

École doctorale de Saint-Étienne

Inférence Grammaticale de Langages Hors-Contextes

Thèse présentée à la Faculté des Sciences et Techniques de Saint-Étienne
pour obtenir le grade de :

DOCTEUR DE L'UNIVERSITÉ JEAN MONNET DE SAINT-ÉTIENNE

Mention : Informatique.

par Rémi EYRAUD

EURISE

Faculté des Sciences et Techniques

Soutenance le 22 novembre 2006, devant le jury composé de :

Président du Jury : Antoine Cornuéjols.....Professeur des universités
Rapporteur : Géraud Sénizergues.....Professeur des universités
Rapporteur : Isabelle Tellier.....Maître de conférences
Directeur : Colin de la Higuera.....Professeur des universités
Co-directeur : Jean-Christophe Janodet Maître de conférences
Membre : Alexander Clark.....Professor
Membre : Takashi Yokomori Professor

Sommaire

Introduction	1
Inférence grammaticale	9
1 Apprendre des langages formels	11
1.1 Qu'est-ce qu'un langage?	11
1.1.1 Définition des langages	11
1.1.2 Représentation des langages	12
1.1.3 Hiérarchie de Chomsky	16
1.1.4 Propriétés des grammaires hors-contextes	17
1.2 Qu'est-ce qu'apprendre des langages?	19
1.2.1 Approches empiriques	21
1.2.2 Le cadre PAC	27
1.2.3 Le cadre de l'identification à la limite	29
1.3 Pourquoi apprendre les langages hors-contextes?	32
2 Sur la difficulté d'apprentissage des grammaires hors-contextes	35
2.1 D'où vient cette difficulté?	35
2.1.1 Opérabilité	35
2.1.2 Indécidabilité	36
2.1.3 Ambiguïté et non-déterminisme	38
2.1.4 Expansivité	39
2.1.5 Générativité	40
2.1.6 Indivisibilité	41
2.1.7 Structuralité	42
2.2 Quelles solutions?	43
2.2.1 Données structurées <i>a priori</i>	43
2.2.2 Restrictions à des sous-classes	45
2.2.3 Changement de représentation	47
2.3 Conclusion	48

Nos contributions	49
3 Structuration a priori des données	51
3.1 Motivation	51
3.2 L'algorithme SEQUITUR	52
3.2.1 Fonctionnement	52
3.2.2 Exemples	54
3.2.3 Propriétés	55
3.3 L'algorithme RT	56
3.3.1 Fonctionnement	56
3.3.2 Propriétés	57
3.4 Modifications de SEQUITUR	58
3.5 Expérimentations	59
3.5.1 Résultats	59
3.5.2 Discussion	60
3.6 Conclusion	61
4 Changement de représentation	63
4.1 Motivation	63
4.2 Définir des langages avec des systèmes de réécriture	64
4.3 Représenter efficacement des langages par des SDRMs	66
4.3.1 La propriété de terminaison	66
4.3.2 La propriété de confluence	69
4.3.3 Sur les SDRMs hybrides et PNCs	71
4.4 LARS: fonctionnement et exemples	73
4.5 Apprendre des SDRMs hybrides et PNCs	77
4.5.1 Un résultat d'identification	77
4.5.2 Sur la classe apprise et sa position dans la hiérarchie de Chomsky	81
4.6 Résultats expérimentaux	83
4.6.1 Expérimentations en inférence grammaticale	84
4.6.2 Comportement de LARS sur des petits langages	86
4.7 Conclusion et perspectives	87
5 Restriction à une sous-classe	89
5.1 Motivation	89
5.2 Substituabilité: définitions et exemples	91
5.3 Un algorithme d'apprentissage des langages hors-contextes substi- tuables	92
5.3.1 Apprendre des grammaires	93
5.3.2 SGL-G identifie à la limite les langages substituables	95

5.3.3	Apprendre des systèmes de réécriture	97
5.4	Sur les langages substituables	100
5.4.1	Exemples de langages substituables	100
5.4.2	Relations avec les autres classes	100
5.4.3	Propriétés des langages substituables	101
5.5	Application à un problème de langue naturelle	102
5.5.1	Problématique	102
5.5.2	Expérimentations	103
5.5.3	Discussion	104
5.6	Conclusion et perspectives	106

Conclusions

109

Introduction

Une des plus grandes évolutions humaines de ces dernières décennies est sans aucun doute ce qu'il est désormais convenu d'appeler la "révolution numérique". L'essor de l'informatique a ainsi radicalement changé nos habitudes, tant sur un plan culturel que social. Jamais depuis l'invention de Gutenberg une avancée scientifique n'a autant fait évoluer nos mœurs, notre conception du monde, nos connaissances, notre manière de travailler, de se distraire, de s'informer. Et pourtant, d'un point de vue scientifique, cette science n'en est qu'à ses débuts, certains considérant même qu'elle ne sort qu'à peine de sa préhistoire. . .

De tous les domaines de l'informatique, l'*intelligence artificielle* est celui qui cristallise le plus d'espoirs, mais aussi de peurs [AW99]. Mais si l'on est encore loin de construire le *terminator* [Cam85], les progrès réalisés en une très courte période sont impressionnants. Pour ne donner qu'un exemple, l'aéroport de Sydney, en Australie, est entièrement géré par des agents artificiels, l'intervention humaine étant réduite à une supervision ne requérant pas une réelle interaction.

Un sous-domaine de l'intelligence artificielle, en plein essor, nous intéresse plus particulièrement ici. Il s'agit de l'*apprentissage artificiel* ou *apprentissage automatique* [CM02]. S'il est difficile de définir correctement et complètement ce que l'on entend par "intelligence", il est indéniable que, quelle que soit cette définition, les capacités d'apprentissage en font partie. Ainsi, tout artefact capable d'apprendre peut être considéré comme possédant une intelligence. Mais qu'est-ce qu'apprendre? Sous quelles conditions peut-on considérer qu'un apprentissage est réussi? Est-il possible de formaliser mathématiquement ce qu'apprendre veut dire?

Intuitivement, apprendre c'est être capable d'extraire des règles de portée générale à partir de constatations particulières. C'est ainsi qu'un enfant ayant appris à parler sera en mesure de comprendre et de former des phrases qu'il n'a jamais entendues. De manière analogue, un algorithme est capable d'apprendre si, à partir d'observations, que l'on appelle *données d'apprentissage*, il acquiert la capacité de prendre une décision correcte sur des données qu'il n'a pas rencontrées lors de sa phase d'apprentissage [Mit97]. Ceci est fortement lié aux capacités de catégorisation, de classification de l'univers dans lequel évolue l'apprenant : grâce aux données d'apprentissage et s'il réussit dans son entreprise, il peut ensuite faire la distinction entre ce qui correspond à ce qu'il a cherché à apprendre et ce

qui n'est pas du même type, de la même classe. Deux façons de voir l'apprentissage s'opposent et se complètent à ce niveau. La première part du principe qu'il existe un *concept cible* qu'il faut découvrir. Un apprentissage sera alors réussi si l'algorithme parvient à inférer une représentation de cette cible, ou du moins, une représentation très similaire à ce concept. La seconde se base sur la minimisation de l'erreur en généralisation : l'existence d'un concept cible n'est pas nécessaire, et l'algorithme apprendra correctement si, sur des données ne faisant pas partie de son apprentissage, le nombre d'erreurs de classification qu'il fait, c'est-à-dire le nombre de données qu'il acceptera alors qu'il aurait dû les refuser (et vice versa), est inférieur à un certain seuil. Par exemple, il n'existe pas de définition exhaustive de la langue française. On considère donc que quelqu'un parle français lorsqu'il possède une certaine maîtrise de la langue, sans toutefois connaître l'intégralité du vocabulaire ou des structures grammaticales. Bien entendu, plus l'apprenant dispose de données d'apprentissage, plus l'erreur en généralisation doit être faible.

Il existe une autre façon de formaliser différents types d'apprentissage. Au lieu de se focaliser sur le but recherché (l'existence ou non d'un concept cible), la classification se fait sur les différents types de données d'apprentissage dont on peut disposer. Ainsi, on parle d'*apprentissage supervisé* si les données portent une étiquette permettant de savoir si elles font parties ou non de ce que l'on cherche à apprendre. Dans le cas contraire, on parle d'*apprentissage non-supervisé*. L'*apprentissage transductif* [Vap98] correspond à une voie intermédiaire, où seulement une partie des données est étiquetée. Nous nous plaçons dans le cadre de l'apprentissage supervisé et nous n'aborderons pas les autres types dans ce qui suit.

Le but de cette thèse est de proposer et d'étudier des algorithmes réalisant un apprentissage de langages formels. Ce sous-domaine de l'apprentissage automatique, qui a une quarantaine d'années, est appelé *inférence grammaticale* [dlH03]. Sa motivation première était de modéliser l'apprentissage de la langue par l'enfant [Cho56]. Mais, par la suite, de multiples autres domaines ont bénéficié de ses avancées, allant de la génétique [BJVU98] à la reconnaissance de la parole [GSVG94], en passant par le traitement de documents structurés [Fer01] ou la compression de textes [NMW97].

