Département de formation doctorale en informatique
UFR STMIA

École doctorale IAE + M

# Motifs dans les mots et les arbres

## MÉMOIRE

présenté et soutenu publiquement le 21 decembre 2000

pour l'obtention de l'

**Habilitation à diriger les recherches**

**(Spécialité Informatique)**

par

Grégory KUCHEROV

(Grigori KOUTCHEROV)

**Composition du jury**

| | |
|---|---|
| *Rapporteurs :* | Alexander Bockmayr |
| | Maxime Crochemore |
| | Anatol Slissenko |
| *Examinateurs :* | Nachum Dershowitz |
| | Philippe Flajolet |
| | Juhani Karhumäki |
| | Pierre Lescanne |

**Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503**
**Institut National de Recherche en Informatique et Automatique**

Mis en page avec la classe thloria.

*À tous ceux et toutes celles*
*qui m'empêchaient parfois de travailler*

## Résumé

Ce mémoire d'habilitation est une présentation de certains travaux que j'ai effectués dans les dix dernières années. Le cadre commun de ces travaux est fourni par la notion de motif dans les structures discrètes. En particulier, nous étudions divers problèmes autour de motifs dans les mots et arbres – deux structures fondamentales en informatique. Les résultats présentés sont variés et relèvent des domaines de la combinatoire du mot, l'algorithmique, la théorie des langages formels, la déduction automatique. Une large partie de ces études portent sur la complexité algorithmique de problèmes sous-jacents.



## Abstract

This thesis presents some of the work I have done for the last ten years. A common framework for this work is provided by the notion of pattern in discrete structures. In particular, we study various problems related to motifs in trees and words - two fundamental structures in Computer Science. The results presented are of different nature, and belong to such areas as word combinatorics, algorithm design, formal language theory, and automated deduction. A large part of these studies is devoted to the algorithmic complexity of underlying problems.

# Remerciements

Je remercie en premier lieu Pierre Lescanne de m'avoir beaucoup aidé dans ma carrière scientifique et d'avoir accepté de faire partie du jury de ce travail, qui lui doit beaucoup.

Je remercie Maxime Crochemore qui m'a apporté un soutien précieux au moment de mon changement thématique. Ses travaux m'ont souvent servi de « point d'appui » dans mes recherches. À ce titre, c'est pour moi un très grand honneur qu'il ait accepté d'être rapporteur de ce mémoire.

Je remercie Anatol Slissenko de m'avoir également fait l'honneur d'être rapporteur de ce travail. Il symbolise pour moi le lien avec mes origines scientifiques (et pas seulement scientifiques), d'où je garde un souvenir de lui comme d'un grand spécialiste de la complexité algorithmique et d'un pionnier dans le domaine du *pattern matching*.

Je remercie Alexander Bockmayr d'avoir accepté la lourde tâche de rapporteur interne.

Je remercie beaucoup Nachum Dershowitz, Philippe Flajolet et Juhani Karhumäki d'avoir accepté d'être examinateurs de ce travail et de se déplacer pour participer au jury.

Je remercie les coauteurs de mes articles dont les résultats sont repris dans ce mémoire ; ce sont (dans l'ordre chronologique) Mohammed Tajine, Michaël Rusinowitch, Maria Huber et Dieter Hofbauer, Roman Kolpakov, Yuri Tarannikov, David Plaisted, Mathieu Giraud. Les collaborations avec tous ces chercheurs m'ont laissé d'excellents souvenirs et m'ont fait découvrir le plaisir du travail en commun que je n'avais pas connu jusqu'alors.

Je remercie Vladimir Grébinski, avec qui j'ai travaillé sur la recherche combinatoire et ses applications à la bioinformatique pendant les trois années de sa thèse et grâce à qui j'ai beaucoup appris. Bien que les résultats de ces travaux ne fassent pas partie de ce mémoire, je tiens à le remercier ici.

Enfin, je remercie tous les chercheurs que j'ai pu côtoyer durant mes années au Loria, et tout particulièrement mes collèges de l'équipe EURECA puis de PolKA pour leur soutien. Je tiens à exprimer ma gratitude à Guillaume Hanrot, Michaël Rusinowitch et Gilles Schaeffer pour la relecture du présent mémoire.

*Блаженство и ужас вызывало в нем скольжение наклонной линии вверх по другой, вертикальной,– в примере, указывавшем тайну параллельности. Вертикальная была бесконечна, как всякая линия, и наклонная, тоже бесконечная, скользя по ней и поднимаясь все выше, обречена была двигаться вечно, соскользнуть ей было невозможно, и точка их пересечения, вместе с его душой, неслась вверх по бесконечной стезе. Но, при помощи линейки, он принуждал их расцепиться: просто чертил их заново, параллельно друг дружке, и чувствовал при этом, что там, в бесконечности, где он заставил наклонную соскочить, произошла немыслимая катастрофа, неизъяснимое чудо, и он подолгу замирал на этих небесах, где сходят с ума земные линии.*

Владимир Набоков, Защита Лужина

*Le glissement d'une ligne oblique remontant sur une verticale – dans l'exemple qui expliquait le mystère des lignes parallèles – faisait naître en lui un sentiment de béatitude et d'effroi. La verticale était infinie, comme toute ligne, et l'oblique, qui l'était également, glissant sur elle et s'élevant toujours plus haut, était condamnée à se mouvoir perpétuellement ; il était impossible qu'elle décrochât jamais, et le point d'intersection de ces deux lignes fuyait, avec l'âme de Loujine, toujours plus haut, sur une route sans fin. Cependant, à l'aide d'une règle, il les forçait à se séparer ; il les retraçait simplement à nouveau, en ligne parallèles – et il sentait que, dans l'infini, là où il avait forcé l'oblique à se détacher de l'autre ligne, une inconcevable catastrophe s'était produite, un miracle inexplicable, et il s'absorbait longtemps dans cet empyrée où les lignes terrestres deviennent folles.*

Vladimir Nabokov, La défence Loujine

# Table des matières

# Première partie

# Résultats scientifiques

# Chapitre 1

# Introduction

Ce document se veut une synthèse de certains des travaux que j'ai effectués pendant les dix dernières années. Ces travaux sont de nature et de contenu mathématique très divers ; certains visent à construire des algorithmes efficaces, d'autres se focalisent sur des propriétés combinatoires, ou cherchent à établir la complexité de problèmes sous-jacents, ou encore étudient des propriétés grammaticales de langages associés. Cependant, tous ces travaux sont reliés à plusieurs niveaux et en particulier, ils portent tous sur des propriétés de structures discrètes qui peuvent être décrites à l'aide de la notion de *motifs*.

Le motifs sont un moyen simple et souvent naturel de spécifier des familles non-triviales de structures discrètes. L'idée d'utiliser les motifs consiste à définir les structures en spécifiant des sous-structures que ces structures contiennent, ou ne contiennent pas. Par exemple, on définit certaines classes d'arbres binaires de recherche, comme les arbres rouges-noirs par exemple, avec les motifs locaux qu'ils doivent suivre (ou éviter). Il est intéressant de noter qu'en définissant ces motifs locaux, nous imposons une structure *globale* à l'arbre, à savoir la propriété d'être équilibré. Cela illustre la puissance de cette approche.

Inversement, étant donné une classe de structures, on cherche souvent à la caractériser en spécifiant les sous-structures qui sont présentes, ou au contraire absentes, dans les structures de la classe. Les mathématiques contiennent des résultats très puissants illustrant cette approche, comme le théorème de Kuratovski qui affirme que les graphes planaires sont exactement ceux qui ne contiennent pas de sous-graphes $K_{3,3}$ et $K_5$. De façon plus générale, la théorie de graphes possède une riche variété de résultats sur des classes de graphes caractérisées en termes de sous-graphes interdits (*obstruction sets*) [Brandstädt *et al.*, 1999]. Ces résultats sont confortés par le célèbre théorème de Robertson et Seymour [Robertson et Seymour, 1990], affirmant que toute classe de graphes fermée par rapport aux mineurs peut en effet être caractérisée par un ensemble *fini* de mineurs interdits.

D'un point de vue général, le processus de caractérisation d'une classe de structures en termes de motifs permet de mieux comprendre leurs propriétés communes. Dans ce cas, les motifs peuvent être associés à des *généralisations* de ces structures, et la construction de ces motifs relève de l'*apprentissage*. De ce fait, des modèles visant à construire des motifs qui apparaissent dans les structures données sont étudiés dans la théorie de l'apprentissage automatique [Angluin, 1980; Lassez et Marriot, 1987; Shinohara, 1982]. Récemment, cette approche a trouvé des applications intéressantes dans l'analyse de séquences génomiques [Shinohara et Arikawa, 1995; Yokomori et Kobayashi, 1998]. Ce lien est en effet naturel : les séquences d'ADN sont devenues disponibles en grande quantité, mais le problème majeur reste à les comprendre. Or, le processus de « compréhension » passe inévitablement par la recherche de certains motifs (régularités) com-

muns à un groupe de séquences, ou au contraire absents dans le groupe. Le même problème se pose dans les séquences de protéines où les motifs servent à caractériser des familles de protéines liées et correspondent à leur propriétés structurelles ou fonctionnelles communes.

Les motifs sont également largement utilisés en informatique dans les formalismes de spécification. Ils sont au cœur des systèmes de réécriture [Dershowitz et Jouannaud, 1990], de la programmation fonctionnelle [Bird et Wadler, 1988], ou encore des systèmes experts [Buchanan et Shortliffe, 1984]. Dans tous ces domaines, les motifs sont utilisés pour exprimer les règles de transformation d'objets. La signification typique d'une règle $\alpha \to \beta$ est la suivante : si l'objet contient le motif $\alpha$, son occurrence est remplacée par une instance correspondante du motif $\beta$.

Supposons que nous avons une classe $\mathcal{G}$ de structures combinatoires (telles que graphes, mots, arbres, etc.) pour laquelle il a été défini une notion de sous-structure (comme celle de sous-graphe, sous-mot, sous-arbre, ...). Très souvent, on peut naturellement définir pour la classe $\mathcal{G}$ une notion de *motif* comme une partie d'une structure possible ou comme une structure contenant des parties non-spécifiées. Toutes les structures qui contiennent un motif donné sont appelées des *instances* de ce motif.

Ce cadre général nous permet de définir deux notions qui vont jouer un rôle central dans ce mémoire, du fait que la plupart de problèmes que nous allons traiter vont s'exprimer par ces notions. Étant donné une classe $\mathcal{G}$ et un ensemble $P$ de motifs définis pour cette classe, nous allons noter $Inst(P) \subseteq \mathcal{G}$ l'ensemble de toutes les instances de motifs de $P$, et $Cont(P) \subseteq \mathcal{G}$ l'ensemble de toutes les structures de $\mathcal{G}$ contenant une sous-structure de $Inst(P)$.

Dans ce mémoire, nous allons projeter cette définition générique sur deux types de structures probablement parmi les plus utilisées en informatique : les mots et les arbres. C'est aussi pour ces deux structures que la notion de motif est particulièrement naturelle. Ces deux structures correspondent aux deux chapitres de ce document – le chapitre 2 est consacré au cas des mots et le chapitre 3 au cas des arbres.

Le chapitre 2 contient à son tour trois parties. La première (section 2.3) présente des résultats portant sur la complexité algorithmique (décidabilité, NP-complétude, ...) de divers problèmes de décision liés aux *langages de motifs* (langages de mots définis à l'aide de motifs), tels que les problèmes d'inclusion, de la vacuité ou de finitude. La partie suivante (section 2.4) est consacrée aux motifs répétés dans les mots. Les résultats qui y sont présentés relèvent de la combinatoire du mot. Enfin, la troisième partie (section 2.5) porte sur les algorithmes efficaces de recherche de motifs dans les mots et se situe donc dans le domaine de l'algorithmique des mots.

Le chapitre 3 porte sur le cas des arbres ; il contient deux parties. La section 3.2 est consacrée à des problèmes de décision de langages de motifs dans le cas des arbres. Ensuite, dans la section 3.3 nous étudions certaines classes de langages d'arbres, définies du point de vue grammatical mais liées également aux langages de motifs.

Dans la section 4 nous présentons un survol de résultats de complexité pour des problèmes de décision clefs, en les comparant pour les cas des mots et des arbres. En particulier, nous revenons dans ce chapitre aux résultats des sections 2.3 et 3.2 en les replaçant dans un contexte plus général.

Avant de terminer cette introduction, je tiens à souligner que le but de ce document est de présenter l'ensemble de ces résultats d'une façon synthétique et relativement courte (mais néanmoins rigoureuse). Tous les détails techniques relatifs à ces résultats sont donc hors de portée de ce mémoire. Aucune preuve n'est donnée, hormis une seule exception dans la section 2.3. Cependant, nous avons essayé de porter un soin particulier à ce que tout ce qui est resté « derrière la scène » puisse être reconstitué, soit à partir de nos articles, soit à partir d'articles auxquels

nous faisons référence.

# Chapitre 2

# Motifs dans les mots

## 2.1 Définitions de base

Nous commençons par les définitions de base. Soit $A$ un alphabet fini de *lettres*. Les *mots* sur l'alphabet $A$ sont des suites finies de lettres de $A$. Du point de vue algébrique, les mots sont les éléments du monoïde libre engendré par $A$. L'ensemble des mots est noté $A^*$, et le mot vide $\varepsilon$. Les mots *infinis* dont l'ensemble est noté $A^\omega$ sont des suites infinies de lettres de $A$, ou des fonctions de $\mathbb{N}$ dans $A$.

Soit $w = a_1 \dots a_n$ un mot. L'entier $n$ est appelé la *longueur* de $w$, notée $|w|$. Un mot $a_i \dots a_j$ pour $i \leq j$, que l'on note $w[i..j]$, est appelé un *facteur* de $w$. Une position dans $w$ est un entier entre 0 et $n$. À chaque position $\pi$ dans $w$ on associe une décomposition $w = w_1 w_2$ où $|w_1| = \pi$. La position de la lettre $a_i$ dans $w$ est $i - 1$.

Les *motifs* sur $A$ sont des mots sur l'alphabet $A \cup X$, où $X$ est un alphabet infini de *variables*.

**Exemple 1** *$v = abaababaabaab$ est un mot sur l'alphabet $\{a, b\}$, et $abaxbayb$, $xabax$, $xaxxaxax$ sont des motifs sur $\{a, b\}$ pourvu que $x, y \in X$.*

Une variable $x \in X$ qui a au moins deux occurrences dans un motif est appelée *non-linéaire* dans ce motif, et elle est *linéaire* si elle n'a qu'une occurrence dans le motif. Une *substitution* est un morphisme $\sigma : (A \cup X)^* \to A^*$ tel que $\sigma(a) = a$ pour toute lettre $a \in A$. Une substitution $\sigma$ est appelée *non-effaçante* si pour tout $x$, $\sigma(x) \neq \varepsilon$ ; sinon elle est dite *effaçante*. Un mot $w \in A^*$ est une *instance* d'un motif $p \in (A \cup X)^*$, s'il existe une substitution $\sigma$ telle que $w = \sigma(p)$. Une substitution peut être vue comme une fonction remplaçant les occurrences de variables dans un motif par des mots, à condition que les occurrences d'une même variable soient remplacées par le même mot. Dans l'exemple 1, le mot $v$ est une instance de chacun des trois motifs indiqués.

## 2.2 Langages de motifs

En utilisant la notation générique de l'introduction, on note $Inst(P)$ l'ensemble des instances d'un ensemble $P \subseteq (A \cup X)^*$ de motifs. On considère uniquement ici les instances par les substitutions non-effaçantes. Si l'on autorise les substitutions effaçantes, l'ensemble des instances est noté $Inst_\varepsilon(P)$. D'une façon analogue, $Cont(P)$ est l'ensemble des mots contenant un facteur qui est une instance d'un motif de $P$ par une substitution non-effaçante. Si les substitutions effaçantes sont autorisées, on utilisera la notation $Cont_\varepsilon(P)$. On note que $Cont(P) = A^* Inst(P) A^*$ et $Cont_\varepsilon(P) = A^* Inst_\varepsilon(P) A^*$. Si $P$ est composé d'un seul motif $p$, on utilisera la notation $Inst(p)$, $Cont(p)$ etc. au lieu de $Inst(\{p\})$, $Cont(\{p\})$, etc.

On remarque que pour tout ensemble de motifs $P$, il existe un autre ensemble $S$ tel que $Inst(P) = Inst_\varepsilon(S)$. Appelons $\sigma$ une *substitution élémentaire* si elle envoie chaque variable $x$ sur le mot $ax$ pour $a \in A$. L'ensemble $S$ peut alors être obtenu de $P$ en agissant sur les motifs de $P$ avec toutes les substitutions élémentaires possibles :

$$S = \bigcup_{p \in P} \{\sigma(p) \mid \forall x \in X,\ \sigma(x) = ax \text{ pour } a \in A\}. \tag{2.1}$$

Inversement, $Inst_\varepsilon(S)$ peut toujours être représenté comme $Inst(P)$. Ici, $P$ peut être obtenu à partir de $S$ en substituant toutes les occurrences de certaines variables par le mot vide :

$$P = \bigcup_{p \in S} \{\sigma(p) \mid \forall x \in X,\ \sigma(x) = x \text{ ou } \sigma(x) = \varepsilon\}. \tag{2.2}$$

Un langage $L \subseteq A^*$ représentable comme $Inst(P)$ (ou $Inst_\varepsilon(P)$) pour un ensemble fini de motifs $P \subset (A \cup X)^*$ est appelé un *langage de motifs* (*pattern language*). Notons que selon la remarque ci-dessus, la distinction entre les cas effaçant et non-effaçant n'a pas d'importance pour la définition de la classe des langages de motifs. En revanche, pour certains problèmes avec des restrictions sur le nombre de motifs, la différence entre les cas effaçant et non-effaçant peut s'avérer capitale. Nous en verrons des exemples dans la sections 2.3.1.

Une autre observation, importante pour nous, concerne la relation entre les langages $Inst(P)$ et $Cont(P)$. On peut facilement exprimer $Cont(P)$ en termes de $Inst(P)$ en ajoutant une variable linéaire au début et à la fin de chaque motif de $P$ :

$$Cont(P) = Inst(\bigcup_{p \in P} \{xpy\,,\ xp\,,\ py\,,\ p \mid p \in P,\ x, y \in X \text{ et } x, y \text{ n'apparaissent pas dans } p\}) \tag{2.3}$$

Cette réduction implique qu'un problème de décision pour pour les langages $Cont(P)$ est d'une complexité qui ne dépasse pas la complexité du même problème pour les langages $Inst(P)$. En particulier, si un problème est décidable pour $Inst(P)$, il l'est aussi pour $Cont(P)$. Par contre, si un problème est indécidable pour $Inst(S)$, l'indecidabilité du même problème pour $Cont(P)$ n'en découle pas nécessairement et peut être plus difficile à prouver. Nous rencontrerons cette situation par la suite.

## 2.3 Problèmes de décision liés aux langages de motifs

L'étude des langages de motifs dans le cadre de la théorie des langages (problèmes de la vacuité, d'appartenance, inclusion, ambiguïté, etc.) constitue une direction de recherche relativement nouvelle, dont le début semble être marqué par le travail d'Angluin [Angluin, 1980] dans le contexte de l'apprentissage automatique. En revanche, l'étude des motifs du point de vue de leur *évitabilité* est un sujet bien plus ancien et remonte aux travaux de Thue du début du siècle [Thue, 1906; Thue, 1912]. Les résultats présentés plus loin dans la section 2.4.1 s'inscrivent dans cette dernière direction.

Dans les années 1990, l'intérêt pour les langages de motifs a été réveillé par le résultat impressionnant de [Jiang *et al.*, 1993] (version complète dans [Jiang *et al.*, 1995]) qui a été suivi d'autres travaux [Kari *et al.*, 1995]. Les publications [Salomaa, 1994; Salomaa, 1995] donnent un bon aperçu de cette direction de recherche.

Dans cette section nous étudions quelques problèmes de décision liés aux langages de motifs. Pour chacun de ces problèmes notre but est d'établir sa complexité algorithmique. Par la suite

dans le chapitre 4 nous ferons un résumé des résultats de complexité de certains problèmes de décision pour les langages de motifs, en les comparant avec la complexité des problèmes homologues dans le cas des arbres.

La section 2.3.1 décrit les résultats de l'article [Kucherov et Rusinowitch, 1994] dont la version journal a été publiée dans [Kucherov et Rusinowitch, 1995c]. L'un des deux résultats de la section 2.3.2 a été énoncé dans [Kucherov et Rusinowitch, 1999]. La construction principale de la section 2.3.3 a été introduite dans [Kucherov et Rusinowitch, 1995a], ainsi que certains résultats de cette section.

### 2.3.1 Inclusion de langages de motifs

Cette section porte sur les problèmes d'inclusion de langages de motifs. Le résultat suivant a été prouvé dans [Jiang *et al.*, 1993] :

**Théorème 1 ([Jiang *et al.*, 1993])** *Il n'existe pas d'algorithme qui pour tous motifs $p_1, p_2 \in (A \cup X)^*$ vérifie l'inclusion $Inst(p_1) \subseteq Inst(p_2)$. Il en est de même pour l'inclusion $Inst_\varepsilon(p_1) \subseteq Inst_\varepsilon(p_2)$.*

L'inclusion de deux langages de motifs est donc indécidable même dans le cas réduit à un seul motif. Ce résultat, surprenant au premier abord, peut être mieux compris en remarquant que l'inclusion $Inst(p_1) \subseteq Inst(p_2)$ peut exprimer le problème d'*évitabilité* d'un motif sur un alphabet donné, problème dont la décidabilité n'est pas connue [Currie, 1993]. Nous reviendrons sur ce point dans le chapitre 4 où nous discuterons plus de détails l'évitabilité de motifs.

Concernant le théorème 1 il est également à noter que l'équivalence $Inst(p_1) = Inst(p_2)$ est facile à vérifier : elle se produit si et seulement si les motifs $p_1$ et $p_2$ sont égaux à une permutation de variables près [Angluin, 1980]. Quant à l'équivalence $Inst_\varepsilon(p_1) = Inst_\varepsilon(p_2)$, on ne sait pas si elle est décidable [Salomaa, 1993] (voir aussi [Ohlebusch et Ukkonen, 1997]). Si les motifs $p_1$ et $p_2$ sont composés uniquement de variables, alors $Inst(p_1) \subseteq Inst(p_2)$ ou $Inst_\varepsilon(p_1) \subseteq Inst_\varepsilon(p_2)$ implique que $p_1 = \gamma(p_2)$ pour une substitution $\gamma : X \to (A \cup X)^*$ [Filè, 1988].

Dans le travail [Kucherov et Rusinowitch, 1994] nous nous sommes intéressés à la décidabilité de l'inclusion

$$Inst(p) \subseteq Cont(P) \tag{2.4}$$

pour un ensemble de motifs $P$. À part son intérêt dans le cadre pur de la théorie des langages de motifs, cette question a des liens avec les systèmes de réécriture de termes [Dershowitz et Jouannaud, 1990]. Considérons les *systèmes de réécriture de mots avec variables* [Kucherov et Rusinowitch, 1994]. Un tel système se compose de règles de réécriture $l \to r$, où $l, r \in (A \cup X)^*$ sont des motifs tels que chaque variable de $r$ apparaît dans $l$. En utilisant la terminologie de la réécriture [Dershowitz et Jouannaud, 1990], il s'agit de systèmes de réécriture sur un symbole associatif (concaténation) et un nombre fini de symboles constants (lettres de $A$). Notons que ce type de système peut être vu aussi comme une généralisation de systèmes de semi-Thue (*semi-Thue systems*) [Book, 1987; Book et Otto, 1993] par l'introduction de variables.

Comme tout système de réécriture, un système de réécriture de mots avec variables engendre une relation de réduction sur les objets sous-jacents, ici les mots de $A^*$. Toutes les questions qui sont traditionnellement posées dans le contexte de systèmes de réécriture (telles que la confluence, la terminaison, la propriété de Church-Rosser, etc.) peuvent donc être étudiées dans le cadre de systèmes de réécriture de mots. L'inclusion (2.4) représente une de ces questions, à savoir la propriété de *réductibilité inductive* dont nous parlerons plus dans le chapitre 4 où nous analyserons cette propriété dans le cas des arbres.

Le résultat principal de l'article [Kucherov et Rusinowitch, 1994] est le théorème suivant :

**Théorème 2 ([Kucherov et Rusinowitch, 1994])** *Il n'existe pas d'algorithme qui pour tout motif $p \in (A \cup X)^*$ et tout ensemble fini de motifs $P \subseteq (A \cup X)^*$ vérifie l'inclusion $Inst(p) \subseteq Cont(P)$.*

En d'autres termes, l'inclusion (2.4) est indécidable.

Notons que le théorème 2 ne découle pas du théorème 1 en raison du remplacement de *Inst* par *Cont* dans la partie droite de l'inclusion (voir la remarque dans la section 2.2). D'autre part, le théorème 1 n'est pas non plus une conséquence du théorème 2, à cause de la restriction (très forte) de n'avoir qu'un seul motif dans la partie droite de l'inclusion.

La preuve du théorème 2 donnée dans [Kucherov et Rusinowitch, 1994] a une conséquence très intéressante : elle montre que l'inclusion (2.4) reste indécidable même lorsque le motif $p$ dans la partie gauche est fixe et a une forme extrêmement simple, à savoir si $p = axa$ pour $x \in X$ et $a \in A$. Nous reviendrons à cette propriété par la suite dans les sections 2.3.2 et 4.3.

Sans donner les détails de la preuve du théorème 2, nous en exposerons maintenant l'idée générale. Elle est basée sur les machines de Minsky (machines à deux registres) qui sont des machines abstraites capable de simuler les machines de Turing et donc de calculer toute fonction calculable [Minsky, 1961]. La preuve consiste à construire un ensemble fini $P_{M,d}$ de motifs tel que les instances de $p = axa$ qui ne contiennent pas de motifs de $P_{M,d}$ codent une exécution finie d'une machine de Minsky $M$ sur une donnée $d$. Une exécution est représentée par une suite de configurations de la machine $M$. L'exécution doit commencer par une configuration initiale et, si elle est finie, terminer par une configuration terminale. Les motifs de $P_{M,d}$ sont construits de telle façon que toute instance de $p = axa$ qui n'est pas un encodage valide d'une exécution doit contenir un motif de $P_{M,d}$. En particulier, si deux configurations successives ne correspondent pas à un pas d'exécution valide, ce couple doit contenir un motif de $P_{M,d}$. Ici réside la plus grande difficulté de la preuve, car on doit par ailleurs assurer que les motifs de $P_{M,d}$ n'apparaissent pas dans les instances qui codent les exécutions valides. On obtient la solution en utilisant, de façon essentielle, des variables non-linéaires dans les motifs. Les motifs de $P_{M,d}$ doivent également garantir que l'exécution finie commence par une configuration initiale et termine par une configuration d'arrêt. Les occurrences de $a$ au début et à la fin du motif $p$ servent à assurer ces conditions, plus précisément à forcer certains motifs à ne s'appliquer qu'au début ou à la fin d'une instance de $p$.

Selon la construction de l'ensemble de motifs $P_{M,d}$, seules les instances de $p$ qui codent une exécution *finie* de la machine $M$ sur la donnée $d$ ne contiennent pas de motifs de $P_{M,d}$. Cela signifie qu'il existe une instance de $p$ ne contenant pas de motifs de $P_{M,d}$, si et seulement si la machine $M$ s'arrête sur $d$. Il est donc impossible, pour un ensemble $P \subseteq (A \cup X)^*$, de décider algorithmiquement si toutes les instances de $p$ contiennent un motif de $P$, ce qui prouve le théorème 2.

Pour conclure cette section, notons que du point de vue logique, l'inclusion (2.4) peut être exprimée par une $\forall\exists$-formule positive dans la théorie du premier ordre du semi-groupe libre. En effet, si $\bar{x}$ sont les variables du motif $p$ et $\bar{y}$ celles de l'ensemble $P$, alors l'inclusion (2.4) s'exprime par la formule suivante :

$$\forall \bar{x} \exists u, z \exists \bar{y} \bigvee_{q \in P} p = u\, q\, z. \tag{2.5}$$

Il est connu [Marchenko, 1982] que la théorie du premier ordre des $\forall\exists$-formules positives est indécidable. Le théorème 2 peut donc être vu comme un renforcement du résultat de [Marchenko, 1982] pour ce type particulier de formules ne contenant pas de conjonction.

### 2.3.2 Problèmes de la vacuité et de la finitude du complément

La preuve du théorème 2 décrite dans la section précédente admet des modifications permettant d'obtenir d'autres résultats d'indécidabilité. Nous en présentons deux dans cette section. Le premier affirme qu'il n'existe pas d'algorithme qui vérifie si le complément de l'ensemble $Inst(P)$ est vide, c'est-à-dire si tous les mots de $A^*$ sont couverts par $Inst(P)$.

**Théorème 3** *Il n'existe pas d'algorithme qui pour tout ensemble fini de motifs $P \subseteq (A \cup X)^*$ vérifie si $Inst(P) = A^*$.*

Une modification mineure de la preuve du théorème 2 permet de démontrer le théorème 3. Soit $P_{M,d}$ l'ensemble de motifs de la preuve du théorème 2. Considérons l'ensemble de motifs

$$\tilde{P}_{M,d} = A \cup A^2 \cup \{\alpha x | \alpha \in A,\ \alpha \neq a\} \cup \{x\alpha | \alpha \in A,\ \alpha \neq a\} \cup \{xqy, xq, qy | q \in P_{M,d} \text{ et } x, y \text{ n'apparaissent pas dans } q\}, \quad (2.6)$$

où $a$ est la même lettre que dans le motif $p$ de la preuve du théorème 2. La définition (2.6) assure que les mots qui n'appartiennent pas à $Inst(\tilde{P}_{M,d})$ sont exactement ceux de la forme $awa$ qui ne contiennent pas de motifs de $P_{M,d}$ et qui donc codent une exécution finie de la machine de Minsky $M$ sur la donnée $d$. De déterminer si un tel mot existe est donc un problème indécidable. Cela prouve le théorème 2.

