de l'automate et F l'ensemble de ses états terminaux. On considère que la distribution sur les ensembles d'états terminaux est indépendante de la structure de transition, ce qui équivaut à dire :

$$\mathbb{P}(\mathcal{A}) = \mathbb{P}(\tau) \times \mathbb{P}(F)$$

On a donc :

$$\sum_{j=\gamma_n+1}^{n-2} \left(\sum_{\mathcal{A} \in \mathcal{A}_n^j} \mathbb{P}(\mathcal{A}) \times (j+1) \right) = \sum_{\tau \in \mathcal{T}_n} \left[\mathbb{P}(\tau) \times \sum_{j=\gamma_n+1}^{n-2} \left(\sum_{\substack{F \subset \{1,\dots,n\} \\ (\tau,F) \in \mathcal{A}_n^j}} \mathbb{P}(F) \times (j+1) \right) \right]$$

Dans cette section, on montre que pour toute distribution de probabilité sur les structures de transitions déterministes et complètes, et la distribution uniforme sur l'ensemble des états terminaux, cette somme est égale à $\mathcal{O}(1)$, ce qui implique directement que $\mathfrak{M}_n = \mathcal{O}(\log n)$.

Rappelons que si l'on prend la distribution uniforme sur l'ensemble des états terminaux, alors pour tout ensemble F, on a $\mathbb{P}(F) = \frac{1}{2^n}$.

On va montrer que pour toute structure de transitions τ fixée, on a :

$$\frac{1}{2^n} \times \sum_{j=\gamma_n+1}^{n-2} \left(\sum_{\substack{F \subset \{1,\dots,n\} \\ (\tau,F) \in \mathcal{A}_n^j}} (j+1) \right) = \mathcal{O}(1)$$

Pour une structure de transitions τ fixée et un entier ℓ tel que $1 \leq \ell \leq n-2$, on définit $\mathcal{F}_\tau(\ell)$, l'ensemble des ensembles d'états terminaux tel que l'automate est minimisé en plus de ℓ itérations. Dit autrement :

$$\mathcal{F}_\tau(\ell) = \{F \subset \{1, \cdots, n\} \mid (\tau, F) \in \bigcup_{i>\ell} \mathcal{A}_n^i\}$$

Pour tout automate minimisé en plus de ℓ itérations, l'algorithme de Moore impose qu'au moins deux états soient séparés lors de la ℓ-ème itération. Dans le cas contraire, la partition ne varie pas et l'algorithme s'arrête. On modifie la définition précédente en conséquence :

$$\mathcal{F}_\tau(\ell) = \{F \subset \{1, \cdots, n\} \mid \text{pour l'automate } (\tau, F),\ \exists p, q \in Q \\ p \sim_{\ell-1} q,\ p \nsim_\ell q\}$$

On a :

$$\frac{1}{2^n} \times \sum_{j=\gamma_n+1}^{n-2} \left(\sum_{\substack{F \subset \{1,\dots,n\} \\ (\tau,F) \in \mathcal{A}_n^j}} (j+1) \right) \leq \frac{n-1}{2^n} \times |\mathcal{F}_\tau(\gamma_n)|$$

On va montrer qu'il existe une majoration du cardinal de l'ensemble $\mathcal{F}_\tau(\ell)$, commune à toutes les structures de transitions τ. Pour cela, on commence par introduire un outil de modélisation : le *graphe de $\mathcal{F}$-dépendance*.

2 Le graphe de $\mathcal{F}$-dépendance

Définition

Soit une structure de transitions τ, un entier ℓ tel que $1 \leq \ell < n$ et deux états p et q tels que $p \neq q$. On souhaite caractériser l'ensemble des ensembles F d'états terminaux pour lesquels, dans l'automate (τ, F), on a :

$$p \sim_{\ell-1} q \text{ et } p \nsim_\ell q$$

Cette propriété implique qu'il existe un mot u de longueur ℓ tel que $p \cdot u = p'$ et $q \cdot u = q'$, avec $[\![p' \in F]\!] \neq [\![q' \in F]\!]$.
On définit les ensembles d'ensemble d'états terminaux $\mathcal{F}_\tau(p, q, p', q', \ell)$:

$$\begin{aligned}\mathcal{F}_\tau(p,q,p',q',\ell) = \{F \subset \{1,\cdots,n\} \mid & \text{ pour tout automate } (\tau,F), \\ & p \sim_{\ell-1} q, \\ & [\![p' \in F]\!] \neq [\![q' \in F]\!], \\ & \exists u \in A^\ell,\ p \cdot u = p' \text{ and } q \cdot u = q'\}\end{aligned}$$

On remarquera que, dans une structure de transitions τ fixée, il peut très bien n'y avoir aucun chemin étiqueté par un mot de longueur ℓ qui part de p et q et arrive en p' et q'. En fonction des paramètres fixés, l'ensemble pourra donc être vide. En revanche, s'il existe plusieurs mots de longueur ℓ étiquetant des chemins allant de p à p' et de q à q', on choisit par convention le mot u comme étant le plus petit dans l'ordre lexicographique.

Afin de faire le lien avec les définitions précédentes, précisons que l'on a :

$$\mathcal{F}_\tau(\ell) = \bigcup_{\substack{p,q,p',q' \in \{1,\dots,n\} \\ p \neq q \\ p' \neq q'}} \mathcal{F}_\tau(p,q,p',q',\ell)$$

A partir de l'ensemble $\mathcal{F}_\tau(p, q, p', q', \ell)$, on définit le graphe non orienté $\mathcal{G}_\tau(p, q, p', q', \ell)$, appelé le *graphe de $\mathcal{F}$-dépendance* :

- l'ensemble de ses sommets $\{1, \cdots, n\}$ correspond à l'ensemble des états de τ ;
- il existe une arête (s, t) entre deux sommets s et t si et seulement si pour tout $F \in \mathcal{F}_\tau(p, q, p', q', \ell)$, $[\![s \in F]\!] = [\![t \in F]\!]$.

Les graphes de $\mathcal{F}$-dépendance sont donc une représentation pratique de conditions nécessaires à un ensemble d'états terminaux pour être dans $\mathcal{F}_\tau(p, q, p', q', \ell)$.

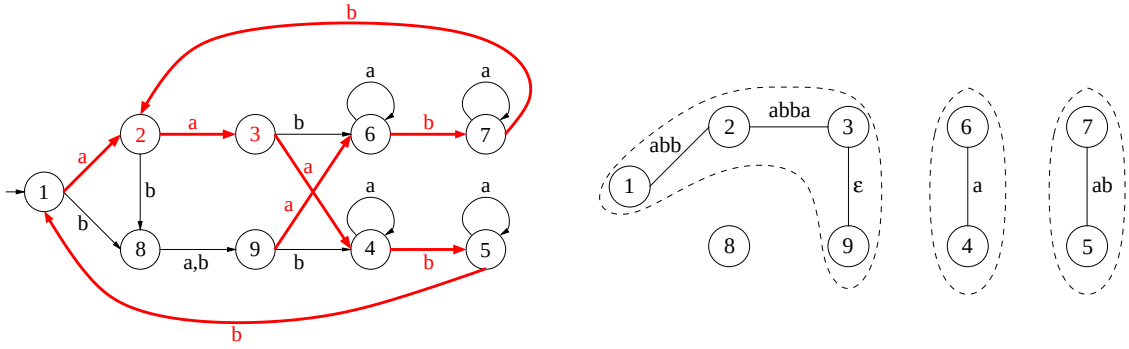


FIG. 8.1 – **(a)** est une structure de transitions et **(b)** un sous-graphe du **graphe de $\mathcal{F}$-dépendance** où $\ell = 5$, $p = 3$, $q = 9$, $p' = 3$ et $q' = 4$. On en déduit que $u = abbaa$. Les arêtes de **(b)** ont été étiquetées avec les préfixes de u dans un soucis de lisibilité : s'il existe une arête étiquetée par u' entre deux sommets s et t de **(b)**, alors $s = p.u'$ et $t = q.u'$ dans **(a)**. Comme tous les états d'une même composante connexe de **(b)** sont soit tous terminaux, soit tous non-terminaux, il y a au plus 2^4 ensembles d'états terminaux possibles, au lieu de 2^9.

Lemme 31. *Si un graphe de $\mathcal{F}$-dépendance contient un sous-graphe acyclique composé de i arêtes, le nombre d'ensemble d'états terminaux modélisés par ce graphe est inférieur ou égal à 2^{n-i}.*

Démonstration. La preuve découle directement de la définition des graphes de $\mathcal{F}$-dépendance : deux états dans une même composante connexe sont soit tous deux terminaux, soit tous deux non terminaux. Ainsi, si m est le nombre de composantes connexes dans le graphe, alors le nombre d'ensemble d'états terminaux modélisés par le graphe est borné par 2^m. Si le graphe de $\mathcal{F}$-dépendance contient un sous-graphe acyclique avec exactement i arêtes, alors il y a au plus $n - i$ composantes connexes dans le graphe. Ceci conclut la preuve. □

La Figure 8.1 contient un exemple d'un sous-graphe acyclique d'un graphe de $\mathcal{F}$-dépendance.

Caractérisation d'un sous-graphe acyclique ubiquitaire

On va à présent caractériser un sous-graphe acyclique, présent dans tout graphe $\mathcal{G}_\tau(p, q, p', q', \ell)$, afin d'obtenir une majoration commune sur le cardinal de tous les ensembles $\mathcal{F}_\tau(p, q, p', q', \ell)$.

On considérera uniquement les cas où $\mathcal{F}_\tau(p, q, p', q', \ell)$ est non vide.

Pour tout entier $\ell \in \{1, \ldots, n-1\}$ et tous états $p, q, p', q' \in \{1, \ldots, n\}$, tels que $p \neq q$, $p' \neq q'$, on a $u = u_1 \ldots u_\ell$, où $u_i \in A$, le plus petit mot selon l'ordre lexicographique de longueur ℓ tel que $p \cdot u = p'$ et $q \cdot u = q'$. Tout mot u de longueur ℓ tel que $p \cdot u = p'$ et $q \cdot u$ pourrait être utilisé, mais un choix non-ambigu garantit une description complète de la construction qui suit.

Pour tout entier $i \in \{0, \ldots, \ell - 1\}$, on note $\mathcal{G}_{\ell,i}$ le sous-graphe de $\mathcal{G}_\tau(p, q, p', q', \ell)$ dont les arêtes sont définies de la façon suivante : une arête (s, t) est dans $\mathcal{G}_{\ell,i}$ si et seulement si il existe un préfixe strict v de u de longueur inférieure ou égale à i tel

que $s = p \cdot v$ et $t = q \cdot v$. En d'autres termes, les arêtes de $\mathcal{G}_{\ell,i}$ sont exactement les arêtes $(p \cdot v, q \cdot v)$ entre les états $p \cdot v$ et $q \cdot v$ où v est un préfixe de u de longueur inférieure ou égale à i.

Lemme 32. *Pour tout sous-graphe $\mathcal{G}_{\ell,i}$ les propriétés suivantes sont satisfaites :*

1. *Pour tout $i \in \{0, \ldots, \ell-2\}$, $\mathcal{G}_{\ell,i}$ est un sous-graphe strict de $\mathcal{G}_{\ell,i+1}$.*
2. *Pour tout $i \in \{0, \ldots, \ell-1\}$, $\mathcal{G}_{\ell,i}$ contient $i+1$ arêtes.*
3. *Pour tout $i \in \{0, \ldots, \ell-1\}$, $\mathcal{G}_{\ell,i}$ ne contient pas de boucle.*
4. *Pour tout $i \in \{0, \ldots, \ell-1\}$, s'il existe dans $\mathcal{G}_{\ell,i}$ un chemin allant d'un nœud s à un nœud t, alors $s \sim_{\ell-1-i} t$ pour tout automate (τ, F) pour lequel $F \in \mathcal{F}_\ell(p, q, p', q')$.*

Démonstration.

1. Le graphe $\mathcal{G}_{\ell,i+1}$ est obtenu à partir de $\mathcal{G}_{\ell,i}$ en ajoutant une arête entre $p \cdot w$ et $q \cdot w$, où w est le préfixe de u de longueur $i+1$. Cette nouvelle arête n'appartient pas à $\mathcal{G}_{\ell,i}$. En effet, raisonnons par l'absurde, si c'est le cas, alors il existe un préfixe z de w tel que, soit $p \cdot z = p \cdot w$ et $q \cdot z = q \cdot w$, soit $p \cdot z = q \cdot w$ et $q \cdot z = p \cdot w$. La Figure 8.2 illustre ces deux cas.

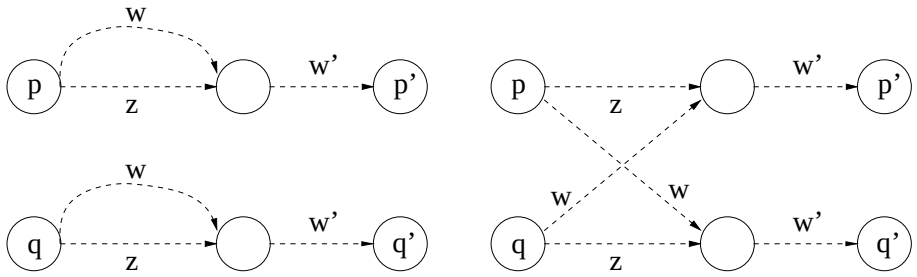


FIG. 8.2 – Tout sous-graphe $\mathcal{G}_{\ell,i+1}$ contient $i+1$ arêtes

   Dans les deux cas, si w' est le mot tel que $u = ww'$, alors soit $p \cdot zw' = p'$ et $q \cdot zw' = q'$, soit $p \cdot zw' = q'$ et $q \cdot zw' = p'$. Il existe donc un mot de longueur inférieur à ℓ, noté zw', tel que pour tout $F \in \mathcal{F}_\tau(p, q, p', q', \ell)$, $[\![p \cdot zw' \in F]\!] \neq [\![q \cdot zw' \in F]\!]$, ce qui est impossible puisque $p \sim_{\ell-1} q$ et que $\mathcal{F}_\tau(p, q, p', q', \ell)$ est non vide.
2. Il s'agit d'une conséquence de la propriété 1, obtenue par récurrence, en remarquant que $\mathcal{G}_{\ell,0}$ ne contient qu'une unique arête entre p et q.
3. Pour tout automate $(\mathcal{T}, F)$, où F est non vide et tel que $F \in \mathcal{F}_\tau(p, q, p', q', \ell)$, p n'est pas équivalent à q. Donc pour tout préfixe v de u, $p \cdot v \neq q \cdot v$.
4. On prouve cette propriété par induction sur $i \in \{0, \ldots, \ell-1\}$. Pour $i = 0$, $\mathcal{G}_{\ell,0}$ contient seulement une arête, celle entre p et q, et on a bien $p \sim_{\ell-1} q$. On admet à présent que la propriété est vraie pour $i \in \{0, \ldots, \ell-2\}$. Soit x et y deux sommets tels qu'il existe un chemin simple de x à y dans $\mathcal{G}_{\ell,i+1}$. Si ce chemin était déjà dans $\mathcal{G}_{\ell,i}$, c'est-à-dire s'il ne contient pas la nouvelle arête, alors d'après notre hypothèse de récurrence $x \sim_{\ell-1-i} y$, et donc par hypothèse $x \sim_{\ell-2-i} y$. Si tel n'était pas le cas, le chemin utiliserait l'arête entre $p \cdot v$ et

$q \cdot v$, où v est le préfixe de u de longueur $i+1$. Admettons par symétrie que le chemin allant de x vers y atteigne $p \cdot v$ en premier. La partie du chemin entre x et $p \cdot v$ est bien dans $\mathcal{G}_{\ell,i}$, donc $x \sim_{\ell-1-i} p \cdot v$ d'après l'hypothèse. De la même façon, on obtient $y \sim_{\ell-1-i} q \cdot v$. Puisque $p \sim_{\ell-1} q$, on a $p \cdot v \sim_{\ell-2-i} q \cdot v$ et $x \sim_{\ell-2-i} y$ par transitivité, ce qui conclut la preuve par récurrence.