Un *langage* est un ensemble, fini ou non, de mots formés sur un *alphabet* donné [Aut87]. Un *mot* est simplement une concaténation de symboles de l'alphabet. Un langage peut être représenté par une *grammaire formelle*, c'est-à-dire un ensemble de règles formées sur l'alphabet du langage (les *lettres*) et sur un alphabet complémentaire distinct (appelé *alphabet des non-terminaux*). Ces règles permettent de réécrire un ensemble de symboles des alphabets par un autre ensemble. Le langage engendré par une grammaire est l'ensemble des mots que l'on peut obtenir en utilisant les règles de la grammaire à partir d'un symbole non-terminal particulier, appelé *axiome* de la grammaire.

Les différentes formes que peuvent prendre les règles des grammaires définissent des classes auxquelles sont associées des classes de langages formels. Dans cette thèse, nous nous intéressons à une classe en particulier, la classe dite des *grammaires hors-contextes*, ou *grammaires algébriques* (et donc aux *langages hors-contextes* ou *langages algébriques*). Cette classe est définie par une contrainte simple : les parties gauches des règles, c'est-à-dire les parties qui se réécrivent, doivent être composées d'un unique symbole appartenant à l'alphabet des non-terminaux. La classe de langages associée correspond à un large panel de structures grammaticales. Elle comprend, par exemple, les langages contenant des imbrications non bornées de parenthèses, c'est-à-dire une parenthèse ouvrante suivi d'un élément de taille indéterminée, puis d'une parenthèse fermante [vD35]. Les expressions algébriques, couramment utilisés en mathématiques, forment ainsi un langage algébrique. De la même façon, les langages informatiques à base de balises, comme HTML ou XML, sont aussi des langages hors-contextes [Fer01]. Mais le champ couvert par cette classe de langages ne se résume pas aux structures de parenthèses : par exemple, les avancées en bioinformatique tendent à montrer que les structures génétiques sont, pour la plupart, hors-contextes [BJVU98]. En ce qui concerne les langues naturelles, quelques structurations n'appartenant pas à la classe ont été mises en évidence, mais elles appartiennent presque toutes à la langue écrite et une très grande partie des langues naturelles sont hors-contextes [Abe88]. D'autre part, les langages hors-contextes sont couramment utilisés en informatique, dans un cadre autre que l'apprentissage. En effet, les langages de programmation, comme *Java* [AGH00] ou *Caml* [WAL⁺90], sont des langages hors-contextes, représentés dans leur compilateur respectif par des grammaires de ce type.

Outre la diversité formidable de langages que couvre la classe des hors-contextes, un autre élément nous a poussé à tenter d'apprendre ces langages. Il existe une autre classe de langages, plus restreinte et contenue dans celle qui nous intéresse : les *langages réguliers*. Or, en 2003, quand ont commencé les travaux présentés ici, les chercheurs de la communauté de l'inférence grammaticale avaient fini par obtenir de bons résultats sur cette classe, que ce soit avec une approche exacte [OG92] ou dans le cas stochastique [TDdlH00]. Apprendre des langages hors-contextes représentait donc le nouveau challenge qui s'offrait à la communauté. C'est dans cette optique qu'a commencé cette thèse.

Si désormais notre but et nos motivations sont clairs, il nous reste à spécifier ce que l'on entend par apprendre, et sous quelles formes nous allons essayer de le faire. Cette thèse se place dans un cadre d'étude théorique, ce qui explique que nous nous refusons à toute approche empirique (même si nous rappellerons les principaux résultats obtenus dans ce cas dans la première partie de ce document). Le cadre formel le plus utilisé en apprentissage automatique pour étudier les propriétés d'apprenant d'un algorithme est le cadre PAC [Val84], pour Proba-

blement Approximativement Correct. Ce formalisme correspond à la définition de l'apprentissage ne nécessitant pas la présence d'un concept cible dans l'ensemble d'hypothèses dont dispose l'algorithme. Il peut se résumer informellement à dire que l'apprentissage est réussi si la probabilité que l'erreur en généralisation soit supérieure à un seuil est proche de zéro. Nous verrons que, malheureusement, il est impossible de PAC-apprendre les langages algébriques (mais aussi des sous-classes plus restreintes). En fait, ce cadre est trop restrictif pour permettre des résultats d'apprentissage de langages. Il est plus approprié à une approche stochastique se basant sur la distribution des données d'apprentissage. Or nous avons choisi de travailler dans une approche exacte et non-stochastique : nous verrons dans la suite qu'apprendre un langage algébrique est rendu difficile par le fait qu'il faille à la fois découvrir le langage ainsi que sa structure. Chercher à découvrir, en sus, la loi de probabilité se cachant derrière les données d'apprentissage rajoute une difficulté à laquelle nous ne nous sommes pas attaqués.

Nous avons aussi mis de côté un autre cadre d'apprentissage, pourtant initialement introduit pour modéliser l'acquisition de langages. Il s'agit du cadre d'Angluin [Ang87], où l'apprenant dispose d'un *oracle* connaissant le concept cible et qu'il peut interroger à l'aide de requêtes. Nous savons que dans ce cadre il n'est pas possible d'apprendre les langages hors-contextes, du moins avec les types de requêtes utilisés pour apprendre la classe des réguliers. Travailler dans ce cadre nous aurait forcément amené à introduire de nouveaux types de requêtes ce qui nous est apparu comme inutile et très peu pertinent : les requêtes jusqu'ici utilisées sont déjà très gourmandes, certaines étant même incalculables pour des langages hors-contextes. Comme nous ne disposions pas d'oracle et comme cette approche ne nous apparaissait pas si pertinente, il nous a semblé déraisonnable de nous engager dans cette voie. Nos contributions se placent donc dans un troisième paradigme d'apprentissage, propre lui aussi à l'apprentissage de langages, appelé *identification à la limite* [Gol67]. Nous détaillerons dans la suite sa formalisation mais donnons ici l'intuition qui a amené à sa formulation. L'idée repose sur l'existence d'un langage cible à apprendre. Montrer qu'un algorithme identifie à la limite une classe de langages revient à montrer que, quel que soit le langage considéré dans la classe, si l'on donne suffisamment de données d'apprentissage à l'algorithme, alors il infèrera une représentation correspondant exactement au langage cible.

Ce qui suit est structuré en deux parties. La première regroupe deux chapitres dont le premier introduit les définitions et les notations élémentaires qui seront utilisées dans la suite du document. Il contient aussi une discussion sur trois façons de voir et de définir ce qu'est un apprentissage artificiel de langages. Le deuxième chapitre de cette première partie contient une réflexion sur les difficultés inhérentes à l'inférence grammaticale de langages hors-contextes ainsi que les

principaux résultats théoriques. Une étude des meilleurs algorithmes répondant à cette problématique et de leur façon de gérer ces difficultés complète ce chapitre.

La deuxième partie de ce document de thèse contient nos contributions à l'inférence grammaticale de langages algébriques. Cette partie est divisée en trois chapitres, chacun correspondant à une approche différente du problème. À l'exception du premier (chapitre numéroté 3), ces travaux ont donné lieu chacun à plusieurs publications : le chapitre 4 résume des travaux publiés dans les actes de la conférence ICGI en 2004 [EdlHJ04] et à paraître dans la revue scientifique *Machine Learning Journal*; le chapitre 5 est une synthèse d'un article publié dans les actes de la conférence ALT en 2005 [CE05] et de deux autres articles présentés aux conférences *Cognitive Science* et *ConLL* en 2006 [CE06].

Deuxième partie

Inférence grammaticale

1 **Apprendre des langages formels**

1.1 Qu'est-ce qu'un langage ?

Le but de cette section est d'introduire les définitions qui serviront de bases au reste de ce document. Les notations définies et utilisées sont, pour la plupart, issues de [Aut87]. Aucun résultat nouveau de théorie des langages n'est présenté ici, ni aucune étude algébrique ou topologique des structures sous-jacentes. Un lecteur connaissant la théorie des langages formels peut directement se rendre à la section 1.2.

1.1.1 Définition des langages

Un *alphabet* est un ensemble fini non vide de symboles appelés *lettres*. Un *mot* sur un alphabet Σ est une séquence finie $w = a_1a_2 \dots a_n$ de lettres de Σ . On note $|w|$ la longueur du mot w , c'est-à-dire le nombre de lettres qui le composent. $|w|_x$ représente le nombre d'occurrences de la lettre x dans le mot w . Par exemple, *abba* est un mot sur l'alphabet $\Sigma = \{a,b\}$ tel que $|w| = 4$, $|w|_a = |w|_b = 2$. Dans la suite on notera $a,b,c \dots$ les lettres et $u,v,w \dots$ les mots. Le mot vide, c'est-à-dire le mot de longueur zéro, sera noté λ . Σ^* dénote l'ensemble des mots sur l'alphabet Σ . Ces définitions permettent de lever les ambiguïtés qui peuvent être trouvées dans certains articles scientifiques, en particulier ceux du domaine de la linguistique computationnelle : il n'est pas rare que l'unité lexicale soit appelée "mot", une séquence de mots devenant une "phrase". Dans la suite, nous éviterons ces ambiguïtés, le vocabulaire employé répondra donc aux définitions qui viennent d'être introduites.