Un autre résultat lié affirme que le problème de savoir, pour un ensemble $P$ de motifs, si $Inst(P)$ contient tous les mots de $A^*$ sauf un nombre fini est indécidable.

**Théorème 4** *Il n'existe pas d'algorithme qui pour tout ensemble fini de motifs $P \subseteq (A \cup X)^*$ vérifie si l'ensemble de mots $\overline{Inst(P)} = A^* \setminus Inst(P)$ est fini.*

Pour prouver le théorème 4, il suffit de modifier la preuve du théorème 2 de la façon suivante. Nous transformons l'ensemble de motifs $P_{M,d}$ en un ensemble $P_M$ qui ne « spécifie » que l'exécution valide de la machine $M$, et ne spécifie pas de donnée $d$. Autrement dit, $P_M$ est tel qu'une instance de $p = axa$ ne contient pas de motifs de $P_M$ si et seulement si elle code une exécution valide finie de la machine $M$ sur une donnée de départ *quelconque*. L'ensemble $P_M$ est facile à construire. En effet, seuls les motifs de $P_{M,d}$ qui portent sur la configuration initiale de l'exécution doivent être modifiés (ou supprimés) (voir [Kucherov et Rusinowitch, 1995c] pour les détails).

Considérons maintenant l'ensemble de motifs

$$\tilde{P}_M = \{\alpha x | \alpha \in A,\ \alpha \neq a\} \cup \{x\alpha | \alpha \in A,\ \alpha \neq a\} \cup \{xpy, xp, py | p \in P_M \text{ et } x, y \text{ n'apparaissent pas dans } p\}, \quad (2.7)$$

La définition (2.7) implique que les mots qui forment $\overline{Inst(\tilde{P}_M)}$ sont, à part les mots d'une ou deux lettres, ceux de la forme $awa$ qui ne contiennent pas de motifs de $P_M$. Ce sont donc les codages de toutes les exécution finies de la machine $M$ sur différentes données. Comme il est indécidable de savoir si une machine de Minsky s'arrête sur un nombre fini ou infini de données, il est indécidable non plus de savoir s'il y un nombre fini ou infini de mots dans $\overline{Inst(\tilde{P}_M)}$, ce que affirme le théorème 4.

Les questions de la vacuité et de la finitude de $\overline{Inst(P)}$ se posent elles aussi très naturellement dans le cadre des systèmes de réécriture et de la déduction automatique en général. Nous y reviendrons dans la section 3.2.2 dans le cas des arbres. D'autres questions liées, comme par exemple la finitude du complément $\overline{Cont(P)} = A^* \setminus Cont(P)$, seront discutées dans le chapitre 4.

### 2.3.3 Cas des motifs linéaires

Dans cette section, nous restreignons les problèmes considérés dans les sections 2.3.1 et 2.3.2 au cas où l'on impose aux motifs de ne contenir que des variables linéaires. Rappelons que cela signifie que chaque variable ne peut avoir qu'une seule occurrence dans le motif.

Si $P \subseteq (A \cup X)^*$ est un ensemble de motifs linéaires, $Inst(P)$ et $Cont(P)$ sont des langages réguliers spécifiés par des expressions régulières de la forme

$$\bigcup_{i=1}^{n} (A^*) w_{i1} A^* w_{i2} \dots A^* w_{ik_i} (A^*), \tag{2.8}$$

où $w_{ij}$ sont des mots de $A^*$ et les parenthèses signifient que (dans le cas de $Inst(P)$) $A^*$ peut ne pas apparaître au début et à la fin de l'expression. La régularité des langages $Inst(P)$ et $Cont(P)$ implique immédiatement que tous les problèmes d'inclusion entre ces langages, le test de la vacuité et de la finitude de ces langages et de leurs compléments, sont décidables en raison de la décidabilité de ces problèmes pour les langages réguliers généraux.

Considérons le problème de l'inclusion $Inst(p) \subseteq Cont(P)$ – problème de la réductibilité inductive de la section 2.3.1 – où $P$ est un ensemble de motifs linéaires. Il se trouve que même si $p$ reste non-linéaire, le problème devient décidable.

**Lemme 1 ([Kucherov et Rusinowitch, 1994])** *L'inclusion (2.4) est décidable pour les motifs $p$ arbitraires et les ensembles $P$ de motifs linéaires.*

L'idée de la preuve est de démontrer qu'il existe une constante $C_{P,p}$ telle que si l'inclusion (2.4) n'est pas vérifiée, alors il existe un contre-exemple $\sigma(p) \notin Cont(P)$, tel que $|\sigma(x)| \leq C_{P,p}$ pour toute variable $x$ de $p$. Cela implique que pour tester l'inclusion (2.4), il suffit de tester l'appartenance à $Cont(P)$ d'un nombre fini d'instances de $p$, d'où un algorithme de décision.

Comme le langage $Cont(P)$ est régulier, son complément $\overline{Cont(P)} = A^* \setminus Cont(P)$ est lui aussi régulier. La constante $C_{P,p}$ est définie en terme d'un automate déterministe $\mathcal{A}$ qui reconnaît $\overline{Cont(P)}$. L'argument ici est, d'une certaine façon, inverse au lemme de pompage pour les langage réguliers : si un facteur d'un mot du langage est suffisamment long, on peut le contracter de sorte que le mot résultant appartienne toujours au langage[1]. On applique ce principe aux images $\sigma(x)$ des variables $x$ de $p$. La difficulté ici est que l'on doit contracter de la même façon les facteurs $\sigma(x)$ qui correspondent aux occurrences distinctes d'une même variable $x$ non-linéaire. Le « principe des tiroirs » (*pigeon hole principle*) est utilisé pour montrer que cela est toujours possible.

Le lemme 1 affirme que l'inclusion (2.4) est décidable mais ne donne pas la complexité exacte du problème. Nous allons maintenant établir cette complexité pour le cas où le motif $p$ est lui aussi un motif linéaire.

Dans le cas général des langages réguliers spécifiés par des expressions régulières, le test d'inclusion et de la vacuité sont des problèmes d'une complexité très élevée, à savoir PSPACE-complets [Garey et Johnson, 1979]. Dans le reste de cette section nous démontrerons que le test de l'inclusion (2.4) et de la vacuité du langage $\overline{Inst(P)}$, les problèmes considérés dans les sections 2.3.1 et 2.3.2, ainsi que certains autres problèmes liés, sont co-NP-complets dans les cas des motifs linéaires.

Nous poursuivrons d'abord l'analyse de l'inclusion (2.4) pour le cas où $p$ est un motif linéaire et démontrerons qu'elle est co-NP-complète.

**Théorème 5** *Le test de l'inclusion $Inst(p) \subseteq Cont(P)$ pour les motifs $p$ linéaires et les ensembles $P$ de motifs linéaires est un problème co-NP-complet.*

1. Notons que la même idée est également utilisée dans le cas des arbres [Kapur *et al.*, 1987; Kucherov et Tajine, 1992] qui sera traité dans la section 3.2.1.

Pour prouver que le problème est co-NP-difficile, nous avons établi, dans [Kucherov et Rusinowitch, 1995a], une réduction à partir du problème MONOTONE-ONE-IN-THREE-SAT, dont on sait qu'il est NP-complet (voir [Garey et Johnson, 1979]), vers le problème $Inst(p) \not\subseteq Cont(P)$, complémentaire de l'inclusion (2.4).

Pour démontrer que le problème $Inst(p) \subseteq Cont(P)$ appartient à la classe co-NP, il convient encore d'exhiber un algorithme non-déterministe résolvant la relation $Inst(p) \not\subseteq Cont(P)$ en temps polynomial. Un tel algorithme consisterait en deux étapes :

1. (*guess*) deviner une substitution de variables $\sigma : X \to A^*$,
2. tester si $\sigma(p)$ ne contient aucun des motifs de $P$.

Pour prouver que l'algorithme est polynomial, il faut prouver que d'une part, la taille de la substitution devinée à l'étape 1 peut être bornée polynomialement, et d'autre part, que l'étape 2 prend un temps polynomial déterministe. Nous commençons par l'étape 2 : comme $P$ est composé de motifs linéaires, l'étape 2 peut être accomplie en temps linéaire par rapport à la taille de $\sigma(p)$ plus la taille de $P$. Pour s'en convaincre, il suffit de remarquer que la recherche d'un motif linéaire dans un mot revient à la recherche successive de ces fragments, ce qui peut être fait en temps linéaire, par exemple, en utilisant à plusieurs reprises l'algorithme de Knuth-Morris-Pratt.

La preuve qu'à l'étape 1 la taille de la substitution $\sigma$ peut être bornée polynomialement est plus complexe et constitue la partie centrale de la preuve du théorème 5. Cette preuve est basée sur la construction générale d'un automate *déterministe complet* reconnaissant le langage $\overline{Cont(P)}$ pour un ensemble $P$ de motifs linéaires. La propriété clef est la suivante : bien que le nombre total d'états de l'automate soit en général exponentiel, la *longueur maximale d'un chemin sans boucle* est bornée polynomialement.

**Exemple 2 ([Kucherov et Rusinowitch, 1995a])** *Pour $k > 0$, considérons l'ensemble de motifs*

$$P = \{\#a\#x\#a\#,\ \#aa\#x\#aa\#, \ldots, \#a^k\#x\#a^k\#\}$$

*sur l'alphabet à deux lettres $\{a, \#\}$. On peut démontrer que l'automate minimal déterministe reconnaissant $Cont(P)$ (et donc $\overline{Cont(P)}$, puisqu'il s'agit d'un automate déterministe et complet et il suffit donc d'inverser les états acceptants et non-acceptants pour obtenir un automate reconnaissant le complément) a un nombre d'états exponentiel en $k$. Intuitivement, cela s'explique par le fait que l'état de l'automate résultant de la lecture d'un mot $w$ doit mémoriser l'ensemble $\{i \mid 1 \leq i \leq k,\ \#a^i\#$ est un facteur de $w\}$, et il y a $2^k$ tels ensembles. Les états qui correspondent à deux ensembles différents ne peuvent pas être factorisés, puisqu'il existe un mot qui transforme un de ces deux états dans un état terminal et pas l'autre.*

*D'autre part, la longueur maximale d'un chemin sans boucle dans cet automate est polynomiale en $k$, et donc polynomiale en la taille de $P$. On peut montrer (voir [Kucherov et Rusinowitch, 1995a]) que cette longueur est $\Theta(k^2)$.*

La construction générale d'un automate reconnaissant $\overline{Cont(P)}$ dont la longueur d'un chemin sans boucle est bornée polynomialement en la taille de $P$ est décrite dans [Kucherov et Rusinowitch, 1995a]. Une fois que nous avons démontré que l'on peut borner polynomialement la longueur d'un chemin sans boucle dans l'automate, il s'en suit, par un argument analogue à la preuve du lemme 1, que si $Inst(p) \not\subseteq Cont(P)$, il existe toujours une instance $\sigma(p) \notin Cont(P)$ telle que $\sigma(x)$ est de taille polynomiale par rapport à la taille de $P$. Autrement dit, la taille du *guess* (étape 1 de l'algorithme) peut être bornée polynomialement. Cela implique que décider l'inclusion $Inst(p) \subseteq Cont(P)$ appartient à la classe co-NP, ce qui prouve le théorème 5.

La construction de l'automate décrite ci-dessus s'est avérée fructueuse. D'une part, elle a inspiré le principe d'un algorithme efficace reconnaissant un ensemble de motifs linéaires, que

nous présenterons dans la section 2.5.1. D'autre part, cette construction permet d'établir la NP-complétude d'autres problèmes liés dont nous énumérons quelques uns dans le reste de cette section.

Avec la construction de l'automate, on peut également démontrer que le problème de tester si l'ensemble $Cont(P)$ est co-fini appartient aussi à la classe co-NP. La construction démontre que s'il n'y a qu'un nombre fini de mots dans $\overline{Cont(P)}$, leur longueur est bornée par un polynôme $\tau(|P|)$. En particulier, tous les mots de longueur $\tau(|P|)+1$ contiennent un motif de $P$. Inversement, si tous les mots de longueur $\tau(|P|)+1$ contiennent un motif de $P$, c'est trivialement le cas pour tous les mots plus longs. Par conséquent, pour tester si $Cont(P)$ est infini, il suffit de deviner, de façon non-déterministe, un mot de longueur $\tau(|P|)+1$ et de tester s'il contient un motif de $P$. Cela prouve que ce test est dans NP, et donc le problème complémentaire est dans co-NP. Enfin, la preuve que ce dernier est co-NP-difficile se fait en réduisant le problème MONOTONE-ONE-IN-THREE-SAT d'une façon similaire à la preuve du théorème 5. Nous avons donc le résultat suivant.

**Théorème 6 ([Kucherov et Rusinowitch, 1995a])** *Le test de la finitude du langage $\overline{Cont(P)}$ pour les ensembles $P$ de motifs linéaires est un problème co-NP-complet.*

On peut modifier la construction de l'automate afin qu'il reconnaisse l'ensemble plus général $Inst(P)$. La différence avec le langage $Cont(P)$ est que l'on doit en plus « forcer » les mots qui correspondent aux instances de certains motifs à commencer par un préfixe donné et/ou terminer par un suffixe donné. Cela entraîne une modification technique de l'automate, qui n'affecte pas sa propriété principale assurant la longueur polynomiale de chaque chemin sans boucle. Cette modification permet de prouver la co-NP-complétude d'autres problèmes liés au langage $Inst(P)$.

**Théorème 7** *Les problèmes suivants sont co-NP-complets :*

*(1) pour un ensemble $P$ de motifs, tester si $\overline{Inst(P)}$ est vide,*

*(2) pour un ensemble $P$ de motifs et un motif $p$, tester si $Inst(p) \subseteq Inst(P)$.*

La preuve que les deux problèmes sont dans co-NP se fait avec l'automate modifié, mentionné ci-dessus. Le problème (2) est plus général que le problème de la réductibilité inductive du théorème 5 ; on en conclut qu'il est co-NP-difficile, et donc co-NP-complet. La preuve que le problème (1) est co-NP-difficile s'inspire de celle du théorème 6. Cependant, comme ces problèmes ne sont pas exactement de même nature (on verra en particulier dans le chapitre 4 que ces deux problèmes sont très différents dans le cas des motifs non-linéaires), et comme la preuve que la relation $\overline{Inst(P)} \neq \emptyset$ est co-NP-difficile n'a pas été publiée, nous allons la donner ici.

**Preuve que le problème (1) du théorème 7 est co-NP-difficile.** Pour prouver que la relation $\overline{Inst(P)} \neq \emptyset$ est NP-difficile, nous allons lui réduire le problème MONOTONE-ONE-IN-THREE-SAT, connu NP-complet [Garey et Johnson, 1979]. Soit $\mathcal{Y}$ un ensemble de variables booléennes et $\mathcal{C} = \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \ldots \wedge \mathcal{C}_m$ une formule booléenne en forme conjonctive, où chaque clause $\mathcal{C}_i$ contient trois variables de $\mathcal{Y}$ (littéraux positifs). Rappelons que le problème MONOTONE-ONE-IN-THREE-SAT est de vérifier s'il existe une affectation de variables $\sigma : \mathcal{Y} \to \{\mathtt{t}, \mathtt{f}\}$ telle que chaque clause $\mathcal{C}_i$ contient précisément une variable envoyée dans $\mathtt{t}$ par $\sigma$.

La preuve consiste à construire, à partir d'une formule $\mathcal{C}$, un ensemble de motifs $P$ avec la propriété clef suivante : chaque mot de $\overline{Inst(P)}$ code une solution de MONOTONE-ONE-IN-THREE-SAT pour la formule $\mathcal{C}$. Cela impliquera qu'il est NP-difficile de savoir si $\overline{Inst(P)}$ est non vide.

Soit $\{y_1, y_2, \ldots, y_n\}$ les variables booléennes de $\mathcal{C}$. Nous allons construire $P$ sur l'alphabet[2] $A = \{\tilde{1}, \ldots, \tilde{n}\} \cup \{\mathtt{t}, \mathtt{f}\} \cup \{\#\}$ et les variables $X = \{x, x_1, x_2, \ldots\}$. Nous allons introduire les motifs de $P$ par groupes. Naturellement, chaque groupe élargit en général l'ensemble des instances et donc réduit son complément.

D'abord, nous allons « forcer » les mots de $\overline{Inst(P)}$ à commencer par un #, à terminer par # et à contenir au moins deux symboles #. Pour ce faire, nous allons introduire les motifs suivants :

$$\alpha,\ x\alpha,\ \alpha x, \qquad \text{pour chaque } \alpha \in A \setminus \{\#\} \tag{2.9}$$

$$\# \tag{2.10}$$

Les motifs suivants expriment qu'un # ne peut être suivi que par un des symboles $\{\tilde{1}, \ldots, \tilde{n}\}$, qui à son tour doit être suivi par un $\mathtt{t}$ ou un $\mathtt{f}$ suivi par un # ou un symbole de $\{\tilde{1}, \ldots, \tilde{n}\}$ :

$$\#\alpha,\ \#\alpha x,\ x_1\#\alpha x_2, \qquad \text{pour chaque } \alpha \in A \setminus \{\tilde{1}, \ldots, \tilde{n}\}, \tag{2.11}$$

$$xi\alpha,\ x_1 i\alpha x_2, \qquad \text{pour chaque } i \in \{\tilde{1}, \ldots, \tilde{n}\}, \alpha \in A \setminus \{\mathtt{t}, \mathtt{f}\}, \tag{2.12}$$

$$x_1\mathtt{t}\alpha x_2,\ x_1\mathtt{f}\alpha x_2, \qquad \text{pour chaque } \alpha \in A \setminus \{\tilde{1}, \ldots, \tilde{n}\} \setminus \{\#\}. \tag{2.13}$$

Les mots qui ne sont pas des instances des motifs introduits jusqu'à maintenant sont des mots du langage $\#((\{\tilde{1}, \ldots, \tilde{n}\}\{\mathtt{t}, \mathtt{f}\})^+\#)^*$. Le pas suivant consiste à « forcer » trois paires de symboles entre les deux symboles # successifs. Pour cela, nous introduisons tous les motifs schématisés par les expressions suivantes :

$$\#\tilde{j_1}\{\mathtt{t}, \mathtt{f}\}\#,\ \#\tilde{j_1}\{\mathtt{t}, \mathtt{f}\}\#x,\ x\#\tilde{j_1}\{\mathtt{t}, \mathtt{f}\}\#,\ x_1\#\tilde{j_1}\{\mathtt{t}, \mathtt{f}\}\#x_2, \qquad \text{pour tous } j_1 \in \{1, \ldots, n\}, \tag{2.14}$$

$$\#\tilde{j_1}\{\mathtt{t}, \mathtt{f}\}\tilde{j_2}\{\mathtt{t}, \mathtt{f}\}\#,\ \#\tilde{j_1}\{\mathtt{t}, \mathtt{f}\}\tilde{j_2}\{\mathtt{t}, \mathtt{f}\}\#x,\ x\#\tilde{j_1}\{\mathtt{t}, \mathtt{f}\}\tilde{j_2}\{\mathtt{t}, \mathtt{f}\}\#,\ x_1\#\tilde{j_1}\{\mathtt{t}, \mathtt{f}\}\tilde{j_2}\{\mathtt{t}, \mathtt{f}\}\#x_2, \qquad \text{pour tous } j_1, j_2 \in \{1, \ldots, n\}, \tag{2.15}$$

$$x_1\tilde{j_1}\{\mathtt{t}, \mathtt{f}\}\tilde{j_2}\{\mathtt{t}, \mathtt{f}\}\tilde{j_3}\{\mathtt{t}, \mathtt{f}\}\tilde{j_4}x_2, \qquad \text{pour tous } j_1, j_2, j_3, j_4 \in \{1, \ldots, n\} \tag{2.16}$$

Après avoir introduit ces motifs, les mots du complément ont la forme

$$\#((\{\tilde{1}, \ldots, \tilde{n}\}\{\mathtt{t}, \mathtt{f}\})^3\#)^*. \tag{2.17}$$

Nous allons maintenant définir comment ces mots vont coder les instances de la formule $\mathcal{C} = \mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \ldots \wedge \mathcal{C}_m$. Chaque instance de $\mathcal{C}$ par une affectation de variables $\sigma$ sera codée par le mot

$$\#\mathbf{C}_1^\sigma\#\mathbf{C}_2^\sigma\#\ldots\#\mathbf{C}_m^\sigma\# \tag{2.18}$$

où $\mathbf{C}_i^\sigma$ est obtenu de $\mathcal{C}_i$ de la façon suivante : si la clause $\mathcal{C}_i$ contient des variables $y_{j_1}, y_{j_2}, y_{j_3}$, $1 \leq j_1, j_2, j_3 \leq n$, alors $\mathbf{C}_i^\sigma = \tilde{j_1}\sigma(y_{j_1})\tilde{j_2}\sigma(y_{j_2})\tilde{j_3}\sigma(y_{j_3})$.

Étant donné une formule $\mathcal{C}$, l'étape suivante consiste à « éliminer » les mots de (2.17) qui ne codent pas des instances valides de $\mathcal{C}$. Pour des raisons techniques et sans restreindre la généralité nous supposons que toutes les clauses de $\mathcal{C}$ sont distinctes et qu'il y en a au moins deux. Soit $\mathbf{C}_i^\sigma$ l'ensemble des (huit) mots codant toutes les instances de $\mathcal{C}_i$ :

$$\mathbf{C}_i = \{\mathbf{C}_i^\sigma \mid \sigma : \{y_1, y_2, y_3\} \to \{\mathtt{t}, \mathtt{f}\} \text{ où } y_1, y_2, y_3 \text{ sont les variables de } \mathcal{C}_i\}.$$

2. Le fait que la taille de l'alphabet dépende de la formule $\mathcal{C}$ n'affecte pas la validité de la preuve. Cependant, il serait aussi possible d'utiliser un alphabet de taille constante ; cela rendrait toutefois la preuve légèrement plus complexe.

Les motifs suivants assurent que les mots de $\overline{Inst(P)}$ ont au plus $(m+1)$ symboles #, qu'ils commencent par un mot de $\mathbf{C}_1$, se terminent par un mot de $\mathbf{C}_m$, et que chaque mot de $\mathbf{C}_i$ est suivi par un mot de $\mathbf{C}_{i+1}$ :

$$\#x_1\#x_2\#\dots x_n\#x_{m+1}\# \tag{2.19}$$

$$\#vx, \qquad \text{pour tout } v \in (\{\tilde{1},\dots,\tilde{n}\}\{\mathtt{t},\mathtt{f}\})^3 \setminus \mathbf{C}_1 \tag{2.20}$$

$$xv\#, \qquad \text{pour tout } v \in (\{\tilde{1},\dots,\tilde{n}\}\{\mathtt{t},\mathtt{f}\})^3 \setminus \mathbf{C}_m \tag{2.21}$$

$$x_1v_1\#v_2x_2, \qquad \text{pour chaque } i,\ 1 \le i \le m-1 \text{ et pour tous } v_1 \in (\{\tilde{1},\dots,\tilde{n}\}\{\mathtt{t},\mathtt{f}\})^3 \setminus \mathbf{C}_i, v_2 \in (\{\tilde{1},\dots,\tilde{n}\}\{\mathtt{t},\mathtt{f}\})^3 \setminus \mathbf{C}_{i+1} \tag{2.22}$$

Les mots qui restent non couverts par les motifs introduits plus haut sont de la forme (2.18). Cependant, il reste à assurer que tous les $\mathbf{C}_i^\sigma$ dans (2.18) correspondent en effet à une *même* affectation $\sigma$. Pour cela, il faut garantir que la même variable apparaissant dans des clauses différentes est affectée d'une même valeur booléenne. Les motifs suivantes remplissent cette tâche en demandant que la valeur qui suit le même $\tilde{j}$ soit la même :

$$x_1\tilde{j}\mathtt{f}x_2\tilde{j}\mathtt{t}x_3,\ x_1\tilde{j}\mathtt{t}x_2\tilde{j}\mathtt{f}x_3, \qquad \text{pour tout } j \in \{1,\dots,n\}. \tag{2.23}$$

Enfin, pour qu'un mot (2.18) code une solution de MONOTONE-ONE-IN-THREE-SAT, il faut que parmi les trois valeurs de chaque clause il n'y ait qu'un et un seul $\mathtt{t}$. Pour assurer cela, nous ajoutons les motifs suivants :

$$x_1\mathtt{f}\tilde{j}_1\mathtt{f}\tilde{j}_2\mathtt{f}\tilde{j}_3x_2,\ x_1\mathtt{t}\tilde{j}_1\mathtt{t}\tilde{j}_2\mathtt{f}\tilde{j}_3x_2,\ x_1\mathtt{t}\tilde{j}_1\mathtt{f}\tilde{j}_2\mathtt{t}\tilde{j}_3x_2,\ x_1\mathtt{f}\tilde{j}_1\mathtt{t}\tilde{j}_2\mathtt{t}\tilde{j}_3x_2,\ x_1\mathtt{t}\tilde{j}_1\mathtt{t}\tilde{j}_2\mathtt{t}\tilde{j}_3x_2, \qquad \text{pour tous } j_1,j_2,j_3 \in \{1,\dots,n\}. \tag{2.24}$$

La construction est finie. Il est facile de vérifier que la taille totale des motifs introduits est polynomiale en la taille de $\mathcal{C}$. Les mots de $\overline{Inst(P)}$ sont exactement les mots (2.18) qui codent les solutions du problème MONOTONE-ONE-IN-THREE-SAT sur la formule $\mathcal{C}$. Décider si de tels mots existent est donc un problème NP-complet.

Nous reviendrons aux questions considérées dans cette section dans le chapitre 4.

## 2.4 Répétitions dans les mots

Ce chapitre est consacré aux motifs (ici, au sens informel du terme) très particuliers dans les mots, à savoir aux répétitions successives de facteurs. Il peut paraître surprenant que l'étude des répétitions dans les mots ait anticipé la combinatoire des mots et la théorie des langages formels moderne [Salomaa, 1981].

Dans ce chapitre, nous présentons deux études combinatoires. La première, de la section 2.4.1, porte sur les mots *sans puissance* $n$, c'est-à-dire ceux qui ne contiennent pas (ou *évitent*, voir aussi la section 4.3) de facteur $u^n$ pour un mot $u$ non-vide. Avec la notation introduite précédemment, il s'agit des mots de l'ensemble $\overline{Cont(x^n)}$. Plus généralement, on considéra les *mots sans puissance* $x$, pour $x \in \mathbb{R}$. Les résultats décrits correspondent à l'article [Kolpakov *et al.*, 1999] et ont été présenté à la conférence MFCS consécutivement en 1997 et 1998 [Kolpakov et Kucherov, 1997; Kolpakov *et al.*, 1998].

Dans la deuxième étude (section 2.4.3), on s'intéresse au nombre des répétitions dans un mot. Ces résultats théoriques seront appliqués à l'analyse de l'algorithme présenté par la suite dans la section 2.5.2. L'ensemble de ces travaux, incluant ceux de la section 2.5.2, est décrit dans

l'article [Kolpakov et Kucherov, 2000b] dont les résultats préliminaires ont été présenté en 1999 aux conférences FCT et FOCS [Kolpakov et Kucherov, 1999b; Kolpakov et Kucherov, 1999a].

La plupart des résultats de cette section ont été obtenus en collaboration avec Roman Kolpakov, que je tiens ici à remercier de nouveau.

### 2.4.1 Mots sans puissance $x$ : les bases

Nous commençons par des définitions de base. On dit qu'un entier $p$ est une *période* d'un mot $w = w[1..n]$ si $w[i] = w[i+p]$ pour toute position $i$, $1 \leq i \leq n-p$. De façon équivalente, $p$ est une période de $w$ si $w[1..n-p] = w[p+1..n]$. Parmi toutes les périodes d'un mot il en existe une, la plus petite, que l'on appellera *la* période de ce mot et notera $p(w)$. On appelle le rapport $e(w) = \frac{|w|}{p(w)}$ *l'exposant* de $w$ et on dit que $w$ est une *répétition* si $e(w) \geq 2$. Si $e(w)$ est un entier supérieur ou égal à deux, on dit que $w$ est une *répétition entière*.

**Exemple 3** *Le mot $w' = abaabaabaaba$ a les périodes 3 et 6, donc $p(w') = 3$ et $e(w') = 4$. $w'$ est donc une répétition entière. Le mot $w'' = abbab$ a la période 3, par conséquent $e(w'') = 5/3$. $w''$ n'est donc pas une répétition. Dans le cas du mot $w''' = ababa$, $p(w''') = 2$ et $e(w''') = 5/2$. $w'''$ est donc une répétition qui n'est pas entière.*

Dans la section suivante nous allons étudier les mots ne contenant pas de répétitions d'un exposant donné. Cette thématique est classique dans la combinatoire du mot et la théorie des langages formels, car elle remonte au travaux d'Axel Thue du début du siècle [Thue, 1906; Thue, 1912] (voir aussi [Berstel, 1992]), où sont construits des mots infinis sans carré et sans chevauchement, respectivement dans l'alphabet à trois et à deux lettres. Un mot est appelé *sans carré* s'il ne contient pas deux occurrences successives d'un facteur $u$ non-vide. Cela équivaut à dire qu'il ne contient pas de répétitions d'exposant 2 (et donc pas de répétitions du tout). Un mot est *sans chevauchement* s'il ne contient pas deux occurrences chevauchantes d'un facteur $u$ non-vide. On peut aisement se convaincre que cela équivaut à dire que le mot ne contient pas de répétition d'un exposant strictement supérieur à deux.

On peut reformuler les résultats de Thue en termes de motifs : les ensembles $\overline{Cont(xx)}$ et $\overline{Cont(xyxyx)}$ sont infinis si l'alphabet sous-jacent contient respectivement au moins trois ou au moins deux lettres. Cela rejoint donc les problèmes considérés dans la section 2.3.2. En particulier, ces résultats s'inscrivent dans le problème général d'*évitabilité* de motifs. On dit qu'un motif, ou un ensemble de motifs, est *évitable* sur un alphabet (fini) donné s'il existe un nombre infini de mots (ou de façon équivalente, un mot infini) évitant ce motif (ensemble de motifs). Les résultats de Thue peuvent alors être reformulés sous une nouvelle forme : les motifs $xx$ et $xyxyx$ sont évitables sur l'alphabet respectivement à deux lettres et à trois lettres. Comme la propriété d'évitabilité dépend de la taille de l'alphabet, on dira qu'un motif (ensemble de motifs) est $k$-évitable s'il est évitable sur un alphabet à $k$ lettres. Le motif $xx$ est donc 3-évitable (mais pas 2-évitable) et le motif $xyxyx$ est 2-évitable. Dans le chapitre 4 on reviendra aux résultats de décidabilité autour de l'évitabilité de motifs.