□

Lemme 33. *Le sous-graphe $\mathcal{G}_{\ell,\ell-1}$ est un sous-graphe acyclique de $\mathcal{G}_\tau(p,q,p',q',\ell)$ contenant exactement ℓ arêtes.*

Démonstration. D'après la Propriété 2. du lemme 32, $\mathcal{G}_{\ell,\ell-1}$ contient exactement ℓ arêtes. On prouve par l'absurde que $\mathcal{G}_{\ell,\ell-1}$ est acyclique. Soit $j \geq 1$ le plus petit entier tel que $\mathcal{G}_{\ell,j}$ contient un cycle. D'après la Propriété 1. du lemme 32, $\mathcal{G}_{\ell,j}$ est obtenu à partir de $\mathcal{G}_{\ell,j-1}$ en ajoutant une arête $(p \cdot w, q \cdot w)$, où w est le préfixe de u de longueur j. Comme $\mathcal{G}_{\ell,j-1}$ est acyclique, cette arête fait apparaître un cycle dans $\mathcal{G}_{\ell,j}$. Par suite, dans $\mathcal{G}_{\ell,j-1}$, il existe déjà un chemin entre $p \cdot w$ et $q \cdot w$. D'après la Propriété 4., on a donc $p \cdot w \sim_{\ell-j} q \cdot w$ dans tout automate $(\mathcal{T}, F)$, tel que $F \in \mathcal{F}_\tau(p,q,p',q',\ell)$. La Figure 8.3 illustre la preuve.

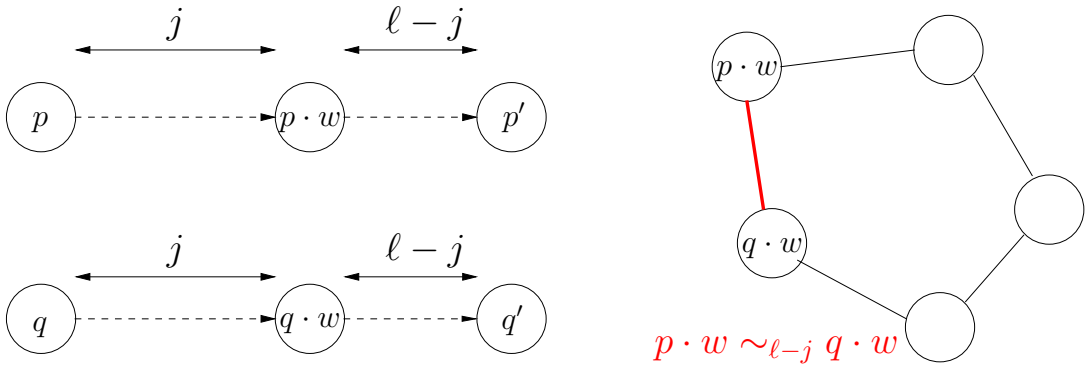


FIG. 8.3 – Dans cet exemple, le graphe $\mathcal{G}_{\ell,j}$ contient un cycle : la partie de gauche est une partie de la structure de transitions et celle de droite une partie du sous-graphe $\mathcal{G}_{\ell,j}$. D'après le graphe, on a $p' \sim_0 q'$, ce qui est une contradiction avec la définition.

Soit w' le mot tel que $u = ww'$. La longueur du mot w' est $\ell - j$, donc $p \cdot u$ et $q \cdot u$ sont soit tous deux terminaux soit tous deux non terminaux, ce qui est impossible puisque $F \in \mathcal{F}_\tau(p,q,p',q',\ell)$. Le graphe $\mathcal{G}_{\ell,\ell-1}$ est donc un sous-graphe acyclique de $\mathcal{G}_\tau(p,q,p',q',\ell)$. □

3 Majoration de $\mathcal{F}_\tau(\ell)$

Lemme 34. *Pour toute structure de transitions τ fixée, de taille $n \geq 1$, tout entier ℓ tel que $1 \leq \ell < n$, tous états p,q,p',q' tels que $p \neq q$ et $p' \neq q'$, la propriété suivante est satisfaite :*

$$|\mathcal{F}_\tau(p,q,p',q',\ell)| \leq 2^{n-\ell}.$$

Démonstration. Si $\mathcal{F}_\tau(p,q,p',q',\ell)$ est vide, le résultat est vrai. Sinon, on sait d'après le lemme 33, qu'il existe un sous-graphe acyclique dans $\mathcal{G}_\tau(p,q,p',q',\ell)$ qui contient exactement ℓ arêtes. On conclut en utilisant le lemme 31. □

Corollaire 35. *Pour toute structure de transitions τ fixée, de taille $n \geq 1$, tout entier ℓ tel que $1 \leq \ell < n$, la propriété suivante est satisfaite :*

$$|\mathcal{F}_\tau(\ell)| \leq n^4 \times 2^{n-\ell}.$$

Démonstration. D'après la définition donnée dans la section 2 page 94, on a :

$$|\mathcal{F}_\tau(\ell)| \leq \sum_{\substack{p,q,p',q' \in \{1,\ldots,n\} \\ p \neq q \\ p' \neq q'}} |\mathcal{F}_\ell(p,q,p',q')|$$

On conclut la preuve en utilisant le lemme 34 et en majorant le nombre de quadruplets p, q, p', q' par n^4. □

On utilise à présent ce résultat pour conclure sur la complexité moyenne de l'algorithme de Moore, pour différentes distributions.

4 Complexité en moyenne

Théorème 36. *Pour tout entier fixé $k \geq 1$, pour toute distribution sur l'ensemble des automates déterministes complets de taille n sur un alphabet à k lettres, telle que l'ensemble des ensembles des états terminaux est tiré uniformément parmi $\mathcal{P}(Q)$, la complexité de l'algorithme de Moore est $\mathcal{O}(n \log n)$.*

Démonstration. On rappelle que si pour la distribution uniforme sur les ensembles d'états terminaux, on a :

$$\frac{n-1}{2^n} \times |\mathcal{F}_\tau(\gamma_n)| = \mathcal{O}(1),$$

avec $\gamma_n = \lceil 5 \log_2 n \rceil$, alors $\mathfrak{M}_n = \mathcal{O}(\log n)$ pour toute distribution sur l'ensemble des structures de transitions déterministes et complètes à n états. En utilisant le Corollaire 35, on obtient la majoration suivante :

$$\frac{n-1}{2^n} \times |\mathcal{F}_\tau(5 \log_2 n)| \leq \frac{n-1}{2^n} \times n^4 \times 2^{n - 5 \log_2 n}$$

Ceci conclut la preuve car $2^{n-5\log_2 n} = \frac{2^n}{n^5}$. □

Théorème 37. *Pour tout entier fixé $k \geq 1$ et pour la distribution uniforme sur l'ensemble des automates déterministes accessibles complets de taille n sur un alphabet à k lettres, la complexité de l'algorithme de Moore est $\mathcal{O}(n \log n)$.*

Théorème 38. *Pour tout entier fixé $k \geq 1$ et pour la distribution uniforme sur l'ensemble des automates déterministes complets fortement connexes de taille n sur un alphabet à k lettres, la complexité de l'algorithme de Moore est $\mathcal{O}(n \log n)$.*

Démonstration. Les distributions sur des structures de transitions des théorèmes 37 et 38. sont incluses dans le théorème 36. □

5 Distribution de Bernoulli

Une des extensions naturelles du théorème 36 est d'utiliser une distribution de Bernoulli pour la probabilité qu'a chaque état d'être final, ce qui donne une distribution binomiale sur le nombre d'états terminaux.

Théorème 39. *Soit τ une structure de transitions à n états sur un alphabet à k lettres. Pour la distribution sur les ensembles d'états terminaux où chaque état a une probabilité $x \in]0,1[$ d'être final, la complexité moyenne de l'algorithme de Moore est $\mathcal{O}(n \log n)$.*

Démonstration. Soit x un nombre réel fixé tel que $0 < x < 1$. Soit τ une structure de transitions à n états. Considérons la distribution sur les ensembles d'états terminaux telle que chaque état a la probabilité x d'être final. La probabilité d'un sous-ensemble F de $\{1, \ldots, n\}$ est $\mathbb{P}(F) = x^{|F|}(1-x)^{n-|F|}$. Pour les paramètres fixés p, q, p', q' et ℓ, on note $P_\tau(p, q, p', q', \ell)$ la probabilité qu'a un ensemble d'être dans $\mathcal{F}_\ell(p, q, p', q')$.

Puisque $\mathcal{G}_\tau(p, q, p', q', \ell)$ ne fait que modéliser un sous-ensemble de contraintes sur les ensembles d'états terminaux de $\mathcal{F}_\tau(p, q, p', q', \ell)$, $P_\tau(p, q, p', q', \ell)$ est inférieur à la probabilité qu'a un ensemble d'états terminaux de vérifier les contraintes données par ce graphe.

Soit m le nombre de composantes connexes dans $\mathcal{G}_{\ell,\ell-1}$ contenant au moins deux sommets, et $c_1, ..., c_m$ les tailles de ces composantes. Puisque $\mathcal{G}_{\ell,\ell-1}$ est acyclique et contient exactement ℓ arêtes, l'égalité suivante est vérifiée :

$$\sum_{i=1}^{m} c_i = m + \ell$$

La probabilité qu'un ensemble aléatoire vérifie les conditions imposées par la i-ème composante connexe est égale à $x^{c_i} + (1-x)^{c_i}$ (soit tous les états sont terminaux, soit aucun ne l'est). Soit r le nombre réel défini par $r = \max\{x, 1-x\} \in [1/2, 1[$. Pour tout $n \geq 1$, $x^n + (1-x)^n \leq r^{n-1}(x + 1 - x) = r^{n-1}$. Donc

$$P_\tau(p, p', q, q', \ell) \leq \prod_{i=1}^{m} r^{c_i - 1} = r^{\sum_{i=1}^{m}(c_i - 1)} = r^\ell$$

Pour une structure de transitions fixée τ, la probabilité que l'ensemble F appartienne à l'ensemble $\mathcal{F}_\tau(\ell)$ est donnée par :

$$\sum_{\substack{p,q,p',q' \in \{1,\ldots,n\} \\ p \neq q,\ p' \neq q'}} P_\tau(p, p', q, q', \ell) \leq n^4 \left(\frac{1}{r}\right)^{-\ell}$$

On conclut la preuve en prenant $\ell = \lceil 5 \log_{1/r} n \rceil$. Remarquons qu'en prenant $x = \frac{1}{2}$, on a $\ell = \gamma_n = \lceil 5 \log_2 n \rceil$. □

6 Optimalité de la borne pour les automates unaires

On prouve dans cette section que la borne $\mathcal{O}(n \log n)$ donnée dans le théorème 37 est optimale pour la distribution uniforme sur les automates lorsque l'alphabet est unaire, *i.e.* lorsqu'il ne contient qu'une seule lettre.

On utilise pour cela le résultat suivant, énoncé par Nicaud [59] :

Proposition 40 ([59]). *Pour la distribution uniforme sur les automates unaires à n états, la probabilité qu'un automate déterministe et accessible soit minimal est asymptotiquement égale à $\frac{1}{2}$.*

Proposition 41. *Pour la distribution uniforme sur les automates unaires à n états, la complexité moyenne de l'algorithme de Moore est $\Theta(n \log n)$.*

Démonstration. On sait d'après le théorème 37 que pour la distribution uniforme sur les automates unaires à n états la complexité moyenne l'algorithme de Moore est $\mathcal{O}(n \log n)$. Il reste donc à montrer que cette borne est atteinte.

D'après la proposition 30, page 90, il existe une constante positive $C > 0$ pour laquelle, la complexité de l'algorithme de Moore appliqué à un automate unaire minimal à n états est au moins $Cn \log n$. Soit m_n le nombre d'automates unaires minimaux à n états et a_n le nombre d'automates unaires déterministes accessibles complets à n états. En prenant en compte uniquement la contribution apportée à la complexité moyenne par les automates minimaux, que l'on calcule en utilisant la proposition 40, on obtient la borne inférieure suivante :

$$\frac{m_n}{a_n} Cn \log n \sim \frac{1}{2} Cn \log n,$$

la complexité moyenne est donc $\Omega(n \log n)$, ce qui conclut la preuve. □

7 Automates possiblement incomplets

Avant d'appliquer l'algorithme de Moore à un automate incomplet, on effectue une phase de prétraitement : la phase de complétion (usuelle) d'automate, au cours de laquelle on ajoute un état puits, qui devient l'état d'arrivée de toutes les transitions jusqu'alors indéfinies. Du point de vue de l'analyse en moyenne, cela va légèrement modifier la distribution de probabilité, mais pas suffisamment pour changer l'ordre de grandeur de la complexité.

Proposition 42. *Pour tout entier fixé $k \geq 2$ et pour la distribution uniforme sur les automates déterministes accessibles possiblement incomplets à n états sur un alphabet de taille k, la complexité moyenne de l'algorithme de Moore est $\mathcal{O}(n \log n)$.*

Démonstration. Soit τ une structure de transitions possiblement incomplète, et τ' la complétion de τ (si τ était déjà complet, alors $\tau = \tau'$). D'après le lemme 34, et puisqu'il y a soit n, soit $n+1$ états dans τ', il y a au plus $2^{n+1-\ell}$ ensembles dans $\mathcal{F}_\tau(p, p', q, q', \ell)$, pour tout $\ell \geq 1$ et tous états p, p', q, q' tels que $p \neq p'$ et $q \neq q'$. On remarque que cette majoration est plus grossière que dans le cas complet, puisque l'état puits ne peut être final. Néanmoins, la majoration en $2^{n+1-\ell}$ est suffisante pour déduire que la contribution des ensembles d'états terminaux pour lesquels l'algorithme de Moore effectue au moins $\lceil 5 \log n \rceil$ raffinements de partitions est négligeable, en utilisant le même raisonnement que dans le cas complet. □

8 Automates émondés

On peut également considérer la distribution uniforme sur l'ensemble des automates émondés à n états. Cela présente un intérêt particulier puisqu'il est possible d'émonder un automate en temps linéaire, avant de le minimiser. La complexité en moyenne de la minimisation reste la même. La raison principale est qu'il existe une proportion suffisante d'automates co-accessible parmi les automates accessibles, d'après le résultat suivant du à Korshunov [53] :

Théorème 43 ([53]). *Il existe une constante réelle $0 < c < 1$, dépendant de la taille de l'alphabet, telle que pour la distribution uniforme sur les automates déterministes, accessibles et complets à n états, la probabilité qu'un tel automate soit fortement connexe tend vers c lorsque n tends vers l'infini.*

Proposition 44. *Pour la distribution uniforme sur l'ensemble des automates déterministes, complets et émondés à n états sur un alphabet à k lettres, la complexité moyenne de l'algorithme de Moore est en $\mathcal{O}(n \log n)$.*

Démonstration. Soit $\mathcal{C}_n$ l'ensemble des automates émondés à n états. Soient $\mathcal{C}_n^{\leq \ell}$ et $\mathcal{C}_n^{>\ell}$ des sous-ensembles de $\mathcal{C}_n$ constitués des automates $\mathcal{A}$ pour lesquels on a respectivement $\mathcal{M}oore(\mathcal{A}) \leq \ell$ et $\mathcal{M}oore(\mathcal{A}) > \ell$.

Soit $\ell = \lceil 5 \log n \rceil$, et soit t_n le nombre de structures de transitions déterministes, accessibles et complètes.

On sait d'après les résultats précédents que le nombre d'automates accessibles $\mathcal{A}$ à n états pour lesquels on a $\mathcal{M}oore(\mathcal{A}) > \ell$ est $\mathcal{O}(\frac{t_n}{n})$ lorsque $\ell = \lceil 5 \log n \rceil$. Il existe donc une constante positive C telle que $|\mathcal{C}_n^{>\ell}| \leq C \frac{t_n}{n}$.

Par suite, le nombre moyen d'itérations de l'algorithme de Moore est :

$$\begin{aligned}\frac{1}{|\mathcal{C}_n|} \sum_{\mathcal{A} \in \mathcal{C}_n} \mathcal{M}oore(\mathcal{A}) &= \frac{1}{|\mathcal{C}_n|} \sum_{\mathcal{A} \in \mathcal{C}_n^{\leq \ell}} \mathcal{M}oore(\mathcal{A}) + \frac{1}{|\mathcal{C}_n|} \sum_{\mathcal{A} \in \mathcal{C}_n^{>\ell}} \mathcal{M}oore(\mathcal{A}) \\ &\leq \frac{|\mathcal{C}_n^{\leq \ell}|}{|\mathcal{C}_n|} \ell + n \frac{|\mathcal{C}_n^{>\ell}|}{|\mathcal{C}_n|} \leq \ell + \frac{n}{|\mathcal{C}_n|} \mathcal{O}\left(\frac{1}{n} t_n\right)\end{aligned}$$

D'après le théorème 43, $t_n/|\mathcal{C}_n| = \mathcal{O}(1)$, et par conséquence on a :

$$\frac{1}{|\mathcal{C}_n|} \sum_{\mathcal{A} \in \mathcal{C}_n} \mathcal{M}oore(\mathcal{A}) \leq \ell + \mathcal{O}(1) = \mathcal{O}(\log n),$$

ce qui conclut la preuve. □

9 Nombre d'états terminaux fixés

Dans tous les cas étudiés précédemment, les distributions de probabilités employées produisaient des automates possédant, avec une forte probabilité, un grand nombre d'états terminaux. Il est cependant naturel que certaines applications considèrent une distribution où les automates ne contiennent que peu de ces états terminaux. Dans ce cas, la méthode du graphe de $\mathcal{F}$-dépendance est inutile et les résultats énoncés jusqu'alors semblent ne pas fonctionner.