On suppose un ordre total $<$, fixé mais arbitraire, sur les lettres d'un alphabet Σ . Cet ordre est étendu à Σ^* par l'*ordre hiérarchique*, noté \triangleleft et défini comme

suit:

$$\forall w_1, w_2 \in \Sigma^*, w_1 \triangleleft w_2 \text{ ssi } \begin{cases} |w_1| < |w_2| \text{ ou} \\ |w_1| = |w_2| \text{ et } \exists u, v_1, v_2 \in \Sigma^*, \exists x_1, x_2 \in \Sigma \\ \text{t.q. } w_1 = ux_1v_1, w_2 = ux_2v_2 \text{ et } x_1 < x_2. \end{cases}$$

Cet ordre est total et strict sur Σ^* et, par exemple, si $\Sigma = \{a, b\}$ et si $a < b$ alors $\lambda \triangleleft a \triangleleft b \triangleleft aa \triangleleft ab \triangleleft ba \triangleleft bb \dots$

Un *langage* formel L sur un alphabet Σ est un ensemble, fini ou non, de mots sur Σ , c'est-à-dire $L \subseteq \Sigma^*$. Le complémentaire \bar{L} d'un langage L est l'ensemble des mots de Σ^* n'appartenant pas à L . Étant donnés deux langages L_1 et L_2 sur un alphabet Σ , l'union $L_1 \cup L_2$ est l'ensemble des mots appartenant à l'un des deux langages; l'intersection $L_1 \cap L_2$ est l'ensemble des mots appartenant aux deux langages; la concaténation $L_1 L_2$ est l'ensemble $\{w : \exists w_1 \in L_1, \exists w_2 \in L_2, w = w_1 w_2\}$; la différence symétrique $L_1 \oplus L_2$ est l'ensemble $\{w \in \Sigma^* : w \in L_i, w \notin L_j, i, j \in \{1, 2\}, i \neq j\}$.

Étant donné un mot w d'un langage L tel qu'il existe $l, u, r \in \Sigma^*$, $w = lur$, on dit que l, u et r sont des *facteurs* de w , que l est un *préfixe* de w , que r un *suffixe* de w , et que la paire (l, r) est un *contexte* du facteur u dans L . L'ensemble des contextes d'un facteur u dans un langage L est noté $C_L(u) = \{(l, r) \in \Sigma^* : lur \in L\}$, ou simplement $C(u)$ quand il n'existe pas d'ambiguïté sur le langage.

Exemple 1.1 *Supposons que $\Sigma = \{a, b\}$. $L_1 = \{aa, ab\}$ est un langage fini sur Σ , tandis que $L_2 = \{a^n b^n : n > 0\}$ est un langage infini, composé des mots formés d'un certain nombre de a suivi du même nombre de b . Le mot $w = aaabbb$ appartient à L_2 , ab et $aaab$ en sont des facteurs tels que $aaab$ est un préfixe de w et (aa, bb) est un contexte de ab dans L_2 . On peut facilement montrer que $C_{L_2}(ab) = \{(a^n, b^n) : n \geq 0\}$.*

1.1.2 Représentation des langages

De nombreuses *classes de langages* ont été étudiées jusqu'ici. La plupart du temps, la définition d'une classe de langage \mathbb{L} est liée à celle d'une classe de machines abstraites \mathbb{R} , appelées ici *représentations*, caractérisant tous les langages de \mathbb{L} et seulement ces langages. La relation entre ces deux classes est formalisée par la *fonction de nommage* $\mathcal{L}_{\mathbb{R}, \mathbb{L}}$ ou plus simplement $\mathcal{L} : \mathbb{R} \rightarrow \mathbb{L}$ telle que :

(1) $\forall R \in \mathbb{R}, \mathcal{L}(R) \in \mathbb{L}$ et (2) $\forall L \in \mathbb{L}, \exists R \in \mathbb{R}$ tel que $\mathcal{L}(R) = L$. Deux représentations R_1 et R_2 sont *équivalentes* si et seulement si $\mathcal{L}(R_1) = \mathcal{L}(R_2)$. Dans ce document, nous nous préoccuperons principalement de deux classes de langages : celle des langages réguliers (REG) caractérisée par les automates finis déterministes (AFD), et celle des langages hors-contextes représentée, entre autre, par les grammaires hors-contextes (GHC). La taille d'une représentation R , notée $\|R\|$ est polynomialement reliée à son codage binaire sur une machine.

Automate fini déterministe

Un *automate fini déterministe* est un quintuplet $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ où Q est l'ensemble des *états*, $q_0 \in Q$ est l'*état initial*, $F \subseteq Q$ est l'*ensemble des états finaux* et $\delta : Q \times \Sigma \rightarrow Q$ est la *fonction de transition*. Le langage reconnu par A est $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$, où δ^* est la *fonction de transition étendue* définie sur $Q \times \Sigma^* \rightarrow Q$. Le nombre d'états d'un automate fini déterministe est une mesure pertinente de la taille de l'automate. Un automate peut être représenté graphiquement, comme sur la Fig. 1.1.

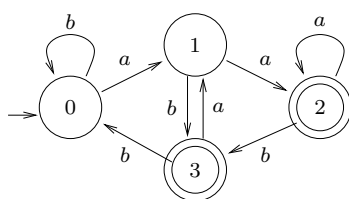


FIG. 1.1 – Un automate fini déterministe reconnaissant le langage $(a+b)^*a(a+b)$.

À noter qu'il existe d'autres représentations possibles des langages réguliers. Ils sont, par exemple, représentables par les *grammaires régulières* (introduites dans la section 1.1.3) et les *expressions rationnelles* définies comme suit :

- $\forall a \in \Sigma$, a est une expression rationnelle,
- L'ensemble vide \emptyset et le mot vide λ sont des expressions rationnelles,
- Si e_1 et e_2 sont des expressions rationnelles, alors
 - leur concaténation, notée e_1e_2 et définie par $\{w \in \Sigma^* : \exists w_1 \in e_1, w_2 \in e_2, w = w_1w_2\}$, est une expression rationnelle (e_1e_1 est noté e_1^2),
 - leur union, notée $(e_1 + e_2)$ et définie par $\{w \in \Sigma^* : w \in e_1 \text{ ou } w \in e_2\}$ est une expression rationnelle,
- Si e est une expression rationnelle alors sa fermeture de Kleene, notée e^* , est une expression rationnelle et $e^* = \{w \in \Sigma^* : \exists n \geq 0, w = e^n\}$.

Grammaire hors-contextes

Une *grammaire hors-contexte*, ou *grammaire algébrique*, est un quadruplet $G = \langle \Sigma, V, P, S \rangle$ où Σ est l'alphabet des *symboles terminaux* (ou lettres), V celui des *symboles non-terminaux* (ou simplement des *non-terminaux*), $P \subseteq V \times (\Sigma \cup V)^*$ est un ensemble fini de *règles de production* et $S \in V$ l'*axiome*. La somme des longueurs des règles de production d'une grammaire algébrique est une mesure pertinente de la taille de cette grammaire. On note $uTv \Rightarrow uvv$ quand $(T, w) \in P$. \Rightarrow^* est la fermeture réflexive et transitive de \Rightarrow . Le langage engendré par une

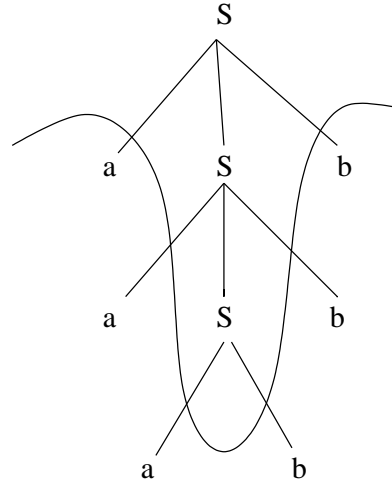


FIG. 1.2 – Un arbre de dérivation du mot $aaabbb$ dans la grammaire G_2 .

grammaire G , noté $\mathcal{L}(G)$, est $\{w \in \Sigma^* \mid S \xRightarrow{*} w\}$. On appelle *forme sententielle*, notée dans la suite $\alpha, \beta \dots$, toute séquence d'éléments de $(\Sigma \cup V)^*$ contenant au moins un élément de V (sinon c'est un mot).