On peut généraliser la définition des mots sans carrés de façon évidente : on dit qu'un mot est *sans puissance $n$*, $n \in \mathbb{N}$, s'il ne contient pas de facteur $u^n = \underbrace{uu\ldots u}_{n}$ pour un mot $u$ non-vide. En termes de motifs, il s'agit des mots de $Cont(x^n)$. Comme l'exposant d'un mot peut prendre une valeur rationnelle quelconque, on peut généraliser cette définition encore plus en l'étendant aux puissances rationnelles et même réelles. Pour un nombre $x \in \mathbb{R}$, on dira qu'un mot est sans puissance $x$ s'il ne contient pas comme facteur une répétition d'exposant au moins égal à $x$.

Considérer les répétitions *fractionnaires* (celles dont l'exposant est un nombre rationnel quelconque), et non pas uniquement les répétitions entières, est une généralisation souvent très utile permettant d'obtenir une compréhension plus fine de propriétés combinatoires de mots [Mignosi et Pirillo, 1992; Mignosi *et al.*, 1995; Choffrut et Karhumäki, 1997; Justin et Pirillo, 1999]. En particulier, l'étude des mots évitant les répétitions d'exposant réel a déjà été entreprise dans la littérature. Dans [Dejean, 1972], F. Dejean a obtenu un renforcement d'un des résultats de Thue, en démontrant que l'on peut construire un mot infini sur trois lettres qui non seulement ne contient pas de facteur d'exposant 2, mais ne contient pas non plus d'exposant strictement supérieur à $7/4$ ; de plus, cette borne a été prouvée la meilleure possible. Une autre formulation de ce résultat est la suivante : il existe un mot infini sur trois lettres qui ne contient pas deux occurrences d'un facteur $u$ non vide séparées par moins de $|u|/3$ lettres. Des généralisations de ce résultat pour des alphabets plus grands ont été obtenues (voir [Berstel, 1992] pour plus de références).

Les résultats que nous allons présenter dans la section suivante sont dans le même esprit, car ils présentent certains « cas limites » dans les propriétés de mots sans répétitions.

### 2.4.2 Mots sans puissance $x$ de densité minimale

#### Densité minimale d'une lettre : définition abstraite et propriétés

Le problème de cette section peut être motivé comme suit. Supposons qu'un motif (ou un ensemble de motifs) soit $k$-évitable et ne soit pas $(k-1)$-évitable. Quel est le taux minimal limite de la $k$-ème lettre, permettant de construire un mot infini évitant le motif ? Avant d'aborder cette question, nous devons d'abord définir la notion de *taux minimal limite* d'une lettre, que nous appelons également la *densité minimale*. Pour ce faire, nous allons nous abstraire de la propriété des mots en question (à savoir, de ne pas contenir de répétitions), et nous allons définir la notion de la densité minimale dans un cadre plus général.

Soit $\mathcal{H} \subseteq A^*$ un ensemble (fini ou infini) de mots. Nous allons nous intéresser à l'ensemble des mots qui évitent (ne contiennent pas comme facteurs) les mots de $\mathcal{H}$. On notera cet ensemble $\overline{Cont(\mathcal{H})}$, en accord avec la notation utilisée auparavant. Il est évident que l'ensemble $\overline{Cont(\mathcal{H})}$ est factoriel, c'est-à-dire que s'il contient un mot, il contient également tous ces facteurs. Inversement, tout ensemble factoriel $\mathcal{F} \subseteq A^*$ peut être représenté comme $\overline{Cont(\mathcal{H})}$ pour $\mathcal{H} = A^* \setminus \mathcal{F}$. Par conséquent, la propriété d'être factoriel peut être considérée comme une caractérisation des ensembles des mots évitant un ensemble de facteurs.

Considérons un ensemble factoriel $\mathcal{F} \subseteq A^*$ et supposons qu'il est *infini*. En vertu du lemme de König, il existe un mot infini de $A^\omega$ dont chaque facteur fini appartient à $\mathcal{F}$. On note $\mathcal{F}^\omega$ l'ensemble des tels mots infinis. Pour une lettre $a \in A$ donnée, nous souhaitons définir une proportion minimale limite des lettres $a$ dans les mots de $\mathcal{F}^\omega$. Pour $w \in \mathcal{F}$, on note $c_a(w)$ le nombre d'occurrences de $a$ dans $w$ et $\rho_a(w) = \frac{c(w)}{|w|}$. Soit $\mathcal{F}(l) = \{w \in \mathcal{F} \mid |w| = l\}$. Pour tout $l \in \mathbb{N}$, soit $\rho_a(\mathcal{F}, l) = \frac{1}{l} \min_{w \in \mathcal{F}(l)} c_a(w)$ et $\rho_a(\mathcal{F}) = \underline{\lim}_{l \to \infty} \rho_a(\mathcal{F}, l)$. On appelle $\rho_a(\mathcal{F})$ la *densité minimale* de $a$ dans $\mathcal{F}$.

On peut prouver que pour tout $l \in \mathbb{N}$, $\rho_a(\mathcal{F}, l) \leq \rho_a(\mathcal{F})$, ce qui implique que la limite inférieure dans la définition de $\rho_a(\mathcal{F})$ peut être remplacée par la limite simple : $\rho_a(\mathcal{F}) = \lim_{l \to \infty} \rho_a(\mathcal{F}, l) = \sup_{l \geq 1} \rho_a(\mathcal{F}, l)$. Cela suggère que la définition $\rho_a(\mathcal{F}) = \lim_{l \to \infty} \min_{w \in \mathcal{F}(l)} \frac{c_a(w)}{l}$ est correcte et définit bien la quantité à laquelle on s'intéresse. Cela est effectivement le cas, car de plus, d'autres façons naturelles de définir cette quantité lui sont équivalentes. Par exemple, pour un mot $v \in \mathcal{F}^\omega$ on peut considérer la limite $\lim_{j \to \infty} \rho_a(v[1..j])$. Cette limite n'existe pas pour tout mot $v$, cependant on peut démontrer que parmi tous les mots pour lesquels cette limite existe, il en existe

un qui réalise le minimum de toutes ces limites, qui de plus est égal à $\rho_a(\mathcal{F})$. Cette équivalence montre que $\rho_a(\mathcal{F})$ est bien la bonne quantité à étudier.

### Mots sans puissance $x$ de densité minimale

Pour cette section nous fixons l'alphabet binaire $\{0,1\}$. Soit $\mathcal{R}(x)$, pour $x \in \mathbb{R}$, $x \geq 2$, l'ensemble des répétitions $r \in \{0,1\}^*$ tels que $e(r) \geq x$, et $\mathcal{R}(x+\varepsilon)$, pour $x \geq 2$, l'ensemble des répétitions $r \in \{0,1\}^*$ tels que $e(r) > x$. Nous allons nous intéresser aux ensembles $\overline{Cont(\mathcal{R}(x))}$ et $\overline{Cont(\mathcal{R}(x+\varepsilon))}$ – les mots binaires qui ne contiennent pas comme facteur les répétitions d'exposant respectivement supérieur ou égal ou strictement supérieur à $x$. Dans cette section nous allons étudier les valeurs des fonctions $\rho_1(\overline{Cont(\mathcal{R}(x))})$ et $\rho_1(\overline{Cont(\mathcal{R}(x+\varepsilon))})$ (considérées comme fonctions en $x$), que l'on notera respectivement $\rho(x)$ et $\rho(x+\varepsilon)$.

Intuitivement, $\rho(x)$ (respectivement $\rho(x+\varepsilon)$) est la proportion minimale d'une lettre que l'on peut avoir dans les mots binaires évitant les répétitions d'un exposant au moins égal à (respectivement, strictement plus grand que) $x$. On peut démontrer que $\rho(x+\varepsilon)$ coïncide avec la limite à droite de $\rho(x)$. Comme $\mathcal{R}(x) \subseteq \mathcal{R}(y)$ pour $x \leq y$, les fonctions $\rho(x)$ et $\rho(x+\varepsilon)$ sont décroissantes. Trivialement, $0 < \rho(x), \rho(x+\varepsilon) \leq \frac{1}{2}$.

L'ensemble $\overline{Cont(\mathcal{R}(2+\varepsilon))}$ contient exactement les mots sans chevauchement (voir la section 2.4.1) dont les propriétés ont été étudiées dans [Restivo et Salemi, 1983; Kfoury, 1988]. Les résultats de ces articles impliquent, en particulier, que $\rho(2+\varepsilon) = \frac{1}{2}$. Nous avons établi un résultat plus fort :

**Lemme 2 ([Kolpakov et Kucherov, 1997])** *$\rho(x) = \frac{1}{2}$ pour tout $x \in (2, \frac{7}{3}]$.*

Cela veut dire que les mots binaires qui évitent les répétitions d'exposant jusqu'à $\frac{7}{3}$ doivent contenir (asymptotiquement) la même proportion de chacune des deux lettres. De plus, $\frac{7}{3}$ est la « dernière » valeur avec cette propriété, car nous avons démontré dans [Kolpakov et Kucherov, 1997] que

$$\rho\left(\frac{7}{3}+\varepsilon\right) \leq \frac{10}{21}. \tag{2.25}$$

Autrement dit, un mot (infini) qui évite les répétitions d'exposant *strictement* supérieur à $\frac{7}{3}$ peut être « déséquilibré » en ce sens qu'une lettre peut apparaître avec une proportion strictement inférieure à l'autre. Avec le lemme 2 cela implique que $\rho(x)$ est discontinue en $\frac{7}{3}$. Dans la lignée de ce résultat de discontinuité, nous avons pu démontrer que $\rho(x)$ possède en fait une infinité de points de discontinuité. Plus spécifiquement, nous avons étudié les valeurs $\rho(n)$ aux points entiers $n \in \mathbb{N}$ et nous avons obtenu des estimations asymptotiques pour $\rho(n)$ et $\rho(n+\varepsilon)$. Le théorème suivant résume les résultats obtenus.

**Théorème 8 ([Kolpakov *et al.*, 1998; Kolpakov *et al.*, 1999])**
*(i)* $\rho(n) = \frac{1}{n} + \frac{1}{n^3} + \frac{1}{n^4} + \mathcal{O}(\frac{1}{n^5})$,
*(ii)* $\rho(n+\varepsilon) = \frac{1}{n} - \frac{1}{n^2} + \frac{2}{n^3} - \frac{2}{n^4} + \mathcal{O}(\frac{1}{n^5})$,
*(iii) pour tout $n \geq 3$, $\rho(n)$ est discontinue à droite, car $\rho(n) > \rho(n+\varepsilon)$.*

La preuve de ces résultats utilise la technique de mots engendrés par morphismes (DOL-systèmes). En particulier, la preuve de l'inéquation (2.25) utilise un morphisme de 21 lettres, construit à l'aide de l'ordinateur.

À notre connaissance, la notion de densité minimale n'avait pas été étudiée auparavant et donc ces résultats sont les premiers dans cette direction. Certains travaux ont été faits sur la fréquence de facteurs dans les mots engendrés par les DOL-systèmes [Dekking, 1992] et sur la

*fonction de récurrence* [Rauzy, 1983; Shallit et Breitbart, 1994; Allouche et Bousquet-Mélou, 1995] caractérisant, d'une façon uniforme, la fréquence des facteurs. Le théorème 8 montre (ou plutôt confirme) que la structure et les propriétés combinatoires des mots sans répétitions est très complexe. En particulier, la discontinuité de la fonction $\rho(x)$ en certains points est surprenante et difficile à interpréter[3]. Les résultats ne dévoilent que très partiellement la nature et les propriétés de la fonction $\rho(x)$. En particulier, ils laissent supposer que $\rho(x)$ pourrait être constante par morceaux, ce qui cependant reste une conjecture. Une des conséquences intéressantes (et, à notre connaissance, nouvelle) du théorème 8 est que pour tout $n \geq 3$, il existe des mots arbitrairement longs n'appartenant pas à $\overline{Cont(\mathcal{R}(n))}$ mais appartenant à $\overline{Cont(\mathcal{R}(n+1))}$.

### 2.4.3 Répétitions maximales

Si la section 2.4.2 est consacrée à l'étude des mots ne contenant pas de répétitions, dans cette section nous nous intéressons, au contraire, aux propriétés des apparitions de répétitions dans les mots. La question centrale de notre étude ici sera la suivante : *combien de répétitions un mot de longueur $n$ peut-il contenir?* À part son aspect combinatoire, cette question s'avère avoir des applications importantes dans l'analyse d'algorithmes, comme on le verra par la suite dans la section 2.5.2.

Afin d'aborder cette question, on doit d'abord préciser le type des répétitions que l'on souhaite compter, et pour cela il convient d'introduire de la terminologie supplémentaire.

Rappelons que l'on appelle répétition tout mot d'un exposant $\geq 2$. Les mots d'un exposant entier pair, appelés *carrés*, sont précisément ceux qui peuvent s'écrire sous la forme $uu$ pour un mot $u$ non-vide. De même, les mots dont l'exposant est un entier multiple de 3, sont appelés *cubes*. Généralement, un mot dont l'exposant est entier supérieur ou égal à 2 est appelé une *répétition entière*, dans le cas contraire il est dit mot *primitif*. On dit qu'un mot est un *carré à racine primitive* si son exposant est exactement 2, auquel cas il s'écrit $uu$, où $u$ est un mot primitif. Plus généralement, pour un exposant $k$ donné, une répétition entière d'un exposant multiple de $k$ est appelé une *$k$-répétition*. Si, de plus, son exposant vaut exactement $k$, on l'appelle une *$k$-répétition à racine primitive*[4].

#### Dénombrement de répétitions entières

Les répétitions entières étant des objets très naturels, de nombreuses études leur ont été consacrées. Quant aux occurrences de répétitions entières, on peut facilement constater que dans le mot $a^n$ tout facteur de longueur au moins 2 est une répétition, ce qui réalise évidemment le nombre maximum de répétitions dans un mot de longueur $n$. Nous voulons « coder » l'ensemble de toutes les répétitions par une structure plus compacte dont la taille dans le cas le pire est la plus petite possible. Une façon naturelle d'aborder ce problème est de restreindre les répétitions en considération tout en préservant la « complétude » de la représentation, qui se traduit par la possibilité de pouvoir reconstruire toutes les répétitions présentes dans le mot à partir de cette représentation. Une première idée serait de considérer les $k$-répétitions ($k$ fixé) à racine primitive. Par exemple, $a^n$ ne contient que $(n-1)$ carrés primitifs (facteur $aa$ commençant aux positions $1, \ldots, n-1$) alors qu'il contient environs $n^2/4$ carrés (tous les facteurs de longueur paire).

3. Parmi les résultats de ce type, notons celui de J. Cassaigne [Cassaigne, 1993] qui démontre que le nombre asymptotique de mots sans chevauchement ne converge pas vers un quelconque polynôme.

4. Il faut bien noter que ce terme n'a de sens que si $k$ est fixé, car toute répétition entière peut s'écrire comme $u^m$ où $u$ est un mot primitif, et de ce fait peut être légitimement appelé une répétition à racine primitive.

Il est connu qu'un mot de longueur $n$ contient $O(n \log n)$ carrés à racine primitive. Cela est une conséquence, en particulier, du lemme 10 de l'article [Crochemore et Rytter, 1995] qui affirme qu'un mot de longueur $n$ ne peut contenir parmi ses préfixes qu'au plus $\log_\phi n$ carrés à racine primitive ($\phi = 1,618...$ est le nombre d'or). Cela implique immédiatement la borne supérieure $n \log_\phi n$ pour le nombre de carrés à racine primitive dans un mot de longueur $n$. D'autre part, on sait que la borne asymptotique $O(n \log n)$ est exacte, car il existe des familles de mots qui la réalisent. Les mots de Fibonacci en sont un exemple.

**Mots de Fibonacci – définition et quelques propriétés**

Les mots de Fibonacci sont des mots sur l'alphabet binaire $\{0, 1\}$ définis par la récurrence $f_0 = 0$, $f_1 = 1$, $f_k = f_{k-1} f_{k-2}$ pour $k \geq 2$. La longueur du $k$-ème mot de Fibonacci est le $k$-ème nombre de Fibonacci que l'on notera $F_k$. Les mots de Fibonacci ont de nombreuses propriétés combinatoires intéressantes et servent très souvent de tests dans des conjectures ou des algorithmes sur les mots, car ils fournissent souvent des exemples de « pire des cas » (voir [Iliopoulos *et al.*, 1997]). Vu l'importance de cette famille de mots par rapport aux problèmes qui nous intéressent ici, ainsi que dans la section 2.5.2, nous allons maintenant présenter certaines propriétés des mots de Fibonacci.

En termes informels, les mot de Fibonacci contiennent beaucoup de répétitions qui sont toutes de petit exposant. [Mignosi et Pirillo, 1992] ont démontré que les mots de Fibonacci ne contiennent pas de répétition d'exposant supérieur à $2 + \phi$ mais contiennent cependant des répétitions d'exposant supérieur à $2 + \phi - \varepsilon$ pour tout $\varepsilon > 0$. Cela implique en particulier que les mots de Fibonacci ne contiennent des répétitions entières que d'exposants 2 ou 3. On peut aussi dire que tout carré ou cube dans les mots de Fibonacci est à racine primitive.

Il est connu depuis longtemps que le nombre de carrés dans le mot de Fibonacci $f_k$ est $\Theta(F_k \log F_k)$ (voir [Crochemore, 1981]). La formule exacte pour ce nombre a été récemment obtenue dans [Fraenkel et Simpson, 1999] ; l'asymptotique de cette formule est

$$\frac{2}{5}(3 - \varphi)kF_k + O(F_k) \approx 0.7962 \cdot F_k \log F_k + O(F_k). \tag{2.26}$$

Remarquons que dans cette section nous nous intéressons au nombre d'*occurrences* de répétitions dans les mots et non pas au nombre de répétitions syntaxiquement distinctes. Cette différence peut être importante : tandis que les mots de Fibonacci contiennent $O(n \log n)$ occurrences de carrés ($n$ la longueur du mot), le nombre de *carrés distincts* n'est que $O(n)$. Plus précisément, A. Fraenkel et J. Simpson démontrent que le nombre de carrés distincts dans $f_k$ est

$$2(F_{k-2} - 1) = 2(2 - \varphi)F_k + O(1). \tag{2.27}$$

De même que pour les carrés primitifs, les mots de Fibonacci réalisent également, modulo une constante multiplicative, le maximum de carrés distincts parmi tous les mots, car [Fraenkel et Simpson, 1998] ont par ailleurs démontré que le nombre de carrés distincts dans les mots généraux de longueur $n$ (sur un alphabet arbitraire) est borné par une fonction linéaire de $n$.

**Répétitions entières non-extensibles**

Revenons maintenant à notre problème de comptage de répétitions dans les mots généraux. Une autre façon de restreindre la notion de répétition consiste à compter les répétitions entières *non-extensibles*, c'est-à-dire les facteurs d'un mot $w$ qui ont la forme $u^n$ ($u$ un mot primitif) et ne sont pas suivies ni précédées par une autre occurrence de $u$. Intuitivement, si un facteur d'un

mot est une répétition $u^n$ qui se prolonge à droite ou à gauche d'une autre occurrence de $u$, cette dernière est forcement incluse dans la répétition. Il est clair que le nombre des répétitions entières non-extensibles est inférieur à celui des carrés primitifs, car chaque carré primitif $uu$ fait partie d'une et une seule répétition entière non-extensible $u^n$ obtenue par son extension maximale à droite et à gauche. D'autre part, chaque répétition entière non-extensible $u^n$ ($u$ primitif) est associée avec $(n-1)$ carrés primitifs $uu$. Par conséquent, l'ensemble des répétitions entières non-extensibles est une représentation plus succincte de toutes les répétitions du mot que l'ensemble des carrés primitifs. Par exemple, le mot $a^n$ ne contient qu'une seule répétition entière non-extensible qui est le mot lui-même.

**Exemple 4** *Le mot* 10101101101 *contient 8 carrés primitifs (de gauche à droite :* $(10)^2$, $(01)^2$, $(101)^2$, $(011)^2$, $1^2$, $(110)^2$, $(101)^2$, $1^2$*) alors qu'il ne contient que 7 répétitions entières non-extensibles car deux carrés* $(101)^2$ *forment en effet un cube* $(101)^3$.

Cependant, la brève présentation ci-dessus des propriétés des mots de Fibonacci implique immédiatement que, de même que dans le cas des carrés primitifs, les mots de Fibonacci contiennent $\Theta(n \log n)$ répétitions entières non-extensibles. En effet, comme les mots de Fibonacci ne contiennent pas de répétition entières d'exposant 4 ou plus, et comme chaque cube primitif est associé à deux carrés primitifs, le nombre des répétitions entières non-extensibles est au moins la moitié de celui des carrés primitifs, et il est donc $\Theta(n \log n)$. Cela avait été observé par M. Crochemore dans l'article [Crochemore, 1981], où les répétitions entières non-extensibles ont été étudiées. Pour conclure, l'ensemble des répétitions entières non-extensibles n'est, dans le pire des cas, pas une représentation significativement plus compacte de l'ensemble des répétitions par rapport aux carrés primitifs.

### Répétitions maximales – définition

Pour représenter l'ensemble des répétitions de façon compacte, nous choisissons, tout d'abord, de considérer les répétitions *fractionnaires*. Comme cela a déjà été mentionné dans la section 2.4.1, le « passage » aux répétitions fractionnaires permet très souvent d'obtenir une compréhension plus profonde de propriétés combinatoires. Le résultat de Dejean mentionné dans la section 2.4.1 ainsi que nos résultats sur la fonction de densité minimale, présentés dans la section 2.4.2, en sont des exemples.

Cependant, il est évident que, pris isolement, ce changement de définition amène plutôt à un agrandissement de la représentation, car il y a bien sûr plus de répétitions fractionnaires dans un mot que de répétitions entières. L'idée est d'ajouter la condition de *maximalité* qui est en quelque sorte analogue à la condition de non-extensibilité pour les répétitions entières : une répétition (fractionnaire) est maximale si on ne peut pas l'étendre *d'une lettre* à gauche ou à droite sans que la période ne soit changée (augmentée). Voici la définition formelle :

**Définition 1** *Soit un facteur* $r = w[i..j]$ *d'un mot* $w$ *est une répétition, i.e.* $e(r) \geq 2$. *Le facteur* $r$ *est appelé une* répétition maximale *dans* $w$ *si*

*(i)* $p(w[i-1..j]) > p(w[i..j])$ *à condition que* $i > 1$, *et*

*(ii)* $p(w[i..j+1]) > p(w[i..j])$ *à condition que* $j < n$.

**Exemple 5** *Le mot* 10101101101 *considéré dans l'exemple 4 ne contient que 4 répétition maximales. Ce sont les répétitions* 10101 *(période 2, exposant* $5/2$*) et* 101101101 *(période 3, exposant 3), ainsi que deux carrés* 11. *Cet exemple illustre pourquoi les répétitions maximales constituent une représentation compacte : les carrés* $(10)^2$ *et* $(01)^2$*, par exemple, ne forment plus qu'une répétition* 10101. *Cela est naturel, car les deux ne sont que des « morceaux » d'une même périodicité.*

*Formellement, les deux carrés ne sont pas des répétitions maximales, car ils s'étendent en formant une seule répétition. De même, à l'intérieur du cube* $(101)^3$ *on ne distingue plus les carrés* $(011)^2$ *et* $(110)^2$ *comme des répétitions séparées.*

D'autre part, il est clair que l'ensemble des répétitions maximales caractérise toute la structure des répétitions du mot. Une fois cet ensemble obtenu, on peut facilement reconstruire les répétitions de tout autre type, comme les carrés (primitifs ou non), les $k$-répétitions, les répétitions entières non-extensibles, etc. (Nous allons d'ailleurs utiliser ce fait dans la section 2.5.2.)

Il nous faut maintenant estimer le nombre maximal de répétitions maximales dans un mot de longueur $n$, afin de vérifier si l'ensemble des répétitions maximales est bien, dans le cas le pire, plus petit que l'ensemble des répétitions entières. Une fois de plus, nous allons d'abord analyser cette question pour les mots de Fibonacci.

### Répétitions maximales dans les mots de Fibonacci : leur nombre et la somme des exposants

L'article [Iliopoulos *et al.*, 1997] démontre que le nombre de répétitions maximales dans le mot de Fibonacci $f_k$ est $O(F_k)$. Ce fait intéressant qui contraste avec le nombre $\Theta(F_k \log F_k)$ de répétitions entières, a été obtenu dans l'article d'une façon indirecte, à savoir à l'aide d'un algorithme qui trouve toutes les répétitions maximales dans $f_k$ en temps $O(F_k)$. Cela implique que le nombre des répétitions maximales (la taille de sortie de l'algorithme) est également linéaire en la longueur du mot. Nous avons calculé le nombre $R_k$ des répétitions maximales dans $f_k$ de façon exacte :

**Théorème 9 ([Kolpakov et Kucherov, 1999b])** *Pour tout* $k \geq 4$, $R_k = 2F_{k-2} - 3$.

Il est intéressant de remarquer que $R_k$ est 1 de moins que le nombre des carrés distincts dans $f_k$ (voir la formule (2.27)). Nous ne savons pas s'il s'agit d'une pure coïncidence ou si cela a une explication combinatoire.

Le théorème 9 affirme que le nombre de répétitions maximales dans les mots de Fibonacci est linéaire en la longueur (asymptotiquement $2(2-\phi)F_k \approx 0,764 \cdot F_k$). Comme les exposants des répétitions des mots de Fibonacci sont inférieurs à $(2+\phi)$, la somme des exposants de toutes les répétitions maximales est elle aussi bornée linéairement par

$$(2+\phi)(2F_{k-2} - 3) = 2(2-\phi)(2+\phi)F_k + O(1) = 2(3-\phi)F_k + O(1) \approx 2.764 \cdot F_k.$$

Nous avons établi une estimation plus précise. Soit $SR(k)$ la somme des exposants de toutes les répétitions maximales dans $f_k$.

**Théorème 10 ([Kolpakov et Kucherov, 1999b])** $SR(k) = C \cdot F_k + o(F_k)$, *où* $1.922 \leq C \leq 1.926$.

### Répétitions maximales dans les mots généraux

L'analyse des répétitions maximales des mots de Fibonacci, présentée dans la section précédente, conduit à conjecturer que le nombre de répétitions maximales, et même la somme de leurs exposants, reste linéaire dans les mots généraux. Cela confirmerait que l'ensemble des répétitions maximales est en effet une représentation plus compacte de toutes les répétitions du mot, par rapport à toute autre représentation (sous-classe de répétitions).

Soit $R(w)$ l'ensemble de toutes les répétitions maximales d'un mot $w$ (sur un alphabet arbitraire) et soit $Sexp(w) = \sum_{r \in R(w)} e(r)$, $Sexp(n) = \max_{|w|=n} Sexp(w)$. Nous avons démontré le résultat suivant.

**Théorème 11 ([Kolpakov et Kucherov, 1999a])** $Sexp(n) = O(n)$.

Un corollaire important du théorème 11 est que le nombre des répétitions maximales dans les mots généraux est borné linéairement en la longueur du mot. Soit $\#R(w)$ le nombre des répétitions maximales dans un mot $w$.

**Corollaire 1** $\max_{|w|=n} \#R(w) = O(n)$.

La preuve du théorème 11, présentée en entier dans [Kolpakov et Kucherov, 2000b], est très technique et ne permet pas en pratique de donner une valeur à la constante multiplicative implicite dans la borne linéaire asymptotique. Cependant, nos expériences sur ordinateur montrent que cette constante est très petite. En particulier, selon ces expériences, le nombre maximal de répétitions maximales (Corollaire 1) est inférieur à $n$. Trouver une preuve simple du théorème 11 et/ou du corollaire 1 permettant de faire apparaître une petite constante multiplicative reste un problème intéressant.

Le théorème 11 et le corollaire 1 sont les premiers résultats de ce type. Le corollaire 1, en particulier, répond positivement à la question, posée dans l'article [Iliopoulos *et al.*, 1997], de savoir s'il existe une représentation linéaire de l'ensemble de toutes les répétitions et si l'ensemble des répétitions maximales forment une telle représentation. Le théorème 11 répond en outre positivement à la conjecture de Gusfield et Stoye [Stoye et Gusfield, 1998a] sur la linéarité du nombre de *carrés branchants* (*branching tandem repeats*). Dans notre terminologie, les carrés branchants sont des carrés (pas nécessairement à racine primitive) qui se trouvent en suffixe de répétitions maximales. Comme chaque répétition maximale $r$ contient $\lfloor e(r)/2 \rfloor$ carrés branchants, le théorème 11 implique que le nombre de ces derniers est linéaire.

En dehors de son intérêt combinatoire, le théorème 11 et le corollaire 1 ont des applications importantes dans l'analyse de certains algorithmes de recherche de motifs. Nous aborderons cet aspect dans la section 2.5.2.

## 2.5 Algorithmes de recherche de motifs dans les mots

Si le chapitre 2.3 portait sur des problèmes de complexité algorithmique élevée (typiquement indécidables ou NP-complets), et le chapitre 2.4 présentait des résultats combinatoires sans forcément un contenu algorithmique immédiat (typiquement des résultats de comptage ou d'estimation de valeurs limites), ce chapitre-ci est consacré aux problèmes « de basse complexité », pour lesquels il existe des algorithmes très efficaces travaillant en temps linéaire ou quasi-linéaire. La finalité de cette problématique est le développement d'algorithmes les plus efficaces possibles, souvent basés sur des structures de donnés ou des techniques algorithmiques avancées. Le développement de tels algorithmes et surtout leur analyse sont très souvent basés sur des propriétés combinatoires de structures sous-jacentes, qui s'appuient sur des méthodes mathématiques profondes. En particulier, les algorithmes qui sont présentés dans ce chapitre utilisent la théorie des langages formels et la théorie des automates, ainsi que des résultats profonds de combinatoire des mots, y compris ceux présentés dans le chapitre 2.4.