Proposition 45. *Pour la distribution uniforme sur les automates unaires possédant un seul état final, la complexité moyenne de l'algorithme de Moore est* $\Theta(n^2)$.

Démonstration. Dans [59], il est dit que la structure de transitions d'un automate unaire $\mathcal{U}$ sur un alphabet $\{a\}$ est caractérisée par son nombre d'états n et l'état d'arrivée m de la transition $n \cdot a$. Si $m = 1$, c'est-à-dire si la transition sortant du dernier état de la structure de transitions arrive dans le premier état, l'automate est un cycle. Autrement, l'automate unaire contient un cycle et une queue, où le cycle est l'ensemble des états $B = \{m, .., n\}$ et la queue est l'ensemble $Q \setminus B$.

Soit $\mathcal{U}$ un automate unaire à n états et un unique état final f. Pour tout état p, on note i_p le plus grand entier tel que pour tout $j \in \mathbb{N}$ avec $j < i_p$, l'état $p \cdot a^j$ n'est pas final. Soit q l'état pour lequel i_q est maximal. Si $i_q \geq 1$ alors, d'après la forme de l'automate, on a $i_{q \cdot a} = i_q - 1$. En particulier, on a $\mathcal{M}oore(\mathcal{U}) = i_q + 1$, car q et $q \cdot a$ sont $(i_q - 1)$-équivalent mais ne sont pas i_q-équivalent. Ceci est également vrai si $i_q = 0$, lorsque l'état initial est aussi l'état final et que $\mathcal{U}$ n'est pas un cycle. On a alors $\mathcal{M}oore(\mathcal{U}) = 1$.

Les états pour lesquels i_q est maximal sont soit l'état initial, soit l'état $f \cdot a$ (ou les deux). Si $f \notin B$, c'est-à-dire si $f < m$, alors l'état initial maximise i_q et $\mathcal{M}oore(\mathcal{U}) = i_1 + 1 = (f-1)+1 = f$. Si f est dans le cycle, on a $i_1 = f - 1$ et $i_{f \cdot a} = n - l$, puisque la longueur du cycle est $n - l + 1$. Alors $\mathcal{M}oore(\mathcal{U}) = \max\{f, n - l + 1\}$ lorsque $f \in L$.

Le nombre d'automates unaires possédant un unique état final est n^2 car tous les choix possibles de $l \in \{1, .., n\}$ et $f \in \{1, .., n\}$ correspondent à des états distincts. Le nombre moyen d'itérations est donc égal à :

$$\frac{1}{n^2} \sum_{l=1}^{n} \left(\sum_{f=1}^{l-1} f + \sum_{f=l}^{n-l} (n - l + 1) + \sum_{f=n-l+1}^{n} f \right) = \frac{1}{6n^2}(n^3 + 3n^2 - n) = \Theta(n).$$

Ceci conclut la preuve. □

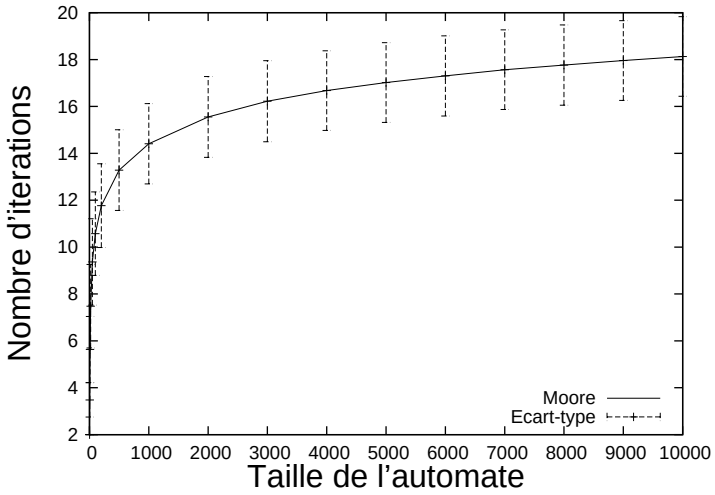


FIG. 8.4 – Étude expérimentale du nombre moyen d'itérations effectuées par l'algorithme de Moore, pour la distribution uniforme sur les automates possédant un unique état final. Pour chaque taille, les valeurs ont été calculées à partir de 20 000 automates aléatoires sur un alphabet à 2 lettres.

Néanmoins, lorsque l'alphabet contient au moins 2 lettres, l'algorithme de Moore appliqué aux automates possédant un unique état final semble s'exécuter en moins

de $\Theta(n^2)$ opérations, comme on peut le voir sur la Figure 8.4. L'analyse de la complexité en moyenne, dans ce cas, ne peut se faire par un traitement indépendant des structures de transitions et des ensembles d'états terminaux. Il en va de même pour obtenir une meilleure borne dans le cas général. Dans la deuxième partie de l'analyse de la complexité moyenne de l'algorithme, on utilisera également des propriétés sur les structures de transitions.

Chapitre 9

Complexité moyenne de l'algorithme de Moore : 2^{eme} partie

Dans ce chapitre, on étudie la complexité en moyenne de l'algorithme de Moore en tenant compte à la fois de l'ensemble des états terminaux et de l'ensemble des structures de transitions. Il est donc nécessaire de fixer un ensemble de structures de transitions sur lequel l'étude sera effectuée. L'énumération des structures de transitions de taille n pour lesquels on fixe l'état d'arrivée d'une transition donnée étant un problème ouvert, on a choisi d'étudier l'ensemble des automates déterministes et complets.

Le principal résultat est le suivant : pour la distribution uniforme sur l'ensemble des automates déterministes et complets, la complexité moyenne de l'algorithme de Moore est $\mathcal{O}(n \log\log n)$ (théorème 49). Ce résultat a été publié dans [30].

Dans la section 6, on montre que le résultat est valable pour une distribution de Bernoulli sur l'ensemble des états terminaux.

Dans la section 7, on conjecture que pour la distribution uniforme sur l'ensemble des automates déterministes accessibles et complets, la complexité en moyenne de l'algorithme est égale à $\Theta(n \log\log n)$.

Dans la section 8, on utilise les arguments de la preuve du théorème 49 pour montrer que la complexité générique de l'algorithme de Moore est $\mathcal{O}(n \log\log n)$.

1 L'idée

Distribution

Dans cette deuxième partie, on va devoir énumérer des ensembles de structures de transitions possédant un ensemble de propriétés. On se fixe donc la distribution la plus simple dans ce contexte, à savoir la distribution uniforme sur les automates déterministes complets. On note $\mathcal{T}_n$ l'ensemble des structures de transitions déterministes complètes à n états sur un alphabet à k lettres et $\mathcal{A}_n$ l'ensemble des automates associés. On a $|\mathcal{A}_n| = |\mathcal{T}_n| \times 2^n$, puisque pour toute structure de transitions, on peut choisir 2^n ensembles d'états terminaux. La condition d'accessibilité n'est pas garantie, et certains automates sont isomorphes dans cet ensemble. L'ensemble des états d'arrivée possibles pour une transition ne dépend pas des autres transitions de la structure de transitions. L'automate étant déterministe et complet, il y a donc kn transitions pouvant chacune aller dans n états. On a donc $|\mathcal{T}_n| = n^{kn}$.

Découpage

On rappelle la définition du nombre moyen d'itérations effectuées par l'algorithme de Moore, pour la distribution uniforme sur l'ensemble des automates déterministes complets :

$$\mathfrak{M}_n = \frac{1}{|\mathcal{A}_n|} \sum_{i=1}^{n-2} \left[|\mathcal{A}_n^i| \times (i+1) \right],$$

où $\mathcal{A}_n^i$ représente l'ensemble des automates déterministes à n états pour lesquels i est le plus petit entier tel que la partition $\sim_i$ est égale à l'équivalence de Myhill-Nerode.

On veut à présent montrer que $\mathfrak{M}_n = \mathcal{O}(\log\log n)$. On rappelle que $\gamma_n = \lceil 5\log_2 n \rceil$ et on pose $\lambda_n = \lceil \log_k \log_2 n^3 + 2 \rceil$:

$$\mathfrak{M}_n \leq \underbrace{\frac{\lambda_n+1}{|\mathcal{A}_n|} \sum_{i \leq \lambda_n} |\mathcal{A}_n^i|}_{\textbf{(S1)}} + \underbrace{\frac{\gamma_n+1}{|\mathcal{A}_n|} \sum_{i=\lambda_n+1}^{\gamma_n} |\mathcal{A}_n^i|}_{\textbf{(S2)}} + \underbrace{\frac{n-1}{|\mathcal{A}_n|} \sum_{i=\gamma_n+1}^{n-2} |\mathcal{A}_n^i|}_{\textbf{(S3)}}$$

(S1) est égal à $\mathcal{O}(\log\log n)$ et on a montré dans la section précédente que **(S3)** est égal à $\mathcal{O}(1)$. Il nous reste donc à montrer que :

$$\textbf{(S2)} = \frac{\gamma_n+1}{|\mathcal{A}_n|} \sum_{i=\lambda_n+1}^{\gamma_n} |\mathcal{A}_n^i| = \mathcal{O}(\log\log n) \tag{9.1}$$

Plus précisément

Pour tout $\ell > 0$, on définit l'ensemble $\mathcal{A}_n(p,q,\ell)$ des automates à n états, où p et q sont deux états séparés lors de la ℓ-ème itération :

$$\mathcal{A}_n(p,q,\ell) = \{(\tau,F) \in \mathcal{A}_n \mid \tau \subset \mathcal{T}_n,\ F \subseteq \{1,..,n\},\ p \sim_{\ell-1} q,\ p \nsim_\ell q\}$$

Remarque 46. *Notons que si dans un automate (τ,F), pour toute lettre a de l'alphabet, on a $p\cdot a = q\cdot a$ alors soit $p \nsim_0 q$, soit $p \sim q$. Donc $(\tau,F) \notin \mathcal{A}_n(p,q,\ell)$ pour $\ell > 0$. En conséquence, dans le reste de la preuve, pour toute structure de transitions où p et q ont été fixés, on suppose qu'il existe au moins une lettre a telle que $p\cdot a \neq q\cdot a$.*

On fait une fois de plus le lien avec les définitions précédentes (on se reportera à la page 92) :

$$\bigcup_{i>\lambda_n} \mathcal{A}_n^i = \bigcup_{p,q\in\{1,..,n\}} \mathcal{A}_n(p,q,\lambda_n+1)$$

Dans la première partie, on a vu que quelque soit la structure de transitions τ que l'on fixe, tout graphe de $\mathcal{F}$-dépendance modélisant les ensembles d'états terminaux tels que $(\tau,F) \in \mathcal{A}_n(p,q,\ell)$ possède un sous-graphe acyclique avec exactement ℓ arêtes.

Dans cette deuxième partie, on va caractériser un ensemble de structures de transitions pour lesquelles on peut garantir que le graphe de $\mathcal{F}$-dépendance associé contiendra un sous-graphe acyclique avec $\mathcal{O}(2^{\lambda_n})$ arêtes. Pour des raisons combinatoires, on commence par modifier légèrement la définition du graphe (Section 2). On introduit ensuite d'autres outils de modélisation (Sections 3 et 4) qui vont permettre de montrer que l'ensemble des automates, dont la structure de transitions ne garantit pas une telle propriété, est négligeable.

2 Modification du graphe de $\mathcal{F}$-dépendance

A l'instar de l'ensemble $\mathcal{F}_\tau(p, q, p', q', \ell)$ défini dans la section précédente, on définit ici l'ensemble $\mathcal{F}_\tau(p, q, u)$ comme suit :

$$\mathcal{F}_\tau(p,q,u) = \{F \subset \{1,\dots,n\} \mid \text{pour tout automate } (\mathcal{T}, F), \\ p \sim_{|u|-1} q, \\ [\![p \cdot u \in F]\!] \neq [\![q \cdot u \in F]\!]\}$$

Cet ensemble est très proche de celui défini dans la section précédente : on fixe le mot u au lieu des trois paramètres ℓ, p' et q'. En effet, on a :

$$\bigcup_{\substack{p',q' \in \{1,..,n\} \\ p' \neq q'}} \mathcal{F}_\tau(p,q,p',q',\ell) = \bigcup_{u \in A^\ell} \mathcal{F}_\tau(p,q,u)$$

Les deux définitions sont cependant utiles d'un point de vue combinatoire. En effet, puisque l'on souhaite ici que le mot u soit de longueur $\mathcal{O}(\log\log n)$, le nombre de mots possibles sera $\mathcal{O}(\log n)$, ce qui est négligeable par rapport aux n^2 choix possibles pour les états p' et q'. À l'inverse, dans la section précédente, puisque le mot u était de longueur $\lceil 5\log_2 n \rceil$, on aurait eu $\mathcal{O}(n^5)$ choix possibles pour $k = 2$ si on utilisait ce découpage.

A partir de l'ensemble $\mathcal{F}_\tau(p, q, u)$, on définit le graphe de $\mathcal{F}$-dépendance $\mathcal{G}_\tau(p, q, u)$:

- l'ensemble de ses sommets $\{1, \dots, n\}$ correspond à l'ensemble des états de τ ;
- il existe une arête (s, t) entre deux sommets si et seulement si pour tout $F \in \mathcal{F}_\tau(p, q, u)$, on a $[\![s \in F]\!] = [\![t \in F]\!]$ et $s = p \cdot v$, $t = q \cdot v$ où v est un mot sur A tel que $|v| < |u|$.

Le lemme 31 s'applique également à ce graphe.

3 L'arbre de dépendance

On définit ici l'*arbre de dépendance*, qui permet de modéliser un ensemble de structures de transitions. Dans un souci de clarté, on commencera par expliquer comment un arbre, noté $\mathcal{R}(p)$, peut être obtenu à partir d'une structure de transitions τ et d'un d'état p, puis comment cet arbre peut nous permettre d'estimer le cardinal d'un ensemble de structures de transitions. On rappelle que l'ordre *militaire* (ou *lexicographique gradué*) sur les mots, noté $<_{mil}$, est défini comme suit : pour tous mots $w, w' \in A^*$, on note $w <_{mil} w'$ si et seulement si, soit le mot w est strictement plus court que le mot w', soit ils sont de même longueur et w est plus petit que w' dans l'ordre lexicographique. Dit autrement, on a :

$$\forall w, w' \in A^*,\ w <_{mil} w' \iff \begin{cases} |w| < |w'| \\ \text{ou} \\ |w| = |w'| \text{ et } w <_{lex} w' \end{cases}$$

Soit une structure de transitions fixée, à n états, définie sur un alphabet à k lettres, et un état p, lui aussi fixé. On définit la fonction *estnoeud* de A^* dans $\{0, 1\}$

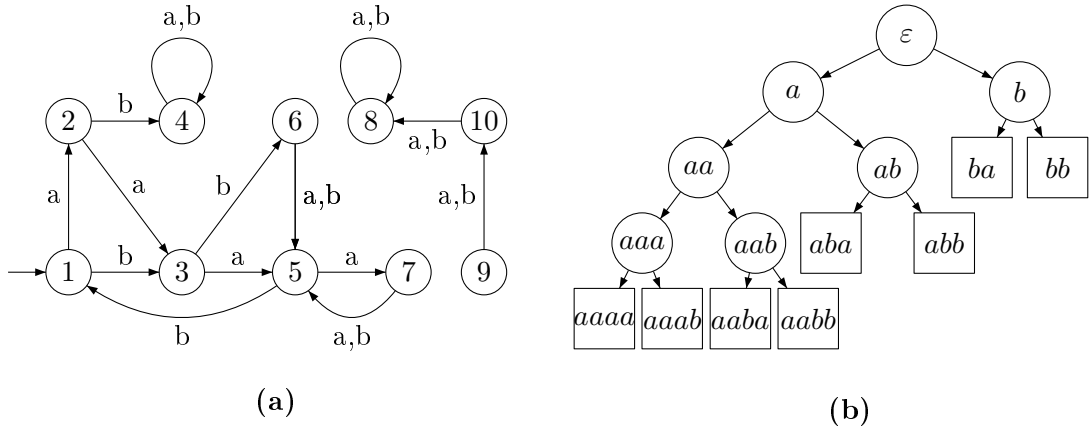


FIG. 9.1 – **(a)** est la structure de transitions fixée et 2 l'état fixé, **(b)** est l'**arbre de dépendance** associé, noté $\mathcal{R}(2)$. On a également $S_2(2) = \{\varepsilon, a, b, aa, ab\}$, $L_2(2) = \{aa, ab\}$ et $s_2(2) = \{2, 3, 4, 5, 6\}$.

comme suit :

$$estnoeud(w) = \begin{cases} 0 \text{ si il existe } v \in A^* \text{ tel que } p \cdot w = p \cdot v \text{ et } v <_{mil} w, \\ 1 \text{ sinon.} \end{cases}$$

autrement dit, $estnoeud(w)$ vaut 1 si et seulement si il existe un mot v inférieur à w dans l'ordre militaire et tel que l'état $p \cdot w$ est égal à l'état $p \cdot v$.