Un *arbre ordonné* [Kil92] t est un quadruplet $\langle N_t, A_t, \text{root}(t), \geq \rangle$ où N_t est un ensemble de noeuds, $A_t \subseteq N_t \times N_t$ et $\text{root}(t) \in N_t$ est un noeud spécial appelé *racine de l'arbre*. Chaque paire $(u, v) \in A_t$ est appelé *arête* de l'arbre, u est le *parent* de v , v un *fil* de u . A_t doit vérifier que: (1) $\text{root}(t)$ est le seul noeud sans parent, (2) tous les autres noeuds ont un unique parent, (3) pour tout noeud n de l'arbre, il existe n_1, \dots, n_k tel que $(n_i, n_{i+1}) \in A_t$, $(\text{root}(t), n_1) \in A_t$ et $n_k = n$. Pour tout noeud n possédant $k (> 0)$ fils n_1, \dots, n_k , \geq est un ordre total sur les fils: $n_1 \geq n_2 \dots \geq n_k$. Un noeud possédant au moins un fil est appelé *noeud interne*; un noeud sans fils est appelé *noeud externe* ou *feuille* de l'arbre.

Un *arbre de dérivation*, ou *arbre syntaxique*, d'un mot m dans une grammaire G est un arbre ordonné dont la racine est l'axiome; un noeud N possède des fils n_1, n_2, \dots, n_n si et seulement si $N \rightarrow n_1 n_2 \dots n_n \in P$, et le parcours préfixe de l'arbre (c'est-à-dire le parcours en profondeur commençant par la racine et, pour chaque noeud interne, visitant les fils de gauche à droite) donne le mot m . Les noeuds internes sont donc des non-terminaux, les feuilles, des lettres de l'alphabet.

Exemple 1.2 La grammaire $G_1 = \langle \{a, b\}, \{S, N\}, \{S \rightarrow aaN, N \rightarrow a, N \rightarrow b\}, S \rangle$ engendre le langage $L_1 = \{aaa, aab\}$.

La grammaire $G_2 = \langle \{a, b\}, \{S\}, \{S \rightarrow aSb, S \rightarrow ab\}, S \rangle$ engendre le langage $L_2 = \{a^n b^n : n > 0\}$. $aaabbb$ est engendré par G_2 car $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$. L'arbre de cette dérivation est représenté sur la Fig. 1.2.

Automate à pile

Les langages hors-contextes peuvent aussi être représentés par les *automates à pile*. Un automate à pile est un septuplet $\langle Q, q_0, T, \Sigma, \Gamma, \tau, \Delta \rangle$ tel que Q est un ensemble fini d'états, $q_0 \in Q$ est l'état initial, $T \subseteq Q$ est un ensemble fini d'états finaux, Σ est l'alphabet des symboles d'entrée, Γ est l'alphabet des symboles de pile, $\tau \in \Gamma$ est le symbole de fond de pile et Δ est la relation de transition $\Delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})^* \rightarrow Q \times (\Gamma \cup \{\lambda\})^*$. Une transition $\Delta(q, a, s) = (q', s')$ veut dire que l'automate dans l'état q , lisant la lettre a avec en sommet de pile l'enchaînement de symboles s se rend dans l'état q' et met en sommet de pile s' . Si $a = \lambda$ alors on est arrivé à la fin du mot; si $s' = \lambda$ alors l'automate consomme les symboles de s (il les dépile). Le langage reconnu par un automate à pile est l'ensemble des chaînes de Σ qui permettent de passer de l'état initial à un état final en démarrant avec une pile contenant uniquement τ et en terminant avec une pile vide.

Exemple 1.3 *Le langage $\{a^n b^n : n > 0\}$ est reconnu par l'automate à pile $A = \langle \{q_i, q_a, q_b, q_f, q_e\}, q_i, \{q_f\}, \{a, b\}, \{s_{vide}, s_a\}, s_{vide}, \Delta \rangle$, où*

- $\Delta(q_i, a, s_{vide}) = (q_a, s_a), \Delta(q_i, b, s_{vide}) = (q_e, s_{vide}), \Delta(q_i, \lambda, s_{vide}) = (q_e, s_{vide})$ (lecture du premier caractère),
- $\Delta(q_a, a, s_a) = (q_a, s_a s_a)$ (lecture d'un a en début de mot),
- $\Delta(q_a, b, s_a) = (q_b, \lambda)$ (lecture d'un b après un certain nombre de a),
- $\Delta(q_b, b, s_a) = (q_b, \lambda)$ (lecture d'un b après un certain nombre de a suivi d'un nombre inférieur de b),
- $\Delta(q_b, a, -) = (q_{e,-})^1$ (lecture d'un a après un certain nombre de b),
- $\Delta(q_b, \lambda, s_{vide}) = (q_f, s_{vide})$ (fin de la lecture d'un mot appartenant au langage),
- $\Delta(q_b, -, s_{vide}) = (q_e, s_{vide})$ (lecture d'une lettre alors que la pile redevenue vide),
- $\Delta(q_e, -, -) = (q_{e,-})$ (lecture de n'importe quelle lettre dans l'état puits).

Globalement, le comportement de cet automate à pile est le suivant : pour chaque a lu en début de mot, il empile un symbole s_a et reste dans l'état q_a ; puis quand le premier b est lu il passe dans l'état q_b et dépile tant qu'il lit des b ; si la lettre suivante est un a , ou si la pile devient vide mais que le mot n'a pas fini d'être lu, il rentre dans l'état puits q_e qui correspond à un échec de reconnaissance du mot. L'automate passe dans l'état final s'il est dans l'état q_b , s'il n'y a plus de lettre à lire et si la pile est vide.

¹ $-$ peut être remplacé par n'importe quel élément.

Automate d'arbre ascendant déterministe

Nous n'utiliserons pas dans ce document ce formalisme permettant, entre autre, de représenter les langages hors-contextes, mais certains travaux présentés dans cette première partie s'en servent, ce qui constitue un motif suffisant pour les introduire ici.

Un *automate d'arbre ascendant déterministe* [CDG⁺97] est un quadruplet $\langle Q, \Sigma_g, \delta, F \rangle$ où Q est un ensemble d'états, Σ_g est un alphabet gradué $(A, Arite)$, $\delta : \Sigma_g \times Q^* \rightarrow Q$ la fonction de transition et $F \subseteq Q$ l'ensemble des états finaux. $Arite$ est une fonction de A dans \mathbb{N} . L'alphabet Σ du langage est contenu dans Σ_g et $\forall a \in \Sigma, Arite(a) = 0$. Pour faire le parallèle avec une grammaire hors-contexte $\langle \Sigma, V, P, S \rangle$, si $N \rightarrow u_1 \dots u_k \in P$, alors il existe un symbole N' dans A et $Arite(N') = k$. La fonction de transition δ est définie comme suit : pour tout $a \in \Sigma_g$ tel que $Arite(a) = n$, et $q_1, q_2, \dots, q_n \in Q$, $|\{q : \delta(a, q_1, q_2, \dots, q_n) = q\}| \leq 1$.

Exemple 1.4 Soit A l'automate d'arbre déterministe $\langle \{p, q, r\}, (\{a, b, s, s'\}, Arite), \delta, \{r\} \rangle$, avec $Arite(a) = 0$, $Arite(b) = 0$, $Arite(s) = 2$ et $Arite(s') = 3$. Si la fonction de transition est définie comme suit : $\delta(a) = p$, $\delta(b) = q$, $\delta(s, p, q) = r$ et $\delta(s', p, r, q) = r$, alors le langage du feuillage reconnu par A est $\{a^n b^n : n > 0\}$.

À noter qu'il existe d'autres classes de langages formels et d'autres représentations. Elles seront introduites au besoin dans la suite de ce document.

1.1.3 Hiérarchie de Chomsky

Si nous avons déjà défini les grammaires hors-contextes, l'étude de ces représentations de langages ne saurait être complète sans une vision plus globale. En fait, les grammaires hors-contextes sont une sous-classe des grammaires formelles, introduites dans les années cinquante par Noam Chomsky [Cho56] lors de ses travaux sur les langues naturelles. Il en a extrait quatre grandes classes hiérarchisées par une relation de contenance. Une grammaire formelle est un quadruplet $\langle \Sigma, V, P, S \rangle$ où Σ est l'alphabet des terminaux (ou lettres), V l'alphabet des non-terminaux, $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ un ensemble de règles de production et $S \in V$ l'axiome. Les classes de la *hiérarchie de Chomsky* sont définies par des contraintes syntaxiques sur la forme des règles de production :

- Les grammaires de type 0 n'ont aucune contrainte sur les règles de production,
- Les grammaires de type 1, ou grammaires sensibles au contexte, ont des règles de production qui ne contiennent qu'un seul non-terminal en partie gauche et une partie droite différente de λ , c'est-à-dire, si $(\alpha_1 \rightarrow \alpha_2) \in P$ alors $\alpha_1 \in \Sigma^* V \Sigma^*$ et $\alpha_2 \neq \lambda$,

- Les grammaires de type 2, ou grammaires hors-contextes, ou encore grammaires algébriques, ont des règles de production dont les parties gauches sont formées d'un unique non terminal, c'est-à-dire, si $(\alpha_1 \rightarrow \alpha_2) \in P$ alors $\alpha_1 \in V$,
- Les grammaires de type 3, ou grammaires régulières, ou encore grammaires rationnelles, ont des règles de production formées d'un non terminal en partie gauche et en partie droite soit d'une unique lettre, soit d'une lettre puis d'un non terminal, c'est-à-dire, si $(\alpha_1 \rightarrow \alpha_2) \in P$ alors $\alpha_1 \in V$ et $\alpha_2 \in \Sigma \cup \Sigma V$.