Les algorithmes de recherche de motifs sont, de toute évidence, importants pour les applications, dont la recherche sur le Web et l'analyse de séquences génomiques sont des exemples très d'actualité. Des monographies récentes [Crochemore et Rytter, 1994; Stephen, 1994; Gusfield, 1997] sont consacrées à ce domaine ; elles montrent la richesse et la beauté de ces algorithmes.

Dans ce chapitre nous présentons trois algorithmes de recherche de motifs que nous avons élaborés. Le premier (section 2.5.1) résout le problème de recherche d'un ensemble de motifs *avec espaces non-bornés*. Ce travail, commun avec Michaël Rusinowitch, est relié aux résultats de la section 2.3.3, et en particulier s'inspire de la construction de l'automate de [Kucherov et Rusinowitch, 1995a]. L'algorithme proposé a été présenté à la conférence *Combinatorial Pattern*

*Matching* en 1995 [Kucherov et Rusinowitch, 1995b] et a été ensuite publié dans *Theoretical Computer Science* en 1997 [Kucherov et Rusinowitch, 1997].

L'algorithme de la section 2.5.2 recherche, en temps linéaire, toutes les répétitions maximales dans un mot. Cet algorithme est une extension de l'algorithme de Main [Main, 1989] et son analyse repose sur les résultats combinatoires présentés dans la section 2.4.3. L'algorithme a été présenté à FOCS en 1999 [Kolpakov et Kucherov, 1999a] ; il est également décrit dans l'article [Kolpakov et Kucherov, 2000b] déjà mentionné dans la section 2.4.3. Les deux algorithmes, présentés dans les sections 2.5.1 et 2.5.2, ont donné lieu aux logiciels `grappe` et `mreps` respectivement.

Enfin, le troisième algorithme (section 2.5.3) résout le problème de recherche de répétitions à écartement fixe. Ce travail récent est décrit dans [Kolpakov et Kucherov, 2000a]. Les résultats des sections 2.5.2 et 2.5.3 ont été obtenus en collaboration avec Roman Kolpakov.

### 2.5.1 Recherche de motifs avec espaces non-bornés

#### Énoncé du problème

Le problème consiste à reconnaître, le plus efficacement possible, si un texte (un mot, une chaîne de caractères) contient un motif parmi un ensemble donné de motifs. Un motif est composé d'un certain nombre de mots appelés *mots-clefs*, séparés par un symbole spécial qui peut être remplacé par n'importe quel mot. Formellement, étant donné un alphabet $A$, un motif $p$ s'écrit $v_1\#v_2\#\ldots\#v_m$, où $v_1, v_2, \ldots, v_m \in A^*$ et $\# \notin A$. Ce motif apparaît dans un texte $t \in A^*$ si $t = u_0v_1u_1v_2u_2\ldots v_mu_m$ pour $u_0, u_1, u_2, \ldots, u_m \in A^*$. Le symbole $\#$ que l'on appellera l'*espace non-borné* (en anglais *unlimited wildcard* ou *variable-length don't care*) joue donc le même rôle que `*` dans le shell UNIX ou `.*` dans les expressions régulières de `grep`. Étant donné un ensemble fini de motifs $P = \{p_1, p_2, \ldots, p_n\}$ nous voulons, pour un texte donné $t$, déterminer rapidement si $t$ contient *un des motifs* de $P$.

Comme le notent les auteurs de [Fisher et Paterson, 1974] à propos du problème de recherche d'un ensemble de motifs avec des espaces non-bornés, « un bon algorithme pour ce problème aurait des applications pratiques évidentes ». On peut reformuler le problème comme celui de la recherche dans un texte d'une expression régulière du type $\cup_{i=1}^{n} A^*u_1^iA^*u_2^iA^*\ldots A^*u_{m_i}^iA^*$. Notons que le meilleur algorithme de recherche d'expressions régulières générales [Myers, 1992] s'exécute en temps $O(|t||P|/\log|t|)$ où $|t|$ et $|P|$ sont respectivement la taille du texte et de l'expression régulière. L'algorithme que nous présentons dans cette section montre que pour la classe d'expressions régulières que nous considérons, la recherche peut se faire en temps quasi-linéaire et même linéaire (à la fois par rapport à $|P|$ et à $|t|$).

Au lieu de présenter notre algorithme en détail, nous en donnons ici les idées et les propriétés principales en renvoyant le lecteur à l'article [Kucherov et Rusinowitch, 1997] pour une description complète.

#### DAWG et ses propriétés

Une première caractéristique de l'algorithme est qu'il fonctionne selon le principe d'un automate qui marche sur le texte $t$ en le lisant lettre par lettre *en ligne*. Cet automate stocke les mots-clefs (un mot-clef par motif) qui sont actuellement recherchés. Le rôle de l'automate est joué par la structure de données appelée DAWG (*Directed Acyclic Word Graph*).

Le DAWG est une structure de données qui sert à représenter un mot ou un ensemble de mots. Elle est apparentée à l'*arbre des suffixes*, structure bien connue et universellement utilisée

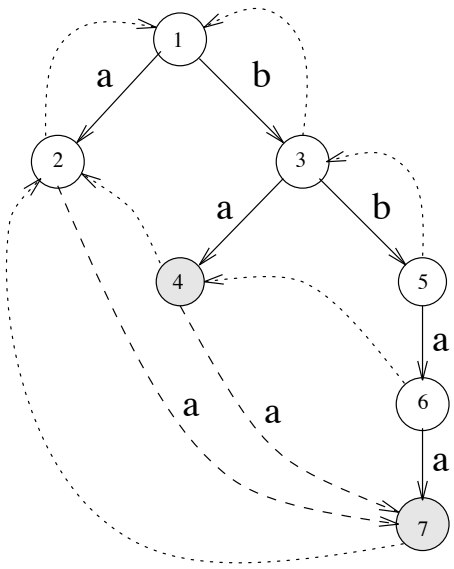


FIG. 2.1 – *DAWG pour l'ensemble* $P = \{ba, bbaa\}$

dans l'algorithmique des mots (voir [McCreight, 1976; Ukkonen, 1995; Gusfield, 1997])[5]. Une des propriétés remarquables du DAWG est qu'il admet une définition algébrique simple : dans le cas d'un mot, le DAWG n'est rien d'autre que l'automate minimal déterministe (non-complet) reconnaissant l'ensemble des suffixes du mot. Le DAWG d'un mot a été introduit et étudié dans [Blumer *et al.*, 1985] et [Crochemore, 1986] où il était appelé l'*automate de suffixes*. Le DAWG pour un ensemble de mots, introduit dans [Blumer *et al.*, 1987], peut être défini comme l'automate minimal déterministe reconnaissant les suffixes de cet ensemble de mots, mais qui de plus distingue les suffixes appartenants aux mots différents. Formellement, pour un ensemble $P$ de mots, le DAWG peut être obtenu en ajoutant le symbole distinct $\$_i$ à la fin de chaque mot $p_i \in P$, en construisant l'automate minimal déterministe pour l'ensemble de mots ainsi obtenu, et en omettant ensuite l'état final avec toutes les $\$_i$-transitions entrantes. Le DAWG pour l'ensemble $P = \{ba, bbaa\}$ est représenté sur la figure 2.1. (La différence entre les transitions continues et celle en petits traits n'est pas importante pour cette présentation. Les flèches vers le haut, montrées en pointillé, seront expliquées par la suite.)

Le DAWG peut être construit en temps linéaire en la taille de l'ensemble de mots $P$ en traitant les mots de $P$ un par un et en rajoutant chaque mot dans le DAWG construit pour les mots précédents. On appellera ce processus le *chargement* de mots dans le DAWG. Le chargement d'un mot prend un temps linéaire en la longueur du mot et se fait en lisant les lettres du mot en ligne une par une. Qui plus est, nous avons montré qu'un mot présent dans le DAWG peut être *déchargé* du DAWG, toujours en temps linéaire, en défaisant les modifications qui auraient été faites lors de son chargement. Comme le DAWG est unique, l'ordre d'une suite de chargements et déchargements de mots n'a pas d'importance.

Une autre propriété remarquable du DAWG, mise en évidence par M. Crochemore [Croche-

5. La relation entre l'arbre des suffixes et le DAWG est décrite dans [Blumer *et al.*, 1987; Crochemore et Rytter, 1994]. Notons au passage que la domination de l'arbre des suffixes dans l'algorithmique des mots n'est, à notre avis, pas justifié. La plupart des applications de l'arbre des suffixes (voir [Gusfield, 1997]) peuvent être résolues avec le DAWG, certaines même plus élégamment ou plus simplement. L'algorithme présenté dans cette section en est une illustration.

more, 1988], est qu'il peut être utilisé comme un automate de recherche des mots qui y sont stockés, de façon similaire à l'automate de Aho-Corasick par exemple. Les *liens de suffixe* du DAWG, représentés en pointillé sur la Figure 2.1, servent de *transitions d'échec* (*failure transitions*) suivant la terminologie des algorithmes de Knuth-Morris-Pratt ou d'Aho-Corasick. À la différence de ces algorithmes, en plus de recalculer l'état courant de l'automate à la lecture de chaque lettre du texte, il faut remettre à jour un compteur. Dans le cas du DAWG d'un mot, ce mot est reconnu si l'état courant est l'état terminal du DAWG (c'est-à-dire l'état qui correspond au mot entier) et si de plus la valeur du compteur est égale à la longueur du mot. La procédure de recherche d'un mot reste donc très analogue à l'algorithme d'Aho-Corasick, d'une complexité linéaire par rapport à la longueur du texte. Dans le cas du DAWG associé à plusieurs mots, la situation est plus complexe, car un mot de l'ensemble est reconnu s'il se trouve sur la *chaîne de liens de suffixe* issue de l'état courant. Sur l'exemple de la figure 2.1, le mot *ba* peut être reconnu au moment où l'algorithme se trouve dans l'état 6, car l'état 4 (l'état terminal pour le mot *ba*) se trouve sur la chaîne de liens de suffixe issue de l'état 6. Cela nécessite de retrouver, à la lecture de chaque lettre du texte, les états terminaux éventuels se trouvant sur la chaîne de liens de suffixe issue de l'état courant.

**Recherche de dictionnaires dynamiques**

Les bonnes propriétés du DAWG mentionnées plus haut fournissent une solution au problème de la *recherche de dictionnaires dynamiques* (*dynamic dictionary matching*). Ce problème consiste à réaliser efficacement la recherche d'un ensemble de motifs, appelé dictionnaire, ainsi que les opérations d'ajout et de suppression de motifs du dictionnaire. La difficulté (et l'intérêt) de ce problème s'explique par le fait que l'ajout d'un nouveau motif dans l'automate d'Aho-Corasick peut nécessiter la reconstruction de *tout* l'automate et dépend donc de la taille de *tout* le dictionnaire. Le but consiste donc à réduire la complexité d'ajout/suppression de motifs, si nécessaire au prix d'une légère augmentation du temps de recherche. La solution à ce problème proposée dans [Amir et Farach, 1991; Amir *et al.*, 1994] est basée sur l'arbre des suffixes. Selon elle, l'ajout/suppression d'un motif $p$ se fait en temps $O(|p| \log |P|)$ et la recherche dans un texte $t$ en temps $O(|t| \log |P|)$, où $|P|$ est la taille du dictionnaire. Les mêmes complexités ont été obtenues dans l'article [Idury et Schäffer, 1994] avec l'automate d'Aho-Corasick. Une combinaison assez complexe de ces approches [Amir *et al.*, 1993] a permis d'améliorer légèrement les complexités : $O(|p| \log |P| / \log\log |P|)$ et $O(|t| \log |P| / \log\log |P|)$ respectivement pour l'ajout/suppression et la recherche.

Il est important de noter que le « goulot d'étranglement » dans ces solutions, qui entraîne les facteurs poly-logarithmiques dans les bornes de complexité, réside dans un sous-problème indépendant, appelé le problème de la recherche de l'ancêtre commun le plus proche (*lowest common ancestor*) de deux noeuds d'un arbre qui change dynamiquement. Plusieurs versions de ce problème ont été étudiées dans la littérature [Harel et Tarjan, 1984; Schieber et Vishkin, 1988; Wen, 1994]. La solution utilisée par [Amir *et al.*, 1994] est basée sur la méthode d'*arbres dynamiques* (*dynamic trees*) de Sleator et Tarjan [Sleator et Tarjan, 1983] qui permet de réaliser plusieurs types d'opérations dans les arbres dynamiques en temps logarithmique (d'où les facteurs logarithmiques dans les bornes de complexité de [Amir *et al.*, 1994]).

La structure de DAWG se prêtant très bien à l'ajout/suppression de motifs qu'elle stocke, ainsi qu'à la recherche de ces motifs, elle fournit une nouvelle solution au problème de la recherche de dictionnaires dynamiques. Cependant, la question, mentionnée à la fin de la section précédente, d'identification des états terminaux sur la chaîne des liens de suffixe pose problème, car cette information ne peut pas être associée « statiquement » à chaque état du DAWG à cause de sa

nature dynamique.

Il est intéressant de noter que bien que la structure de données soit différente, le problème rencontré revient, dans sa forme abstraite, au problème de la recherche de l'ancêtre commun le plus proche, rencontré par les approches [Amir *et al.*, 1994; Idury et Schäffer, 1994][6]. Ainsi, en empruntant la méthode des arbres dynamiques, nous obtenons une autre solution au problème de la recherche de dictionnaires dynamiques, avec les mêmes bornes de complexité que celles de [Amir *et al.*, 1994]. Une amélioration analogue à celle de [Amir *et al.*, 1993] peut être également intégrée naturellement dans l'approche utilisant le DAWG.

Un progrès considérable dans le problème de la recherche de l'ancêtre commun le plus proche a été réalisé récemment dans l'article [Cole et Hariharan, 1999]. Il démontre que pour la version du problème qui nous intéresse, les opération de modification de l'arbre ainsi que l'opération même de recherche de l'ancêtre commun le plus proche peuvent toutes être réalisées *en temps constant*. En intégrant cette solution dans une des méthodes pour le problème de la recherche de dictionnaires dynamiques, on obtient une solution complètement linéaire à ce dernier : ajout/suppression d'un motif $p$ en temps $O(|p|)$ et recherche d'un ensemble de motifs dans un texte $t$ en temps $O(|t|)$.

### Algorithme de recherche de motifs avec espaces non-bornés

Revenons maintenant au problème initial de la recherche de motifs avec espaces non-bornés. Soit $P = \{p_1, \ldots, p_n\}$ un ensemble de motifs, où $p_i = v_1^i \# v_2^i \# \ldots \# v_{m_i}^i$. Rappelons que le but est de tester efficacement si un texte donné contient un des motifs $p_i$. L'idée générale de l'algorithme est la suivante. L'algorithme commence par construire le DAWG pour les premiers mots-clefs de chaque motif $\{v_1^1, \ldots, v_1^n\}$ et par les rechercher dans le texte en utilisant le DAWG comme automate de recherche. Une fois qu'un mot-clef $v_1^i$ est trouvé, il est déchargé du DAWG et le mot suivant $v_2^i$ y est chargé. D'une façon générale, à chaque instant l'algorithme recherche un mot-clef de chaque motif. Après avoir trouvé un mot parmi les mots actuellement dans le DAWG, il est déchargé du DAWG et le mot suivant du même motif y est chargé.

Cette description montre que le problème en question peut être résolu avec les mêmes techniques que le problème de la recherche de dictionnaires dynamiques. Certes, il y a quelques difficultés supplémentaires qu'il faut surmonter. Par exemple, lorsqu'un mot $v_j^i$ est trouvé et que l'on commence à rechercher le mot suivant $v_{j+1}^i$, il faut s'assurer qu'il ne peut être reconnu qu'après $|v_{j+1}^i|$ lettres du texte (s'il était reconnu avant, cela signifierait un chevauchement de l'occurrence trouvée avec celle de $v_j^i$, ce qui n'est pas autorisé par l'énoncé du problème). Cette question est traitée, de façon élégante, en « étalant dans le temps » le chargement de mots-clefs. Plus précisement, nous tenons à jour une liste des mots-clefs en cours de chargement et nous chargons une lettre de chaque mot-clef de la liste à la lecture d'une lettre du texte. Cela permet de synchroniser naturellement la recherche du mot avec l'intervalle de ses occurrences possibles.

L'algorithme résultant est représenté dans la figure 2.2. À la lecture de chaque lettre du texte, son travail consiste en trois étapes. La première est l'étape de « synchronisation » destinée à ne maintenir dans le DAWG que les prefixes de mots-clefs qui peuvent potentiellement être reconnus à cet endroit du texte (voir le paragraphe précédent). La deuxème étape est la mise à jour de l'état courant du DAWG et du compteur (voir la section sur les propriétés du DAWG). Enfin, la troisième étape consiste à identifier des mots-clefs reconnus et de les decharger. Nous renvoyons le lecteur à l'article original [Kucherov et Rusinowitch, 1997] pour la description détaillée de l'algorithme, ainsi que pour la preuve de correction et l'analyse de complexité. Notons que l'algorithme représente un phénomène intéressant et nouveau : contrairement aux algorithmes traditionnels

6. Cela montre un lien algorithmique profond entre ces deux problèmes, apparemment de nature différente.

ENTRÉE : texte $t$ et un ensemble de motifs $P = \{p_1, \ldots, p_n\}$
SORTIE : une occurrence d'un des motifs $p_1, \ldots, p_n$ s'il y en a une

`Pour chaque lettre de` $t$ `faire :`

ÉTAPE 1 :
`Charger la lettre suivante de chaque mot-clef` $v_j^i$ `en cours de chargement. Si toutes les lettres de ce mot-clef ont été chargées, l'enlever de la liste des mots-clefs en cours de chargement et créer un noeud terminal correspondant.`

ÉTAPE 2 :
`Remettre à jour le noeud (état) courant et le compteur.`

ÉTAPE 3 :
`Rechercher les noeuds terminaux sur la chaîne des liens de suffixe du noeud courant. Pour chaque occurrence trouvée, décharger le mot-clef correspondant. Si c'était le dernier mot-clef du motif, signaler l'occurrence de ce motif et arrêter l'execution, sinon remettre le mot-clef suivant du motif dans la liste des mots-clefs en cours de chargement.`

`signaler l'absence d'occurrences de motifs de` $P$

FIG. 2.2 – *Algorithme de recherche de motifs avec espaces non-bornés*

de recherche de motifs, l'automate sous-jacent change dynamiquement lors du parcours du texte en s'adaptant à la situation de recherche.[7]

Pour conclure, nous indiquons la complexité de l'algorithme compte-tenu de la dernière avancée [Cole et Hariharan, 1999] dans le problème de l'ancêtre commun le plus proche.

**Théorème 12** *Le problème de la recherche de motifs avec espaces non-bornés peut être résolu en temps* $O(|t| + |P|)$ *où* $|t|$ *et* $|P|$ *sont respectivement la taille du texte et la taille des motifs.*

### Logiciel `grappe`

L'algorithme proposé dans l'article [Kucherov et Rusinowitch, 1997] a été implanté dans un logiciel nommé `grappe`. Une première version de ce logiciel, `grappe-1`, avait été réalisée dans les années 1996-97. Le logiciel s'est avéré compétitif par rapport aux logiciels existants tels que `agrep` et `egrep`. Contrairement à ces derniers, `grappe` n'a pas de limitation sur la taille des motifs à rechercher et se montre particulièrement efficace lorsqu'un grand nombre de motifs sont recherchés en parallèle.

En 1999-2000, `grappe` a été étendu pour permettre le traitement d'*espaces bornés*, c'est-à-dire les distances entre les occurrences de mots-clefs spécifiées par des intervalles de nombres entiers. Cela rend `grappe` plus adapté, en particulier, aux applications à la recherche de motifs dans les séquences génomiques. De nombreuses options ont également été ajoutées, qui servent à

7. Notons au passage une implantation expérimentale destinée à visualiser cet automate dynamique [Bertault et Kucherov, 1998]. Il s'agit d'une interface du logiciel `grappe`, décrit dans la section suivante, avec le logiciel `Padnon` de tracé de graphes.

augmenter la convivialité du logiciel. La dernière version, `grappe-3`, est accessible sur le Web à l'adresse `http://www.loria.fr/~kucherov/SOFTWARE/grappe-3.0/`. Le logiciel `grappe`, déposé à l'APP, est également distribué avec le CD-ROM de logiciels libres de l'INRIA. `grappe-3` prévoit également une version spécialement adaptée au traitement de séquences d'ADN/ARN.

### 2.5.2 Recherche des répétitions maximales

Les résultats combinatoires sur les répétitions maximales présentés dans la section 2.4.3 ont eu des conséquences algorithmiques très intéressantes que nous décrivons dans cette section. Le corollaire 1 du théorème 11 affirme que l'ensemble des répétitions maximales fournit une représentation linéaire de toutes les répétitions du mot. Cela conduit à conjecturer que l'on peut trouver cet ensemble en temps linéaire. Nous proposons un algorithme confirmant cette conjecture.

#### Bref survol des travaux existants

La recherche de répétitions dans les mots a fait objet de nombreux travaux. Au début des années 80, A. Slissenko [Slisenko, 1983] propose un algorithme linéaire capable de trouver en temps réel toutes les répétitions *distinctes*. Indépendamment, M. Crochemore [Crochemore, 1983] propose un algorithme linéaire simple pour trouver un carré dans un mot, ce qui permet de vérifier si le mot est sans carré ou non. Un autre algorithme linéaire pour accomplir cette tâche a été proposé dans [Main et Lorentz, 1985].

Plusieurs algorithmes ont été proposés pour rechercher *toutes* les occurrences de répétitions dans un mot en temps $O(n \log n)$. Cependant, chacun de ces algorithmes traite un type de répétitions légèrement différent. L'algorithme proposé par M.Crochemore [Crochemore, 1981] trouve toutes les répétitions entières non-extensibles alors que celui de A. Apostolico et F. Preparata [Apostolico et Preparata, 1983] considère les répétitions fractionnaires *maximales à droite* (c'est-à-dire celles qui ne s'étendent pas à droite sans que la période ne soit augmentée). M. Main et R. Lorentz [Main et Lorentz, 1984] trouvent en temps $O(n \log n)$ *toutes* les répétitions maximales. Ils démontrent également dans cet article que $O(n \log n)$ est la borne optimale à condition que l'algorithme n'utilise, pour seule opération de base, que la comparaison de lettres. En 1989, M.Main propose un algorithme *linéaire* qui trouve toutes les répétitions maximales *distinctes*, ou plus précisément qui garantit de trouver l'occurrence la plus à gauche de chaque répétition maximale distincte.

En ce qui concerne d'autres travaux liés, R.Kosaraju [Kosaraju, 1994] propose un algorithme assez complexe qui trouve en temps $O(n)$ le carré le plus court commençant à chaque position du mot. Il affirme également, sans donner de démonstration, pouvoir trouver tous les carrés primitifs dans un mot en temps $O(n + S)$, où $S$ est le nombre de ces carrés. Très récemment (et après nos travaux que nous décrivons dans la section suivante) J. Stoye et D. Gusfield ont proposé dans [Stoye et Gusfield, 1998b] un algorithme permettant de trouver en temps $O(n)$ un représentant de chaque carré *distinct* dans le mot. Rappelons que le nombre de carrés distincts est linéaire selon le résultat de [Fraenkel et Simpson, 1998] mentionné dans la section 2.4.3.

Malgré tous ces travaux, restait un problème ouvert de savoir si toutes les occurrences de répétitions maximales pouvaient être trouvées en temps linéaire. M. Main [Main, 1989] conjecture qu'un tel algorithme pourrait exister. La même question est posée dans [Iliopoulos *et al.*, 1997]. Cependant, on ne savait pas si le nombre de répétitions maximales était borné linéairement ; il n'y avait donc pas d'argument de comptage supportant cette conjecture. Nos résultats de la section 2.4.3 fournissent cet argument.

ENTRÉE : mot $t$
SORTIE : toutes les répétitions maximales de $t$

ÉTAPE 1 :
`Calculer la s-factorisation du mot` $t$

ÉTAPE 2 :
`Trouver toutes les répétitions maximales traversant les frontières entre les s-facteurs à l'aide de l'algorithme de Main (cet ensemble contient l'occurrence la plus à gauche de chaque répétition maximale distincte).`

ÉTAPE 3 :
`Trouver toutes les répétitions maximales apparaissant à l'intérieur des s-facteurs`

FIG. 2.3 – *Algorithme de recherche des répétitions maximales*

**Notre algorithme**

L'algorithme recherchant toutes les répétitions maximales, que nous avons proposé dans [Kolpakov et Kucherov, 1999a], est une extension de l'algorithme de Main [Main, 1989]. Ce dernier est basé sur deux idées principales. La première est une factorisation spéciale du mot appelée la *s-factorisation* [Crochemore, 1981]. La s-factorisation $w = w_1 w_2 \dots w_m$ d'un mot $w$ est définie récursivement : le *s-facteur* $w_i$ est une lettre si cette lettre n'apparaît pas dans $w_1 \dots w_{i-1}$, sinon $w_i$ est le plus long facteur apparaissant au moins deux fois dans $w_1 \dots w_{i-1} w_i$.

**Exemple 6** *La s-factorisation du mot* 1011010110110 *est* 1|0|1|101|01101|10.

La s-factorisation se calcule en temps linéaire en utilisant une structure de données du type arbre des suffixes [Gusfield, 1997] ou DAWG [Blumer *et al.*, 1985; Crochemore, 1986] (voir la section 2.5.1).

La deuxième idée est fournie par les fonctions de Main et Lorentz [Main et Lorentz, 1984]. Ces fonctions, calculées également en temps linéaire, permettent d'extraire les répétitions qui traversent une position donnée. Étant donné un mot $w$, l'algorithme de Main calcule d'abord la s-factorisation $w = w_1 w_2 \dots w_m$ et ensuite, pour chaque s-facteur $w_i$, toutes les répétitions maximales qui traversent le début de $w_i$ et terminent à l'intérieur de $w_i$. Un lemme de [Main, 1989] montre que pour chaque facteur $w_i$ les fonctions de Main et Lorentz ne doivent être calculées que sur un mot de longueur $O(|w_{i-1}| + |w_i|)$. Cela fournit un argument clef pour établir que toute la tâche peut être accomplie en temps linéaire en la longueur du mot. D'autre part, d'après la définition de la s-factorisation, calculer toutes ces répétitions garantit de trouver toutes celles qui n'ont pas de copie à gauche. En effet, les répétitions qui n'ont potentiellement pas été trouvées sont celles qui sont situées entièrement à l'intérieur du s-facteur, et par conséquent ont une copie à gauche.

Il reste donc à identifier les répétitions se trouvant entièrement à l'intérieur des s-facteurs. Dans [Kolpakov et Kucherov, 1999a] nous avons montré que cela peut se faire avec des techniques algorithmiques classiques, telles que listes doublement chaînées, tri par cases (*basket sort*).

L'algorithme entier est représenté schématiquement dans la figure 2.3. De même que l'algorithme de la section précédente, cet algorithme contient également trois étapes. La première et la deuxième prennent chacune un temps linéaire, comme cela a été indiqué précédemment. Selon la méthode proposée dans [Kolpakov et Kucherov, 1999a], l'étape 3 prend un temps proportionnel

au nombre de répétitions trouvées. Le corollaire 1 du théorème 11 assure que ce nombre est linéaire en la longueur du mot et tout l'algorithme reste donc linéaire en temps. Nous le résumons dans le théorème suivant :

**Théorème 13** *Toutes les occurrences de répétitions maximales dans un mot de longueur $n$ peuvent être trouvées en temps $O(n)$.*

Une fois trouvé, l'ensemble des répétitions maximales fournit toute l'information sur la structure des répétitions du mot. En particulier, nous pouvons facilement en extraire d'autres types de répétitions, tels que les carrés (primitifs ou non), cubes, répétitions entières non-extensibles, etc. Toutes ces tâches prennent un temps $O(n+T)$ où $T$ est le nombre de répétitions trouvées. Par exemple, le fait de pouvoir trouver tous les carrés primitifs en temps $O(n+T)$ confirme la borne annoncée dans [Kosaraju, 1994]. Cette borne a également été confirmée par la suite dans l'article [Stoye et Gusfield, 1998b]. Les carrés branchants mentionnés dans la section 2.4.3 peuvent être également extraits de l'ensemble des répétitions maximales. Comme le nombre de carrés branchants est linéaire (conséquence du théorème 11), ils peuvent être trouvés tous en temps linéaire. Un autre exemple d'application du théorème 13 en combinaison avec le théorème 11 : on peut calculer en temps linéaire, pour un $k$ donné, le nombre de $k$-répétitions à racine primitive commençant à chaque position du mot [Kolpakov et Kucherov, 1999a].

#### Logiciel `mreps`

L'algorithme de recherche des répétitions maximales a été implanté à l'aide de Mathieu Giraud, étudiant de l'ENS Lyon, lors de son stage au LORIA[8]. Le logiciel, appelé `mreps`, s'est avéré très efficace. Des tests sur des séquences d'ADN (de l'ordre de centaines de milliers de paires de base) ont été faits et des répétitions intéressantes ont été trouvées. Une mini-interface graphique a été réalisée pour permettre d'afficher les répétitions à l'écran de façon visuelle. Ce travail a été présenté à la conférence JOBIM'2000 [Giraud et Kucherov, 2000]. Une interface Web a été mise en œuvre permettant d'utiliser le programme `mreps` via l'Internet[9].

### 2.5.3 Recherche de répétitions à écartement fixe

La section 2.5.2 a été consacrée à la recherche de répétitions *successives* dans un mot. Un autre problème consiste à rechercher des facteurs ayant plusieurs occurrences, pas forcement successives, dans le mot. On peut s'intéresser, par exemple, à la recherche de plus longs facteurs répétés dans un mot [Gusfield, 1997].