$\mathcal{R}(p)$ est un arbre dont les nœuds et les feuilles de profondeur h sont étiquetés par des mots de longueur h. On le construit de façon récursive, en utilisant un parcours en largeur des nœuds de l'arbre, en partant de la racine. Pour chaque nœud de profondeur h, étiqueté par un mot w, et chaque lettre a de l'alphabet, on ajoute un nœud étiqueté par wa à la profondeur $h+1$ si la fonction $estnoeud(wa)$ renvoie 1, et une feuille si elle renvoie 0. La Figure 9.1 montre un exemple d'arbre de dépendance.

Il est assez intuitif qu'un même arbre peut être obtenu à partir de plusieurs structures de transitions. Pour un état p et un arbre de dépendance $\mathcal{R}(p)$ fixés, l'ensemble des structures de transitions *associées* à $\mathcal{R}(p)$ est l'ensemble des structures des transitions pour lesquelles les conditions modélisées par $\mathcal{R}(p)$ sont satisfaites.

Notations associées à l'arbre $\mathcal{R}(p)$:

- $S_h(p)$ désigne l'ensemble des nœuds de profondeur inférieure ou égale à h dans $\mathcal{R}(p)$.
- $L_h(p)$ désigne l'ensemble des nœuds de profondeur h dans $\mathcal{R}(p)$.
- $s_h(p)$ est l'ensemble des états atteints dans une structure de transitions associée à $\mathcal{R}(p)$, à partir de l'état p, en suivant un chemin étiqueté par un mot de longueur inférieure ou égale à h.

Chaque nœud de l'arbre étant étiqueté par un mot, on utilisera les notations $w \in S_h(p)$ et $w \in L_h(p)$ si le mot w étiquette un nœud appartenant à l'un de ces ensembles. Pour toute structure de transitions associée à l'arbre de dépendance $\mathcal{R}(p)$, on a $|s_h(p)| = |S_h(p)|$.

Lemme 47. *Pour tout état p fixé, si un arbre de dépendance $\mathcal{R}(p)$ contient f feuilles de profondeur inférieure ou égale à h, alors le nombre de structures de transitions associées à $\mathcal{R}(p)$ est inférieur ou égal à $|\mathcal{T}_n| \left(\frac{|S_h(p)|}{n}\right)^f$.*

Démonstration. On rappelle que $|\mathcal{T}_n|$ est égal au produit des cardinaux des ensembles d'états d'arrivée possibles pour chaque transition. Soit wa l'étiquette d'une feuille de profondeur inférieure ou égale à h. Pour toute structure de transitions associée à l'arbre $\mathcal{R}(p)$, la transition étiquetée par a sortant de l'état $p \cdot w$ aboutit dans un état $p \cdot v$, avec $v \in S_h(p)$. Le nombre d'états d'arrivées possibles pour cette transition est donc borné par $|S_h(p)|$, au lieu de l'être par n. Cette majoration est certes grossière mais suffit pour les besoins de la preuve. □

4 Le graphe de $\mathcal{T}$-dépendance

On introduit également un graphe de dépendance pour modéliser les ensembles de structures de transitions. Sa fonction de modélisation est complémentaire de celle proposée par l'arbre de dépendance. Pour deux états fixés p et q, deux x-uplets fixés (x est un entier) de mots non vides $\overrightarrow{u} = (u_1, \ldots, u_x)$ et $\overrightarrow{v} = (v_1, \ldots, v_x)$, deux ensembles fixés φ_p et φ_q de couples de mots (w,w') tels que $w' <_{mil} w$, on définit l'ensemble $\mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ comme suit :

$$\begin{aligned}\mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v}) = \{\tau \in \mathcal{T}_n \mid & \forall (w_p, w'_p) \in \varphi_p, p \cdot w_p = p \cdot w'_p, \\ & \forall (w_q, w'_q) \in \varphi_q, q \cdot w_q = q \cdot w'_q, \\ & \forall i \leq x,\ p \cdot u_i = q \cdot v_i\}\end{aligned}$$

On définit les x-uplets de mots $\overrightarrow{u'} = (u'_1, \ldots, u'_x)$ et $\overrightarrow{v'} = (v'_1, \ldots, v'_x)$ et les x-uplets de lettres $\overrightarrow{\alpha'} = (\alpha'_1, \ldots, \alpha'_x)$ et $\overrightarrow{\beta'} = (\beta'_1, \ldots, \beta'_x)$, tels que pour tout $i \leq x$, on a $u_i = u'_i\alpha_i$ et $v_i = v'_i\beta_i$. À partir de l'ensemble $\mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$, on définit le graphe non-orienté $\mathcal{G}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$, appelé *graphe de $\mathcal{T}$-dépendance*, comme suit :

- ses sommets sont des (r, a), avec $r \in Q$ et $a \in A$, qui modélisent des transitions.
- Il existe une arête $((r, a), (t, b))$ dans $\mathcal{G}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ si et seulement si pour tout $\tau \in \mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$, $r \cdot a = t \cdot b$.

Le graphe de $\mathcal{T}$-dépendance $\mathcal{G}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ satisfait les deux propriétés suivantes :

- Pour tout $i \leq x$, il existe une arête $((p \cdot u'_i, \alpha_i), (q \cdot v'_i, \beta_i))$ dans $\mathcal{G}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$.
- Pour tout $(w_1, w_2) \in \varphi_p$ (resp. $(w_3, w_4) \in \varphi_q$), on a $w_1 = w'_1a_1$ et $w_2 = w'_2a_2$ avec $a_1, a_2 \in A$ tels qu'il existe une arête $((p \cdot w'_1, a_1), (p \cdot w'_2, a_2))$ (resp. $((q \cdot w'_3, a_3), (q \cdot w'_3, a_3))$) dans $\mathcal{G}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$.

Lemme 48. *Si $\mathcal{G}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ contient un sous-graphe acyclique avec j arêtes, alors :*

$$|\mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})| \leq \frac{|\mathcal{T}_n|}{n^j}$$

Démonstration. Deux sommets distincts dans une même composante connexe de $\mathcal{G}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ modélisent deux transitions distinctes qui partagent le même état d'arrivée. Si c est le nombre de composantes connexes dans le graphe, alors

$$|\mathcal{T}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})| \leq n^c.$$

Si le graphe $\mathcal{G}_n(p, q, \varphi_p, \varphi_q, \overrightarrow{u}, \overrightarrow{v})$ contient un sous-graphe acyclique composé d'au moins j arêtes, alors il contient au plus $kn - j$ composantes connexes. Ceci conclut la preuve. □

5 Complexité moyenne

Théorème 49. *Pour tout entier $k \geq 2$ et pour la distribution uniforme sur les automates déterministes complets à n états sur un alphabet de taille k, la complexité moyenne de l'algorithme de Moore est $\mathcal{O}(n \log\log n)$.*

D'après les remarques de la section 1, il nous reste à prouver l'équation suivante :

$$\frac{\gamma_n + 1}{|\mathcal{A}_n|} \sum_{p,q \in \{1,\ldots,n\}} |\mathcal{A}_n(p, q, \lambda_n + 1)| = \mathcal{O}(\log\log n).$$

On définit deux propriétés des structures de transitions pour lesquelles τ est une structure de transitions, p et q deux états distincts, A un alphabet de taille k et μ un entier positif :

(1) $intersectionVide(\tau, p, q)$ est *vraie* si et seulement si $s_{\lambda_n - 2}(p) \cap s_{\lambda_n - 2}(q) = \emptyset$.

(2) $arbreLarge(\tau, p, \mu)$ est *vraie* si et seulement si $\mathcal{R}(p)$ contient au plus une feuille de profondeur inférieure ou égale à μ. Notons que cela implique que pour tout entier i, tel que $i \leq \mu$, on a $|s_i(p)| \geq k^i - 1$. En effet, si $\mathcal{R}(r)$ contient au plus une feuille de profondeur inférieure ou égale à μ, alors il existe $k - 1$ lettres $a \in A$ telles que $\mathcal{R}(r \cdot a)$ ne contient pas de feuille de profondeur inférieure à μ. On a ainsi $|S_\mu(p)| \geq k^\mu - 1$. La Figure 9.2 illustre cette preuve.

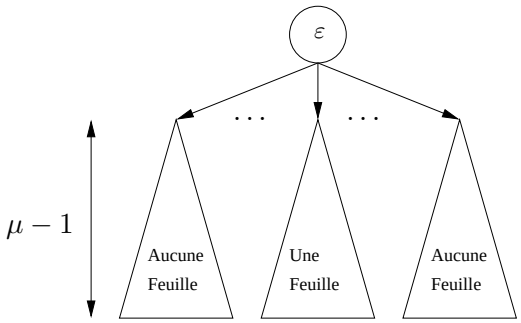


FIG. 9.2 – Pour tout arbre de dépendance $\mathcal{R}(r)$, si il existe au plus une feuille de profondeur inférieure ou égale à μ et k transitions partant de r, alors il existe $k - 1$ sous-arbres qui ne contiennent pas de feuilles de profondeur μ, ce qui revient à dire que chaque sous-arbre contient $\frac{k^\mu - 1}{k - 1}$ nœuds.

Pour p et q deux états fixés, on partitionne l'ensemble des structures de transitions $\mathcal{T}_n$ en trois sous-ensembles :

- $\mathcal{X}_n$ est l'ensemble des structures de transitions τ pour lesquelles il existe un état $r \in Q$ tel que $arbreLarge(\tau, r, \lambda_n)$ est $faux$. On remarque que cet ensemble ne dépend pas des valeurs de p et q.
- $\mathcal{Y}_n(p,q)$ est l'ensemble des structures de transitions τ pour lesquelles les deux conditions suivantes sont satisfaites :
  - Pour tout état $r \in Q$, la propriété $arbreLarge(\tau, r, \lambda_n)$ est $vraie$,
  - Pour tout mot $w \in A^{\leq 2}$, la propriété $intersectionVide(\tau, p \cdot w, q \cdot w)$ est $fausse$.
- $\alpha_n(p,q)$ est l'ensemble des structures de transitions τ pour lesquelles les deux conditions suivantes sont satisfaites :
  - Pour tout état $r \in Q$, la propriété $arbreLarge(\tau, r, \lambda_n)$ est $vraie$,
  - Il existe un mot $w \in A^{\leq 2}$ tel que $intersectionVide(\tau, p \cdot w, q \cdot w)$ est $vraie$.

D'après ces trois définitions, pour tout états p et q fixés, on a bien les égalités suivantes :

$$\mathcal{T}_n = \mathcal{X}_n \cup \mathcal{Y}_n(p,q) \cup \alpha_n(p,q)$$

$$\mathcal{X}_n \cap \mathcal{Y}_n(p,q) \cap \alpha_n(p,q) = \emptyset$$

Ces ensembles nous ont donc permis de partitionner l'ensemble $\mathcal{T}_n$ des structures de transitions. On va à présent les étudier séparément, afin d'en obtenir des propriétés énumératives, ou plus précisément, afin de parvenir à majorer le nombre d'automates appartenant aux ensembles de la forme $\mathcal{A}_n(p, q, \lambda_n + 1)$. Dit de manière informelle, on va montrer que :

- si une structure de transitions τ est dans l'ensemble $\alpha_n(p,q)$, il existe très peu d'ensembles d'états terminaux F tels que $(\tau, F) \in \mathcal{A}_n(p, q, \lambda_n + 1)$,
- il existe peu d'automates dont la structure de transitions appartient à l'ensemble $\mathcal{X}_n$ ou à un ensemble $\mathcal{Y}_n(p,q)$.

Les ensembles $\alpha_n(p,q)$

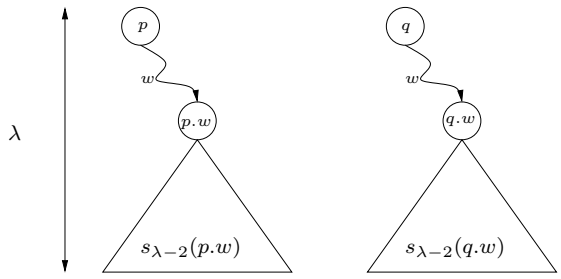


FIG. 9.3 – Pour toute structure de transitions appartenant à l'ensemble $\alpha_n(p,q)$, il existe un mot w de longueur 2 tel que les ensembles d'états atteints en suivant les chemins partant de $p \cdot w$ et $q \cdot w$ et étiquetés par des mots de longueurs inférieures ou égales à $\lambda_n - 2$, sont disjoints et ont tous deux un cardinal supérieur ou égal à $\log n^3 - 1$.

La Figure 9.3 illustre les propriétés des structures de transitions appartenant à l'ensemble $\alpha_n(p,q)$.

Lemme 50. *Pour toute structure de transitions fixée $\tau \in \alpha_n(p,q)$, et un mot fixé u de longueur λ_n+1, la propriété suivante est satisfaite : tout graphe de $\mathcal{F}$-dépendance $\mathcal{G}_\tau(p,q,u)$ contient un sous-graphe acyclique avec au moins $k^{\lambda_n-2}-1$ arêtes.*

Démonstration. On rappelle que pour toute structure de transitions τ appartenant à l'ensemble $\alpha_n(p,q)$, il existe un mot $w \in A^{\leq 2}$ tel que la propriété $intersectionVide(\tau, p\cdot w, q\cdot w)$ est *vraie*. S'il existe plusieurs mots de longueur au plus 2 qui satisfont cette propriété, w désignera par la suite le plus petit dans l'ordre militaire. Soit $\mathcal{G}'$ un sous-graphe de $\mathcal{G}_\tau(p,q,u)$ définit comme suit : il existe une arête (s,t) dans $\mathcal{G}'$, si et seulement si $s = p\cdot wv$ et $t = q\cdot wv$, où w est le mot évoqué précédemment et où v est un mot appartenant à l'ensemble $S_{\lambda_n-2}(p\cdot w)$, *i.e.* qui étiquette un nœud de l'arbre $\mathcal{R}(p\cdot w)$. Ce sous-graphe $\mathcal{G}'$ contient exactement $|s_{\lambda_n-2}(p\cdot w)|$ arêtes, puisque pour tout $v \in S_{\lambda_n-2}(p\cdot w)$, les états $p\cdot wv$ sont tous deux à deux distincts. La propriété $arbreLarge(\tau, r, \lambda_n)$ étant vraie pour tout état $r \in Q$, on a $|s_{\lambda_n-2}(p\cdot w)| \geq k^{\lambda_n-2}-1$. De plus, comme toute arête de $\mathcal{G}'$ a exactement une extrémité dans $S_{\lambda_n-2}(p\cdot w)$ et que les sommets de $S_{\lambda_n-2}(p\cdot w)$ sont de degré 1, le sous-graphe $\mathcal{G}'$ est acyclique. □

On rappelle que $\mathcal{A}_n(p,q,\ell)$ désigne l'ensemble des automates pour lesquels les états p et q sont séparés lors du ℓ-ème raffinement de partitions.