À chaque type de grammaires correspond une classe de langages. Par exemple, un langage est régulier s'il est représentable par une grammaire régulière; on dit qu'il est purement hors-contexte s'il peut être représenté par une grammaire hors-contexte, mais pas par une grammaire régulière.

Exemple 1.5 *Le langage $L_3 = \{a^n : n > 0\}$ est régulier car il peut être représenté par la grammaire régulière $\langle \{a\}, \{S\}, \{S \rightarrow a, S \rightarrow aS\}, S \rangle$. Il est à noter que L_3 peut aussi être représenté par une grammaire hors-contexte : par exemple, $\langle \{a\}, \{S\}, \{S \rightarrow SS, S \rightarrow a\}, S \rangle$. Le langage $L_4 = \{a^n b^n c^n : n > 0\}$ est sensible au contexte car il peut être représenté par une grammaire de type 1, mais pas par une grammaire de type 2 (une preuve de ce résultat se trouve dans [Aut87]).*

Seules les grammaires de type 3 et de type 2 nous intéressent dans ce document, mais il sera intéressant de mettre de temps en temps en perspective les résultats obtenus à l'aide de cette hiérarchie. Le principal intérêt de ces deux classes vient du fait que, contrairement aux grammaires de type 0, la question de l'appartenance d'un mot au langage engendré par une de ces grammaires est décidable efficacement (ce problème est décidable mais P -space complet pour les grammaires de type 1). Sans cette propriété, il est difficile, pour ne pas dire inutile, d'utiliser un formalisme de représentation de langages. En effet, notre but est de représenter des langages pour pouvoir décider l'appartenance d'un mot donné au langage. Pour ce faire, il nous faut donc utiliser une représentation des langages permettant de prendre cette décision en temps polynomial dans la taille du mot et de la représentation.

1.1.4 Propriétés des grammaires hors-contextes

Lorsque l'on utilise une classe de machines abstraites, le but n'est pas uniquement de représenter des langages : par exemple, pouvoir décider rapidement si un mot appartient au langage engendré par une grammaire est un problème important. Cela se fait en temps polynomial dans la taille du mot pour les grammaires de type 3 et 2, mais pas pour les autres.

Une autre question fondamentale est celle de l'équivalence de deux machines abstraites. On rappelle que deux représentations sont *équivalentes* si elles engendrent exactement le même langage. Pour les grammaires régulières, le problème est décidable. Cela découle du fait qu'il existe une forme normale canonique calculable pour ces langages : chaque langage régulier peut être représenté par un unique automate déterministe minimal en nombre d'états (aux noms des états près), calculable à partir de n'importe quel automate (par exemple, la Fig. 1.1 correspond à l'automate minimal du langage $(a + b)^*a(a + b)$).

En ce qui concerne les grammaires hors-contextes, pour un langage donné, il n'existe pas une grammaire unique minimale en nombre de non terminaux et de règles de production qui soit calculable. En effet, parmi toutes celles possédant un nombre minimal de non-terminaux, on peut définir une forme canonique, par exemple en étendant l'ordre hiérarchique aux règles, puis aux ensembles de règles. La grammaire canonique d'un langage hors-contexte peut alors être définie comme la grammaire représentant ce langage dont le nombre de non-terminaux est minimal et dont l'ensemble des règles est minimal vis-à-vis de l'ordre ainsi défini. Malheureusement une telle définition d'une forme canonique n'est que très peu utile car il n'est pas possible de transformer une grammaire quelconque en une telle grammaire. Ce résultat est lié à celui établissant que l'équivalence de deux grammaires hors-contextes n'est pas décidable. En effet, s'il existait une forme canonique calculable, il suffirait de transformer deux grammaires en forme canonique puis de comparer "syntaxiquement" les résultats. Nous verrons dans le chapitre suivant que l'indécidabilité de l'équivalence de deux grammaires hors-contextes est la source de grandes difficultés en apprentissage.

Toutefois, pour des raisons d'utilisation des grammaires hors-contextes, des formes normales ont été introduites. Toute grammaire hors-contexte peut être transformée en une de ces formes et, par conséquent il existe au moins une grammaire sous forme normale pour chaque langage hors-contexte (mais l'unicité n'est pas assurée). Un exemple d'utilisation de ces formes normales est l'algorithme CYK [You67], qui permet de tester l'appartenance d'un mot au langage engendré par une grammaire hors-contexte en un temps cubique dans la taille du mot. Pour ce faire, l'algorithme nécessite une grammaire sous *forme normale de Chomsky*.

Définition 1.1 (Forme normale de Chomsky)

Une grammaire hors-contexte $\langle \Sigma, V, P, A \rangle$ est sous forme normale de Chomsky si, quel que soit $(N \rightarrow \alpha) \in P$, α est sous l'une des formes suivantes :

- $\alpha \in \Sigma \cup \{\lambda\}$
- $\alpha \in V \times V$

Exemple 1.6 La grammaire $G = \langle \{a,b\}, \{S,N,A,B\}, P, S \rangle$, où $P = \{S \rightarrow AN, N \rightarrow SB, N \rightarrow b, A \rightarrow a, B \rightarrow b\}$ est une grammaire hors-contexte sous forme normale de Chomsky engendrant le langage $L_2 = \{a^n b^n : n > 0\}$.

Étant donnée une grammaire hors-contexte quelconque, il est possible de la transformer en forme normale de Chomsky en temps polynomial dans la taille de la grammaire initiale.

Une deuxième forme normale se retrouve souvent dans la littérature : la *forme normale de Greibach*. Initialement introduite pour passer d'une grammaire à un automate à pile, elle est aujourd'hui utilisée principalement pour définir des contraintes de déterminisme sur des grammaires algébriques.

Définition 1.2 (Forme normale de Greibach)

Une grammaire hors-contexte $\langle \Sigma, V, P, A \rangle$ est sous forme normale de Greibach si toutes ses productions sont de la forme $N \rightarrow a\alpha$, avec $a \in \Sigma$ et $\alpha \in V^*$.

Nous rappelons qu'il n'existe pas une unique forme normale calculable, que ce soit de Greibach ou de Chomsky, pour un langage hors-contexte donné.

1.2 Qu'est-ce qu'apprendre des langages ?

“Même les machines ont besoin d'apprendre”. C'est par cette phrase que commence le livre constituant la référence francophone du domaine de l'*apprentissage automatique*, ou *apprentissage artificiel* [CM02]. En effet, dans bien des domaines, on demande de plus en plus souvent aux ordinateurs de gérer des problèmes pour lesquels on ne peut donner une liste exhaustive de solutions, ni un protocole fixe de résolution. C'est la première justification de l'apprentissage automatique : permettre à l'ordinateur de prendre une décision face à une situation qu'il n'a encore jamais rencontrée, et pour laquelle il n'a pas explicitement été programmé. Bien entendu, ceci n'est possible que si cette situation est proche de celles qu'il a déjà vues.

Les tâches d'un tel système apprenant sont très variées, ce qui explique la diversité des solutions apportées : permettre à un ordinateur d'apprendre la langue française, afin de le rendre capable de corriger d'éventuelles fautes grammaticales, ne demandera pas la même mise en oeuvre qu'un programme permettant, par exemple, la reconnaissance de caractères manuscrits.

Nous avons déjà dit que la conséquence d'un apprentissage réside dans la capacité à s'adapter à une situation encore inconnue. Mais cela ne nous dit rien sur ce qu'est réellement un apprentissage, ni comment le réaliser ou le valider. Le concept de base est celui de l'*induction*, c'est-à-dire le processus par lequel on tire des règles de portée générale à partir d'observations particulières. Cela se réalise à l'aide de ce qui s'appelle un *principe inductif*, véritable base de ce que l'on nomme le raisonnement artificiel. Nous ne nous intéressons ici qu'à l'induction dite *supervisée* : au lieu de laisser le programme se débrouiller dans son environnement d'apprentissage, il reçoit les données accompagnées d'une connaissance *a priori*

de ce qu'il faut chercher à apprendre. Reprenons l'exemple de l'apprentissage d'un correcteur grammatical de la langue française. L'induction sera supervisée si les exemples de phrases servant pour l'apprentissage sont étiquetés, de façon à distinguer les exemples correspondant à des structures grammaticales correctes (exemples dits positifs) des autres (exemples dits négatifs).