Dans les applications, telle que l'analyse de séquences génomiques par exemple, il se pose souvent un problème intermédiaire : trouver les copies du même facteur séparées par une distance spécifiée. Ce problème a été traité dans un article récent [Brodal *et al.*, 1999]. Plus précisément, le problème considéré dans [Brodal *et al.*, 1999] consistait à trouver tous les facteurs $uvu$, où la taille de $v$, appelé l'*écartement*, est bornée par un intervalle donné (dépendant éventuellement de la taille de $u$). La solution proposée, basée sur l'arbre des suffixes combiné avec les arbres binaires de recherche, s'exécute en temps $O(n \log n + S)$, où $S$ est la taille de la sortie (nombre de répétitions trouvées).

Ici, nous considérons une restriction de ce problème dans laquelle l'écartement est de taille *fixe*. Dans [Kolpakov et Kucherov, 2000a] nous avons montré le résultat suivant :

**Théorème 14** *Soit $w$ un mot de longueur $n$. Pour un nombre $r \in \mathbb{N}$ donné, on peut trouver dans $w$ tous les facteurs $uvu$ avec $|v| = r$ en temps $O(n \log r + S)$, où $S$ est le nombre de facteurs*

8. `http://www.loria.fr/remag/stages.html`
9. `http://www.loria.fr/~kucherov/SOFTWARE/MREPS/index.html`

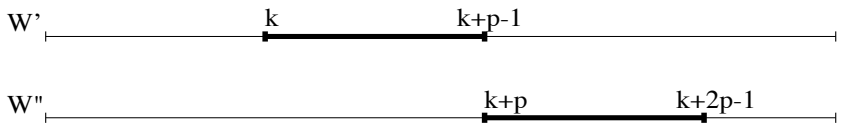


FIG. 2.4 – *Quasi-carrés*

*trouvés.*

Il est intéressant de noter que notre solution utilise fondamentalement les mêmes techniques que celles de la section 2.5.3, à savoir la s-factorisation et les fonctions de Main et Lorentz[10]. On n'utilise une structure du type arbre des suffixes que d'une façon indirecte (pour la construction de la s-factorisation) et on ne fait pas appel aux techniques relativement complexes d'arbres de recherche utilisées dans [Brodal *et al.*, 1999].

Notre algorithme utilise comme *boîte noire* une solution à un autre problème qui est intéressant en soi, car il généralise le problème de recherche de tous les carrés dans un mot. De plus, c'est ce problème qui s'avère critique pour la complexité du problème de départ et qui donne lieu au facteur $\log r$ dans la borne de complexité. Il s'agit de la recherche de *quasi-carrés*. Étant donnés deux mots $w'$ et $w''$ d'une même longueur $n$, on dit qu'ils contiennent un quasi-carré si pour une position $k$, $1 \leq k \leq n$ et un entier $p > 0$ on a $w'[k..k+p-1] = w''[k+p..k+2p-1]$ (voir la figure 2.4).

**Exemple 7** *Il y a cinq quasi-carrés entre les mots* $w' = 01100110$ *et* $w'' = 11011100$ *qui correspondent aux racines* 01 *(*$k = 1$*),* 11 *(*$k = 2$*),* 110 *(*$k = 2$*),* 1 *(*$k = 3$*),* 100 *(*$k = 3$*).*

Il est clair que si $w' = w'' = w$, la recherche des quasi-carrés entre $w'$ et $w''$ revient à la recherche des carrés de $w$. Nous avons démontré dans [Kolpakov et Kucherov, 2000a], en n'utilisant que les fonctions de Main et Lorentz, que les quasi-carrés entre deux mots de longueur $n$ peuvent être extraits en temps $O(n \log n + S)$ où $S$ est leur nombre. C'est une question ouverte intéressante de savoir si cette complexité peut être améliorée et en particulier si elle peut être rendue linéaire. Une réponse positive entraînerait une solution linéaire au problème de recherche de carrés à écartement fixe.

Finalement, notre algorithme peut être modifié pour trouver toutes les occurrences répétées séparées par un *mot fixe*.

10. Pour des raisons techniques et peu importantes, à la place de la s-factorisation nous avons utilisé dans [Kolpakov et Kucherov, 2000a] la *factorisation de Lempel-Ziv*, bien connue grâce à la méthode de compression associée [Lempel et Ziv, 1976; Ziv et Lempel, 1977] et légèrement différente de la s-factorisation mais étroitement liée à cette dernière.

# Chapitre 3

# Motifs dans les arbres

## 3.1 Langages d'arbres engendrés par les motifs : définitions

L'arbre est une structure fondamentale en informatique. Nous n'adoptons pas ici le point de vue de la théorie des graphes qui considère les arbres comme une classe particulière de graphes généraux [Diestel, 1996]. Les arbres que nous considérons dans ce document sont les *termes du premier ordre* qui sont tout simplement les expressions correctement formées de symboles de fonction et de variables. Cette définition d'arbres apparaît lorsque l'on veut parler d'arbres comme des objets syntaxiques qui forment des *langages* et qui sont donc engendrés par des *grammaires* ou reconnus par des *automates* [Gécseg et Steinby, 1984; Nivat et Podelski, 1992; Comon *et al.*, 1997]. C'est le point de vue de la théorie des langages formels [Rozenberg et Salomaa, 1997] mais aussi de la déduction automatique [van Leeuwen, 1990] ou encore de la théorie des systèmes de réécriture [Dershowitz et Jouannaud, 1990].

Plus formellement, un arbre est une expression bien formée sur une *signature* $\Sigma$ de symboles de fonction, où chaque symbole est indexé par un nombre entier naturel appelé l'*arité*. Par exemple, $u = f(f(f(a,a),h(a)),h(a))$ est un arbre sur la signature $\Sigma = \{f,h,a\}$, où les symboles $f,h,a$ ont respectivement l'arité 2,1,0. L'ensemble de tous les arbres sur $\Sigma$ est noté $T(\Sigma)$. Il est clair que pour que $T(\Sigma)$ soit non-vide, $\Sigma$ doit contenir des symboles d'arité 0, appelés *symboles de constante*. Du point de vue algébrique, $T(\Sigma)$ est une $\Sigma$-algèbre libre générée par les symboles de constante de $\Sigma$. Il s'agit donc d'arbres dont les nœuds sont étiquetés par des symboles de $\Sigma$. Ces arbres représentent des *termes du premier ordre* sur $\Sigma$. Nous allons donc employer les mots *arbre* et *terme* comme synonymes.

Un *motif d'arbre* est un arbre sur $\Sigma \cup X$, où $X$ est un ensemble infini de symboles d'arité 0, appelés *variables*. Par exemple, $f(x,h(y))$, $f(f(f(y,a),x),x)$ sont des motifs sur $\{f,h,a\}$, où $x,y$ sont des variables. Un sous-arbre d'un arbre $t$ est une sous-expression de $t$. En d'autres termes, un sous-arbre de $t$ est un arbre enraciné à un nœud interne de $t$. Les sous-arbres de l'arbre $u$ sont $f(f(f(a,a),h(a)),h(a))$, $f(f(a,a),h(a))$, $f(a,a)$, $h(a)$ et $a$. Remarquons que $h(a)$ et $a$ ont plusieurs occurrences dans $u$. Une *substitution* est un homomorphisme $\sigma : T(\Sigma \cup X) \to T(\Sigma)$ tel que $\sigma(a) = a$ pour tout symbole de constante de $\Sigma$. Si $t = \sigma(p)$ pour un terme $t$ et un motif $p$, on dira que $t$ est une *instance* de $p$. Comme dans le cas des mots, la substitution remplace les variables dans le motif par des termes de telle façon que la même variable apparaissant à plusieurs positions est remplacée par le même terme. Par exemple, le terme $u$ est une instance de chacun des motifs $f(x,h(y))$ et $f(f(f(y,a),x),x)$, mais n'est pas une instance de $f(f(x,x),h(a))$. À la différence avec le cas des mots, chaque variable doit être remplacée par un terme « non-vide » et il n'existe donc pas d'analogue de la notion de substitution effaçante. De même que pour le

cas des mots, une variable d'un motif est appelée *linéaire* si elle n'apparaît dans le motif qu'une fois et *non-linéaire* dans le cas contraire.

Les notions de sous-arbre, motif et instance introduites plus haut définissent, pour le cas des arbres, deux ensembles principaux que nous avons définis dans l'introduction d'une façon générique. Il s'agit de l'ensemble $Inst(P) \subseteq T(\Sigma)$ qui est l'ensemble des instances d'un ensemble $P$ de motifs, et de l'ensemble $Cont(P) \subseteq T(\Sigma)$ qui est l'ensemble des termes qui contiennent un sous-terme appartenant à $Inst(P)$. De même que dans le cas des mots, les langages $Inst(P)$ et $Cont(P)$ sont les deux objets principaux étudiés dans ce chapitre, puisque la plupart des problèmes considérés s'expriment en termes de ces deux ensembles.

Notons que contrairement au cas des mots (voir (2.3)), $Cont(P)$ ne s'exprime pas en termes de $Inst(P)$, puisque la notion de sous-arbre ne peut pas être exprimée à l'aide de motifs, ces derniers ne contenant que des variables du premier ordre. De plus, un problème algorithmique portant sur $Inst(P)$ se réduit en général au même problème portant sur l'ensemble $Cont(P')$ pour un autre ensemble $P'$ convenablement construit. Par conséquent, les propriétés du langage $Inst(P)$ sont en général plus simples que les propriétés correspondantes de $Cont(P)$, contrairement au cas des mots. Nous reviendrons à ces questions dans le chapitre 4, où nous ferons une analyse comparative systématique des cas des mots et d'arbres par rapport à certains problèmes de décision.

## 3.2 Propriétés algorithmiques des langages de motifs

Dans cette section nous étudions la complexité de quelques problèmes de décision liés aux langages de motifs. Le premier problème, traité dans la section 3.2.1, consiste à reconnaître si, pour un ensemble fini $P$ de motifs, l'ensemble $Cont(P)$ est un langage régulier d'arbres. Ces résultats ont fait l'objet des communications [Kucherov, 1991; Kucherov et Tajine, 1992] dont une version approfondie a été publiée dans [Kucherov et Tajine, 1995]. La section 3.2.2 est consacrée au problème de complémentation des langages $Inst(P)$. Elle correspond principalement à l'article [Plaisted et Kucherov, 1999].

### 3.2.1 Régularité de langages de motifs

Alors que la régularité de langages de mots est une notion bien classique, la notion de régularité de langages d'arbres est probablement moins connue. Cependant, cette dernière est une généralisation naturelle de la régularité de langages de mots. Une des définitions possibles des langages réguliers d'arbres est celle à l'aide de grammaires d'arbres. Une grammaire régulière d'arbre est un quadruplet $G = (N, \Sigma, S, P)$ où $N$ est un ensemble fini de *symboles non-terminaux*, $\Sigma$ la signature, $S \in N$ symbole de départ, et $P$ un ensemble de règles de dérivation de la forme $A \to t$, où $A \in N$ et $t \in T(\Sigma \cup N)$. Le langage engendré par la grammaire $G$ est défini naturellement comme l'ensemble de tous les termes de $T(\Sigma)$ dérivables à partir du symbole $S$, où la dérivation est une suite d'applications de règles de $P$ remplaçant le non-terminal $A$ de la partie gauche par l'arbre $t$ de la partie droite correspondante.

**Exemple 8** *Le langage des arbres de la forme* $f(g^{i_1}(a), f(g^{i_2}(a), \ldots, f(g^{i_m}(a), g^{i_{m+1}}(a))\ldots))$, $i_1, i_2, \ldots, i_{m+1} \geq 0$, *peut être engendré par la grammaire*

$$\begin{aligned} S &\to C \\ B &\to a \mid g(B) \\ C &\to f(B,B) \mid f(B,C) \end{aligned}$$

Toutes les caractéristiques principales des langages réguliers de mots ont leurs analogues dans le cas des arbres : le lemme de pompage, la caractérisation algébrique, la caractérisation à l'aide d'automates d'états finis ou à l'aide d'expressions régulières, ainsi que beaucoup d'autres propriétés. Nous renvoyons aux livres [Gécseg et Steinby, 1984; Nivat et Podelski, 1992; Comon *et al.*, 1997] ainsi qu'au survol [Gésceg et Steinby, 1997] pour plus de détails sur les langages réguliers d'arbres. Dans la section 3.3 nous allons considérer d'autres classes de langages d'arbres et leurs relations aux langages réguliers.

Afin d'introduire le problème principal de ce chapitre qui porte sur l'ensemble $Cont(P)$, nous nous concentrons d'abord sur certaines propriétés de l'ensemble $Inst(P)$. L'article [Lassez et Marriot, 1987] contient une étude de ce langage. Une des questions étudiées dans cet article est la suivante : dans quel cas l'ensemble $\overline{Inst(P)} = T(\Sigma) \setminus Inst(P)$ peut-il lui-même être représenté comme $Inst(S)$ pour un certain ensemble fini de motifs $S$, que nous appelons *une représentation du complément* (*complement representation*) de $P$ ? Par exemple, soit $P = \{h(x), f(h(x), y)\}$ sur la signature $\{f, h, a\}$. L'ensemble $S = \{a, f(a, x), f(f(x, y), z)\}$ peut alors être choisi comme une représentation du complément de $P$.

En généralisant l'exemple ci-dessus on peut prouver (voir par exemple [Kucherov, 1988]) que si les motifs d'un ensemble ne contiennent pas de variables non-linéaires, alors cet ensemble possède une représentation finie du complément. Cependant, il n'en est pas de même en présence de variables non-linéaires. Par exemple, on peut démontrer que l'ensemble $S = \{f(x, x)\}$ ne possède pas de représentation finie du complément. L'analyse effectuée dans [Lassez et Marriot, 1987] peut être résumée dans le théorème suivant :

**Théorème 15 ([Lassez et Marriot, 1987])** *Un ensemble fini $P$ de motifs possède une représentation finie du complément si et seulement s'il existe un ensemble fini de motifs* linéaires $P_{lin}$ *tel que $Inst(P) = Inst(P_{lin})$. De plus,*

- *si un tel ensemble $P_{lin}$ existe, il peut être obtenu en instanciant les variables non-linéaires dans les motifs de $P$ par des termes.*
- *la propriété de posséder une représentation finie du complément est décidable.*

Illustrons le théorème 15 avec un exemple.

**Exemple 9** *Soit $P = \{a, f(x, h(y)), f(x, x),\ f(x, f(y, z))\}$, toujours sur la signature $\{f, h, a\}$. Cet ensemble contient un motif non-linéaire $f(x, x)$. Cependant, une analyse simple montre que $f(x, x)$ peut être remplacé par $f(a, a)$ sans que l'ensemble d'instances $Inst(P)$ soit changé. Par conséquent, $Inst(P) = Inst(P_{lin})$, où $P_{lin}$ est un ensemble de motifs linéaires obtenu de $P$ en substituant $x$ par $a$ dans le motif $f(x, x)$. Une représentation du complément de ce nouvel ensemble $P_{lin}$ peut être maintenant construit : $S = \{h(x), f(h(x), a), f(f(x, y), a)\}$.*

Le théorème 15 affirme que l'exemple 9 représente une situation caractéristique à tous les ensembles possédant une représentation finie du complément. La propriété de décidabilité du théorème 15 signifie qu'il existe une borne effective sur la taille des termes devant remplacer les variables non-linéaires. Nous poursuivrons l'étude des représentations du complément dans la section suivante où nous aborderons la question de complexité du calcul d'une représentation du complément ainsi que de sa taille. Dans cette section, nous nous intéressons aux propriétés structurelles de l'ensemble $Inst(P)$ et en particulier aux conditions de régularité.

Il n'est pas très difficile de démontrer que $Inst(P)$ est un langage régulier d'arbres si l'ensemble $P$ ne contient que des termes linéaires. Évidemment, si $P$ est « linéarisable » selon le théorème 15, c'est-à-dire $Inst(P) = Inst(P_{lin})$ où $P_{lin}$ est une instanciation linéaire de $P$, alors $Inst(P)$ est régulier aussi. Est-ce également une condition *nécessaire* pour que $Inst(P)$ soit régulier ? Autrement dit, est-ce que la condition de « linéarisabilité » du théorème 15, qui équivaut à l'existence d'une représentation finie du complément, équivaut aussi à la régularité du langage

$Inst(P)$? Cela est effectivement vrai mais n'est pas facile à démontrer. Cette propriété découle des résultats plus généraux de l'article [Kucherov, 1991] que nous décrivons maintenant.

Considérons le langage $Cont(P)$. De même que $Inst(P)$, $Cont(P)$ est un langage régulier si tous les motifs de $P$ sont linéaires. Le résultat suivant a été démontré dans [Kucherov, 1991] :

**Théorème 16 ([Kucherov, 1991])** *Pour un ensemble fini $P$ de motifs, $Cont(P)$ est un langage régulier d'arbres si et seulement s'il existe un ensemble fini de motifs* linéaires *$P_{lin}$ tel que $Cont(P) = Cont(P_{lin})$. De plus, si un tel ensemble $P_{lin}$ existe, il peut être obtenu en instanciant les variables non-linéaires dans les motifs de $P$ par des termes.*

L'analogie entre ce théorème et le théorème 15 est facile à observer : le premier affirme que pour que $\overline{Inst(P)}$ soit représentable comme $Inst(S)$ pour un ensemble fini $S$, il faut et il suffit que $P$ puisse être instancié de façon à donner un ensemble $P_{lin}$ de motifs linéaires tel que $Inst(P) = Inst(P_{lin})$. Le théorème 16, de son côté, affirme que pour que $Cont(P)$ soit régulier, il faut et il suffit que $P$ puisse être instancié de façon à donner un ensemble $P_{lin}$ de motifs linéaires tel que $Cont(P) = Cont(P_{lin})$. Malgré cette similitude apparente, le deuxième théorème est plus puissant car il porte sur l'ensemble plus général $Cont(P)$. En outre, il établit une condition de régularité d'un ensemble – type de résultats de la théorie des langages souvent difficile à obtenir.

Comme cela a été mentionné plus haut, la preuve du théorème 16 (voir les preuves des théorèmes 4 et 6 de [Kucherov et Tajine, 1995]) implique, entre autre, que le théorème 15 énonce également la condition de la régularité de l'ensemble $Inst(P)$ :

**Théorème 17** *Le langage $Inst(P)$ est régulier si et seulement s'il existe un ensemble $P_{lin}$ de motifs linéaires tel que $Inst(P) = Inst(P_{lin})$, où $P_{lin}$ peut être obtenu de $P$ en instanciant chaque variable non-linéaire par un ensemble fini de termes.*

À la différence du théorème 15, le théorème 16 n'affirme pas la décidabilité de la condition de régularité de $Cont(P)$. La raison en est que la preuve du théorème 16 [Kucherov, 1991; Kucherov et Tajine, 1995] utilise un argument combinatoire très puissant mais non-effectif – la version infinie du théorème de Ramsey – qui ne fournit pas de borne sur la taille de termes impliqués dans la substitution des variables non-linéaires. La question de décidabilité de la régularité du langage $Cont(P)$ était donc restée ouverte. Elle avait été mentionnée sur la liste de problèmes ouverts majeurs dans la théorie des systèmes de réécriture, publiée dans les actes de la conférence *Rewriting Techniques et Applications* en 1991 [Dershowitz *et al.*, 1991]. Peu après cette publication, il a été prouvé que ce problème était effectivement décidable [Kucherov et Tajine, 1992; Vágvölgyi et Gilleron, 1992; Hofbauer et Huber, 1992]. Il est intéressant de noter que la preuve de [Kucherov et Tajine, 1992] utilise à nouveau le théorème de Ramsey, mais cette fois sa version finie. Cette preuve permet d'établir d'autres résultats de décidabilité. Par exemple, elle implique que le problème de savoir si le langage $Cont(P)$ est co-fini est également décidable. Ce résultat avait été précédemment démontré dans [Plaisted, 1985; Kapur *et al.*, 1987]. Cependant, la construction de [Kucherov et Tajine, 1992] fournit une borne explicite (en fonction de $P$) pour la taille de l'ensemble $\overline{Cont(P)}$ dans le cas où il est fini. Nous résumons ces résultats dans le théorème suivant.

**Théorème 18** *Les propriétés de régularité et de co-finitude du langage $Cont(P)$ sont décidables.*

Par la suite dans le chapitre 4 nous présenterons d'autres résultats sur les propriétés algorithmiques des langages $Inst(P)$ et $Cont(P)$.

### 3.2.2 Représentation du complément d'un ensemble de motifs linéaires

Le théorème 15 décrit une condition pour qu'un ensemble de motifs (non-linéaires) possède une représentation finie du complément. Notons que le calcul d'une représentation du complé-

ment a des applications pratiques. En particulier, ce problème se pose lors de la vérification de propriétés de systèmes de réécriture de termes [Thiel, 1984; Lazrek *et al.*, 1990], ou lors des preuves automatiques dans les théories spécifiées par ces systèmes [Kucherov, 1988]. Dans [Lassez et Marriot, 1987], le problème est motivé par des problèmes d'apprentissage automatique où la représentation du complément formalise la notion d'apprentissage par contre-exemples. Enfin, un autre exemple est la programmation logique où la représentation du complément a fait l'objet de travaux récents [Gottlob et Pichler, 1999]. L'article [Lassez *et al.*, 1991] contient une discussion autour des applications de la représentation du complément.

Le théorème 15 affirme également que la condition de « linéarisabilité » d'un ensemble de motifs est décidable. Cependant, le théorème a laissé ouverte la complexité exacte de ce problème algorithmique. La réponse à cette question est apportée (selon [Pichler, 2000]) par l'article [Maher et Stuckey, 1995] :

**Théorème 19 ([Maher et Stuckey, 1995])** *Le problème de vérifier si, pour un ensemble de motifs $P$, il existe un ensemble de motifs linéaires $P_{lin}$ tel que $Inst(P) = Inst(P_{lin})$, est co-NP-complet.*

Dans le reste de cette section nous nous focalisons sur les ensembles de motifs *linéaires* pour lesquels il existe *toujours* une représentation finie du complément. La question que nous nous sommes posés dans [Plaisted et Kucherov, 1999] porte sur la taille de cette représentation. En particulier, dans quels cas peut-on construire une représentation du complément telle que la taille de cette représentation est polynomiale en la taille de l'ensemble de départ ?

La taille d'un motif $p \in T(\Sigma \cup X)$ est définie comme le nombre d'occurrences de symboles de $\Sigma$ dans $p$, et la taille d'un ensemble de motifs $P$ est la somme des tailles de ses éléments. On démontre que la taille de la représentation du complément peut toujours être bornée exponentiellement par rapport à la taille de l'ensemble de départ. D'autre part, cette borne exponentielle est effectivement atteignable comme le montre l'exemple suivant. Supposons que la signature contienne deux symboles de constante $a, b$ et un symbole de fonction $f$ d'arité $n$, On peut démontrer qu'une représentation du complément de l'ensemble

$$\{a, b, f(a, x_2, \ldots, x_n), f(x_1, a, x_3, \ldots, x_n), \ldots, f(x_1, x_2, \ldots, a)\}$$

contient au moins $2^n$ motifs [Plaisted et Kucherov, 1999]. Enfin, on peut facilement modifier cet exemple pour que la signature ne contienne que des symboles d'arité borné, en remplaçant les motifs $f(t_1, t_2, \ldots, t_n)$ par un motif du type $f(t_1, f(t_2, f(\ldots, f(t_n, a)\cdots)))$.

Un autre lemme démontré dans [Plaisted et Kucherov, 1999] affirme que dans le cas dégénéré où les motifs de l'ensemble $P$ ne contiennent pas de variables, on peut construire une représentation du complément de $P$ dont la taille est quadratique en la taille de $P$. La question se pose donc de savoir si l'on peut identifier des classes intéressantes d'ensembles de motifs avec variables pour lesquels la taille de la représentation du complément reste polynomialement bornée.

En analysant l'exemple ci-dessus, on peut remarquer qu'il repose sur le fait que les motifs de l'ensemble ont des instances communes. Autrement dit, les motifs sont *unifiables*. Il peut donc paraître plausible que dans le cas où les motifs de l'ensemble $P$ sont non-unifiables deux à deux, c'est-à-dire $Inst(t_1) \cap Inst(t_2) = \emptyset$ pour tous $t_1, t_2 \in P$, il existe toujours une représentation du complément de $P$ de taille polynomiale. Il s'avère que cette conjecture est fausse.

**Théorème 20 ([Plaisted et Kucherov, 1999])** *Il existe une famille d'ensembles $P$ de motifs non-unifiables deux à deux tel que toute représentation du complément de $P$ a une taille exponentielle par rapport à la taille de $P$.*

Malgré ce résultat négatif, il est quand même possible de trouver des classes d'ensembles intéressantes pour lesquelles il existe une représentation du complément de taille polynomiale. Une

telle classe comprenant les ensembles dits *hiérarchiques* est définie dans [Plaisted et Kucherov, 1999]. Nous renvoyons le lecteur à cet article pour les détails correspondants.

Pour terminer cette section, nous revenons à nouveau à la complexité algorithmique de propriétés de l'ensemble $Inst(P)$. D'après le théorème 19, c'est un problème co-NP-complet que de vérifier la condition du théorème 15 qui garantit l'existence d'une représentation finie du complément. Il s'avère qu'il est aussi co-NP-complet de vérifier si la représentation du complément d'un ensemble $P$ est vide, même si $P$ est composé uniquement de motifs linéaires.

**Théorème 21 ([Kapur *et al.*, 1991])** *Le problème de vérifier si, pour un ensemble de motifs linéaires $P$, $Inst(P) = T(\Sigma)$ est co-NP-complet.*

Cependant, contrairement au problème de l'existence d'une représentation du complément de taille polynomiale, le problème du théorème 21 devient plus simple si l'on se restreint aux motifs linéaires non-unifiables deux à deux.

**Théorème 22 ([Plaisted et Kucherov, 1999])** *Pour les ensembles $P$ de motifs linéaires non-unifiables deux à deux il existe un algorithme polynomial (linéaire) pour tester si $Inst(P) = T(\Sigma)$.*

## 3.3 Quelques classes de langages d'arbres

Le théorème 16 affirme que le langage d'arbres $Cont(P)$ est régulier si et seulement si les variables non-linéaires des motifs de $P$ peuvent être remplacées par des langages *finis* sans que le langage $Cont(P)$ soit changé. Les résultats que nous présentons dans cette section ont été motivés par la question suivante : sous quelles conditions le langage $Cont(P)$ est-il un langage algébrique (*context-free*) d'arbres? Autrement dit, que se passe-t-il si l'on monte au niveau au-dessus dans la hiérarchie de Chomsky, en passant des langages réguliers aux langages algébriques?

Cette motivation nous a d'abord amené à étudier les langages *co-réguliers* d'arbres [Arnold et Dauchet, 1976], qui constituent une sous-classe des langages algébriques. Cette étude, qui a fait l'objet de la publication [Hofbauer *et al.*, 1994], est décrite dans la section 3.3.1.

Une deuxième étude, présentée dans la section 3.3.2, porte sur les langages algébriques généraux. Nous nous intéressons dans cette partie aux règles effaçantes dans les grammaires algébriques. Nous verrons que ces règles sont en général indispensables mais nous mettons au point une forme normale de grammaires qui permet d'éviter l'application des règles effaçantes jusqu'une certaine profondeur dans les termes dérivés. Cette étude a été décrite dans l'article [Hofbauer *et al.*, 1998].

### 3.3.1 Langages co-réguliers d'arbres

Dans la section 3.2.1 nous avons introduit les langages réguliers d'arbres. Ici, nous allons monter à l'échelle au-dessus dans la hiérarchie de Chomsky et nous allons considérer les langages algébriques d'arbres. Nous en donnons d'abord la définition.

De même que dans le cas régulier, une grammaire algébrique d'arbres est un quadruplet $G = (N, \Sigma, S, P)$. À la différence du cas régulier, les non-terminaux de $N$ ont chacun une arité, tout comme les symboles de la signature $\Sigma$. Ils pourront donc apparaître lors de la dérivation aux positions intérieures, et non seulement aux feuilles des arbres, comme dans le cas régulier. Le symbole $S$ est le symbole de départ – un non-terminal d'arité 0. Enfin, les règles de $P$ ont la forme

$$A(x_1, \ldots, x_n) \to t, \tag{3.1}$$

où $A \in N$ est un non-terminal d'arité $n$, et $x_1, \ldots, x_n$ sont des variables distinctes et $t \in T(\Sigma \cup N \cup \{x_1, \ldots, x_n\})$. L'application de règles de dérivation est analogue au cas des règles de

réécriture : la règle (3.1) s'applique au symbole non-terminal $A$ en le remplaçant par l'arbre $t$ de telle manière que les fils de ce symbole $A$, correspondant aux variables $x_1, \ldots, x_n$, viennent remplacer, respectivement, les variables $x_1, \ldots, x_n$ dans le terme $t$. Naturellement, le langage se dit algébrique s'il peut être engendré par une grammaire algébrique.

Il est important de noter que si le terme $t$ est non-linéaire, c'est-à-dire si une variable $x_i$ apparaît plusieurs fois dans $t$, alors l'application de la règle donne lieu à une duplication de l'arbre correspondant à la variable $x_i$. S'il n'y a qu'une seule occurrence de chaque variable dans le membre droit de chaque règle, on dira que la grammaire est *linéaire*, sinon elle est *non-linéaire*. Un langage s'appelle linéaire s'il peut être engendré par une grammaire linéaire.

**Exemple 10** *La grammaire algébrique suivante engendre le langage* $\{f(t_1, t_2) \mid t_1, t_2 \in T(\{f, a\}), t_1 \neq t_2\}$ *:*

$$\begin{array}{rcl} S & \to & A(a, f(B, B)) \mid A(f(B, B), a) \\ A(x, y) & \to & f(x, y) \mid A(f(x, B), f(y, B)) \mid A(f(B, x), f(B, y)) \\ B & \to & a \mid f(B, B) \end{array}$$

*Cette grammaire est linéaire.*

*Voici une autre grammaire algébrique qui engendre le langage* $\{t_i \mid i \geq 0\}$ *où les* $t_i$ *sont définis par la récurrence* $t_0 = a$, $t_i = f(t_{i-1}, t_{i-1})$ *(il s'agit donc du langage des arbres binaires complets) :*

$$\begin{array}{rcl} S & \to & a \mid C(a) \\ C(x) & \to & f(x, x) \mid C(f(x, x)) \end{array}$$

*Cette grammaire, en revanche, est non-linéaire.*

Les langages algébriques d'arbres ont été introduits par W. C. Rounds [Rounds, 1970a; Rounds, 1970b]. Nous renvoyons au survol [Gésceg et Steinby, 1997] pour les références à d'autres travaux liés.