Corollaire 51. *Pour $\lambda_n = \lceil \log_k \log_2 n^3+2\rceil$, le nombre d'automates dans l'ensemble $\mathcal{A}_n(p,q,\lambda_n+1)$ dont les structures de transitions sont dans $\alpha_n(p,q)$ est $\mathcal{O}\left(|\mathcal{T}_n|\frac{2^n \log n}{n^3}\right)$.*

Démonstration. La preuve se déduit directement des lemmes 31 et 50 (pages 95 et 112) : pour tous états distincts p et q, tout mot u de longueur λ_n+1 , et toute structure de transitions fixée $\tau \in \alpha_n(p,q)$, on a

$$|\mathcal{F}_\tau(p,q,u)| = \mathcal{O}\left(2^{n-\log_2 n^3}\right).$$

Pour une structure de transitions fixée $\tau \in \alpha_n(p,q)$, le nombre de mots de longueur λ_n+1 étant $\mathcal{O}(\log n)$, le nombre de choix possibles pour les ensembles d'états terminaux pour lesquels l'automate est dans $\mathcal{A}_n(p,q,\lambda_n+1)$ est majoré par :

$$\sum_{u\in A^{\lambda_n+1}} |\mathcal{F}_\tau(p,q,u)| = \mathcal{O}\left(\frac{2^n \log n}{n^3}\right)$$

Cette majoration étant valable pour toutes les structures de transitions de taille n, on obtient le résultat énoncé. □

L'ensemble $\mathcal{X}_n$

Lemme 52. *Le nombre de structures de transitions dans $\mathcal{X}_n$ est $\mathcal{O}\left(|\mathcal{T}_n|\frac{\log^5 n}{n}\right)$.*

Démonstration. Pour un état fixé r et un entier fixé $\mu \in \{1,..,\lambda_n\}$, on définit les ensembles $\mathcal{X}_n(r,\mu)$ de toutes les structures de transitions τ où μ est le plus petit entier pour lequel la propriété $arbreLarge(\tau, r, \mu)$ est fausse. On a :

$$\mathcal{X}_n = \bigcup_{r\in\{1,..,n\}} \left(\bigcup_{\mu\in\{1,..,\lambda_n\}} \mathcal{X}_n(r,\mu) \right) \text{ et } \bigcap_{\mu\in\{1,..,\lambda_n\}} \mathcal{X}_n(r,\mu) = \emptyset$$

Pour toute structure de transitions dans $\mathcal{X}_n(r,\mu)$, l'arbre de dépendance $\mathcal{R}(r)$ contient au moins deux feuilles de profondeur inférieure ou égale à μ et au plus une feuille de profondeur inférieure à μ. Afin d'obtenir une majoration du cardinal de l'ensemble $\mathcal{X}_n(r,\mu)$, on partitionne l'ensemble des arbres de dépendance $\mathcal{R}(r)$ possibles :

1. Soit toutes les feuilles sont au niveau μ. Si k est la taille de l'alphabet et f le nombre de feuilles, le nombre d'arbre possibles est égal à :

$$\sum_{f=2}^{k^\mu} \binom{k^\mu}{f}.$$

2. Soit il existe exactement une feuille de profondeur h (avec $h < \mu$), et au moins une feuille de profondeur μ. Le nombre d'arbres possibles est alors majoré par :

$$\sum_{h=1}^{\mu-1} \left(k^h \sum_{f=1}^{k^\mu} \binom{k^\mu}{f} \right).$$

En utilisant le lemme 47, qui donne une majoration sur le nombre de structures de transitions comptées par chaque arbre, on obtient :

$$|\mathcal{X}_n(r,\mu)| \leq \sum_{f=2}^{k^\mu} \left[\binom{k^\mu}{f} |\mathcal{T}_n| \left(\frac{|S_\mu(r)|}{n} \right)^f \right] + \sum_{h=1}^{\mu-1} \left(k^h \sum_{f=1}^{k^\mu} \left[\binom{k^\mu}{f} |\mathcal{T}_n| \left(\frac{|S_\mu(r)|}{n} \right)^{f+1} \right] \right).$$

Puisque $\mu \leq \lambda_n$, on a $|S_\mu(r)| < k^{\lambda_n+1}$ et :

$$|\mathcal{X}_n(r,\mu)| < |\mathcal{T}_n| \left(\sum_{f=2}^{k^{\lambda_n}} \left[\binom{k^{\lambda_n}}{f} \left(\frac{k^{\lambda_n+1}}{n} \right)^f \right] + \frac{\lambda_n k^{2\lambda_n+1}}{n} \sum_{f=1}^{k_n^\lambda} \left[\binom{k_n^\lambda}{f} \left(\frac{k^{\lambda_n+1}}{n} \right)^f \right] \right).$$

Et comme $\binom{k^{\lambda_n}}{f} \left(\frac{k^{\lambda_n+1}}{n} \right)^f \leq \left(\frac{k^{2\lambda_n+1}}{n} \right)^f$, on a :

$$|\mathcal{X}_n(r,\mu)| < |\mathcal{T}_n| \left(\frac{k^{4\lambda_n+2}}{n^2} \sum_{f=0}^{\infty} \left(\frac{k^{2\lambda_n+1}}{n} \right)^f + \frac{\lambda_n k^{4\lambda_n+2}}{n^2} \sum_{f=0}^{\infty} \left(\frac{k^{2\lambda_n+1}}{n} \right)^f \right).$$

$$|\mathcal{X}_n(r,\mu)| = \mathcal{O}\left(|\mathcal{T}_n| \frac{\lambda_n k^{4\lambda_n}}{n^2} \right) = \mathcal{O}\left(|\mathcal{T}_n| \frac{\log^4 n^3 \times \log\log n^3}{n^2} \right).$$

Cette majoration étant valable pour tout $\mu \in \{1,..,\lambda_n\}$ et tout $r \in Q$, on obtient :

$$|\mathcal{X}_n| \leq \left(\sum_{r\in\{1,..,n\}} \sum_{\mu\in\{1,..,\lambda_n\}} |\mathcal{X}_n(r,\mu)| \right) = \mathcal{O}\left(|\mathcal{T}_n| \frac{\log^4 n^3 \times \log^2 \log n^3}{n} \right).$$

Ce qui conclut la preuve. □

Les ensembles $\mathcal{Y}_n(p,q)$

Lemme 53. *Pour tous états distincts p et q , le nombre de structures de transitions dans $\mathcal{Y}_n(p,q)$ est $\mathcal{O}\left(|\mathcal{T}_n|\frac{\log^6 n}{n^3}\right)$.*

Démonstration. Pour toute structure de transitions appartenant à l'ensemble $\mathcal{Y}_n(p,q)$, tout mot $w \in A^{\leq 2}$, il existe deux mots $u, v \in A^{\leq \lambda_n - 2}$ tels que $p \cdot wu = q \cdot wv$. On partitionne l'ensemble $\mathcal{Y}_n(p,q)$ en fonction des feuilles contenues par les arbres de dépendance $\mathcal{R}(p)$ et $\mathcal{R}(q)$.

Aucun arbre ne contient de feuille de profondeur inférieure ou égale à λ_n : soit E un ensemble de lettres, tel que que pour tout $a \in E$, on a $p \cdot a = q \cdot a$. On définit l'entier e comme le cardinal de l'ensemble E. D'après la remarque 46, on a $e < k$. Pour toutes lettres $b \in A \setminus E$ et $c \in A$, la propriété $intersectionVide(\tau, p \cdot bc, q \cdot bc)$ est $fausse$. Pour $\overrightarrow{u}$ et $\overrightarrow{v}$ de taille $x = e + k(k-e)$, tels que :

- pour tout entier j avec $1 \leq j \leq e$, on a $u_j = v_j = a_j$ avec $a_j \in E$,
- pour tout entier i avec $e < i \leq x$, u_i et v_i ont un préfixe commun w_i de longueur 2, dont la première lettre appartient à l'ensemble $A \setminus E$,

alors ce sous-ensemble de $\mathcal{Y}_n(p,q)$ est inclus dans :

$$\bigcup_{\substack{E \subsetneq A \\ \forall a \in E,\ p \cdot a = q \cdot a}} \ \bigcup_{\overrightarrow{u}, \overrightarrow{v}} \mathcal{T}_n(p, q, \emptyset, \emptyset, \overrightarrow{u}, \overrightarrow{v}).$$

Il y a $2^k - 1$ sous-ensembles E possibles et moins de $k^{2\lambda_n(x-e)}$ choix possibles pour les mots $u_i, v_i \in A^{\leq \lambda_n}$, avec $e < i \leq x$. Pour tous $j, l \leq x$, tels que $j \neq l$, les mots u_j et u_l (resp. v_j et v_l) étiquettent des nœuds dans $\mathcal{R}(p)$ (resp. $\mathcal{R}(q)$). En fixant $u_j = u'_j \alpha_j$ et $u_l = u'_l \alpha_l$, avec $u'_j, u'_l \in A^*$ et $\alpha_j, \alpha_j \in A$, on a donc la garantie que $(p \cdot u'_j, \alpha_j) \neq (p \cdot u'_l, \alpha_l)$ et qu'il n'existe pas de chemin entre $(p \cdot u'_j, \alpha_j)$ et $(p \cdot u'_l, \alpha_l)$ car cela impliquerait que $p \cdot u_j = p \cdot u_l$ et que soit le mot u_j soit le mot u_l étiquette une feuille. Ainsi, le graphe $\mathcal{G}_n(p, q, \emptyset, \emptyset, \overrightarrow{u}, \overrightarrow{v})$ contient un sous-graphe acyclique avec x arêtes de la forme $((p \cdot u'_j, \alpha_j), (q \cdot v'_j, \beta_j))$. La Figure 9.4 est un exemple d'arbres

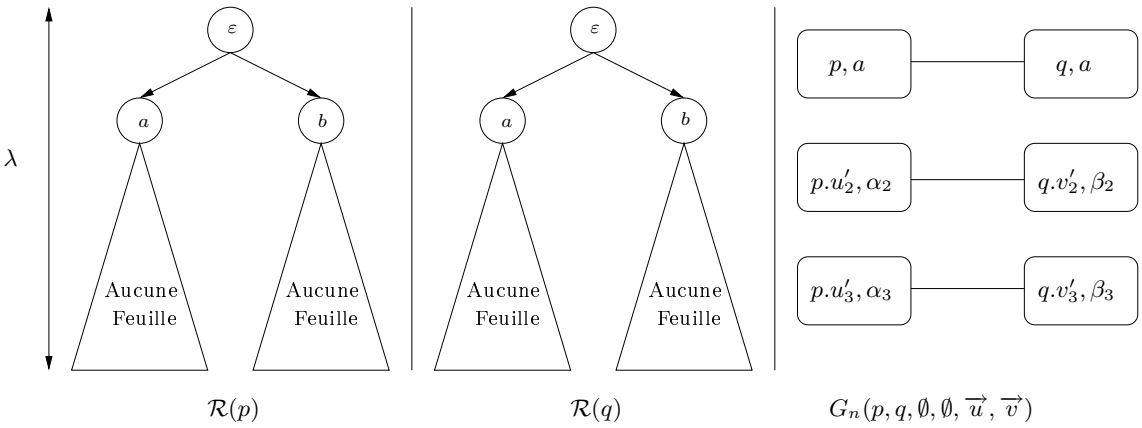


FIG. 9.4 – Premier sous-ensemble de $\mathcal{Y}_n(p,q)$: dans cet exemple, $u_1 = v_1 = a$, u_2 et v_2 ont un préfixe commun ba, u_3 et v_3 ont un préfixe commun bb.

de dépendance et d'un graphe de $\mathcal{T}$-dépendance associés à ce sous-ensemble. En

utilisant le lemme 48, on obtient la majoration suivante sur le cardinal du sous-ensemble de $\mathcal{Y}_n(p,q)$:

$$(2^k-1)\times\frac{|\mathcal{T}_n|}{n^x}\times k^{2\lambda_n(x-e)}$$

On a $x \geq k+1$ (pour $e=k-1$). On majore de nouveau par :

$$(2^k-1)\times\frac{|\mathcal{T}_n|}{n^{k+1}}\times k^{2k\lambda_n}$$

$$\leq (2^k-1)\times\frac{|\mathcal{T}_n|k^{4\lambda_n}}{n^3}\times\left(\frac{k^{2\lambda_n}}{n}\right)^{k-2}$$

$$=\mathcal{O}\left(\frac{|\mathcal{T}_n|\log^4 n}{n^3}\right)$$

Seul l'un des deux arbres contient une feuille de profondeur inférieure ou égale à λ_n : on suppose par symétrie que le mot wc étiquette la feuille dans l'arbre $\mathcal{R}(p)$ et on définit le mot $w'c'$ tel que $w'c' <_{mil} wc$ et $p\cdot wc = p\cdot w'c'$ (ici, c et c' sont des lettres). Pour toute lettre $b\in A$, la propriété $intersectionVide(\tau, p\cdot b, q\cdot b)$ est $fausse$. Soit $\overrightarrow{v}$ et $\overrightarrow{u}$ deux k-uplets. On suppose que tous les éléments de v_i $(1\leq i\leq k)$ commencent par k lettres distinctes. Les structures de transitions de $\mathcal{Y}_n(p,q)$ qui satisfont ces conditions sont incluses dans :

$$\bigcup_{\substack{wc,w'c'\in A^{\leq\lambda_n}\\ w'a'<_{mil}wa}}\ \bigcup_{\overrightarrow{u},\overrightarrow{v}} (\mathcal{Y}_n(p,q,\{(wc,w'c')\},\emptyset,\overrightarrow{u},\overrightarrow{v})\cup\mathcal{Y}_n(p,q,\emptyset,\{(wc,w'c')\},\overrightarrow{u},\overrightarrow{v}))$$

Il y a moins de $k^{2\lambda_n(k+1)}$ choix possibles pour les mots $wc, w'c'\in A^{\leq\lambda_n}$ et $u_i, v_i \in A^{\leq\lambda_n}$, tels que $i\leq k$. On montre que dans le sous-graphe composés des k arêtes $((u'_i,\alpha_i),(v'_i,\beta_i))$, tous les sommets sont d'arité 1. Ce sous-graphe peut-être vu comme une graphe biparti : les sommets $(p\cdot u'_i,\alpha_i)$ d'un côté et les sommets $(q\cdot v'_i,\beta_i)$ de l'autre. Si, au niveau des arbres de dépendance, l'unique feuille se trouve bien dans $\mathcal{R}(p)$, alors pour tous entiers $j,l\leq k$, il n'existe pas de chemin entre les sommets $(q\cdot v'_j,\beta_j)$ et $(q\cdot v'_l,\beta_l)$. En effet, s'il existait un chemin entre ces deux sommets, alors on aurait $q\cdot v_j = q\cdot v_l$, ce qui contredirait le fait que v_j et v_l étiquettent des noeuds et non des feuilles dans l'arbre $\mathcal{R}(q)$. Les sommets $(p\cdot u'_i,\alpha_i)$ sont donc tous d'arité 1 . De plus, tous les v_i sont deux à deux distincts, ce qui conclut la preuve. Dans un cycle, tous les sommets sont d'arité au moins 2. Ce sous-graphe est donc acyclique. L'ajout au sous-graphe de l'arête entre les sommets associés à $p\cdot wc$ et $p\cdot w'c'$ ne peut pas créer de cycle. Ainsi, les deux graphes $\mathcal{G}_n(p,q,\{(wa,w'a')\},\emptyset,\overrightarrow{u},\overrightarrow{v})$ et $\mathcal{G}_n(p,q,\emptyset,\{(wa,w'a')\},\overrightarrow{u},\overrightarrow{v})$ contiennent un sous-graphe acyclique avec $k+1$ arêtes (voir la Figure 9.5). En utilisant le lemme 48, on obtient la majoration énoncée dans le lemme 53.

Les deux arbres contiennent une feuille de profondeur inférieure à λ_n : Soient w_pa_p (resp. w_qa_q) l'étiquette de la feuille dans l'arbre $\mathcal{R}(p)$ (resp. $\mathcal{R}(q)$) et $w'_pa'_p$ le mot pour lequel $w'_pa'_p <_{mil} w_pa_p$ et $p\cdot w_pa_p = p\cdot w'_pa'_p$. Pour tout $b_i\in A$, tel que b_i n'est pas préfixe de w_pa_p, la propriété $intersectionVide(\tau, p\cdot b_i, q\cdot b_i)$ est $fausse$. Soit $\overrightarrow{u}$ et $\overrightarrow{v}$ deux $(k-1)$-uplets, tels que pour tout $i\leq k-1$, la lettre

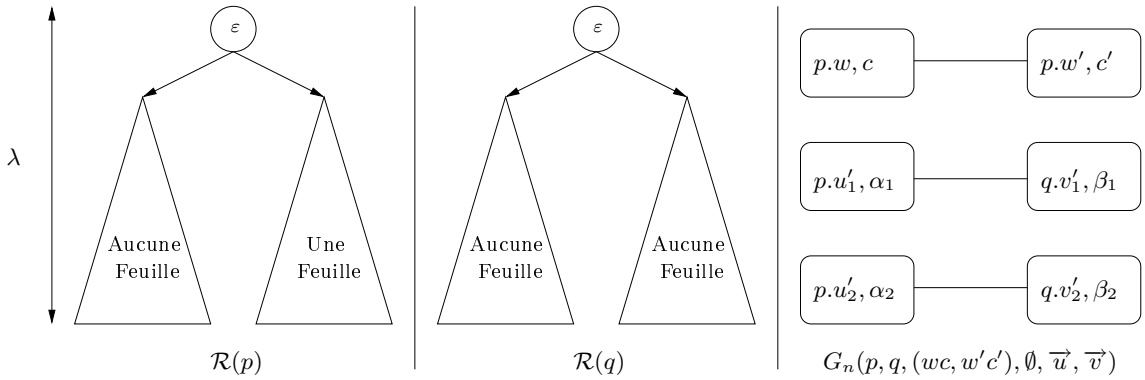


FIG. 9.5 – Deuxième sous-ensemble de $\mathcal{Y}_n(p,q)$: dans cet exemple, w étiquette la feuille de $\mathcal{R}(p)$, u_1 et v_1 partagent le préfixe a, u_2 et v_2 partagent le préfixe b.