Notre but est de définir et d'étudier des algorithmes permettant l'induction de langages formels. Ce domaine de l'apprentissage automatique est connu sous le nom d'*inférence grammaticale*. Nous nous plaçons dans une problématique de classification : un algorithme aura correctement appris s'il est capable de répondre à la question "ce mot appartient-il au langage?". Le fonctionnement d'un algorithme d'inférence grammaticale peut être formulé ainsi : l'algorithme possède un ensemble d'hypothèses, fini ou non, correspondant souvent à une classe de machines abstraites, et, à l'aide des données d'apprentissage et du (ou des) principe(s) inductif(s) choisi(s), retourne une représentation d'un langage. L'apprentissage sera réussi si cette représentation est "proche" de, ou équivalente à, celle du langage ayant généré les données d'apprentissage.

Le principe inductif principalement utilisé en apprentissage artificiel est connu sous le nom de *principe ERM*, de l'anglais Empirical Risk Minimization : l'idée est de choisir l'hypothèse qui correspond le mieux aux données d'apprentissage, c'est-à-dire qui minimise l'erreur empirique (*i.e.* l'erreur sur les données), supposant ainsi que ces données sont représentatives du concept à apprendre, et que donc l'erreur sur des données inconnues sera ainsi minimale. Ce principe est présent en inférence grammaticale mais sa portée est moindre que dans d'autres domaines : nos données d'apprentissage sont principalement composées d'exemples (et éventuellement de contre-exemples) de mots du langage et, si ces données ne sont pas bruitées, il est facile de trouver une hypothèse cohérente avec les données. Ce n'est bien entendu pas le cas quand le concept à apprendre chevauche d'autres concepts et que, par conséquent, il est impossible de totalement séparer les exemples positifs des exemples négatifs (dans le cas de données composées de plusieurs attributs numériques par exemple). En ce qui concerne l'inférence grammaticale, nous verrons que l'ensemble des hypothèses cohérentes avec un ensemble de données est bien souvent trop vaste pour que l'implémentation du principe ERM suffise à permettre un apprentissage.

D'autre part, la mise en oeuvre de principes inductifs n'est pas suffisante pour s'assurer qu'un algorithme est effectivement capable d'apprendre. Il existe plusieurs manières de tester la validité d'un algorithme d'apprentissage :

- On peut utiliser des jeux de données sur lesquels d'autres algorithmes ont été testés pour comparer les résultats obtenus (voir par exemple [DNM98] ou [SCvZ04]).
- On peut aussi faire de la validation croisée, c'est-à-dire n'utiliser qu'une par-

tie des données d'apprentissage pour apprendre et se servir du reste pour vérifier l'adéquation de la représentation induite avec les exemples disponibles. Des résultats formels ont montré la pertinence de cette approche d'un point de vue statistique [ET93].

- Enfin, on peut se placer dans un *paradigme d'apprentissage*, c'est-à-dire un cadre formalisant la notion d'apprentissage.

Les deux premiers points correspondent à une approche empirique, tandis que le dernier donne naissance à des résultats formels d'apprentissage. Nous étudierons dans la suite deux paradigmes d'apprentissage et discuterons de leur pertinence vis-à-vis du problème qui nous intéresse. Comme nous l'avons dit dans ce qui précède, ces paradigmes offrent un cadre définissant formellement ce que "apprendre" veut dire.

Le but de cette thèse est d'étudier et de proposer des algorithmes capables d'apprendre des langages hors-contextes. Jusqu'à récemment, les chercheurs de la communauté de l'inférence grammaticale se sont principalement concentrés sur la plus petite classe de la hiérarchie de Chomsky. De nombreux algorithmes ont ainsi été proposés pour ces langages réguliers. Certains seront explicités au besoin dans la suite, mais là n'est pas ce qui nous intéresse dans ce document. Nous commençons par l'étude d'algorithmes relevant de l'approche empirique, puis nous introduirons les deux principaux paradigmes d'apprentissage utilisés en inférence grammaticale.

1.2.1 Approches empiriques

Dans cette section nous allons présenter plusieurs approches empiriques, c'est-à-dire des algorithmes et des méthodes obtenant de bons résultats en pratique, qui reposent sur l'utilisation de principes inductifs, mais pour lesquels aucun résultat formel d'apprentissage n'a pu être obtenu (ou n'a été obtenu pour l'instant). Dans un premier temps, nous nous préoccuperons d'une classe d'algorithmes appelée "algorithmes génétiques". Ensuite, nous détaillerons deux approches différentes qui constituent des références couramment citées en exemple par les chercheurs de la communauté.

Une méta-heuristique : les algorithmes génétiques.

Les *algorithmes génétiques*, ou *algorithmes évolutionnaires*, constituent une voie de recherche particulièrement innovante en apprentissage automatique. Ainsi, à l'instar d'autres domaines de l'apprentissage automatique, l'inférence grammaticale a donné naissance à des algorithmes génétiques. Avant de détailler quelques unes des idées développées dans la littérature, nous rappelons les principes de

cette approche. Il s'agit dans un premier temps de trouver un codage permettant de représenter toutes les hypothèses. Par exemple, si notre intention est d'apprendre des grammaires hors-contextes, on peut imaginer de coder les grammaires sous forme de suites de lettres. Le but est d'établir un parallèle avec ce qui constitue le moteur de l'évolution des espèces : l'ADN. Viennent ensuite deux phases directement calquées sur les découvertes des généticiens : le *cross-over* et la *mutation*. Ces deux événements constituent des "erreurs" biologiques lors de la division cellulaire, très rares et pourtant principaux responsables de l'évolution des espèces.

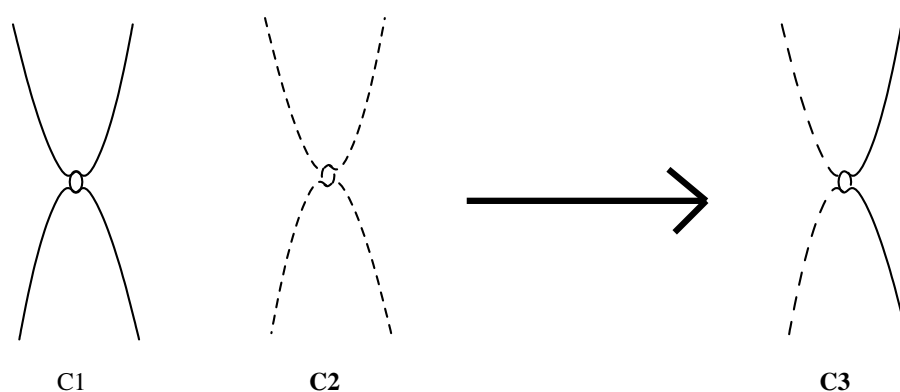


FIG. 1.3 – Représentation d'un *cross-over*. Le chromosome *C3* est le résultat d'un *cross-over* entre le chromosome *C1* et le chromosome *C2*.

Le *cross-over* correspond à un mélange de deux brins d'ADN : un morceau de l'un est échangé avec un morceau d'un autre. Pour se ramener à notre problème d'algorithmique, cela revient à prendre le codage de deux grammaires et à interchanger des bouts de code. À l'instar de ce qu'il se passe en génétique, un tel mélange, réalisé de manière aléatoire, a peu de chances de correspondre au codage d'une grammaire. C'est pourquoi le concepteur d'un algorithme génétique doit s'assurer d'un côté de la robustesse de son codage à de telles modifications, de l'autre de ne pas permettre au *cross-over* de réaliser tout et n'importe quoi. La difficulté réside ici : pour que l'algorithme soit capable d'approcher le langage qu'il cherche à apprendre, il faut qu'il puisse inférer un large éventail d'hypothèses tout en minimisant le risque de création de non-sens.

Le processus de mutation est plus simple à expliquer. Tout comme en génétique, où un nucléotide peut être remplacé par un autre par erreur ou sous l'effet de l'environnement, cette phase d'un algorithme génétique consiste à modifier un codage, par exemple en remplaçant une occurrence d'une lettre par une autre. Les mutations peuvent être encadrées, c'est-à-dire que l'on ne permet pas n'importe quelle modification, mais il n'est pas question de se servir du sens sous-jacent au

codage pour la définir.

Il reste à modéliser la *sélection naturelle*. Un algorithme génétique possède à chaque instant une *population d'hypothèses* générée par des mutations et des cross-overs. Il lui faut donc sélectionner des hypothèses avant de relancer une phase de mutation et de cross-over. Cela se fait à l'aide d'une fonction d'évaluation sensée rendre compte de la pertinence de l'hypothèse.

Plusieurs travaux ont utilisé des algorithmes génétique pour apprendre des grammaires [KB96, SK99]. Plus récemment, une approche utilisant un algorithme génétique [PPK⁺04] a réussi à résoudre un des problèmes de la compétition OMPHALOS (la dernière partie de cette section contient plus plus d'information sur cette compétition).

Combinaison de deux principes inductifs.