A. Arnold et M. Dauchet [Arnold et Dauchet, 1976] ont introduit les langages *co-réguliers*[11], une sous-classe très importante de la classe des langages algébriques. Une grammaire algébrique est *co-régulière* si les symboles non-terminaux n'apparaissent qu'à la racine des termes $t$ dans les membres droits des règles de dérivations (3.1).

**Exemple 11** *Le langages des arbres binaires complets de l'exemple 10 est co-régulier car le non-terminal* $C$ *n'apparaît qu'à la racine dans les membres droits des règles.*

*Un autre exemple est le langage* $\{f(g^i(a), f(g^i(a), \ldots, f(g^i(a), a)\ldots)) \mid i \geq 0\}$ *engendré par le grammaire co-régulière suivante :*

$$\begin{array}{rcl} S & \to & A(a) \\ A(x) & \to & A(g(x)) \mid B(x, a) \\ B(x, y) & \to & B(x, f(x, y)) \mid f(x, y) \end{array}$$

Les langages co-réguliers sont en quelques sorte « orthogonaux » aux langages réguliers (d'où leur nom) : si dans les dérivations de langages réguliers les non-terminaux apparaissent aux feuilles des arbres, dans le cas des langages co-réguliers ils n'apparaissent, au contraire, qu'à la racine.

11. Dans nos travaux [Hofbauer *et al.*, 1994; Hofbauer *et al.*, 1998] nous avons employé le terme anglais *top-context-free*. Dans ce document nous gardons le terme français originel *co-régulier*.

Il y a des langages très simples qui sont réguliers mais ne sont pas co-réguliers et *vice versa*. Par exemple, le langage $\{f(g^i(a), g^i(a))\}$ n'est pas régulier mais il est engendré par la grammaire co-régulière suivante : $S \to A(a)$, $A(x) \to A(g(x)) \mid f(x,x)$. En revanche, le langage de tous les termes $T(\{f,a\})$ est trivialement régulier, mais on peut démontrer (voir le théorème 24 plus loin) qu'il n'est pas co-régulier.

Le théorème suivant a été démontré dans [Arnold et Dauchet, 1976] :

**Théorème 23 ([Arnold et Dauchet, 1976])** *Le langage $L = \{f(t,t) \mid t \in L'\}$ est algébrique si et seulement si $L'$ est co-régulier, auquel cas $L$ est co-régulier aussi.*

On peut remarquer un parallèle entre le théorème 23 et le théorème 17 : si l'on remplace, dans le théorème 23, « algébrique » par « régulier » et « co-régulier » par « fini », on obtient un cas particulier du théorème 17. Nous conjecturons que cette analogie est profonde et qu'elle est valable dans un cadre plus général. En termes informels, les langages co-réguliers jouent le même rôle dans le cas algébrique que les langages finis dans le cas régulier. Cette conjecture nous a motivé pour étudier plus profondément les propriétés des langages co-réguliers.

Le premier résultat, obtenu dans [Hofbauer *et al.*, 1994], établit une condition nécessaire pour qu'un langage soit co-régulier. Il donne alors un critère permettant de prouver qu'un langage n'est pas co-régulier, de même que les lemmes de pompage servent à démontrer qu'un langage n'est pas régulier ou algébrique.

Étant donné un terme $t \in T(\Sigma)$, on défini d'abord une *décomposition* de $t$ comme une suite $D_0, \ldots, D_m$ d'ensembles de termes, où $D_0 = \{t\}$, $D_m = \emptyset$, et pour tout $i$, $\leq i < m$, il existe un terme $f(t_1, \ldots, t_n) \in D_i$ tel que

$$D_{i+1} = (D_i \setminus \{f(t_1, \ldots, t_n)\}) \cup \{t_1, \ldots, t_n\}. \tag{3.2}$$

Le terme $t$ est $k$-mince s'il existe une décomposition $D_0, \ldots, D_m$ de $t$ telle que $|D_i| \leq k$ pour tout $i$, $\leq i \leq m$. Un langage est $k$-mince si tous ses termes sont $k$-minces. Enfin, un langage est *mince* s'il est $k$-mince pour un certain $k$.

**Exemple 12** *Le langage des arbres complets de l'exemple 10 est 1-mince. En effet, comme $t_i = f(t_{i-1}, t_{i-1})$, chaque pas dans la décomposition transforme l'ensemble $\{t_i\}$ en l'ensemble $\{t_{i-1}\}$.*

*Le langage de l'exemple 11 est 2-mince. Il suffit de décomposer $\{f(g^i(a), f(g^i(a), \ldots, f(g^i(a), a)\ldots))\}$ en $\{g^i(a), f(g^i(a), \ldots, f(g^i(a), a)\ldots)\}$, ensuite décomposer le terme $g^i(a)$ jusqu'à ce qu'il disparaisse et répéter la procédure.*

Notez que tous les langages monadiques (c'est-à-dire sur une signature ne contenant que des symboles d'arité 0 et 1) sont 1-minces. Parmi les langages minces on retrouve donc des langages de toute complexité structurelle, mêmes des langages qui ne sont pas récursivement énumérables. En particulier, il existe des langages minces qui ne sont pas co-réguliers. Cependant, l'inverse n'est pas vrai.

**Théorème 24 ([Hofbauer *et al.*, 1994])** *Tous les langages co-réguliers sont minces.*

Le théorème 24 fournit un outil pour prouver qu'un langage n'est pas co-régulier en prouvant qu'il n'est pas mince. Par exemple, on peut démontrer que si la signature $\Sigma$ contient au moins un symbole d'arité 2 ou plus, le langage $T(\Sigma)$ de tous les termes n'est pas mince, et donc n'est pas co-régulier. À l'aide du théorème 23 on peut ensuite déduire que dans ce cas le langage $\{f(t,t\} \mid t \in T(\Sigma)\}$ n'est pas algébrique.

Nous avons déjà remarqué que les classes des langages réguliers et co-réguliers sont incomparables. Cependant, leur intersection n'est pas vide, comme on peut s'en convaincre en remarquant

que les langages finis appartiennent aux deux classes. Dans [Hofbauer *et al.*, 1994] nous avons établi plusieurs caractérisations de l'intersection de ces deux classes, résumées dans le théorème suivant.

**Théorème 25 ([Hofbauer *et al.*, 1994])** *Pour un langage régulier $L$, les conditions suivantes sont équivalentes :*

*(1) $L$ est co-régulier,*

*(2) $L$ est co-régulier linéaire,*

*(3) $L$ est mince,*

*(4) $L$ peut être engendré par une grammaire régulière non-branchante. Plus généralement, toute grammaire régulière réduite qui engendre $L$ est non-branchante,*

*(5) $L$ peut être représenté par une expression régulière linéaire,*

*(6) $L$ est borné,*

*(7) $L$ est polynomial.*

Nous allons maintenant commenter les conditions du théorème.

La condition (2) affirme que tout langage co-régulier qui est en même temps régulier peut être engendré par une grammaire co-régulière linéaire. Notons cependant qu'il n'est bien sûr pas vrai que tout langage co-régulier linéaire soit régulier – le langage $\{f(g^i(a), g^i(a)) \mid i \geq 0\}$ n'est pas régulier, mais il peut être engendré par la grammaire co-régulière linéaire suivante :

$$\begin{aligned} S &\rightarrow A(a,a) \\ A(x,y) &\rightarrow A(g(x),g(y)) \mid f(x,y) \end{aligned}$$

Selon la condition (3), si un langage régulier est mince, il est co-régulier. Autrement dit, l'intersection des langages réguliers avec les langages minces est la même que celle avec les langages co-réguliers (rappelons que tous les langages réguliers sont minces d'après le théorème 24).

La condition (4) caractérise la structure des grammaires régulières qui engendrent des langages co-réguliers ; ce sont les grammaires dites non-branchantes. Une grammaire régulière est non-branchante si aucun non-terminal $A$ ne peut se dériver en un arbre $t$ contenant deux occurrences distinctes de $A$. Informellement, la dérivation ne peut pas « boucler » au long de deux branches ramifiantes. Selon la condition (4), toute grammaire régulière engendrant un langage co-régulier est branchante, à condition qu'elle soit réduite, c'est-à-dire ne contienne pas de non-terminaux « inutiles ». Les grammaires régulières non-branchantes représentent donc une caractérisation syntaxique des langages à la fois réguliers et co-réguliers.

Dans la condition (5) il s'agit d'expressions régulières pour les langages d'arbres. Il faut d'abord préciser que la notion d'expression régulière, connue dans la théorie des langages classique, se généralise très naturellement aux langages d'arbres. En particulier, on peut établir un homologue du théorème de Kleene dans le cas des arbres [Gécseg et Steinby, 1984]. Dans [Hofbauer *et al.*, 1994] nous avons défini une sous-classe d'expressions régulières qui représentent exactement les langages réguliers qui sont co-réguliers. Cette sous-classe d'expressions régulières, que nous ne définissons pas ici, constitue donc une autre caractérisation de l'intersection des langages réguliers et co-réguliers.

La notion de langages *bornés* (en anglais *passable*), introduite dans l'article [Salomaa, 1988], a la même définition que celle des langages minces, à la différence que les ensembles de termes dans la décomposition sont traités comme des multi-ensembles. On peut facilement démontrer, par induction sur la dérivation, que tout langage co-régulier linéaire est borné. Cela veut dire que (2) implique (6) dans le théorème 25. Dans [Salomaa, 1988] on montre également que tout langage borné est *polynomial* (*polynomially size-bounded*), ce qui signifie que la taille des termes

du langage est borné par un polynôme en leur profondeur. Cela implique, à son tour, que (6) implique (7). Enfin, on peut démontrer que la condition (7) implique (4), car si un langage régulier est engendré par une grammaire régulière branchante, la taille des termes est forcément exponentielle par rapport à leur profondeur.

Dans l'article [Hofbauer *et al.*, 1994] nous avons également étudié les propriétés de clôture de différentes classes de langages (finis, réguliers, co-réguliers, co-réguliers linéaires, algébriques, et algébriques linéaires) par rapport aux opérations de *substitution* et *remplacement*. Ces opérations, dont l'importance a été mise en évidence dans [Engelfriet et Schmidt, 1977], sont deux homologues différents de l'opération classique d'itération (étoile) dans le cas des mots. La présentation détaillée de ces résultats dépasse le cadre de ce document.

### 3.3.2 Règles effaçantes dans les grammaires algébriques d'arbres

Comme nous l'avons remarqué dans la section 3.2.1, tous les résultats de base portant sur les langages réguliers de mots (théorème de Kleene, caractérisations algébrique et à l'aide d'automates, formes normales, lemme de pompage, propriétés de clôture, résultats de décidabilité) se généralisent naturellement dans le cas des arbres. Il se trouve qu'il n'en est pas de même dans le cas algébrique, car certains résultats classiques n'ont pas d'analogue dans le cas des arbres. L'étude que nous présentons dans cette section fournit un exemple d'un tel phénomène.

Revenons à la définition de grammaires algébriques d'arbres. On sait que l'on peut, sans réduire la classe des langages algébriques, restreindre les dérivations de façon à ce qu'une règle ne s'applique à une position dans l'arbre que si à toutes les positions ancestrales (sur le chemin menant à la racine) des symboles terminaux ont été installés (le terme « installé » est justifié : il est clair que si un symbole terminal n'a pas de non-terminaux sur ses positions ancestrales, il ne pourra plus être atteint dans le reste de la dérivation). Autrement dit, dans l'analyse de grammaires algébriques on peut se restreindre aux dérivations *descendantes* (*top-down*).

Dans [Maibaum, 1974] on démontre que chaque grammaire algébrique peut être mise sous une forme normale où chaque règle est d'un des trois types :

$$A(x_1, \ldots, x_n) \rightarrow x_i \tag{3.3}$$

$$A(x_1, \ldots, x_n) \rightarrow f(x_{i_1}, \ldots, x_{i_m}) \tag{3.4}$$

$$A(x_1, \ldots, x_n) \rightarrow B(C_1(x_1, \ldots, x_n), \ldots, C_m(x_1, \ldots, x_n)) \tag{3.5}$$

On appelle les règles (3.3) les *règles effaçantes* (en anglais *projection rules* ou *collapsing rules*).

À part les règles effaçantes, les règles (3.4), (3.5) ont une forme analogue à la forme normale de Chomsky dans le cas des mots, où chaque règle est du type $A \rightarrow a$ ou $A \rightarrow BC$. C'est pour cela que la forme normale (3.3)-(3.5) avait été appelée dans [Maibaum, 1974] « la forme normale de Chomsky ». L'analogie n'est cependant pas complète à cause de la présence des règles effaçantes.

La possibilité d'éliminer les règles effaçantes dans le cas des mots, exprimée dans les formes normales de Chomsky et de Greibach, est une propriété importante dans les grammaires algébriques de mots. Elle permet de rendre les dérivations « croissantes », en ce sens que chaque pas de dérivation augmente la taille (ou un certain invariant lié à la taille) de la structure dérivée. Ainsi, sous la forme normale de Chomsky, une dérivation d'un mot de taille $k$ se fait en $2k - 1$ étapes, car chaque règle, soit introduit un non-terminal de plus, soit remplace un non-terminal par un terminal. Sous la forme normale de Greibach, où les règles ont la forme $A \rightarrow aB_1 \ldots B_n$, $n \geq 0$, un mot de taille $k$ est dérivé en $k$ étapes, car chaque règle introduit un et un seul terminal. Cette propriété agréable de « croissance » facilite la démonstration d'autres résultats. Elle

implique, par exemple, que le problème d'appartenance d'un mot à un langage algébrique est décidable, car on peut simplement tester toutes les dérivations engendrant les mots d'une taille donnée, dont le nombre est fini.

Cette discussion soulève donc la question de savoir si l'on peut éliminer les règles effaçantes des grammaires algébriques d'arbres. On peut démontrer que cela est possible dans le cas des grammaires linéaires. Pour ce faire, on peut ajouter dans la grammaire de nouvelles règles qui n'engendrent pas de non-terminaux qui auraient pu ensuite être éliminés par des règles effaçantes (cette technique est également à la base de la construction de [Arnold et Dauchet, 1976]). Or, cette idée d'« anticiper » l'application de règles effaçantes ne marche pas dans le cas de règles non-linéaires. La raison en est que les règles non-linéaires, en combinaison avec les règles effaçantes, permettent de dupliquer une structure pour ensuite en réduire chaque copie *d'une façon indépendante*. Ce phénomène ne peut pas être simulé dans une grammaire sans les règles effaçantes. Il s'est avéré qu'il n'est pas possible, dans le cas général, d'éliminer les règles effaçantes [Leguy, 1980; Dauchet et Tison, 1992]. Par conséquent, des étapes effaçantes sont indispensables dans les dérivations, et il n'existe donc pas d'analogues complets aux formes normales de Chomsky et de Greibach pour le cas des arbres.

Bien que les règles effaçantes soient indispensables, nous avons montré dans [Hofbauer *et al.*, 1998] qu'il est possible de « retarder » leur application autant que l'on veut. Plus exactement, chaque grammaire algébrique peut être transformée en une grammaire équivalente, où l'application de règles effaçantes sur les positions de profondeur inférieure à une profondeur donnée peut être évitée (qui plus est, avec notre construction cela est même impossible). Dans le reste de cette section nous définirons cette construction et en indiquerons quelques conséquences.

Pour $k \in \mathbb{N}$, on dira qu'une grammaire algébrique $G_k$ est une grammaire *terminale de niveau* $k$ (dans [Hofbauer *et al.*, 1998], *k-level terminal grammar*) si $G_k$ vérifie les conditions suivantes :

1. $G_k = (N \cup \bar{N}, \Sigma, P \cup \bar{P}, \bar{S})$, où $N \cap \bar{N} = \emptyset$, $\bar{S} \in \bar{N}$,
2. les règles de $P$ n'utilisent pas de non-terminaux de $\bar{N}$,
3. chaque règle de $\bar{P}$ a une des deux formes suivantes :

   (i)
   $$\bar{A}(x_1, \ldots, x_n) \to \bar{B}(s_1, \ldots, s_m), \tag{3.6}$$
   où $\bar{A}, \bar{B} \in \bar{N}$, et pour chaque $i, 1 \leq i \leq m$, soit $s_i = x_j$ $(1 \leq j \leq n)$, soit $s_i = C_i(x_1, \ldots, x_{k_i})$ pour un $C_i \in N$, $k_i \leq n$,

   (ii)
   $$\bar{A}(x_1, \ldots, x_n) \to t, \tag{3.7}$$
   où $t \in T(\Sigma \cup \{x_1, \ldots, x_n\})$ et toutes les variables n'apparaissent dans $t$ qu'à la profondeur $k+1$.

La grammaire $G_k$ est donc l'union de deux grammaires, l'une avec les non-terminaux $N$ définis par les règles $P$, et l'autre avec les non-terminaux $\bar{N}$ définis par les règles $\bar{P}$. Les règles de $P$ n'ont dans les parties droites que des non-terminaux de $N$, alors que les règles (3.6) de $\bar{P}$ ont des non-terminaux à la fois de $N$ et de $\bar{N}$. De plus, dans les parties droites des règles (3.6), les non-terminaux de $\bar{N}$ apparaissent uniquement à la racine, alors que les non-terminaux de $N$ apparaissent uniquement aux positions immédiatement en-dessous de la racine.

La structure d'une dérivation dans la grammaire $G_k$ est donc la suivante. La dérivation commence avec le symbole de départ $\bar{S} \in \bar{N}$. Ensuite, seules des règles de $\bar{P}$ sont applicables (on se restreint toujours aux dérivations descendantes). Plus précisément, on applique itérativement des règles (3.6) jusqu'à une première application d'une règle (3.7). Les règles (3.7) produisent un arbre avec des symboles terminaux à toutes les positions jusqu'à la profondeur $k$. Aucune règle

de $P$ (y compris les règles effaçantes que $P$ contient éventuellement) ne peut donc s'appliquer à ces positions. Les autres positions de cet arbre contiennent des non-terminaux de $N$. Le reste de la dérivation se fait donc avec les règles de $P$ uniquement – aucune règle de $\bar{P}$ ne sera plus appliquable.

Cette analyse de la structure de dérivations dans une grammaire $G_k$ permet une remarque intéressante : si les non-terminaux de $N$ sont vus comme des symboles terminaux, la grammaire $(\bar{N}, \Sigma \cup N, \bar{P}, \bar{S})$ devient une grammaire co-régulière (voir la section 3.3.1). Toute dérivation dans $G_k$ se décompose donc en deux étapes : la première est une dérivation « co-régulière » avec les règles $\bar{P}$ et la deuxième est une dérivation « algébrique » normale avec les règles $P$.

Notre intérêt pour les grammaires terminales de niveau $k$ se justifie par le théorème suivant.

**Théorème 26 ([Hofbauer *et al.*, 1998])** *Toute grammaire algébrique $G = (N, \Sigma, S, P)$ peut être transformée, pour tout $k \in \mathbb{N}$, en une grammaire terminale de niveau $k$ équivalente $G_k = (N \cup \bar{N}, \Sigma, \bar{S}, P \cup \bar{P})$.*

Notons que le $N$ et le $P$ dans la grammaire $G_k$ sont les mêmes que dans $G$. Il s'agit donc en fait d'une extension propre de la grammaire $G$. Notons aussi que cette extension peut être calculée avec un algorithme fourni par la preuve du théorème 26.

Une conséquence immédiate du théorème 26 est la décidabilité du problème de l'appartenance pour les langages algébriques d'arbres. Pour tester si un terme $t$ appartient au langage engendré par une grammaire $G$, il suffit de construire, selon le théorème 26, une grammaire équivalente $G_k$ où $k$ est la profondeur de $t$. Ensuite, le problème est réduit à un simple problème d'atteignabilité.

Bien que ce résultat de décidabilité ait déjà été connu, le théorème 26 en fournit, à notre connaissance, une première preuve directe. L'approche « traditionnelle » pour démontrer ce résultat [Aho, 1968; Fischer, 1968] consiste à démontrer la décidabilité du problème de la vacuité pour les langages algébriques d'arbres, qui à son tour est réduit au problème de la vacuité pour les langages de *feuillages* (*yield languages*) correspondants (ce sont les *indexed languages* ou *macro languages* [Aho, 1968; Fischer, 1968]). La décidabilité du problème de l'appartenance découle ensuite de la propriété de clôture des langages algébriques par rapport à l'intersection avec les langages réguliers [Rounds, 1970a; Rounds, 1970b]. Les articles [Damm, 1982; Engelfriet et Vogler, 1985] contiennent des preuves pour des classes de langages plus générales. Dans [Engelfriet, 1991] on démontre que le problème de la vacuité pour les langages de feuillages des langages algébriques d'arbres est EXPTIME-complet.

# Chapitre 4

# Arbres et mots – survol comparatif

Le but de cette section est de présenter une synthèse des résultats de complexité de quelques problèmes sur les ensembles $Inst(P)$ et $Cont(P)$ dans les cas des mots et des arbres. Nous reprendrons certains résultats de la section 2.3 (pour le cas des mots) et de la section 3.2 (pour le cas des arbres) en les replaçant dans un contexte plus général. Pour compléter le survol, nous évoquerons d'autres résultats connus de la littérature. Ce survol, contenant également quelques résultats nouveaux, a été publié dans [Kucherov et Rusinowitch, 1999].

## 4.1 Problèmes de décision

En comparant les cas des mots et d'arbres, notons qu'il y a au moins deux façons de représenter les mots à l'aide des arbres. L'une consiste à faire correspondre à chaque lettre un symbole de fonction unaire et d'ajouter dans la signature un symbole de constante spécial #. Ainsi, le mot *abaa* devient l'arbre non-branchant $a(b(a(a(\#))))$. Cependant, pour représenter les motifs de mots nous avons besoin d'introduire des variables à des nœuds internes d'arbres (variables du deuxième ordre) ce qui dépasse notre cadre. Une autre façon est d'associer à chaque lettre un symbole de constante et d'introduire un symbole binaire $\cdot$ correspondant à la concaténation. Dans ce cas, un mot peut être représenté par plusieurs arbres différents. Par exemple, *abaa* peut être représenté par $\cdot(\cdot(a,b),\cdot(a,a))$ mais aussi par $\cdot(a,\cdot(\cdot(b,a),a))$. Cette non-unicité correspond à la propriété d'associativité de la concaténation. Les mots peuvent donc être vus comme des arbres contenant un symbole de fonction associatif. On verra que cette propriété d'associativité joue un rôle crucial pour la complexité en rendant des problèmes dans le cas des mots en général beaucoup plus compliqués que leurs homologues dans le cas des arbres.

Les problèmes que nous allons analyser dans cette section sont listés dans le tableau 4.1. On note $u$ un mot ou un arbre, $p$ un motif (respectivement de mots ou d'arbres) et $P$ un ensemble de motifs. Rappelons que $Inst(p)$ et $Cont(p)$ sont des abréviations de $Inst(\{p\})$ et $Cont(\{p\})$ respectivement.

Ces questions, dont la plupart ont déjà été évoquées précédemment, sont fondamentales du point de vue de la théorie des langages. Les problèmes P1.1 et P1.2 sont des problèmes d'appartenance pour les langages $Inst(p)$ et $Cont(p)$ respectivement. Comme les langages $Inst(p)$ et $Cont(p)$ sont en général infinis, il est intéressant de poser le problème de la vacuité pour le complément $\overline{Inst(S)}$ et le problème de la co-finitude pour $Cont(P)$ (le problème de la vacuité pour ce dernier étant trivial). Les problèmes P2.2 et P5.2 s'intéressent à la structure de ces langages alors que les problèmes P3 et P4 représentent des propriétés d'inclusion.

Nous allons maintenant lister et brièvement commenter les résultats de complexité connus

| | |
|---|---|
| **P1.1** | $u \in Inst(p)$ ? |
| **P1.2** | $u \in Cont(p)$ ? |
| **P2.1** | $\overline{Inst(S)} = \emptyset$ ? |
| **P2.2** | $Inst(P)$ est-il régulier ? |
| **P3** | $Inst(p) \subseteq Inst(P)$ ? |
| **P4** | $Inst(p) \subseteq Cont(P)$ ? |
| **P5.1** | $\overline{Cont(P)}$ est-il fini ? |
| **P5.2** | $Cont(P)$ est-il régulier ? |

TAB. 4.1 – *Problèmes de décision*

pour chacun de ces problèmes dans le cas des arbres et ensuite dans le cas des mot.

## 4.2 Le cas des arbres

Le problème P1.1, dans le cas des arbres, consiste à tester si un terme est une instance d'un motif. Pour cela il suffit de vérifier que le motif coïncide avec le terme à toutes les positions des symboles de fonction dans le motif. De plus, il faut vérifier que les sous-termes correspondant aux occurrences d'une même variable sont égaux. Il est clair que tout ce travail peut se faire en temps $O(|u| + |p|)$.

Le problème P1.2 est le problème de recherche de sous-terme (*subterm matching*) qui a de nombreuses applications à la programmation logique et fonctionnelle, déduction automatique, systèmes de réécriture et d'autres domaines liés au calcul symbolique. Le problème consiste à tester si un motif donné apparaît dans un terme donné, c'est-à-dire tester si ce dernier contient un sous-terme qui est une instance du motif. La version restreinte du problème où le motif ne contient que des variables linéaires est appelé la *recherche de sous-arbre* (*tree matching* dans la littérature anglaise). Au début des années 1980, une solution simple et pratique, dont la complexité est de $O(|p||u|)$, avait été proposée dans l'article [Hoffmann et O'Donnell, 1982]. Une analyse de complexité en moyenne pour le problème de recherche de sous-arbre a été proposée dans [Steyaert et Flajolet, 1983]. Cette analyse montre en particulier que le temps moyen pris par l'algorithme « naïf » (parcours descendant de l'arbre et vérification de l'occurrence du motif à chaque sommet du parcours) est de $O(|p| + |u|)$. Plus récemment, une série de travaux ont été menés visant à améliorer la complexité théorique de ce problème dans le cas le pire. En ne citant que le dernier de ces résultats, les auteurs de [Cole *et al.*, 1999] ont proposé un algorithme résolvant le problème en temps $O(|u| \log^3 |u|)$ (ici, la taille du terme $|u|$ est supposé majorer la taille du motif $|p|$). En présence de variables non-linéaires, la complexité du problème P1.2 est $O(|p||u|)$, d'après les résultats de l'article [Ramesh et Ramakrishnan, 1992].

La complexité du problème P2.1 a été établie dans le théorème 21 : ce problème est co-NP-complet à la fois dans le cas des motifs linéaires et dans le cas général. Notons que le problème 2.1 a également été étudié dans le *cas associatif-commutatif*, c'est-à-dire lorsque l'ensemble des termes est factorisé par rapport aux propriétés d'associativité-commutativité d'un symbole de la signature [Lugiez et Moysset, 1993; Fernández, 1993]. Cependant, on ne sait pas si le problème est décidable ou non dans ce cas [Marcinkowski, 1999]. Le cas des mots, qui est le *cas associatif* (voir la discussion du début de ce chapitre), est discuté dans la section suivante.

Le problème P2.2 a été discuté dans la section 3.2.1. Comme nous l'avons indiqué dans cette section, la régularité de $Inst(P)$ est équivalent, selon [Kucherov, 1991], à l'existence d'une représentation finie du complément de $P$ et est donc décrite par le théorème 15. D'après le

théorème 19, ce problème est également co-NP-complet.

Le problème P3, étudié dans l'article [Lassez et Marriot, 1987], est de même nature que le problème P2.1 et peut être prouvé co-NP-complet par les même arguments [Kapur *et al.*, 1991]. Notons que ce problème est lié à la propriété de *complétude suffisante* (*sufficient completeness*), également étudiée dans la théorie des spécifications algébriques et des systèmes de réécriture [Dershowitz et Jouannaud, 1990].

L'historique du problème P4 remonte aux années 1980. Cette inclusion, connue sous le nom de *réductibilité inductive* (dans la littérature anglaise *ground reducibility, inductive reducibility or quasi-reducibility*), a attiré l'attention de nombreux chercheurs dans le domaine des systèmes de réécriture [Dershowitz et Jouannaud, 1990]. La raison en est que cette propriété a des applications importantes dans la démonstration automatique de théorèmes inductifs dans les théories équationnelles [Dershowitz, 1989; Jouannaud et Kounalis, 1989]. Le problème consiste donc à vérifier si chaque instance d'un motif donné $p$ contient un des motifs d'un ensemble donné $P$. Si l'on identifie les motifs de $P$ avec les membres gauches d'un système de réécriture $\mathcal{R}$, le problème revient donc à vérifier si toute instance du motif $p$ (dans la terminologie des systèmes de réécriture, *terme avec variable*) est réductible par le système de réécriture $\mathcal{R}$. Au milieu des années 1980, plusieurs auteurs ont observé que la réductibilité inductive est décidable si $P$ ne contient que des termes linéaires. Cependant, de même que pour les problèmes de la section 3.2.1, la réelle difficulté apparaît en présence dans $P$ de motifs non-linéaires. Dans ce cas général, D. Plaisted [Plaisted, 1985] et indépendemment plus tard d'autres auteurs [Kapur *et al.*, 1987; Comon, 1988; Kounalis, 1992] ont demontré que ce problème était décidable. Récemment, H. Comon et F. Jacquemard [Comon et Jacquemard, 1997] ont prouvé que ce problème était en fait EXPTIME-complet ; leur démonstration implique également que le problème reste EXPTIME-difficile dans le cas linéaire.

Les problèmes 5.1 et 5.2. ont été discutés dans la section 3.2.1. Le théorème 18 affirme qu'ils sont tous les deux décidables. Nous ne connaissons pas leur complexité exacte, mais nous conjecturons que le problème 5.1 est également EXPTIME-complet.