$b_i \in A$ est préfixe des mots u_i et v_i, ce dernier sous-ensemble de $\mathcal{Y}_n(p,q)$ est inclut dans :

$$\bigcup_{\substack{w_p a_p, w_q a_q \in A^{\leq \lambda_n} \\ w'_p a'_p, w'_q a'_q \in A^{\leq \lambda_n}}} \bigcup_{\overrightarrow{u}, \overrightarrow{v}} \mathcal{Y}_n(p, q, (w_p, w'_p), (w_q, w'_q), \overrightarrow{u}, \overrightarrow{v})$$

Il existe moins de $k^{2\lambda_n(k+1)}$ choix possibles pour les paires $(w_p, w'_p), (w_q, w'_q)$ et toutes les paires (u_i, v_i), avec $i \leq (k-1)$. En utilisant les mêmes arguments que dans le

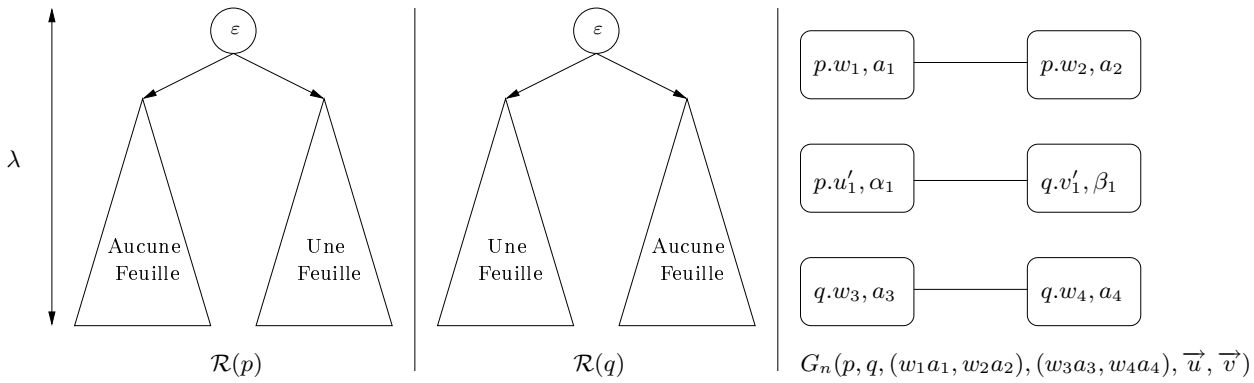


FIG. 9.6 – Troisième sous-ensemble de $\mathcal{Y}_n(p,q)$: w_1 étiquette la feuille de l'arbre $\mathcal{R}(p)$, w_3 étiquette la feuille de l'arbre $\mathcal{R}(q)$.

cas précédent, on montre que dans le sous-graphe composé des $k-1$ arêtes de type $((p \cdot u'_i, \alpha_i), (q \cdot v'_i, \beta_i))$, tous les sommets sont d'arité 1. Le mot w'_p (reps. w'_q) et tous les mots u_i (resp. v_i) étiquettent au total k nœuds distincts dans $\mathcal{R}(p)$ (reps. $\mathcal{R}(q)$). Si on ajoute l'arête entre les sommets associés à $p \cdot w_p$ et $p \cdot w'_p$ (resp. $q \cdot w_q$ et $q \cdot w'_q$)), il ne peut donc pas y avoir un chemin entre un sommet $(p \cdot u'_i, \alpha_i)$ (resp. $(q \cdot v'_i, \beta_i)$) et le sommet associé à $p \cdot w'_p$ (resp. $q \cdot w'_q$). On en déduit que le graphe $\mathcal{G}_n(p, q, \{(w_p, w'_p)\}, \{(w_q, w'_q)\}, \overrightarrow{u}, \overrightarrow{v})$ contient un sous-graphe acyclique avec $k+1$ arêtes (voir la Figure 9.6). □

Conclusion de la preuve

Rappelons que l'on cherche à prouver l'Équation 9.1 :

$$\frac{(\gamma_n+1)}{|\mathcal{A}_n|}\sum_{i=\lambda_n+1}^{\gamma_n}|\mathcal{A}_n^i| = \mathcal{O}(\log\log n)$$

On note $\widetilde{\mathcal{X}_n}$, $\widetilde{\alpha_n}(p,q)$ et $\widetilde{\mathcal{Y}_n}(p,q)$ les ensembles d'automates dont les structures de transitions associées sont respectivement dans $\mathcal{X}_n$, $\alpha_n(p,q)$ et $\mathcal{Y}_n(p,q)$. On a :

$$\bigcup_{i>\lambda_n}\mathcal{A}_n^i = \bigcup_{p,q\in\{1,..,n\}}\mathcal{A}_n(p,q,\lambda_n+1) \subseteq \widetilde{\mathcal{X}_n} \cup \left(\bigcup_{p,q\in\{1,..,n\}}\widetilde{\alpha_n}(p,q)\cup\widetilde{\mathcal{Y}_n}(p,q)\right)$$

En utilisant les lemmes 50,52 et 53 on obtient :

$$\begin{aligned}\sum_{i>\lambda_n}|\mathcal{A}_n^i| &\leq 2^n|\mathcal{T}_n|\frac{\log^5 n}{n^3} + \sum_{p,q\in\{1,..,n\}}2^n|\mathcal{T}_n|\left(\frac{\log n}{n^3}+\frac{\log^6 n}{n^3}\right)\\ &= \mathcal{O}\left(|\mathcal{A}_n|\times\frac{\log^6 n}{n}\right)\\ \frac{(\gamma_n+1)}{|\mathcal{A}_n|}\sum_{i=\lambda_n+1}^{\gamma_n}|\mathcal{A}_n^i| &= \mathcal{O}\left(\frac{\log^7 n}{n}\right) = \mathcal{O}\left(\log\log n\right)\end{aligned}$$

On a donc $\mathcal{N}_n = \mathcal{O}(\log\log n)$, ce qui conclût la preuve du théorème 36.

6 Distribution de Bernoulli

Théorème 54. *Pour tout entier fixé $k \geq 2$, pour la distribution uniforme sur l'ensemble des structures de transitions déterministes complètes à n états, et pour la distribution sur l'ensemble des états terminaux où chaque état a une probabilité $x \in]0,1[$ d'être final, la complexité moyenne de l'algorithme de Moore est $\mathcal{O}(n\log\log n)$.*

Démonstration. Soit x un nombre réel fixé avec $0 < x < 1$ et soit r le nombre réel définit comme $r = \max\{x, 1-x\} \in [1/2, 1[$. En utilisant le théorème 39, on sait que la contribution au nombre moyen d'itérations des automates minimisés en plus de $5\log_{1/r} n$ raffinements est $\mathcal{O}(1)$, de même que celle des automates minimisés en moins de $\log_{1/r}\log_2 n + 2$ itérations est $\mathcal{O}(\log\log n)$.
On définit donc la constante $\lambda_r = \lceil \log_k \log_{1/r} n + 2\rceil$. On modifie légèrement la définition des ensembles $\widetilde{\mathcal{Y}_n}(p,q)$ et $\widetilde{\mathcal{X}_n}$, en remplaçant λ_n par λ_r. Il est facile de voir que leur contributions ne sont pas modifiées par rapport au cas précédent, puisque l'on considérait tous les ensembles d'états terminaux comme étant possibles. Ainsi, seul la contribution des ensembles $\widetilde{\alpha_n}(p,q)$ reste à étudier. Pour deux états p et q fixés, pour un mot $u \in A^{\lambda_r}$ fixé, pour une structure de transitions fixée $\tau \in \alpha_n(p,q)$, on note $P_\tau(p,q,u)$ la probabilité d'un ensemble d'états terminaux

d'être dans $\mathcal{F}_\tau(p,q,u)$. En utilisant les mêmes arguments que dans la preuve du théorème 39, on a

$$P_\tau(p,q,u) \leq \Pi_{i=1}^m r^{c_i-1} = r^{\sum_{i=1}^m (c_i-1)} = r^{k^{\lambda_r}-1}$$

et la contribution d'un ensemble $\widetilde{\alpha_n}(p,q)$ est $\mathcal{O}\left(\frac{1}{n^3}\right)$. □

7 Automates accessibles

En se basant sur les conjectures 21 et 23, on obtient la conjecture suivante sur la complexité moyenne de l'algorithme de Moore lorsque les automates sont accessibles.

Conjecture 55. *Soit $k \geq 2$ un entier fixé. Pour la distribution uniforme sur l'ensemble des automates déterministes accessibles et complets, la complexité moyenne de l'algorithme de Moore est $\Theta(n \log\log n)$.*

8 Complexité générique

Les résultats obtenus sur la complexité moyenne de l'algorithme de Moore selon différentes distributions donnent lieu à des résultats analogues sur la complexité générique. On ne donnera ici qu'un seul exemple, le mode de raisonnement étant toujours le même.

Théorème 56. *Pour la distribution uniforme sur l'ensemble des automates déterministes complets de taille n, la complexité générique de l'algorithme de Moore est $\mathcal{O}(n \log\log n)$.*

Démonstration. On commence par identifier les ensembles négligeables, au sens de la complexité générique. On rappelle que $\lambda_n = \lceil \log_k \log_2 n^3 + 2 \rceil$.

Lemme 57. *Pour la distribution uniforme sur l'ensemble des automates déterministes complets, l'ensemble des automates minimisés en plus de λ_n itérations est négligeable.*

Démonstration. D'après la preuve du théorème 49, on a :

$$\sum_{i>\lambda_n} |\mathcal{A}_n^i| = \mathcal{O}\left(|\mathcal{A}_n| \frac{\log^6 n}{n}\right)$$

On a donc bien :

$$\lim_{n\to+\infty} \frac{\sum_{i>\lambda_n} |\mathcal{A}_n^i|}{|A_n|} = 0$$

On utilise les remarques 4 et 5 (page 30) pour conclure la preuve. □

L'ensemble des automates minimisés en moins de λ_n itérations est générique, ce qui conclut la preuve. □

Chapitre 10

Étude de l'algorithme de Hopcroft

Dans ce chapitre, on étudie la complexité en moyenne de l'algorithme de Hopcroft. Celui-ci étant difficile à analyser, on commence par le modifier légèrement, afin de fixer un ordre partiel sur les blocs extraits de la *Structure*. L'algorithme obtenu possède encore plusieurs implantations possibles, lesquelles sont compatibles avec la définition classique donnée dans le chapitre 6. Toutefois, grâce à cette modification, on montre que le coût d'une d'implantation de l'algorithme est majoré par le coût de l'algorithme de Moore pour la même entrée, à une constante près. On en conclut que la complexité moyenne de l'algorithme de Hopcroft, pour la distribution uniforme sur l'ensemble des automates déterministes complets est $\mathcal{O}(n \log\log n)$.

Dans un deuxième temps, on effectue une étude expérimentale des implantations habituelles de l'algorithme de Hopcroft, c'est-à-dire lorsque celui ci est implanté avec une pile ou une file. On émet la conjecture suivante : lorsque l'algorithme est implanté avec l'une de ces deux structures, sa complexité en moyenne pour la distribution uniforme sur l'ensemble des automates déterministes complets est également $\mathcal{O}(n \log\log n)$.

1 Analogie entre les algorithmes de Hopcroft et Moore

1.1 L'idée

Bien que l'algorithme de Hopcroft calcule l'équivalence de Myhill-Nerode, il n'est pas aisé, à une étape donnée de l'algorithme, de savoir si pour un entier $i \in \{1, .., n-2\}$, deux états d'une même classe sont i-équivalents. Dans la description qui suit, on résout ce problème en donnant un ordre partiel sur les blocs de la *Structure* (voir section 5). Plus précisément, on va utiliser deux *Structures* au lieu d'une seule : les blocs de la *Structure Courante* sont traités avant ceux de *Structure Suivante*.

1.2 Nouvelle description de l'algorithme

On initialise l'algorithme de Hopcroft avec la partition $\{F, Q \setminus F\}$, tout comme l'algorithme de Moore, ou comme dans la description classique de l'algorithme, et on place tous les blocs $(min(F, Q \setminus F), a)$ dans *Courante* (Algorithme 16, Lignes $1-2$).

L'algorithme contient ensuite deux boucles imbriquées :

- Lignes 6 à 11 : Un bloc (P, a) est extrait de *Courante* (Ligne 7). Si une classe B est cassée par (P, a), on le remplace par les classes B' et B'' dans la partition et pour toute lettre a de l'alphabet, on fait une `MiseAJour` des *Structures* de la façon suivante (Algorithme 17) : si le bloc (B, a) appartenait à *Courante*, on le remplace par (B', a) et (B'', a). Si le bloc (B, a) appartenait à *Suivante*, on le remplace par (B', a) et (B'', a). Sinon on ajoute $(min(B', B''), a)$ dans *Suivante*. On appellera désormais ce bloc d'instructions une *itération de l'algorithme de Hopcroft*, par analogie avec celle de l'algorithme de Moore.
- Lignes 3 à 11 : tant que *Suivante* n'est pas vide, ce qui signifie qu'au moins une classe a été cassée au cours de l'itération précédente, on intervertit les deux *Structures* (*Courante* contient alors tous les blocs de *Suivante* et ce dernier est vidé) et on commence une nouvelle itération.

L'algorithme s'arrête lorsque les deux *Structures* sont vides. Tout comme dans la description classique, cela implique que la partition est celle induite par l'équivalence de Myhill-Nerode.

Algorithme 16 : Hopcroft

Données : $Partition = \{F, \overline{F}\}$, $C = min\{F, \overline{F}\}$, $Courante = \emptyset$, $Suivante = \emptyset$

```
1  pour chaque a ∈ A faire
2      Ajouter (C, a) dans Suivante
3  tant que Suivante ≠ ∅ faire
4      Courante = Suivante
5      Suivante = ∅
6      tant que Courante ≠ ∅ faire
7          (P, a) = First(Courante)
8          pour chaque B ∈ Partition tel que B est raffiné par (P, a) faire
9              B', B'' ← Raffiner(B, P, a)
10             Casser le bloc B en B' et B'' dans la Partition
11             MiseAJour(Courante, Suivante, B, B', B'')
```

Résultat : le quotient de $\mathcal{A}$ par *Partition*

Algorithme 17 : $MiseAJour(Courante, Suivante, B, B', B'')$

Données : $C = min\{B', \overline{B''}\}$

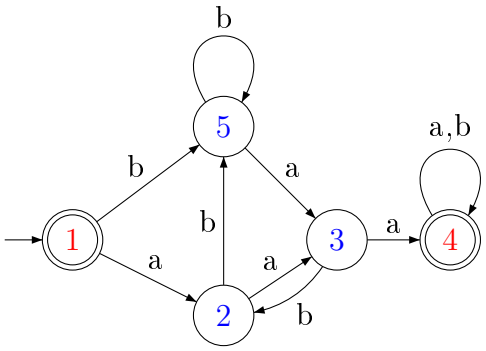
```
1  pour chaque b ∈ A faire
2      si (B, b) ∈ Courante alors
3          Remplacer (B, b) par (B', b) et (B'', b) dans Courante
4      sinon
5          si (B, b) ∈ Suivante alors
6              Remplacer (B, b) par (B', b) et (B'', b) dans Suivante
7          sinon
8              Ajouter (C, b) dans Suivante
```

FIG. 10.1 – Redescription de l'algorithme de Hopcroft

Puisqu'un bloc (C, a) correspond à l'ensemble des transitions arrivant dans les états de la classe C, *Courante* et *Suivante* peuvent être vus comme des ensembles de transitions.