Pour le moment, nous n'avons introduit qu'un seul principe inductif, le principe ERM. Un autre principe est connu sous le nom de *principe de compression de l'information*, ou *principe MDL*, de l'anglais *Minimum Description Length*. Il part du présupposé qu'apprendre c'est représenter de manière compacte les données d'apprentissage. Un algorithme reposant sur ce principe tente d'éliminer les redondances de ce qu'il connaît du concept à apprendre dans le but d'extraire les régularités sous-jacentes. Ainsi, si les régularités présentes dans les données correspondent à celles du langage que l'on essaie de trouver, une telle approche peut permettre un apprentissage: par exemple, si nous cherchons à apprendre le langage régulier $(aa)^*$, un algorithme cherchant à minimiser la représentation des données se rendra certainement compte que tous les exemples d'apprentissage contiennent un nombre pair de a .

L'algorithme GRIDS [LS00] utilise une combinaison des deux principes sus-cités: l'induction est menée à l'aide du principe de compression de l'information, sa validité est testée grâce au principe ERM. L'idée est de partir de la grammaire qui engendre les exemples d'apprentissage, et seulement ceux-là. Ensuite, des non-terminaux sont créés quand deux parties droites de règle partagent une même sous-séquence de symboles. Des non-terminaux sont fusionnés si leurs règles de production sont suffisamment proches. La Table 1.1 explicite un exemple de création de non-terminal et un exemple de fusion de deux non-terminaux.

Le fonctionnement de l'algorithme est assez proche de celui décrit pour les algorithmes génétiques: dans un premier temps il considère toutes les créations possibles de non-terminaux. Ensuite, il fait de même avec les grammaires résultantes pour les fusions de non-terminaux. Il utilise alors une fonction d'évaluation lui permettant de sélectionner la "meilleure" grammaire et de tester si cette grammaire obtient une évaluation supérieure à celle qu'il avait sélectionnée précédemment. Si c'est le cas, l'algorithme recommence le cycle des deux étapes

Création d'un nouveau non-terminal	Fusion de deux non-terminaux
$NP \rightarrow ART\ ADJ\ NOM$ $NP \rightarrow ART\ ADJ\ ADJ\ NOM$ \Downarrow	$NP \rightarrow ART\ NT_1$ $NP \rightarrow ART\ NT_2$ $NT_1 \rightarrow ADJ\ NOM$ $NT_2 \rightarrow ADJ\ NT_1$ \Downarrow
$NP \rightarrow ART\ NT_1$ $NP \rightarrow ART\ ADJ\ NT_1$ $NT_1 \rightarrow ADJ\ NOM$	$NP \rightarrow ART\ NT$ $NT \rightarrow ADJ\ NOM$ $NT \rightarrow ADJ\ NT$

TAB. 1.1 – *Exemples d'actions que réalise l'algorithme GRIDS. Remarquons que la fusion des non-terminaux NT_1 et NT_2 entraîne une généralisation : le langage engendré est infini, car le nouveau non-terminal NT est récursif, alors que ce n'était pas le cas avant la fusion.*

”création-fusion”. Sinon, il retourne la grammaire précédente.

La fonction d'évaluation est l'implémentation du principe de compression de l'information. Cette fonction est une métrique rendant compte de la ”simplicité” de la grammaire : elle prend en considération la taille de la grammaire (le nombre de non-terminaux, la taille des parties droites des règles, etc.) et la longueur des dérivations des mots de l'échantillon d'apprentissage dans cette grammaire. Ainsi, une grammaire est meilleure qu'une autre si le nombre de bits nécessaires pour réaliser le codage de cette grammaire et celui des mots de l'ensemble d'apprentissage dans cette grammaire est plus petit que le nombre de bits utilisés par le codage de la seconde grammaire (et des exemples d'apprentissage dans cette grammaire). Nous ne détaillons pas les raisons, mais cette fonction permet d'éviter ce que l'on appelle le *sur-apprentissage* : comme cet algorithme utilise seulement des exemples du langage à apprendre, la grammaire inférant Σ^* répond au principe ERM, ce qui implique que, sans contrôle, ce serait toujours elle qui serait inférée.

L'utilisation du principe ERM est explicite : initialement, la grammaire n'engendre que les exemples d'apprentissage, et chaque étape ne fait que *généraliser* le langage engendré (le nouveau langage inclut l'ancien).

Dans l'article, les auteurs montrent sur quelques expérimentations simples que l'algorithme obtient de bons résultats. Mais, de leur propre aveu, le processus est trop gourmand en temps et en mémoire pour être utilisé tel quel sur de grands échantillons d'apprentissage. De plus, l'utilisation d'une fonction d'évaluation heuristique ne leur permet pas d'obtenir un résultat formel d'apprentissage.

Toutefois cette approche est prometteuse et d'autres chercheurs l'ont reprise pour l'adapter à des problèmes particuliers. On peut citer par exemple l'algorithme E-GRIDS [PPK⁺04] qui combine cette approche avec un algorithme génétique, ou encore l'algorithme BOISDALE [SF04].

Une approche statistique : utilisation d'une entropie.

En 2003 a eu lieu la première compétition en ligne d'apprentissage de langages hors-contextes, appelée OMPHALOS [SCvZ04]. Signe du renouveau de l'intérêt suscité par cette classe, elle était constituée de dix problèmes pour lesquels étaient donnés un échantillon d'apprentissage et un échantillon de test. Le but était d'apprendre une représentation du langage cible à partir de l'échantillon d'apprentissage, puis de classer les mots de l'échantillon de test en deux classes : ceux qui appartiennent au langage et les autres. On considérait qu'un problème était résolu si tous les mots étaient correctement étiquetés. C'est pourquoi les six premiers problèmes possédaient deux ensembles à étiqueter : un petit, permettant un apprentissage "approché", et un grand. Les échantillons d'apprentissage étaient constitués soit uniquement de mots du langage (exemples positifs), soit d'exemples positifs et d'exemples négatifs. Seuls les six premiers petits échantillons de test et deux des grands ont été correctement étiquetés. Au premier septembre 2006, les problèmes sept à dix n'ont toujours pas été résolus.

Le but ici est de présenter rapidement la méthode ayant obtenu les meilleurs résultats. L'approche gagnante, développée par Alexander Clark [Cla06a], a pour but de trouver les constituants du langage :

Définition 1.3 (Constituant)

Soit $G = \langle \Sigma, V, P, S \rangle$ une grammaire hors-contexte et $L = \mathcal{L}(G)$ son langage engendré. Une occurrence d'un facteur u dans un mot lur du langage L est un constituant de L si il existe $N \in V$ tel que $S \xRightarrow{*} lNr$ et $N \xRightarrow{*} u$.

En général, le même facteur peut être un constituant pour certaines de ses occurrences et un non-constituant pour les autres. Cela ne se produit pas pour les grammaires hors-contextes dites NTS [Boa80].

Définition 1.4 (Grammaire NTS)

Une grammaire hors-contexte $G = \langle \Sigma, V, P, S \rangle$ est NTS (Non Terminally Separated) si pour tout $N_1, N_2 \in V$ et mots $u, l, r \in \Sigma^*$ tels que $N_1 \xRightarrow{*} lur$ et $N_2 \xRightarrow{*} u$ il existe une dérivation $N_1 \xRightarrow{*} lN_2r$.

Clark remarque qu'une grammaire générée aléatoirement, comme celles de OMPHALOS, a de grandes probabilités d'être NTS (chances qui augmentent avec

la taille de l'alphabet). Il ne reste plus qu'à trouver le moyen de mettre en évidence les constituants, et on pourra inférer une grammaire.

Pour ce faire, il faut dans un premier temps construire l'ensemble des facteurs des exemples positifs contenus dans l'échantillon d'apprentissage. Le présupposé est que tous les constituants ont une fréquence d'apparition importante, ce qui permet d'éliminer un certain nombre de facteurs. Mais cela ne suffit pas : on trouve aussi souvent, voir plus fréquemment, les facteurs des constituants que les constituants eux-mêmes. C'est pourquoi dans un deuxième temps il faut sélectionner parmi les facteurs restants ceux qui ont le plus de chance d'être des constituants. Cela se fait par une mesure statistique, une entropie, appelée *information mutuelle* [Cla01]. L'information mutuelle d'un facteur u est donnée par la formule suivante :

$$MI(w) = \sum_{l \in \Sigma'} \sum_{r \in \Sigma'} \frac{f(lwr)}{f(w)} \log \frac{f(lwr)f(w)}{f(lw)f(wr)}$$

où $f(x)$ est la fréquence du facteur x dans les données, Σ' est l'alphabet du langage Σ auquel est ajouté un marqueur de début et de fin de mot (pour ne pas pénaliser les facteurs apparaissant à ces endroits dans les exemples). Une étude rapide de la formule nous permet de dire que le MI sera d'autant plus élevé que w apparaît dans des contextes différents. La méthode consiste donc à sélectionner les facteurs dont l'information mutuelle est la plus élevée.