## 4.3 Le cas des mots

Nous allons maintenant passer en revue les résultats de complexité des problèmes P1.1-P5.2 dans le cas des mots. Nous verrons en particulier que la plupart de ces problèmes deviennent indécidables.

Selon un résultat de D.Angluin [Angluin, 1980], le problème P1.1 est co-NP-complet. Cela implique que le P1.2 est co-NP-complet aussi, car $u \in Inst(p)$ si et seulement si $\#u\# \in Cont(\#p\#)$ où $\#$ est une nouvelle lettre. Ces résultats illustrent déjà la hausse de complexité par rapport au cas des arbres où les problèmes P1.1 et P1.2 sont d'une complexité polynomiale. L'algorithme naïf résolvant les problèmes P1.1 et P1.2 marche respectivement en temps $O(|u|^{\Delta})$ et $O(|u|^{\Delta+2})$, où $\Delta$ est le nombre des variables distinctes dans $p$. J.Néraud [Néraud, 1995] a montré que l'exposant dans ces bornes de complexité peut être réduit de 2 (modulo un facteur polylogarithmique), il a également obtenu des algorithmes spécialisés pour le cas de petit $\Delta$ (1 ou 2). Notons cependant que si le motif $p$ est linéaire, les problèmes P1.1 et P1.2 deviennent linéaires, car ils reviennent au problème bien connu de recherche de motifs dans les mots et peuvent être résolus à l'aide d'un algorithme du type Knuth-Morris-Pratt [Crochemore et Rytter, 1995].

Selon la section 2.3.1, les problèmes P3 et P4 sont indécidables. Le problème P3 est indécidable même si $P$ ne contient qu'un seul motif (théorème 1) et le problème P4 indécidable même pour le motif $p$ fixé à $axa$ où $a$ est une lettre et $x$ une variable (théorème 2). Dans la section 2.3.2

nous avons démontré que le problème P2.1 est également indécidable. Enfin, dans la section 2.3.3 nous avons établi que dans le cas des motifs linéaires, les problèmes P2.1, P3, P4 et P5.1 sont co-NP-complets. Pour compléter ce tableau, nous analysons ici les autres cas.

Le statut de décidabilité du problème P2.2 est inconnu [Kari *et al.*, 1995]. On ne sait pas non plus si le problème inverse, qui consiste à décider si un langage régulier donné s'exprime comme $Inst(P)$, est décidable. Enfin, on ne sait pas s'il existe un algorithme décidant si un langage $Inst(P)$ est algébrique (*context-free*). Le problème inverse, savoir si un langage algébrique donné s'exprime comme $Inst(P)$, a été prouvé indécidable dans [Kari *et al.*, 1995].

Analysons maintenant le problème P5.1. Le résultat d'indécidabilité du théorème 4 pourrait suggérer que le problème P5.1 est également indécidable. En effet, en analysant la preuve du théorème 4 on s'aperçoit que ce qui empêche de la transformer en une preuve de l'indécidabilité du problème P5.1, ce sont deux les premiers groupes de motifs dans la construction (2.7). Ces deux groupes de motifs, extrêmement simples, expriment que les mots non couverts par les motifs doivent commencer et terminer par une lettre donnée. Or, cela ne peut pas être exprimé à l'aide l'ensemble $Cont(P)$ (l'introduction d'un motif $bx$ éliminerait tous les mots qui commencent par un $b$ mais aussi ceux qui contiennent la lettre $b$ à l'intérieur). Cette différence, d'apparence mineure, s'avère capitale. La raison en est que le fait de pouvoir « forcer » le mot à commencer et terminer par une lettre donnée permet d'appliquer l'idée de la preuve du théorème 2 où ces deux lettres jouent un rôle très important en permettant de spécifier la configuration initiale et finale du calcul de machine de Minsky. Dans le cas de $Cont(P)$, nous n'avons pas ce moyen.

On ne sait pas si le problème P5.1 est décidable ou non. Ce problème est en fait la version la plus générale du célèbre problème d'*inévitabilité* (*unavoidability*). Rappelons (voir la section 2.4.1) qu'un ensemble de motifs $P$ est *évitable* sur un alphabet donné s'il existe un nombre infini de mots ne contenant pas de motifs de $P$. Le problème d'inévitabilité a été posé dans la combinatoire des mots sous une forme restreinte où l'ensemble $P$ contient un seul motif $p$ composé uniquement de variables et ne contenant pas de lettres de l'alphabet $A$ sous-jacent (on appelle $p$ un motif *sans constantes*). Le statut de décidabilité de cette version restreinte est également ouvert [12]. Le terme *inévitabilité* s'explique par le fait que si $\overline{Cont(p)}$ est infini, il existe un mot infini de $A^\omega$ ne contenant pas d'occurrences de motif $p$. Dans la section 2.4.1 nous avons déjà constaté que l'évitabilité d'un motif dépend de l'alphabet sous-jacent. Par exemple, le motif $xx$ est évitable dans un alphabet à trois lettres qui n'est pas évitable dans un alphabet à deux lettres. Un exemple d'un motif évitable sur quatre lettres mais inévitable sur trois lettres est donné dans [Bean *et al.*, 1979]. Cependant, on ne connaît pas de motif évitable sur $k$ lettres mais inévitable sur $k-1$ lettres pour $k > 4$. Les articles [Bean *et al.*, 1979; Zimin, 1984] ont indépendamment proposé un algorithme capable de vérifier, pour un motif donné sans constantes, *s'il existe un alphabet* sur lequel ce motif devient évitable. Cependant, ces algorithmes ne donnent pas la borne inférieure exacte de la taille de l'alphabet en cas de réponse positive, et par conséquent le problème de l'évitabilité d'un motif sans constantes *pour une taille d'alphabet donnée* reste ouverte. La thèse [Cassaigne, 1994] contient une excellente présentation de la théorie de l'inévitabilité dans le cas d'un motif sans constantes. Le survol [Choffrut et Karhumäki, 1997] est également une référence très utile.

Le tableau 4.2 récapitule les résultats de complexité pour les problèmes P1.1-P5.2 dans les cas des mots et des arbres.

12. L'auteur de l'article [Currie, 1993] offre 100 dollars américains pour une solution de ce problème.

| Problème | Cas des arbres | | Cas des mots | |
|---|---|---|---|---|
| | Cas linéaire | Cas général | Cas linéaire | Cas général |
| P1.1 $u \in Inst(p)$? | $O(\|p\|)$ | $O(\|u\|)$ | $O(\|p\| + \|u\|)$ | co-NP-complet<br>[Angluin, 1980] |
| P1.2 $u \in Cont(p)$? | $O(\|u\| \log^3 \|u\|)$<br>[Cole *et al.*, 1999] | $O(\|p\|\|u\|)$<br>[Ramesh et Ramakrishnan, 1992] | $O(\|p\| + \|u\|)$ | co-NP-complet<br>[Angluin, 1980] |
| P2.1 $\overline{Inst(S)} = \emptyset$? | co-NP-complet<br>[Kapur *et al.*, 1991] | co-NP-complet<br>[Kapur *et al.*, 1991] | co-NP-complet<br>théorème 7(1) | indécidable<br>théorème 3 |
| P2.2 $Inst(P)$ est-il régulier? | oui | co-NP-complet<br>[Lassez et Marriot, 1987]<br>[Kucherov, 1991]<br>[Maher et Stuckey, 1995] | oui | ouvert<br>[Kari *et al.*, 1995] |
| P3 $Inst(p) \subseteq Inst(P)$? | co-NP-complet<br>[Kapur *et al.*, 1991] | co-NP-complet<br>[Kapur *et al.*, 1991] | co-NP-complet<br>théorème 7(2) | indécidable<br>[Jiang *et al.*, 1993] |
| P4 $Inst(p) \subseteq Cont(P)$? | EXPTIME-complet<br>[Comon et Jacquemard, 1997] | EXPTIME-complet<br>[Comon et Jacquemard, 1997] | co-NP-complet<br>[Kucherov et Rusinowitch, 1995a] | indécidable<br>[Kucherov et Rusinowitch, 1994] |
| P5.1 $\overline{Cont(P)}$ est-il fini? | décidable<br>(complexité exacte inconnue) | décidable<br>[Plaisted, 1985]<br>[Kapur *et al.*, 1987] | co-NP-complet<br>[Kucherov et Rusinowitch, 1995a] | ouvert |
| P5.2 $Cont(P)$ est-il régulier? | oui | décidable<br>[Kucherov et Tajine, 1995] | oui | ouvert |

TAB. 4.2 – *Complexité des problèmes P1.1-P5.2 dans les cas des mots et des arbres*

# Bibliographie


[Aho, 1968] A. V. Aho. Indexed Grammars–An Extension of Context-Free-Grammars. *Journal of the ACM*, 15(4):641–671, 1968.

[Allouche et Bousquet-Mélou, 1995] J.-P. Allouche et M. Bousquet-Mélou. On the conjectures of Rauzy and Shallit for infinite words. *Comment. Math. Univ. Carolinae*, 36:705–711, 1995.

[Amir *et al.*, 1993] A. Amir, M. Farach, R. M. Idury, J. A. La Poutré et A. A. Schäffer. Improved dynamic dictionary matching. Dans *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, Austin (TX)*, pages 392–401, Janvier 1993. to appear in Information and Computation.

[Amir *et al.*, 1994] A. Amir, M. Farach, Z. Galil, R. Giancarlo et P. Kunsoo. Fully dynamic dictionary matching. *Journal of Computer and System Sciences*, 49:208–222, 1994.

[Amir et Farach, 1991] A. Amir et M. Farach. Adaptive dictionary matching. Dans *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, San Juan (Puerto Rico)*, pages 760–766. IEEE computer society press, Octobre 1991.

[Angluin, 1980] D. Angluin. Finding patterns common to a set of strings. *J. Comput. System Sci.*, 21:46–62, 1980.

[Apostolico et Preparata, 1983] A. Apostolico et F.P. Preparata. Optimal off-line detection of repetitions in a string. *Theoretical Computer Science*, 22(3):297–315, 1983.

[Arnold et Dauchet, 1976] A. Arnold et M. Dauchet. Un Théorème de Duplication pour les Forêts Algébriques. *Journal of Computer and System Sciences*, 13:223–244, 1976.

[Bean *et al.*, 1979] D.R. Bean, A. Ehrenfeucht et G.F. McNulty. Avoidable patterns in strings of symbols. *Pacific J. Math.*, 85(2):261–294, 1979.

[Berstel, 1992] J. Berstel. Axel thue's work on repetitions in words. Invited Lecture at the 4th Conference on Formal Power Series and Algebraic Combinatorics, Montreal, 1992, June 1992. disponible à l'adresse `http://www-igm.univ-mlv.fr/~berstel/index.html`.

[Bertault et Kucherov, 1998] F. Bertault et G. Kucherov. Visualization of Dynamic Automata using `Padnon`. Dans *Workshop on Implementing Automata, Ontario, Canada, 1997*, volume 1436 de *Lecture Notes in Computer Science*, pages 25–28. Springer Verlag, Septembre 1998.

[Bird et Wadler, 1988] R. J. Bird et Ph. Wadler. *Introduction to functional programming.* Prentice-Hall, 1988.

[Blumer *et al.*, 1985] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen et J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

[Blumer *et al.*, 1987] A. Blumer, J. Blumer, D. Haussler, R. McConnell et A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, Juillet 1987.

[Book et Otto, 1993] R. V. Book et F. Otto. *String-Rewriting Sytsems.* Springer-Verlag, 1993.

[Book, 1987] R. V. Book. Thue systems as rewriting systems. *Journal of Symbolic Computation*, 3(1 & 2):39–68, 1987.

[Brandstädt *et al.*, 1999] A. Brandstädt, V. B. Le et J. P. Spinrad. *Graph Classes: A Survey.* SIAM Monographs on Discrete Mathematics and Applications. SIAM, 1999.

[Brodal *et al.*, 1999] G. Brodal, R. Lyngsø, Ch. Pedersen et J. Stoye. Finding maximal pairs with bounded gap. Dans *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching*, rédacteurs M. Crochemore et M. Paterson, volume 1645 de *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[Buchanan et Shortliffe, 1984] rédacteurs Bruce G. Buchanan et Edward H. Shortliffe. *Rule-Based Expert Systems.* Addison-Wesley, Reading, MA, 1984.

[Cassaigne, 1993] J. Cassaigne. Counting overlap-free binary words. Dans *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS'93)*, rédacteurs Patrice Enjalbert, Alain Finkel et Klaus W. Wagner, volume 665 de *Lectures Notes in Computer Science*, pages 216–225, Berlin, Germany, February 1993. Springer Verlag.

[Cassaigne, 1994] J. Cassaigne. *Motifs évitables et régularités dans les mots.* Thèse de doctorat, Université Paris VI, 1994.

[Choffrut et Karhumäki, 1997] Ch. Choffrut et J. Karhumäki. Combinatorics of words. Dans *Handbook on Formal Languages*, rédacteurs G. Rozenberg et A. Salomaa, volume I. Springer Verlag, Berlin-Heidelberg-New York, 1997.

[Cole *et al.*, 1999] R. Cole, R. Hariharan et P. Indyk. Tree pattern matching and subset matching in deterministic $o(n \log^3 n)$-time. Dans *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltymore, Maryland, January 17-19, 1999*, pages 245–254. ACM, SIAM, 1999.

[Cole et Hariharan, 1999] R. Cole et R. Hariharan. Dynamic LCA queries on trees. Dans *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 235–244, N.Y., January 17–19 1999. ACM-SIAM.

[Comon *et al.*, 1997] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison et M. Tommasi. Tree automata techniques and applications. Disponible à l'adresse `http://www.grappa.univ-lille3.fr/tata`, 1997.

[Comon et Jacquemard, 1997] H. Comon et F. Jacquemard. Ground reducibility is EXPTIME-complete. Dans *Proceedings, Twelth Annual IEEE Symposium on Logic in Computer Science*, pages 26–34, Warsaw, Poland, 29 Juin–2 Juillet 1997. IEEE Computer Society Press.

[Comon, 1988] H. Comon. *Unification et disunification. Théories et applications.* Thèse de Doctorat d'Université, Institut Polytechnique de Grenoble (France), 1988.

[Crochemore et Rytter, 1994] M. Crochemore et W. Rytter. *Text algorithms.* Oxford University Press, 1994.

[Crochemore et Rytter, 1995] M. Crochemore et W. Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13:405–425, 1995.

[Crochemore, 1981] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12:244–250, 1981.

[Crochemore, 1983] M. Crochemore. Recherche linéaire d'un carré dans un mot. *Comptes Rendus Acad. Sci. Paris Sér. I Math.*, 296:781–784, 1983.

[Crochemore, 1986] M. Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45:63–86, 1986.

[Crochemore, 1988] Maxime Crochemore. String matching with constraints. Dans *Proceedings*

*International Symposium on Mathematical Foundations of Computer Science*, volume 324 de *Lecture Notes in Computer Science*, pages 44–58. Springer-Verlag, 1988.

[Currie et Shelton, 1996] J.D. Currie et R.O. Shelton. Cantor sets and Dejean's conjecture. *Journal of Automata, Languages and Combinatorics*, 1(2):113–128, 1996.

[Currie, 1993] James Currie. Open problems in pattern avoidance. *American Mathematical Monthly*, 100:790–793, 1993.

[Damm, 1982] W. Damm. The IO- and OI-Hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.

[Dauchet et Tison, 1992] M. Dauchet et S. Tison. Structural Complexity of Classes of Tree Languages. Dans *Tree Automata and Languages*, pages 327–353. Elsevier Science Publishers B. V. (North-Holland), 1992.

[Dejean, 1972] F. Dejean. Sur un théorème de Thue. *J. Combinatorial Th. (A)*, 13:90–99, 1972.

[Dekking, 1992] M. Dekking. On the Thue-Morse measure. *Acta Univ. Carolin. Math. Phis*, 33(2):35–40, 1992.

[Dershowitz *et al.*, 1991] N. Dershowitz, J.-P. Jouannaud et J. W. Klop. Open problems in rewriting. Dans *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, rédacteur R. V. Book, volume 488 de *Lecture Notes in Computer Science*, pages 445–456. Springer-Verlag, 1991.

[Dershowitz et Jouannaud, 1990] N. Dershowitz et J.-P. Jouannaud. Rewrite Systems. Dans *Handbook of Theoretical Computer Science*, rédacteur J. van Leeuwen, chapitre 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[Dershowitz, 1989] N. Dershowitz. Completion and its applications. Dans *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, rédacteurs H. Aït-Kaci et M. Nivat, pages 31–86. Academic Press inc., 1989.

[Diestel, 1996] R. Diestel. *Graph Theory*, volume 173 de *Graduate Texts in Mathematics*. Springer, 1996.

[Engelfriet et Schmidt, 1977] J. Engelfriet et E. M. Schmidt. IO and OI. I. *Journal of Computer and System Sciences*, 15:329–353, 1977.

[Engelfriet et Vogler, 1985] J. Engelfriet et H. Vogler. Macro Tree Transducers. *Journal of Computer and System Sciences*, 31:71–146, 1985.

[Engelfriet, 1991] J. Engelfriet. Iterated Stack Automata and Complexity Classes. *Information and Computation*, 95:21–75, 1991.

[Fernández, 1993] M. Fernández. AC-complement problems: Validity and negation elimination. Dans *Proceedings of the Fifth International Conference on Rewriting Techniques and Applications (Montreal, Canada)*, rédacteur C. Kirchner, volume 690 de *Lecture Notes in Computer Science*, pages 358–373, Berlin, 1993. Springer-Verlag.

[Filè, 1988] G. Filè. The relation of two patterns with comparable languages. Dans *Proceedings of the 5th Annual Symposium on Theoretical Aspects of Computer Science (STACS '88)*, rédacteur R. Cori M. Wirsing, volume 294 de *Lecture Notes in Computer Science*, pages 184–192, Bordeaux, France, February 1988. Springer.

[Fischer, 1968] M.J. Fischer. Grammars with Macro-Like Productions. Dans *9th Annual IEEE Symposium on Switching and Automata Theory*, pages 131–142, 1968. voir aussi la PhD thesis, Harvard University, 1968.

[Fisher et Paterson, 1974] M. J. Fisher et M. S. Paterson. String-matching and other products. Dans *Complexity of Computation*, rédacteur R. M. Karp, volume 7 de *SIAM-AMS Proceedings*, pages 113–125. American Mathematical Society, Providence, RI, 1974.

[Fraenkel et Simpson, 1998] A.S. Fraenkel et J. Simpson. How many squares can a string contain? *J. Combinatorial Theory (Ser. A)*, 82:112–120, 1998.

[Fraenkel et Simpson, 1999] A.S. Fraenkel et J. Simpson. The exact number of squares in Fibonacci words. *Theoretical Computer Science*, 218(1):83–94, 1999.

[Garey et Johnson, 1979] M. Garey et D. Johnson. *Computers and Intractability. A guide to the theory of NP-completeness.* W. Freeman and Compagny, New York, 1979.

[Gécseg et Steinby, 1984] F. Gécseg et M. Steinby. *Tree automata.* Akadémiai Kiadó, Budapest, 1984.

[Gésceg et Steinby, 1997] F. Gésceg et M. Steinby. Tree languages. Dans *Handbook on Formal Languages*, rédacteurs G. Rozenberg et A. Salomaa, volume 3, pages 1–68. Springer Verlag, 1997.

[Giraud et Kucherov, 2000] M. Giraud et G. Kucherov. Maximal repetitions and application to DNA sequences. Dans *Proceedings of the* Journées Ouvertes : Biologie, Informatique et Mathématiques, pages 165–172, Montpelier, 3-5 mai 2000.

[Gottlob et Pichler, 1999] G. Gottlob et R. Pichler. Working with ARMs: Complexity results on atomic representations of Herbrand models. Dans *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science, 2-5 July, 1999, Trento, Italy*, pages 306–315. IEEE Computer Society, 1999.

[Gusfield, 1997] D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.

[Harel et Tarjan, 1984] D. Harel et R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal of Computing*, 13(2):338–355, 1984.

[Hofbauer *et al.*, 1994] D. Hofbauer, M. Huber et G. Kucherov. Some results on top-context-free tree languages. Dans *Proceedings of the 19th International Colloquium on Trees in Algebra and Programming*, volume 787 de *Lecture Notes in Computer Science*, pages 157–171. Springer Verlag, 1994.

[Hofbauer *et al.*, 1998] D. Hofbauer, M. Huber et G. Kucherov. *How to get rid of projection rules in context-free tree grammars*, chapitre 15, pages 235–247. Studies in Logic, Language and Information. Center for the Study of Language and Information (CSLI), Stanford and The European Association for Logic, Language and Information (FoLLI), 1998.

[Hofbauer et Huber, 1992] D. Hofbauer et M. Huber. Computing linearizations using test sets. Dans *Proceedings 3rd International Workshop on Conditional Term Rewriting Systems, Pont-à-Mousson (France)*, rédacteurs M. Rusinowitch et J.-L. Rémy, volume 656 de *Lecture Notes in Computer Science*, pages 145–149. Springer-Verlag, Avril 1992. version complète dans [Hofbauer et Huber, 1994].

[Hofbauer et Huber, 1994] D. Hofbauer et M. Huber. Linearizing term rewriting systems using test sets. *Journal of Symbolic Computation*, 17(1):91–129, January 1994.

[Hoffmann et O'Donnell, 1982] C. M. Hoffmann et M. J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, 1982.

[Idury et Schäffer, 1994] R. M. Idury et A. A. Schäffer. Dynamic dictionary matching with failure functions. *Theoretical Computer Science*, 131:295–310, 1994.

[Iliopoulos *et al.*, 1997] C.S. Iliopoulos, D. Moore et W.F. Smyth. A characterization of the squares in a Fibonacci string. *Theoretical Computer Science*, 172:281–291, 1997.

[Jiang *et al.*, 1993] T. Jiang, A. Salomaa, K. Salomaa et S. Yu. Inclusion is undecidable for pattern languages. Dans *Automata, Languages and Programming, 20th International Colloquium*,

rédacteur Svante Carlsson Andrzej Lingas, Rolf G. Karlsson, volume 700 de *Lecture Notes in Computer Science*, pages 301–312, Lund, Sweden, 5–9 Juillet 1993. Springer-Verlag.

[Jiang *et al.*, 1995] T. Jiang, A. Salomaa, K. Salomaa et Sh. Yu. Decision problems for patterns. *Journal of Computer and System Sciences*, 50(1):53–63, February 1995.

[Jouannaud et Kounalis, 1989] J.-P. Jouannaud et E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82:1–33, 1989.

[Justin et Pirillo, 1999] J. Justin et G. Pirillo. Fractional powers in Sturmian words. Rapport Technique LIAFA 99/01, Laboratoire d'Informatique Algorithmique: Fondements et Applications (LIAFA), 1999.

[Kapur *et al.*, 1987] D. Kapur, P. Narendran et H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24:395–415, 1987.

[Kapur *et al.*, 1991] D. Kapur, P. Narendran, D. J. Rosenkrantz et H. Zhang. Sufficient completeness, ground-reducibility and their complexity. *Acta Informatica*, 28:311–350, 1991.

[Kari *et al.*, 1995] L. Kari, A. Mateescu, G. Paun et A. Salomaa. Multi-pattern languages. *Theoretical Computer Science*, 141:253–268, 1995.

[Kfoury, 1988] A.J. Kfoury. A linear time algorithm testing whether a word contains an overlap. *RAIRO Inf. Th.*, 22:135–145, 1988.

[Kolpakov *et al.*, 1998] R. Kolpakov, G. Kucherov et Yu. Tarannikov. On repetition-free binary words of minimal density. Dans *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS), Brno (Czech Republic)*, volume 1450 de *Lecture Notes in Computer Science*, pages 683–692. Springer Verlag, 1998. version complète dans [Kolpakov *et al.*, 1999].

[Kolpakov *et al.*, 1999] R. Kolpakov, G. Kucherov et Yu. Tarannikov. On repetition-free binary words of minimal density. *Theoretical Computer Science*, 218(1), 1999.

[Kolpakov et Kucherov, 1997] R. Kolpakov et G. Kucherov. Minimal letter frequency in $n$-power-free binary words. Dans *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS), Bratislava (Slovakia)*, rédacteurs Igor Privara et Peter Ružička, volume 1295 de *Lecture Notes in Computer Science*, pages 347–357. Springer Verlag, 1997.

[Kolpakov et Kucherov, 1999a] R. Kolpakov et G. Kucherov. Finding maximal repetitions in a word in linear time. Dans *Proceedings of the 1999 Symposium on Foundations of Computer Science, New York (USA)*. IEEE Computer Society, October 17-19 1999.

[Kolpakov et Kucherov, 1999b] R. Kolpakov et G. Kucherov. On maximal repetitions in words. Dans *Proceedings of the 12-th International Symposium on Fundamentals of Computation Theory, 1999, Iasi (Romania)*, Lecture Notes in Computer Science, August 30 - September 3 1999.

[Kolpakov et Kucherov, 2000a] R. Kolpakov et G. Kucherov. Finding repeats with fixed gap. Dans *Proceedings of the 7th International Symposium on String Processing and Information Retrieval (SPIRE), A Coruña, Spain (27 - 29 Septembre, 2000)*, pages 162–168. IEEE, Septembre 2000.

[Kolpakov et Kucherov, 2000b] R. Kolpakov et G. Kucherov. On maximal repetitions in words. *Journal on Discrete Algorithms*, 2000. à paraître.

[Kosaraju, 1994] S. R. Kosaraju. Computation of squares in string. Dans *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, rédacteurs M. Crochemore et D. Gusfield, numéro 807 dans Lecture Notes in Computer Science, pages 146–150. Springer Verlag, 1994.

[Kounalis, 1992] E. Kounalis. Testing for the ground (co-)reducibility property in term-rewriting systems. *Theoretical Computer Science*, 106:87–117, 1992.

[Kucherov et Rusinowitch, 1994] G. Kucherov et M. Rusinowitch. On Ground-Reducibility Problem for Word Rewriting Systems with Variables. Dans *Proceedings 1994 ACM/SIGAPP Symposium on Applied Computing*, rédacteurs E. Deaton et R. Wilkerson, Phoenix (USA), mar 1994. ACM-Press.

[Kucherov et Rusinowitch, 1995a] G. Kucherov et M. Rusinowitch. Complexity of testing ground reducibility for linear word rewriting systems with variables. Dans *Proceedings 4th International Workshop on Conditional and Typed Term Rewriting Systems, Jerusalem (Israel)*, volume 968 de *Lecture Notes in Computer Science*, pages 262–275. Springer-Verlag, 1995.

[Kucherov et Rusinowitch, 1995b] G. Kucherov et M. Rusinowitch. Matching a set of strings with variable length don't cares. Dans *Proceedings of the 6th Symposium on Combinatorial Pattern Matching*, rédacteur E. Ukkonen, volume 937 de *Lecture Notes in Computer Science*, pages 230–247, Helsinki (Finland), Juillet 1995. Springer-Verlag.

[Kucherov et Rusinowitch, 1995c] G. Kucherov et M. Rusinowitch. Undecidability of ground reducibility for word rewriting systems with variables. *Information Processing Letters*, 53:209–215, 1995.

[Kucherov et Rusinowitch, 1997] G. Kucherov et M. Rusinowitch. Matching a set of strings with variable length don't cares. *Theoretical Computer Science*, 178:129–154, 1997.

[Kucherov et Rusinowitch, 1999] Gregory Kucherov et Michaël Rusinowitch. Patterns in words vs patterns in trees: a brief survey and some new results. Dans *Proceedings of the Andrei Ershov 3rd International Conference "Perspectives of System Informatics" (6 - 9 July 1999, Novosibirsk, Akademgorodok, Russia)*, volume 1755 de *Lecture Notes in Computer Science*, pages 280–293. Springer Verlag, 1999.

[Kucherov et Tajine, 1992] G. Kucherov et M. Tajine. Decidability of regularity and related properties of ground normal form languages. Dans *Proceedings 3rd International Workshop on Conditional Term Rewriting Systems, Pont-à-Mousson (France)*, rédacteurs M. Rusinowitch et J.-L. Rémy, volume 656 de *Lecture Notes in Computer Science*, pages 150–156. Springer-Verlag, Avril 1992. version complète dans [Kucherov et Tajine, 1995].

[Kucherov et Tajine, 1995] G. Kucherov et M. Tajine. Decidability of regularity and related properties of ground normal form languages. *Information and Computation*, 118(1):91–100, Avril 1995.

[Kucherov, 1988] G. Kucherov. A new quasi-reducibilty testing algorithm and its application to proofs by induction. Dans *Proceedings of the 1st International Workshop on Algebraic and Logic Programming, Gaussig (GDR)*, rédacteurs J. Grabowski, P. Lescanne et W. Wechler, volume 343 de *Lecture Notes in Computer Science*, pages 204–213. Springer Verlag, 1988.

[Kucherov, 1991] G. A. Kucherov. On relationship between term rewriting systems and regular tree languages. Dans *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*, rédacteur R. V. Book, volume 488 de *Lecture Notes in Computer Science*, pages 299–311. Springer-Verlag, Avril 1991.

[Lassez *et al.*, 1991] J.-L. Lassez, M. Maher et K. Marriott. Elimination of negation in term algebras. Dans *Proceedings of Mathematical Foundations of Computer Science. (MFCS'91)*, rédacteur A. Tarlecki, volume 520 de *Lecture Notes in Computer Science*, pages 1–16, Berlin, September 1991. Springer.

[Lassez et Marriot, 1987] J.-L. Lassez et K. Marriot. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–318, 1987.

[Lazrek *et al.*, 1990] A. Lazrek, P. Lescanne et J.-J. Thiel. Tools for proving inductive equalities, relative completeness and $\omega$-completeness. *Information and Computation*, 84(1):47–70, Janvier 1990.

[Leguy, 1980] B. Leguy. *Réductions, Transformations et Classification des Grammaires Algébriques d'Arbres*. Thèse de Doctorat de Troisième Cycle, Université des Sciences et Techniques de Lille (France), 1980.

[Lempel et Ziv, 1976] A. Lempel et J. Ziv. On the complexity of finite sequences. *IEEE Trans. Inf. Theory IT-22*, pages 75–81, Jan 1976.