Exemple pour l'algorithme de Hopcroft modifié : deux Piles

Initialisation :

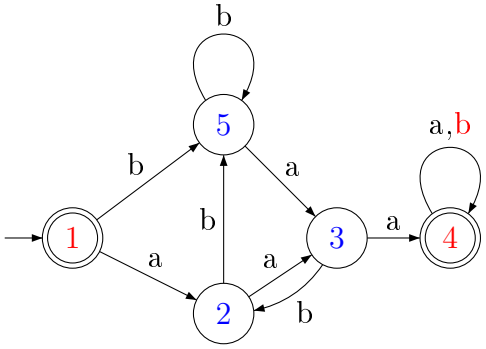


Partition = $\{1,4\}$ $\{2,3,5\}$
Classe = 0 1

Courante				Suivante
	$(0,b)$			
	$(0,a)$			

Exécution :

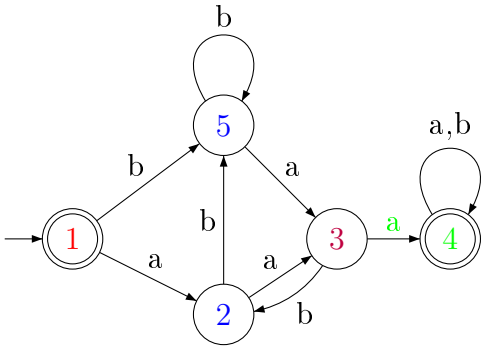
Raffinement par $(0,b)$



Partition = $\{1\}$ $\{2,3,5\}$ $\{4\}$
Classe = 0 1 2

Courante				Suivante
	$(2,a)$			
	$(0,a)$		$(2,b)$	

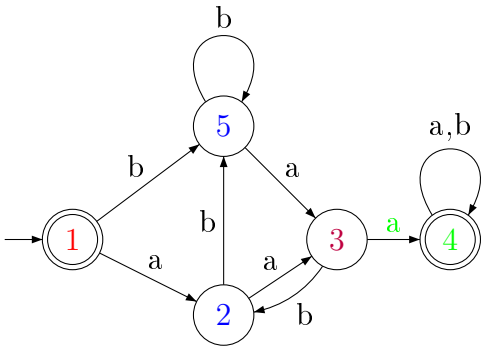
Raffinement par $(2,a)$:



Partition = $\{1\}$ $\{2,5\}$ $\{3\}$ $\{4\}$
Classe = 0 1 3 2

Courante				Suivante
			$(3,b)$	
	$(0,a)$		$(3,a)$	
			$(2,b)$	

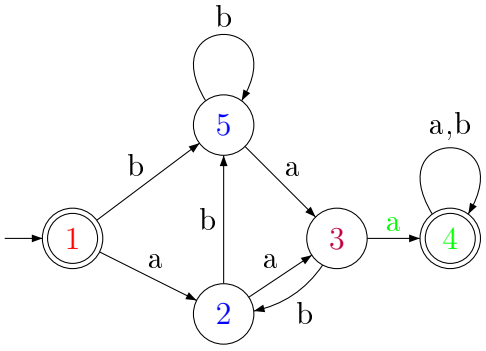
Raffinement par $(0,a)$: la partition n'est pas modifiée.
`Courante` est vide : à ce stade on remarque que, pour toute paire d'états $p,q \in Q$ telle que $p \nsim_1 q$, p et q ne sont pas dans le même sous-ensemble de la Partition. On inverse les piles.



Partition =	$\{1\}$	$\{2,5\}$	$\{3\}$	$\{4\}$
Classe =	0	1	3	2

Courante			Suivante
	$(3,b)$		
	$(3,a)$		
	$(2,b)$		

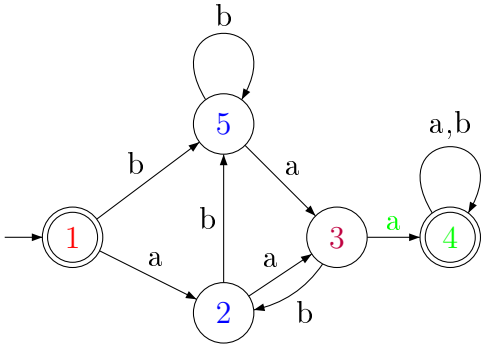
Raffinement par $(3,b)$: la partition n'est pas modifiée



Partition =	$\{1\}$	$\{2,5\}$	$\{3\}$	$\{4\}$
Classe =	0	1	3	2

Courante			Suivante
	$(3,a)$		
	$(2,b)$		

Raffinement par $(3,a)$: la partition n'est pas modifiée



Partition =	$\{1\}$	$\{2,5\}$	$\{3\}$	$\{4\}$
Classe =	0	1	3	2

Courante			Suivante
	$(2,b)$		

Raffinement par $(2,b)$: la partition n'est pas modifiée
Les deux piles sont vides, l'algorithme s'arrête.

Lemme 58. *Pour tout automate déterministe, au début de chaque itération de l'algorithme de Hopcroft, les propriétés suivantes sont satisfaites :*

i) Courante *contient au plus kn transitions.*

ii) Pour tout bloc (B, a) *dans* Courante*, la classe* B *est le fruit d'un raffinement fait lors de l'itération précédente.*

iii) L'ensemble des transitions contenues dans Courante *est exactement l'ensemble des transitions qui seront utilisées au cours de l'itération courante pour raffiner la partition.*

Démonstration. La propriété *i)* vient du fait que l'automate est déterministe : il y a au plus kn transitions dans l'automate et une transition ne peut être dans deux blocs distincts, puisque cela signifierait que la transition possède deux états d'arrivée.

La propriété *ii)* se déduit simplement de l'algorithme : une transition est dans un bloc (C, a) de *Courante* au début d'une itération si et seulement si elle a été ajoutée dans *Suivante* au cours de l'itération précédente. Un bloc ne peut être ajouté dans *Suivante* que par la fonction `MiseAJour`, qui est appelée lorsqu'une classe est raffinée.

La propriété *iii)* se déduit également de l'algorithme : un bloc n'est extrait de *Courante* que pour être utilisé pour raffiner la partition. Dans la fonction `MiseAJour`, *Courante* ne peut être modifiée que pour partitionner un ensemble donné de transitions (C, a) en (C', a) et (C'', a), tel que $(C, a) = (C', a) \cup (C'', a)$. Ainsi, l'opération ne modifie globalement pas l'ensemble des transitions contenu dans *Courante*. □

Remarque 59. *D'après le lemme 58, la complexité en temps d'une itération de l'algorithme de Hopcroft est* $\mathcal{O}(kn)$ *: en effet, les Lignes* 9 *à* 11 *peuvent être réalisées en temps constant, en utilisant les bonnes structures de données. La boucle* ***Foreach*** *(Ligne* 8*) dépend du nombre de transitions contenues dans un bloc, une itération dépend du nombre de transitions dans* Courante*.*

Lemme 60. *Pour tout automate déterministe, tout entier* $i \in \{0, .., n-2\}$*, tous états* p *et* q*, si* $p \nsim_i q$*, alors* p *et* q *ne sont pas dans la même classe d'équivalence à la fin de l'itération* i *de l'algorithme de Hopcroft.*

Démonstration. On prouve ce résultat par récurrence : l'itération 0 représente l'initialisation de l'algorithme, identique à celle de l'algorithme de Moore, qui sépare les états terminaux des non-terminaux. Supposons maintenant que la propriété est vraie jusqu'à une itération i. Soient p et q deux états tels que $p \sim_i q$ et $p \nsim_{i+1} q$. D'après la définition de l'équivalence de Myhill-Nerode, il existe une lettre a telle que $p{\cdot}a \nsim_i q{\cdot}a$. On sait que $p{\cdot}a$ et $q{\cdot}a$ ne sont pas dans la même classe d'équivalence. à la fin de l'itération i de l'algorithme de Hopcroft, ce qui signifie que les transitions étiquetées par a partant des états p et q ont été ajoutées dans deux blocs différents, dans l'une des deux *Structures* lors d'une itération j, pour $j \leq i$. Cela implique que p et q seront séparés soit à l'itération j, soit à l'itération $j+1$. Dans tous les cas, ils ne sont plus dans la même classe d'équivalence à la fin de l'itération $i+1$ de l'algorithme de Hopcroft. Ceci conclut la preuve. □

Corollaire 61. *Pour tout automate déterministe, l'algorithme de Hopcroft s'arrête soit à la même itération que l'algorithme de Moore, soit lors d'une itération précédente.*

Notons que ce lemme ne prouve pas qu'une implantation de l'algorithme de Moore ne sera jamais plus rapide qu'une implantation de l'algorithme de Hopcroft. En effet, l'algorithme de Hopcroft utilise de nombreuses structures de données pour garantir sa complexité dans le pire cas. Pour une simple opération sur une transition, l'algorithme de Hopcroft requiert plus de calculs que l'algorithme de Moore. Il est donc tout à fait possible que pour certains automates, l'algorithme de Moore soit légèrement plus rapide que toute implantation de l'algorithme de Hopcroft.

1.3 Complexité moyenne de l'algorithme de Hopcroft

Théorème 62. *Pour tout entier fixé $k \geq 2$ et pour la distribution uniforme sur l'ensemble des automates déterministes complets à n états sur un alphabet à k lettres, il existe une famille d'implantations de l'algorithme de Hopcroft dont la complexité moyenne est $\mathcal{O}(n \log\log n)$.*

Démonstration. On utilise le théorème 54, le Corollaire 61 et la remarque 59 pour conclure. □

2 Étude expérimentale de différentes implantations de Hopcroft

Dans [4], Baclet et Pagetti comparent deux types d'implantations de l'algorithme de Hopcroft : lorsque la *Structure* est une pile et lorsqu'elle est une file. Les expériences qu'ils effectuent semblent indiquer que la première version est légèrement plus rapide que la seconde. Cependant, les automates utilisés pour les tests appartenaient à des familles particulières d'automates.

On a donc reproduit cette expérience en utilisant la bibliothèque `REGAL`, afin de vérifier si un tel résultat est encore observable dans un cas plus général. Expérimentalement, toutes les implantations qui ont été testées semblent ne différer que d'une constante multiplicative dans leur temps d'exécution. L'utilisation d'une unique pile semble être la méthode la plus rapide en moyenne : l'observation faite dans [4] semble donc être pertinente pour différentes distributions.

Une extension naturelle du théorème 62 serait de montrer que la complexité moyenne de l'algorithme de Hopcroft reste $\mathcal{O}(n \log\log n)$ pour ces deux implantations.

Pour les distributions où les structures de transitions sont choisies uniformément et où soit chaque état a une probabilité fixée $p \in \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}\}$ d'être final, soit un nombre fixé d'états terminaux sont choisis uniformément dans Q, des mesures expérimentales ont été effectuées sur la somme des nombres de transitions associées à chaque bloc extrait de la *Structure*, mesure que l'on considère comme le meilleur indicateur du nombre d'opérations effectuées par l'algorithme de Hopcroft.
En se basant sur les résultats expérimentaux présentés dans la Figure 10.2, on établit les conjectures suivantes :

Conjecture 63. *Soit $k \geq 2$ un entier fixé. Pour la distribution uniforme sur l'ensemble des automates déterministes complets, la complexité moyenne des implantations habituelles de l'algorithme de Hopcroft est $\mathcal{O}(n \log\log n)$.*

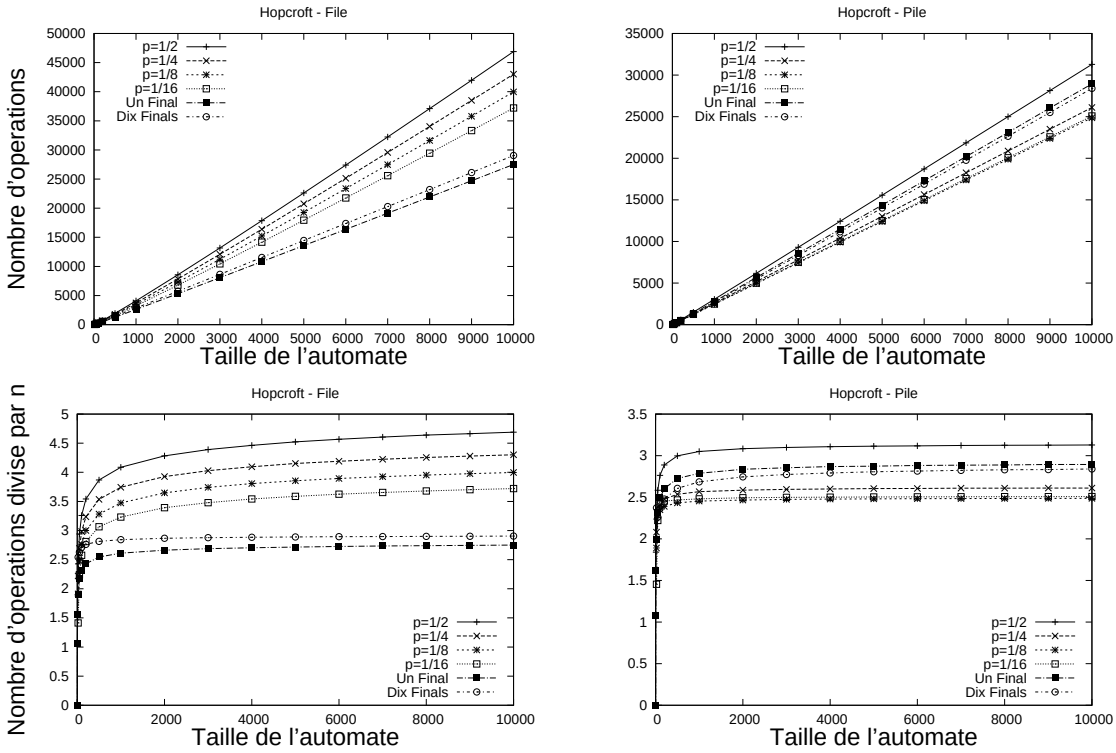


FIG. 10.2 – En haut, les deux graphiques montrent l'évolution du nombre moyen d'opérations effectuées par l'algorithme de Hopcroft. Celui de gauche représente une implantation de la *Structure* avec une file : il semblerait que moins l'automate contient d'états terminaux, plus l'algorithme est rapide. Celui de droite est une implantation de la *Structure* avec une pile : le nombre d'opérations semble moins élevé que dans le cas précédent, en particulier lorsque la probabilité qu'un état soit final ou non suit une loi de Bernoulli. Dans les deux graphiques du bas les valeurs des graphiques du haut ont été normalisées par le nombre d'états de l'automate.

La conjecture 55 et le théorème 17 conduisent à l'énoncé suivant :

Conjecture 64. *Soit $k \geq 2$ un entier fixé. Pour la distribution uniforme sur l'ensemble des automates déterministes accessibles, la complexité moyenne des implantations habituelles de l'algorithme de Hopcroft est $\mathcal{O}(n \log \log n)$.*

Perspectives

Optimiser les générateurs. La complexité moyenne des générateurs d'automates déterministes accessibles présentés dans la partie II est $\mathcal{O}(n\sqrt{n})$ car le nombre moyen de rejets effectués par les générateurs pour engendrer une partition d'ensemble à l'aide de la méthode de Boltzmann est $\Theta(\sqrt{n})$. Décrire un générateur plus efficace pour les partitions d'un ensemble, à l'aide par exemple une méthode *ad hoc*, permettrait d'obtenir un générateur d'automates de complexité linéaire.

Un générateur d'automates où le nombre de transitions indéfinies est fixé. L'algorithme de génération aléatoire d'automate possiblement incomplets présenté dans le chapitre 4 permet d'engendrer des automates incomplets efficacement en utilisant la méthode du rejet. Cependant, si l'on souhaite engendrer aléatoirement des automates où le nombre de transitions indéfinies est précisé par l'utilisateur, la méthode du rejet n'est pas efficace : comme on a pu le voir dans le chapitre 5, expérimentalement, si l'alphabet est de taille 2, un automate incomplet aura en moyenne $1,6$ transitions indéfinies. L'utilisation d'un rejet serait donc inefficace. De plus, si l'on tente d'engendrer directement des partitions d'ensembles respectant la valeur donnée par l'utilisateur, alors le nombre de partitions satisfaisant la propriété k-Dyck devient rapidement insuffisant pour utiliser la méthode par rejet. Il est à noter que dans [41], Héam, Nicaud et Schmitz utilisent la méthode récursive et obtiennent un générateur aléatoire de complexité $\mathcal{O}(n^3)$ en temps. Il s'agit de la seule méthode utilisable à ce jour, mais on peut difficilement envisager de l'utiliser pour engendrer des objets de grande taille.

Énumérer les automates minimaux. Les résultats expérimentaux obtenus à l'aide de *REGAL* semblent indiquer qu'il existe une proportion constante d'automates minimaux parmi les automates déterministes accessibles complets. Une piste possible pour résoudre ce problème ouvert serait de majorer le nombre d'automates non-minimaux de taille n, en étudiant le nombre d'automates reconnaissant le même langage qu'un automate minimal de taille m, pour $n > m$.

Le diamètre moyen d'un automate. Dans le chapitre 5, on a conjecturé que le diamètre moyen d'un automate sur un alphabet d'au moins deux lettres est $\mathcal{O}(\log n)$. Le résultat est non seulement intéressant en soit, mais permettrait également d'obtenir des résultats sur la complexité moyenne des algorithmes de minimisation pour une distribution où le nombre d'états terminaux est fixé.

Complexité de la minimisation des automates accessibles. Dans le chapitre 9, on a montré que la complexité moyenne de l'algorithme de Moore, pour la distribu-

tion uniforme sur l'ensemble des automates déterministes complets, est $\mathcal{O}(n \log\log n)$. On conjecture que le même résultat peut être obtenu sur l'ensemble des automates déterministes accessibles (conjecture 55) et que la borne supérieure soit optimale.

L'algorithme de Hopcroft. La question de l'exécution optimale de l'algorithme de Hopcroft reste ouverte. On émet l'hypothèse suivante : l'algorithme de Hopcroft est plus rapide lorsque l'on ne fait pas de tests pour choisir où insérer les nouveaux blocs. De plus, bien que l'on ait majoré la complexité moyenne d'un ensemble d'implantations de l'algorithme d'Hopcroft par $\mathcal{O}(n \log\log n)$, il est possible que cette borne supérieure ne soit pas optimale sur certaines classes d'automates.

Analyse en moyenne et théorie des automates. On aborde ici des perspectives de recherche moins précise. Le comportement moyen des algorithmes classiques de la théorie des automates est encore peu connu. Sur la base de tests effectués avec `REGAL`, on conjecture que la complexité moyenne de l'algorithme qui calcule le produit de deux automates déterministes accessibles est du même ordre de grandeur que la complexité dans le pire des cas, soit $\Theta(nm)$, où n et m sont les nombres d'états des deux automates d'entrées. Une analyse plus poussée pourrait permettre de découvrir des propriétés moyennes sur les automates produits, et de caractériser des sous-ensembles d'automates pour lesquels la complexité moyenne est meilleure que celle dans le pire des cas.
L'algorithme de déterminisation est également d'un grand intérêt. En effet, il intervient lors de la transformation d'une expression rationnelle en automate minimal. Cette opération nécessite les étapes suivantes :

1. Construire un automate non-déterministe reconnaissant l'expression rationnelle,
2. Déterminiser l'automate.
3. Minimiser l'automate.

On pourrait par exemple caractériser une ou plusieurs familles d'automates pour lesquelles la déterminisation se fait en temps polynomial.

Bibliographie

[1] Alfred Aho, John E. Hopcroft et Jeffrey D. Ullman : *The Design and Analysis of Algorithms*. Addison-Wesley, Reading, MA, 1975.

[2] Jorge Almeida et Marc Zeitoun : Description and analysis of a bottom-up DFA minimization algorithm. *Inf. Process. Lett.*, 107(2):52–59, 2008.

[3] Omar Aït Mous, Frédérique Bassino et Cyril Nicaud : Building the minimal automaton of A^*X in linear time, when X is of bounded cardinality. *In 21st Annual Symposium on Combinatorial Pattern Matching (CPM 2010)*, volume 6129 de *Lecture Notes in Computer Science*, pages 275–287, New York, United States, June 2010. Springer-Verlag.

[4] Manuel Baclet et Claire Pagetti : Around Hopcroft's algorithm. *In Implementation and Application of Automata, 11th International Conference, CIAA 2006, Taipei, Taiwan, August 21-23, 2006. Proceedings*, volume Lecture Notes in Computer Science 4094, pages 114–125, 2006.

[5] Frederique Bassino, Julien David et Cyril Nicaud : REGAL : a library to randomly and exhaustively generate automata. *In* Jan Holub et Jan Žďárek, éditeurs : *CIAA'07*, volume 4783 de *Lecture Notes in Computer Science*, pages 303–305. Springer, 2007.

[6] Frederique Bassino, Julien David et Cyril Nicaud : Enumeration and random generation of possibly incomplete deterministic automata. *Pure Mathematics and Applications*, 13(2-3):1–16, 2008.

[7] Frederique Bassino, Julien David et Cyril Nicaud : Random generation of possibly incomplete deterministic automata. *In Generation Aleatoire de Structure Combinatoire (Gascom'08), Bibbiena, Italy.*, pages 31–40, 2008.

[8] Frederique Bassino, Julien David et Cyril Nicaud : Average case analysis of moore's state minimization algorithm. *Algorithmica*, 2009. Soumis en Juillet 2009, 18 pages.

[9] Frederique Bassino, Julien David et Cyril Nicaud : On the average complexity of Moore's state minimization algorithm. *In* Susanne Albers et Jean-Yves Marion, éditeurs : *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009), Freiburg, Germany.*, volume 3 de *LIPIcs*, pages 123—134. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[10] Frederique Bassino et Cyril Nicaud : Enumeration and random generation of accessible automata. *Theor. Comput. Sci.*, 381:86–104, 2007.

[11] Marie-Pierre Beal et Maxime Crochemore : Minimizing local automata. *In IEEE International Symposium on Information Theory (ISIT'07)*, pages 1376—1380, 2007.

[12] Marie-Pierre Beal et Maxime Crochemore : Minimizing incomplete automata. *In Finite-State Methods and Natural Language Processing (FSMNLP'08)*, pages 9–16, 2008.

[13] Gerard Berry et Ravi Sethi : From regular expressions to deterministic automata. *Theor. Comput. Sci.*, 48(1):117–126, 1986.

[14] Jean Berstel : Automates et grammaires. electonicaly at http ://www-igm.univ-mlv.fr/ berstel/Cours/Licence/CoursAutomates.pdf.

[15] Jean Berstel, Luc Boasson et Olivier Carton : Continuant polynomials and worst-case behavior of Hopcroft's minimization algorithm. *Theor. Comput. Sci.*, 410(30–32):2811–2822, 2009.

[16] Jean Berstel et Olivier Carton : On the complexity of Hopcroft's state minimization algorithm. *In* M. Domaratzki, A. Okhotin, K. Salomaa et S. Yu, éditeurs : *CIAA'04*, volume 3317 de *Lecture Notes in Computer Science*, pages 35–44. Springer, 2004.

[17] Olivier Bodini et Alice Jacquot : Boltzmann samplers for colored combinatorial objects. *In Generation Aleatoire de Structure Combinatoire (Gascom'08), Bibbiena, Italy.*, 2008.

[18] Wilfried Brauer : *Automaten*. B.G. Teubner, Stuttgart, 1984.

[19] Janusz A. Brzozowski : Canonical regular expressions and minimal state graphs for definite events. *In Symposium on the Mathematical Theory of Automata*, volume 12, pages 529–561, Polytechnic Institute of Brooklyn, New York, 1962. Polytechnic Press.

[20] Marie-Pierre Béal, Anne Bergeron, Sylvie Corteel et Mathieu Raffinot : An algorithmic view of gene teams. *Theoretical Computer Science*, 320(2-3): 395–418, 2004.

[21] Alain Cardon et Maxime Crochemore : Partitioning a graph in $O(|A|log_2|V|)$. *Theoretical Computer Science*, 19(1):85–98, 1982.

[22] Giusi Castiglione, Antonio Restivo et Marinella Sciortino : Hopcroft's algorithm and cyclic automata. *In Language and Automata Theory and Applications : Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, pages 172–183, Berlin, Heidelberg, 2008. Springer.

[23] Giusi Castiglione, Antonio Restivo et Marinella Sciortino : On extremal cases of Hopcroft's algorithm. *In* S. Maneth, éditeur : *CIAA'09*, volume 5642 de *Lecture Notes in Computer Science*, pages 14–23. Springer, 2009.

[24] Jean-Marc Champarnaud, Ahmed Khorsi et Thomas Paranthoën : Split and join for minimizing : Brzozowski's algorithm. *In PSC'02 Proceedings*, pages 96–104, 2002.

[25] Jean-marc Champarnaud et Thomas Paranthöen : Random generation of DFAs. *Theoret. Comput. Sci.*, 330:221–235, 2005.

[26] Thomas Claveirole, Sylvain Lombardy, Sarah O'Connor, Louis-Noël Pouchet et Jacques Sakarovitch : Inside Vaucanson. *In* Jacques Farré, Igor Litovsky et Sylvain Schmitz, éditeurs : *CIAA'05*, volume 3845 de *Lecture Notes in Computer Science*, pages 116–128. Springer, 2005.

[27] Robert M. Corless, Gaston H. Gonnet, D. E. G. Hare, David J. Jeffrey et Donald E. Knuth : On the Lambert W-function. *Adv. in Comput. Math.*, 5:329–359, 1996.

[28] Frédéric Dadeau, Jocelyn Levrey et Pierre-Cyrille Héam : On the use of uniform random generation of automata for testing. *Electr. Notes Theor. Comput. Sci.*, 253(2):37–51, 2009.

[29] Julien David : REGAL : une bibliothèque pour la génération des automates déterministes. *In MAnifestation des JEunes Chercheurs en Sciences et Technologies de l'Information et de la Communication (MajecSTIC'07)*, pages 161–170, 2007. 10 pages, en français.

[30] Julien David : The average complexity of Moore's state minimization algorithm is O(n log log n). *In Hlinený, P., Kucera, A., eds. : MFCS*, volume 6281 de *Lecture Notes in Computer Science*, pages 318–329. Springer, 2010.

[31] Philippe Duchon, Philippe Flajolet, Guy Louchard et Gilles Schaeffer : Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability & Computing*, 13(4-5):577–625, 2004.

[32] Philippe Flajolet, Eric Fusy et Carine Pivoteau : Boltzmann sampling of unlabelled structures. *In Analytic Combinatorics and Algorithms Conference (ANALCO'07)*. SIAM, 2007.

[33] Philippe Flajolet et Robert Sedgewick : *Introduction a l'analyse des algorithmes*. International Thomson Publishing France, 1996.

[34] Philippe Flajolet et Robert Sedgewick : *An introduction to the analysis of algorithms*. Addison Wesley, 1996.

[35] Philippe Flajolet et Robert Sedgewick : *Analytic Combinatorics*. Cambridge University Press, 2008.

[36] Philippe Flajolet, Paul Zimmermann et Bernard Van Cutsem : A calculus of random generation of labelled combinatorial structures. *Theoret. Comput. Sci.*, 132:1–35, 1994.

[37] Victor M. Glushkov : The abstract theory of automata. *Russian Math. Surveys*, 16:1–53, 1961.

[38] I. J. Good : An asymptotic formula for the differences of the powers at zero. *Ann. Math. Statist.*, 32(1):249–256, 1961.

[39] David Gries : Describing an algorithm by Hopcroft. *Acta Inf.*, 2:97–109, 1973.

[40] Pierre-Cyrille Héam, Cyril Nicaud et Sylvain Schmitz : Random generation of deterministic tree (walking) automata. *In CIAA*, pages 115–124, 2009.

[41] Pierre-Cyrille Héam, Cyril Nicaud et Sylvain Schmitz : Parametric random generation of deterministic tree automata. *Theoretical Computer Science*, 2010. À paraître.

[42] Charles A. R. Hoare : Quicksort. *The Computer Journal*, 5:10–16, 1962.

[43] John E. Hopcroft : An n log n algorithm for minimizing states in a finite automaton. Rapport technique, Stanford University, Stanford, CA, USA, 1971.

[44] John E. Hopcroft et Jeffrey D. Ullman : *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[45] David Albert Huffman : The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3), 1954.

[46] Ilya Kapovich, Alexei Myasnikov, Paul Schupp et Vladimir Shpilrain : Generic-case complexity, decision problems in group theory and random walks. *J. Algebra*, 264:665–694, 2003.

[47] Donald E. Knuth : *The Art of Computer Programming. Volume 1 : Fundamental Algorithms*. Addison-Wesley, Reading, MA, 1968.

[48] Donald E. Knuth : *The Art of Computer Programming. Volume 2 : Seminumerical Algorithms*. Addison-Wesley, Reading, MA, 1969.

[49] Donald E. Knuth : *The Art of Computer Programming. Volume 3 : Sorting and Searching*. Addison-Wesley, Reading, MA, 1973.

[50] Donald E. Knuth : Two notes on notation. *American Mathematical Monthly*, 99(5):403—422, 1992.

[51] Timo Knuutila : Re-describing an algorithm by Hopcroft. *Theor. Comput. Sci.*, 250(1-2):333–363, 2001.

[52] Dmitry Korshunov : Enumeration of finite automata. *Problemy Kibernetiki*, 34:5–82, 1978.

[53] Dmitry Korshunov : On the number of non-isomorphic strongly connected finite automata. *J. Inf. Process. Cybern.*, 22(9):459–462, 1986.

[54] Jocelyn Levrey : Génération aléatoire d'automates pour le test. Mémoire de D.E.A., Laboratoire d'informatique de l'Université de Franche-Comté, Aout 2008.

[55] Sylvain Lombardy, Yann Régis-Gianas et Jacques Sakarovitch : Introducing Vaucanson. *Theor. Comput. Sci.*, 328(1-2):77–96, 2004.

[56] Lothaire : *Applied Combinatorics on Words*, volume 105 de *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2005.

[57] Edward F. Moore : Gedanken experiments on sequential machines. *In Automata Studies*, pages 129–153. Princeton U., 1956.

[58] Anil Nerode : Linear automaton transformation. *In Proc. American Mathematical Society*, pages 541–544, 1958.

[59] Cyril Nicaud : Average state complexity of operations on unary automata. *In Kutylowski, M., Pacholski, L., Wierzbicki, T., eds. : MFCS*, volume 1672 de *Lecture Notes in Computer Science*, pages 231–240. Springer, 1999.

[60] Cyril Nicaud : *Étude du comportement en moyenne des automates finis et des langages rationnels*. Thèse de doctorat, Universite Paris 7, 2000.

[61] Cyril Nicaud : On the average size of glushkov's automata. *In LATA '09 : Proceedings of the 3rd International Conference on Language and Automata Theory and Applications*, pages 626–637, Berlin, Heidelberg, 2009. Springer-Verlag.

[62] Albert Nijenhuis et Herbert Wilf : *Combinatorial Algorithms*. Academic Press, 1978. Téléchargeable à http ://www.math.upenn.edu/ wilf/website/CombinatorixalAlgorithms.pdf.

[63] Dominique PERRIN : Les débuts de la théorie des automates. *Technique et science informatiques*, 14(4):409–433, 1995.

[64] Jean-Luc REMY : Un procédé itératif de dénombrement d'arbres binaires et son application a leur génération aléatoire. *ITA*, 19(2):179–195, 1985.

[65] Dominique REVUZ : Minimisation of acyclic deterministic automata in linear time. *Theor. Comput. Sci.*, 92(1):181–189, 1992.

[66] Jacques SAKAROVITCH : *Éléments de théorie des automates*. Vuibert Informatique, 2003.

[67] Wojciech SZPANKOWSKI : *Average Case Analysis of Algorithms on Sequences*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[68] Brigitte VALLÉE, Julien CLÉMENT, James Allen FILL et Philippe FLAJOLET : The number of symbol comparisons in Quicksort and Quickselect. *In ICALP (1)*, pages 750–763, 2009.

[69] Antti VALMARI et Petri LEHTINEN : Efficient minimization of DFAs with partial transition. *In* Susanne ALBERS et Pascal WEIL, éditeurs : *25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, volume 1 de *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 645–656, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[70] Bruce W. WATSON : A taxonomy of finite automata minimization algorithms. Rapport technique, Faculty of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands, 1994.

Liste des Notations

On donne ici une liste des notations les plus importantes utilisées dans la thèse.

$\mathcal{D}_n$ Ensemble des structures de transitions de base déterministes accessibles complètes non-isomorphes à n états 42

$\mathcal{B}_n$ Diagrammes k-Dyck marqués de taille n 42

ζ_k 44

ψ Bijection entre $\mathcal{C}_n$ et $\mathcal{B}_n$ 42

χ Bijection entre $\mathcal{C}_n$ et l'ensemble des partition k-Dyck d'un ensemble 49

$\mathcal{E}_n$ Ensemble des structures de transitions complètes à n états en bijection avec $\mathcal{I}_{n-1}$ 52

ϕ Bijection entre $\mathcal{I}_n$ et $\mathcal{E}_{n+1}$ 52

$\rho_{n,k}$ 54

$\mathcal{F}_n$ Ensemble des partitions k-Dyck d'un ensemble de taille $kn+1$ en n sous-ensembles non-vides en bijection avec $\mathcal{I}_{n-1}$ 54

$[\![\ ]\!]$ Crochets d'Iverson 71

L_q Ensemble des mots étiquettant les chemins de q à un état final 71

$\sim$ Équivalence de Myhill-Nerode 72

$\mathcal{M}oore(\mathcal{A})$
Nombre de raffinements effectués par l'algorithe de Moore en prennant $\mathcal{A}$ pour entrée 75

$\mathcal{A}_n^m$ Ensemble d'automates $\mathcal{A}$ pour lesquels $\mathcal{M}oore(\mathcal{A}) = m+1$ 75

$\mathfrak{M}_n$ Nombres moyen de raffinements de partitions effectués par l'algorithme de Moore 92

$\mathcal{F}_\tau(\ell)$ Pour τ fixé, ensemble des ensembles F d'états terminaux tels que $\mathcal{M}oore((\tau, F)) > \ell$ 92

$\mathcal{F}_\tau(p, q, p', q', \ell)$
Sous-ensemble de $\mathcal{F}_\tau(\ell)$ 94

$\mathcal{G}_\tau(p, q, p', q', \ell)$
Graphe de F-dépendance associé à l'ensemble $\mathcal{F}_\tau(p, q, p', q', \ell)$ 94

$\mathcal{F}_\tau(p, q, u)$
Sous-ensemble de $\mathcal{F}_\tau(\ell)$ 107

$\mathcal{G}_\tau(p, q, p', q', \ell)$