[Lugiez et Moysset, 1993] D. Lugiez et J. L. Moysset. Complement problems and tree automata in AC-like theories. Dans *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS '93)*, rédacteurs Patrice Enjalbert, Alain Finkel et Klaus W. Wagner, volume 665 de *Lecture Notes in Computer Science*, pages 515–524, Berlin, Germany, February 1993. Springer.

[Maher et Stuckey, 1995] M. Maher et P. Stuckey. On inductive inference of cyclic structures. *Annals of Mathematics and Artificial Intelligence*, 15(2):167–208, 1995.

[Maibaum, 1974] T. S. E. Maibaum. A Generalized Approach to Formal Languages. *Journal of Computer and System Sciences*, 8:409–439, 1974.

[Main et Lorentz, 1984] M.G. Main et R.J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984.

[Main et Lorentz, 1985] M.G. Main et R.J. Lorentz. Linear time recognition of square free strings. Dans *Combinatorial Algorithms on Words*, rédacteurs A. Apostolico et Z. Galil, volume 12 de *NATO Advanced Science Institutes, Series F*, pages 272–278. Springer Verlag, 1985.

[Main, 1989] M. G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25:145–153, 1989.

[Marchenko, 1982] S.S. Marchenko. Undecidability of the positive $\forall\exists$-theory of a free semigroup. *Sibirskii Matematicheskii Zhurnal*, 23(1):196–198, 1982. in Russian.

[Marcinkowski, 1999] J. Marcinkowski. Undecidability of the $\exists^*\forall^*$ part of the theory of ground term algebra modulo an AC symbol. Dans *Proccedings of the 10th Conference on Rewriting Techniques and Applications*, volume 1631 de *Lecture Notes in Computer Science*, pages 92–104. Springer Verlag, 1999.

[McCreight, 1976] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[Mignosi *et al.*, 1995] F. Mignosi, A. Restivo et S. Salemi. A periodicity theorem on words and applications. Dans *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 969 de *Lecture Notes in Computer Science*, pages 337–348. Springer Verlag, 1995.

[Mignosi et Pirillo, 1992] F. Mignosi et G. Pirillo. Repetitions in the Fibonacci infinite word. *RAIRO Theoretical Informatics and Applications*, 26(3):199–204, 1992.

[Minsky, 1961] M. L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, Novembre 1961.

[Myers, 1992] Gene Myers. A four russians algorithm for regular expression pattern matching. *Journal of the ACM*, 39(4):430–448, Avril 1992.

[Néraud, 1995] J. Néraud. Detecting morphic images of a word: On the rank of a pattern. *Acta Informatica*, 32:477–489, 1995.

[Nivat et Podelski, 1992] rédacteurs M. Nivat et A. Podelski. *Tree Automata and Languages.* Studies in Computer Science and Artificial Intelligence 10. North-Holland, 1992.

[Ohlebusch et Ukkonen, 1997] E. Ohlebusch et E. Ukkonen. On the equivalence problem for E-pattern languages. *Theoretical Computer Science*, 186(1–2):231–248, Octobre 1997.

[Pichler, 1999] R. Pichler. The explicit representability of implicit generalizations. submitted, april 1999.

[Pichler, 2000] R. Pichler. The explicit representability of implicit generalizations. Dans *Proccedings of the 11th Conference on Rewriting Techniques and Applications, Norwich (UK)*, rédacteur L. Bachmair, volume 1833 de *Lecture Notes in Computer Science*, pages 187–202. Springer Verlag, July 2000.

[Plaisted et Kucherov, 1999] D. Plaisted et G. Kucherov. The complexity of some complementation problems. *Information Processing Letters*, 71:159–165, 1999.

[Plaisted, 1985] D. Plaisted. Semantic confluence and completion method. *Information and Control*, 65:182–215, 1985.

[Ramesh et Ramakrishnan, 1992] R. Ramesh et I.V. Ramakrishnan. Nonlinear pattern matching in trees. *Journal of the ACM*, 39(2):295–316, Avril 1992.

[Rauzy, 1983] G. Rauzy. Suites à termes dans un alphabet fini. Dans *Séminaire de Théorie des Nombres, année 1982–1983.* Université Bordeaux 1, 1983. Exposé 25, pages 25-01–25-16.

[Restivo et Salemi, 1983] Antonio Restivo et Sergio Salemi. On weakly square free words. *Bull. of the EATCS*, 21:49–56, 1983.

[Robertson et Seymour, 1990] N. Robertson et P. Seymour. Graph minors iv, tree-width and well-quasi-ordering. *Journal of Combinatorial Theory, Ser. B*, 48(2):227–254, 1990.

[Rounds, 1970a] W. C. Rounds. Mappings and Grammars on Trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.

[Rounds, 1970b] W. C.. Rounds. Tree-Oriented Proofs of some Theorems on Context-Free and Indexed Languages. Dans *Proceedings of 2nd Annual Symposium on Theory of Computing*, pages 109–116, 1970.

[Rozenberg et Salomaa, 1997] G. Rozenberg et A. Salomaa. *Handbook of Formal Languages. (Vol 3) Beyond Words.* Springer Verlag, Berlin, 1997.

[Salomaa, 1981] A. Salomaa. *Jewels of Formal Language Theory.* Computer Science Press, 1981.

[Salomaa, 1988] K. Salomaa. Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems. *Journal of Computer and System Sciences*, 37:367–394, 1988.

[Salomaa, 1993] A. Salomaa. Pattern Languages: Problems of Decidability and Generation. Dans *Proceedings of 9th FCT Conference, Szeged (Hungary)*, rédacteur Z. Ésik, volume 710 de *Lecture Notes in Computer Science*, pages 121–132. Springer Verlag, 1993.

[Salomaa, 1994] A. Salomaa. Patterns. *Bulletin of the European Association for Theoretical Computer Science. The Formal Language Theory Column.*, 54:194–206, October 1994.

[Salomaa, 1995] A. Salomaa. Return to patterns. *Bulletin of the European Association for Theoretical Computer Science. The Formal Language Theory Column.*, 55:144–157, February 1995.

[Schieber et Vishkin, 1988] B. Schieber et U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM Journal of Computing*, 17(6):1253–1262, 1988.

[Shallit et Breitbart, 1994] J. Shallit et Y. Breitbart. Automaticity: Properties of a measure of descriptional complexity. Dans *Proceedings of STACS'94*, volume 775 de *Lecture Notes in Computer Science*, pages 619–630, 1994.

[Shinohara et Arikawa, 1995] T. Shinohara et S. Arikawa. Pattern inference. Dans *Algorithmic Learning for Knowledge-Based Systems*, rédacteurs Klaus P. Jantke et Steffen Lange, volume 961 de *Lecture Notes in Artificial Intelligence*, pages 259–291. Springer-Verlag, 1995.

[Shinohara, 1982] T. Shinohara. Polynomial time inference of pattern langages and its applications. Dans *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science, Mathematical Theory of Computations/The Complexity of Algorithms*, pages 191–209, 1982.

[Sleator et Tarjan, 1983] D. D. Sleator et R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26:362–391, 1983.

[Slisenko, 1983] A.O. Slisenko. Detection of periodicities and string matching in real time. *Journal of Soviet Mathematics*, 22:1316–1386, 1983.

[Stephen, 1994] G.A. Stephen. *String Searching Algorithms*. World Scientific, Singapore, 1994.

[Steyaert et Flajolet, 1983] J.-M. Steyaert et Ph. Flajolet. Patterns and pattern-matching in trees: An analysis. *Information and Control*, 58(1–3):19–58, July/August/September 1983.

[Stoye et Gusfield, 1998a] J. Stoye et D. Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. Dans *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, rédacteur M. Farach-Colton, numéro 1448 dans Lecture Notes in Computer Science, pages 140–152. Springer Verlag, 1998.

[Stoye et Gusfield, 1998b] J. Stoye et D. Gusfield. Linear time algorithms for finding and representing all the tandem repeats in a string. Rapport Technique CSE-98-4, Computer Science Department, University of California, Davis, 1998.

[Thiel, 1984] J.-J. Thiel. Stop losing sleep over incomplete data type specifications. Dans *Proceeding 11th ACM Symp. on Principles of Programming Languages*, pages 76–82. ACM, 1984.

[Thue, 1906] A. Thue. Über unendliche Zeichenreihen. *Norske Vid. Selsk. Skr. I. Mat. Nat. Kl. Christiania*, 7:1–22, 1906.

[Thue, 1912] A. Thue. Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen. *Norske Vid. Selsk. Skr. I. Mat. Nat. Kl. Christiania*, 10:1–67, 1912.

[Ukkonen, 1995] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[Vágvölgyi et Gilleron, 1992] S. Vágvölgyi et R. Gilleron. For a rewriting system it is decidable whether the set of irreducible ground terms is recognizable. *Bulletin of European Association for Theoretical Computer Science*, 48:197–209, Octobre 1992.

[van Leeuwen, 1990] rédacteur Jan van Leeuwen. *Handbook of Theoretical Computer Science - Volume B: Formal Models and Semantics*. Elsevier, Amsterdam, 1 édition, 1990.

[Wen, 1994] Zhaofang Wen. New algorithms for the LCA problem and the binary tree reconstruction problem. *Information Processing Letters*, 51(1):11–16, Juillet 1994.

[Yokomori et Kobayashi, 1998] T. Yokomori et S. Kobayashi. Learning local languages and their application to DNA sequence analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10), October 1998.

[Zimin, 1984] A.I. Zimin. Blocking sets of terms. *Math. USSR Sbornik*, 47:353–364, 1984.

[Ziv et Lempel, 1977] J. Ziv et A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory IT-23*, 3:337–343, May 1977.

# Deuxième partie

# Dossier de présentation

# Chapitre 5

# Curriculum vitæ

## Grégory KUCHEROV

*Cette orthographe de mon nom est celle sous laquelle j'ai signé tous mes travaux scientifiques et sous laquelle je suis connu de mes collègues. L'orthographe légale française de mon nom est* **Grigori KOUTCHEROV** *qui provient de la traduction phonétique française de l'orthographe originale russe* **Григорий КУЧЕРОВ**.

Nationalités française, russe
Né le 07 janvier 1961, à Novossibirsk, Russie
e-mail: `Gregory.Kucherov@loria.fr`

### Emplois

**septembre 1993 - ...** Chargé de recherche INRIA (projet POLKA de l'INRIA-Lorraine/LORIA, précédemment projet EURÉCA).

**septembre 1991 - août 1993** Attaché temporaire d'enseignement et de recherche à l'Université de Nancy 1. Collaborateur extérieur à l'INRIA jusqu'à septembre 1992 (projet EURÉCA).

**juillet 1991 - septembre 1991** Maître de Conférences invité à l'Université de Nancy 1.

**septembre-décembre 1989** Chercheur invité à l'INRIA-Lorraine/CRIN, Nancy (projet EURÉCA).

**1982-1991** Chercheur-stagiaire, puis chercheur au Département d'Informatique du Centre de Calcul de la branche Sibérienne de l'Académie des Sciences d'URSS (réorganisé en 1990 en l'Institut des Systèmes Informatiques).

### Formation

**mars 1988** Thèse de Candidat de Sciences Physiques et Mathématiques (équivalent du doctorat d'Informatique) *Spécification des types abstraits de données à l'aide des systèmes de réécriture de termes*, Centre de Calcul de la branche Sibérienne de l'Académie de Sciences d'URSS

**1977-1982** Études supérieures à l'Institut d'Électrotechnique de Novossibirsk, spécialité mathématiques appliquées. Obtention du diplôme d'enseignement supérieur avec mention excellent en juin 1982

**juin 1977** Diplôme de fin d'études secondaires (équivalent du baccalauréat).

## Domaines de recherche

**Domaines généraux :** Théorie du calcul, calcul formel, informatique théorique, complexité algorithmique.

**Thèmes de recherche actuels :**

- Algorithmes combinatoires. Algorithmes sur les mots et la combinatoire du mot. Recherche de motifs dans les séquences.
- Problèmes combinatoires pour la biologie moléculaire.
- Langages formels de mots et d'arbres, complexité algorithmique des problèmes sous-jacents.

**Thèmes de recherche antérieurs :**

- Déduction automatique, systèmes de réécriture.
- Spécifications algébriques de types abstraits de données.

## Connaissance des Langues

Français, Anglais, Russe (Lus-Écrits-Parlés).

## Expérience d'enseignement

**2000-2001** cours *Algorithmique des structures discrètes* au DEA d'Informatique de l'Université Henri Poincaré Nancy 1

**2000** cours *Éléments d'algorithmique pour la génomique* au DEA d'Informatique de l'Université Henri Poincaré Nancy 1

**1995-1999** cours *Complexité et Analyse d'algorithmes* au DEA d'Informatique de l'Université Henri Poincaré Nancy 1

**1991-1993** enseignement à l'Université de Nancy 1:

- Enseignement de compilation en deuxième année d'ESIAL (École Supérieure d'Informatique et d'Automatique de Lorraine). Cours de LISP et de PROLOG.
- Deux enseignements d'Informatique générale en DEUG *Sciences de la Matière*, basés sur les languages SCHEME et ISETL.

**1983-1984** enseignement des mathématiques supérieures à l'École d'Ingénieurs de Navigation de Novossibirsk

## Expérience d'encadrement

**1995-1998** encadrement de la thèse de Vladimir Grébinski, soutenue en novembre 1998 avec félicitations du jury

**2000** encadrement d'un stage de première année de l'ENS Lyon, d'un stage de licence, et d'un stage d'un étudiant étranger

**1999** encadrement d'un stage de première année de l'ENS Lyon et d'un stage d'un étudiant du CNAM

**1995** encadrement d'un stage de DEA

**1994** encadrement de trois étudiants d'école d'ingénieurs (ESIAL) pour le stage d'initiation à la recherche

## Projets et collaborations

**2000 - ...** Participant du projet *Approches multicritères pour la modélisation et l'analyse in silico des génomes* (appel d'offres commun de CNRS - INRA - INRIA - INSERM, 9 laboratoires impliqués)

**1999 - 2000** Coordinateur de l'action coopérative de l'INRIA *Recherche et Extraction de Motifs dans les Séquences Génomiques (REMAG)* regroupant cinq équipes de recherche françaises[13]

**1997 - 2000** Coordinateur français du projet franco-russe *Algorithmique et Combinatoire des Structures Discrètes* dans le cadre de l'Institut franco-russe A.M.Liapunov d'Informatique et de Mathématiques Appliquées[14]

**2000 - ...** Participant du projet INTAS *Methods, algorithms and software for functional and structural annotation of complete genomes* (projet avec la Russie, incluant des laboratoires d'Allemagne, de France et d'Autriche),

**1997-1999** Participant du réseau national « Informatique et Génome » du CNRS[15]

## Publications

- 10 articles dans des revues internationales (*Information and Computation, Theoretical Computer Science, Discrete Applied Mathematics, Information Processing Letters, Algorithmica, ...*)
- 20 communications à des conférences internationales (*FOCS, ESA, MFCS, SODA, CPM, RTA, FCT, ...*)
- 7 publications dans des Journaux et Ouvrages Collectifs Soviétiques
- 9 rapports internes et notes

## Implantation de Logiciels

- Conception et participation à la réalisation du logiciel MREPS de recherche de répétitions maximales dans les séquences[16]
- Conception et participation à la réalisation du logiciel GRAPPE de recherche de motifs dans les séquences[17]
- Participation au développement du système de programmation de bases de données ATLANT (1985-1988)
- Réalisation de quelques logiciels expérimentaux de démonstration automatique

13. http://www.loria.fr/remag/
14. http://www-direction.inria.fr/international/liapunov.html
15. http://genome.univ-mlv.fr/
16. http://www.loria.fr/~kucherov/SOFTWARE/MREPS/index.html
17. http://www.loria.fr/~kucherov/SOFTWARE/grappe-3.0/grappe-3.0-en.html

## Responsabilités

- responsable du projet PolKA à partir de février 2000.
- membre du comité de programme de la conférence internationale JOBIM'2001 (*Journées Ouvertes : Biologie, Informatique et Mathématiques*)
- membre du comité de programme de la conférence internationale *Perspectives of System Informatics* (Novossibirsk, 2001)
- *publicity chair* de RECOMB'99 (International Conference on Computational Molecular Biology, Lyon).
- membre du comité d'organisation du *Workshop on Discrete Tomography* (Thionville, 1999)
- en 1997-1999, membre du comité de rédaction de la lettre de l'AFIT (Association Française d'Informatique Théorique)
- en 1997-2000, organisateur des séminaires d'Informatique Théorique au Loria et des séminaires d'équipe (PolKA),
- membre du comité de programme de la conférence internationale *Algebraic and Logic Programming* (Nancy, 1990)

## Participation à la vie de la recherche

- membre de l'*European Association for Theoretical Computer Science, Association Française d'Informatique Théorique*
- j'ai été sollicité comme lecteur par plusieurs revues scientifiques (*Algorithmica, Journal of Discrete Algorithms, Theoretical Computer Science, Information Processing Letters, Fundamenta Informaticae, Information and Software Technology, Computers and Chemistry, IEEE Transactions on Knowledge and Data Engineering, ...*) ainsi que par des conférences scientifiques internationales (*STACS'01, SODA'00, ICS'00, STACS'99, PSI'99, CSL'99, EURO-PAR'99, MFCS'97, RTA'97, DISCO'96, ICALP'94, MFCS'93, STACS'93, TAPSOFT'93, STACS'92, ALP'92, ...*).

## Séminaires

Au cours de ma carrière, j'ai été amené à faire des séminaires dans les laboratoires suivants:

- LITA (Université de Metz) (2000)
- École Normale Supérieure de Lyon (2000)
- société CERES Inc., Californie, États-Unis (2000)
- Université de Strasbourg (1999)
- société CompuGen, New-Jersey, États-Unis (1999)
- Institute Engelhardt de Biologie Moléculaire de l'Académie de Sciences de Russie, Moscou (1999)
- LIAFA (Université Paris 7) (1999)
- École Normale Supérieure de Cachan (1999)
- Université Marne-la-Vallée (1994, 1998)
- Université Paris 12 à Creteil (1997)
- Dagstuhl seminar "Applications of Tree Automata in Rewriting, Logic and Programming" (1997)

- Réunion du groupe de travail « Informatique et Génome » du GDR-PRC AMI (1996)
- Réunion du groupe de travail ALEA du GDR-PRC AMI (1996)
- Technische Universität Berlin (1994,1995,1996)
- Workshops franco-russes à l'Université de Moscou (1994, 1995, 1998) et au LORIA (1999)
- Stony Brook University (New York, 1994)
- Réunion du groupe *Langages Formels* de l'AFCET (1993,1994)
- Institute of Computer Science of Polish Academy of Sciences (Gdańsk, 1990)
- Universités VI et VII de Paris (1990)
- Würzburg Universität (Allemagne, 1990)
- LIFIA (Grenoble, 1989)
- CRIN et INRIA-Lorraine (1989)
- Université de Dresde (1988)
- Institut de cybernétique de l'Académie de Sciences de l'Ukraine (Kiev, 1990)
- Centre de Calcul de l'Université P.Stucsky (Riga, Lettonie, 1986)
- Institut de mathématique et de cybernétique de l'Académie de Sciences de la Lituanie (Vilnius, 1984, 1988)
- Institut de mathématique de la branche Sibérienne de l'Académie de Sciences d'URSS (Novossibirsk, 1988,1989)
- conférencier à l'école d'été *Types Abstraits de Données* (Lymanchik, Ukraine, 1984)

# Chapitre 6

# Liste de mes publications

## Thèse de doctorat

[1] G. Kucherov. *Spécification des types abstraits de données à l'aide de systèmes de réécriture de termes*. Thèse de candidat de sciences (équivalent du doctorat), Centre de Calcul de la branche Sibérienne de l'Académie de Sciences d'URSS, mars 1988. En russe.

### Revues Internationales et Chapitres de Livres

[2] G. Kucherov and M. Rusinowitch. Undecidability of ground reducibility for word rewriting systems with variables. *Information Processing Letters*, 53:209–215, 1995.

[3] G. Kucherov and M. Tajine. Decidability of regularity and related properties of ground normal form languages. *Information and Computation*, 118(1):91–100, avril 1995.

[4] G. Kucherov and M. Rusinowitch. Matching a set of strings with variable length don't cares. *Theoretical Computer Science*, 178:129–154, 1997.

[5] V. Grebinski and G. Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discrete Applied Mathematics*, 88:147–165, 1998.

[6] D. Hofbauer, M. Huber, and G. Kucherov. *How to get rid of projection rules in context-free tree grammars*, chapter 15, pages 235–247. Studies in Logic, Language and Information. Center for the Study of Language and Information (CSLI), Stanford and The European Association for Logic, Language and Information (FoLLI), 1998.

[7] R. Kolpakov, G. Kucherov, and Y. Tarannikov. On repetition-free binary words of minimal density. *Theoretical Computer Science*, 218(1):143–160, 1999.

[8] D. Plaisted and G. Kucherov. The complexity of some complementation problems. *Information Processing Letters*, 71:159–165, 1999.

[9] V. Grebinski and G. Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28:104–124, 2000.

[10] R. Kolpakov and G. Kucherov. On maximal repetitions in words. *Journal on Discrete Algorithms*, 2000. à paraître.

### Actes de Conférences Internationales

[11] G. Kucherov. A new quasi-reducibilty testing algorithm and its application to proofs by induction. In J. Grabowski, P. Lescanne, and W. Wechler, editors, *Proceedings of the 1st*

*International Workshop on Algebraic and Logic Programming, Gaussig (GDR)*, volume 343 of *Lecture Notes in Computer Science*, pages 204–213. Springer Verlag, 1988.

[12] G. Kucherov. On relationship between term rewriting systems and regular tree languages. In *Proceedings of the 4th International Conference on Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 299–311. Springer Verlag, 1991.

[13] A. Zamulin, V. Kositov, G. Kucherov, E. Pak, and V. Ryzhkov. The database programming language *atlant*: Principal features and implementation techniques. In *First International East/West Database Workshop*, volume 504 of *Lecture Notes in Computer Science*, pages 385–402. Springer Verlag, 1991.

[14] G. Kucherov and M. Tajine. Decidability of regularity and related properties of ground normal form languages. In *Proceedings of the 3rd International Workshop on Conditional Term Rewriting Systems*, volume 656 of *Lecture Notes in Computer Science*, pages 272–286. Springer Verlag, 1992. Version complète dans [3].

[15] D. Hofbauer, M. Huber, and G. Kucherov. Some results on top-context-free tree languages. In *Proceedings of the 19th International Colloquium on Trees in Algebra and Programming*, volume 787 of *Lecture Notes in Computer Science*, pages 157–171. Springer Verlag, 1994.

[16] G. Kucherov and M. Rusinowitch. On Ground-Reducibility Problem for Word Rewriting Systems with Variables. In E. Deaton and R. Wilkerson, editors, *Proceedings 1994 ACM/SIGAPP Symposium on Applied Computing. Special Track on Computational Logic.* ACM-Press, Phoenix (USA), mars 1994. Version complète dans [2].

[17] D. Hofbauer, M. Huber, and G. Kucherov. How to get rid of projection rules in context-free tree grammars (abstract). In *Proceedings of the Tbilisi Symposium on Logic, Language and Computation*. University of Edinburgh, Human Communication Research Center, RP-72, Gudauri (Georgia), octobre 1995. Version complète dans [6].

[18] G. Kucherov and M. Rusinowitch. Complexity of testing ground reducibility for linear word rewriting systems with variables. In *Proceedings 4th International Workshop on Conditional and Typed Term Rewriting Systems, Jerusalem (Israel)*, volume 968 of *Lecture Notes in Computer Science*, pages 262–275. Springer Verlag, 1995.

[19] G. Kucherov and M. Rusinowitch. Matching a set of strings with variable length don't cares. In E. Ukkonen, editor, *Proceedings of the 6th Symposium on Combinatorial Pattern Matching*, volume 937 of *Lecture Notes in Computer Science*, pages 230–247. Springer Verlag, Helsinki (Finland), juillet 1995. Version complète dans [4].

[20] V. Grebinski and G. Kucherov. Optimal query bounds for reconstructing a hamiltonian cycle in complete graphs. In *Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems*, pages 166–173. IEEE Press, June 1997. Version complète dans [5].

[21] V. Grebinski and G. Kucherov. Optimal reconstruction of graphs under the additive model. In R. Burkard and G. Woeginger, editors, *Proceedings of the 5th Annual European Symposium on Algorithms, Graz (Austria)*, volume 1284 of *Lecture Notes in Computer Science*, pages 246–258. Springer Verlag, 1997. Version complète dans [9].

[22] R. Kolpakov and G. Kucherov. Minimal letter frequency in $n$-power-free binary words. In I. Privara and P. Ružička, editors, *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS), Bratislava (Slovakia)*, volume 1295 of *Lecture Notes in Computer Science*, pages 347–357. Springer Verlag, 1997.

[23] F. Bertault and G. Kucherov. Visualization of Dynamic Automata using `Padnon`. In *Workshop on Implementing Automata, Ontario, Canada, 1997*, volume 1436 of *Lecture Notes in Computer Science*, pages 25–28. Springer Verlag, septembre 1998.

[24] R. Kolpakov, G. Kucherov, and Y. Tarannikov. On repetition-free binary words of minimal density. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS), Brno (Czech Republic)*, volume 1450 of *Lecture Notes in Computer Science*, pages 683–692. Springer Verlag, 1998. Version complète dans [7].

[25] V. Grebinski and G. Kucherov. Reconstructing set partitions. In S. ACM, editor, *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms, Baltimore, Maryland, 17-19 janvier, 1999*, pages 915–916. 1999.

[26] R. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *Proceedings of the 1999 Symposium on Foundations of Computer Science, New York (USA)*, pages 596–604. IEEE Computer Society, New-York, 17-19 octobre 1999.

[27] R. Kolpakov and G. Kucherov. On maximal repetitions in words. In G.Ciobanu and Gh.Păun, editors, *Proceedings of the 12-th International Symposium on Fundamentals of Computation Theory, 1999, Iasi (Romania)*, volume 1684 of *Lecture Notes in Computer Science*, pages 374 – 385. Springer Verlag, 30 août - 3 septembre 1999.

[28] G. Kucherov and M. Rusinowitch. Patterns in words vs patterns in trees: a brief survey and some new results. In *Proceedings of the Andrei Ershov 3rd International Conference "Perspectives of System Informatics" (6 - 9 July 1999, Novosibirsk, Akademgorodok, Russia)*, volume 1755 of *Lecture Notes in Computer Science*, pages 280–293. Springer Verlag, 1999.

[29] M. Giraud and G. Kucherov. Maximal repetitions and application to dna sequences. In *Proceedings of the* Journées Ouvertes : Biologie, Informatique et Mathématiques, pages 165–172. 3-5 mai 2000.

[30] R. Kolpakov and G. Kucherov. Finding repeats with fixed gap. In *Proceedings of the 7th International Symposium on String Processing and Information Retrieval (SPIRE), A Coruña, Spain (27 - 29 Septembre, 2000)*, pages 162–168. IEEE, Septembre 2000.

### Journaux et Recueils Scientifiques Soviétiques

[31] A. Kucherov and G. Kucherov. Noise generator with arbitrary probability distribution. *Radiotechnics*, 7:40–42, 1983. En russe.

[32] G. Kucherov. An algorithm for testing sufficient completeness of algebraic specifications of abstract data types. *Programming*, 4:3–12, 1984. En russe.

[33] G. Kucherov and A. Zamulin. Compiler structure for a language with abstract data types. In *Mathematical models of information representation and problems of data processing*, pages 117–124. Computing Center of Siberian Division of USSR Academy of Sciences, Novosibirsk, 1986. En russe.

[34] G. Kucherov. An algorithm of testing ground reducibility of linear term rewriting systems. In *Computing Systems*, volume 122, pages 19–37. Computing Center of Siberian Division of USSR Academy of Sciences, Novosibirsk, 1987. En russe.

[35] G. Kucherov. On the implementation of Knuth-Bendix completion algorithm for term rewriting systems. In *Methods of theoretical and parallel programming*, pages 50–62. Computing Center of Siberian Division of USSR Academy of Sciences, Novosibirsk, 1987. En russe.

[36] G. Kucherov. Varieties of algebraic specifications of abstract data types and their properties. In *System and Theoretical Programming*, pages 74–82. Rostov, 1987. En russe.

[37] G. Kucherov. On quasi-reducibility and proofs by induction in abstract data types. In *Methods of theoretical and experimental computer science*, pages 57–72. Computing Center of Siberian Division of USSR Academy of Sciences, Novosibirsk, 1989.

### Rapports Internes et Notes

[38] G. Kucherov. An algorithm for testing sufficient completeness of algebraic specifications of abstract data types. Internal Report 491, Computing Center of Siberian Division of the USSR Academy of Science, Novosibirsk, 1983. En russe.

[39] G. Kucherov and N. Ludvina. On the hierarchy of types and interpretation of generic types, 1984. En russe.

[40] G. Kucherov. Term rewriting systems. a survey. Internal Report 601, Computing Center of the Siberian Division of the USSR Academy of Science, Novosibirsk, 1985. En russe.

[41] G. Kucherov. An algorithm of proving properties of abstract data types. Internal Report 750, Computing Center of Siberian Division of the USSR Academy of Science, Novosibirsk, 1987. En russe.

[42] A. Zamulin, V. Kositov, G. Kucherov, E. Pak, and V. Ryzhkov. Organization of a data base programming system. Internal Report 836, Computing Center of Siberian Division of the USSR Academy of Science, Novosibirsk, 1989. En russe.

[43] R. Kolpakov and G. Kucherov. Maximal repetitions in words or how to find all squares in linear time. Rapport Interne 98-R-227, LORIA, 1998.

[44] R. Kolpakov and G. Kucherov. On the sum of exponents of maximal repetitions in a word. Rapport Interne 99-R-034, LORIA, 1999.

[45] D. Plaisted and G. Kucherov. The complexity of some complementation problems. Technical Report RR-3681, INRIA, mai 1999. Paru dans [8].

[46] R. Kolpakov and G. Kucherov. Finding repeats with fixed gap. Technical Report RR-3901, INRIA, mars 2000.