En supposant que les étapes précédentes nous aient permis de trouver les constituants du langage, il nous reste à inférer une grammaire. Cela se fait à l'aide d'un *graphe de substitution*. Les noeuds de ce graphe sont constitués par les facteurs précédemment sélectionnés. Il y a un arc entre deux noeuds si les facteurs correspondants apparaissent au moins une fois dans le même contexte. A chaque *composante connexe* du graphe, c'est-à-dire à chaque ensemble de sommets directement ou indirectement reliés entre eux, est affectée un non-terminal dont les règles de production génèrent chaque noeud de la composante (tous ces constituants ont été générés par le même non terminal). On remplace ensuite toutes les occurrences des facteurs du graphe par le non-terminal correspondant et on recommence le processus de sélection sur les formes sententielles ainsi obtenues. L'algorithme s'arrête quand la grammaire inférée est capable d'engendrer l'échantillon d'apprentissage.

Bien entendu cette méthode est purement heuristique : la fréquence et le MI minimaux pour qu'un facteur soit sélectionné sont des paramètres de l'algorithme qu'il faut patiemment faire évoluer pour trouver les optimaux. Mais l'algorithme n'a une chance d'aboutir que si la grammaire est NTS, ce qui n'est pas forcément le cas de celles de la compétition. C'est pourquoi d'autres heuristiques, utilisant par exemple les exemples négatifs quand ils étaient disponibles, ont été utilisées pour gagner la compétition.

1.2.2 Le cadre PAC

Le premier cadre d'apprentissage que nous présentons ici, appelé *apprentissage PAC*, est le plus utilisé en apprentissage automatique. Malheureusement, nous verrons qu'il n'est pas vraiment adapté à l'inférence grammaticale, du moins à la branche non-stochastique qui nous intéresse. Dans la suite, un autre cadre d'apprentissage sera présenté.

Ce cadre d'apprentissage, connu sous le nom d'*apprentissage PAC*, pour *apprentissage Probablement Approximativement Correct* a été proposé par Valiant au début des années quatre-vingt [Val84]. La définition suivante est celle développée par Pitt [Pit89] pour l'inférence grammaticale de langages réguliers, adaptée aux langages hors-contextes :

Définition 1.5 (Apprentissage PAC)

Une classe de langage \mathbb{L} est apprenable au sens PAC s'il existe un algorithme \mathcal{A} tel que :

- pour tout langage $L \in \mathbb{L}$ dont la représentation est de taille n ,
- pour toute distribution D sur les mots de Σ^* restreinte à $\Sigma^{\geq m}$
- pour tout $\delta > 0$ et $\epsilon > 0$,

l'algorithme \mathcal{A} , prenant en entrée δ et ϵ et utilisant un échantillon de taille polynomiale en n , m , $1/\delta$ et $1/\epsilon$, retourne une hypothèse H , en temps polynomial en n , m , $1/\delta$ et $1/\epsilon$, telle que :

$$P(P_D(L \oplus \mathcal{L}(H)) < \epsilon) > 1 - \delta.$$

Moins formellement, la probabilité des erreurs (*i.e.* la fréquence par rapport à D des éléments de la différence symétrique du langage cible et de celui de l'hypothèse retournée, $L \oplus \mathcal{L}(H)$) doit être inférieure à ϵ avec une probabilité supérieure à $1 - \delta$: l'apprentissage est approximativement correct si l'erreur est proche de zéro, il est probablement approximativement correct si l'approximation correcte est suffisamment probable. Notons que l'apprentissage doit avoir lieu quelle que soit la distribution des exemples.

La grande majorité des résultats théoriques concernant l'apprentissage des langages hors-contextes dans le cadre PAC sont négatifs. Une façon usuelle de procéder pour obtenir de tels résultats est de réduire le problème de l'induction des langages hors-contextes à un autre problème [PW88] : il a été prouvé que la difficulté était analogue à celle de problèmes cryptographiques (revenant, par exemple, à casser le code RSA), que ce soit le problème d'extraction de racines cubiques par le calcul modulaire ou celui de la factorisation de très grands entiers [KV89]. Quelques résultats positifs ont toutefois été obtenus pour des sous-classes extrêmement restreintes, comme les ensembles semi-linéaires [Abe95] (dont

les nombres d'occurrences des symboles sont définis par des systèmes d'équations linéaires).

Une restriction du cadre PAC a été introduite en apprentissage automatique. Il s'agit du cadre *simple* PAC qui restreint les distributions possibles des exemples à être *simple*, dans le sens de la complexité de Kolmogorov [LV91]. Dans ce cadre, la probabilité d'apparition des exemples "simples" doit être supérieure à celle des autres exemples. Cela a permis d'obtenir des résultats positifs concernant la classe des langages réguliers [Den01], mais aucun résultat n'a été jusqu'à présent apporté concernant les langages hors-contextes. Toutefois, cette idée semble plus ou moins être présente dans le système EMILE [AV02]. Enfin, dans un article très récent [Cla06b], un algorithme permet le PAC-apprentissage d'une sous-classe des langages NTS à partir de données dont la distribution possède des propriétés restrictives.

Malgré le très faible nombre de résultats positifs, et les fortes conséquences des résultats négatifs, l'apprentissage PAC de sous-classes des langages hors-contextes reste d'actualité, principalement pour les chercheurs s'intéressant à l'acquisition par les enfants de leur langue maternelle. En effet, si on considère, par exemple, qu'il existe une seule langue française, disons le français de l'Académie, très peu de francophones le maîtrisent complètement. Ce que nous avons appris (et continuons d'apprendre) est une approximation de ce français. Et encore, nul n'est à l'abri d'une erreur grammaticale... Ce qui correspond à la définition du cadre PAC, à la différence que, même si nous n'avons pas tous été exposés à la même distribution d'exemples, il n'est pas concevable de penser que toutes les distributions permettent le même apprentissage (par exemple, l'origine sociale a une grande influence sur la maîtrise de la langue). Les résultats positifs introduits ci-dessus se restreignent d'ailleurs tous à des distributions particulières.

Finalement, notons que le cadre PAC, de par l'utilisation d'une distribution de probabilité sur les exemples d'apprentissage, n'est pas adapté à ce que nous cherchons à faire : une approche visant à apprendre des grammaires stochastiques serait en meilleure adéquation avec ce modèle. Or nous nous intéressons uniquement à l'apprentissage non-stochastique : quelques travaux existent sur l'apprentissage de grammaires hors-contextes stochastiques [SBH⁺94] et de grammaires d'arbres stochastiques [AM97, COCR01, HBJ02], et leurs propriétés ont été clairement étudiées [JLP01]. La plupart des approches sont purement heuristiques pour répondre à un problème pratique particulier [KB96, Cas95] et, à notre connaissance, la question de l'apprentissage de la structure du langage et celle de la loi de probabilité des exemples n'ont jamais été étudiées ensemble de manière formelle (deux résultats empiriques intéressants peuvent toutefois être cités : [SB97] et [SB02]). Le principal résultat non-empirique utilise le principe inductif du *maximum de vraisemblance* [LY90] et suppose que la structure est donnée, ce qui correspond à l'apprentissage d'une loi de probabilité et non d'un langage.

Nous restreignons pour ces raisons notre étude à un autre paradigme d'apprentissage, l'*identification à la limite*.

1.2.3 Le cadre de l'identification à la limite

Le cadre de l'*identification à la limite* a été introduit dans les années 60 par Gold [Gol67]. Il correspond à un apprentissage exact mais pas asymptotique : un algorithme identifiant un langage à la limite retournera une représentation exactement équivalente au langage qu'il cherche à apprendre, mais seulement après avoir reçu un nombre de données d'apprentissage potentiellement très grand; avant qu'il n'ait identifié ce qu'il cherche à apprendre, sa représentation peut en être très éloignée. En fait, dans ce paradigme, du moins dans sa version initiale, les algorithmes travaillent à partir d'une séquence infinie de données d'apprentissage appelée *présentation*. Toutes les valeurs possibles des données d'apprentissage apparaissent au moins une fois dans une présentation d'un langage. Par P_n nous notons les n premiers éléments d'une présentation P . Le paradigme s'écrit alors ainsi :

Définition 1.6 (Identification à la limite)

Une classe de langages \mathbb{L} est identifiable à la limite à partir des présentations de type $\mathbb{P}_{\mathbb{L}}$ s'il existe un algorithme ϕ tel que, pour tout langage $L \in \mathbb{L}$ et quelle que soit la présentation P de L de type $\mathbb{P}_{\mathbb{L}}$, il existe un rang n à partir duquel $\mathcal{L}(\phi(P_n)) = L$.

Différents types de présentations ont été étudiés. Les deux plus fréquemment utilisés sont les *textes* et les *informants*. Les premiers correspondent à des présentations composées uniquement d'exemples de mots du langage, autrement appelés exemples positifs. Les seconds sont des présentations formées d'exemples positifs et de mots n'appartenant pas au langage, appelés contre-exemples ou exemples négatifs. Dans le cas d'un informant, les données prennent la forme d'un couple (x, u) , où x est un mot sur l'alphabet du langage et u une étiquette valant usuellement 1 (ou +) si x est un exemple positif, et -1 (ou $-$) s'il s'agit d'un exemple négatif. D'autres types de présentations ont été développés, nous les introduirons au besoin dans la suite.

Ce paradigme, incrémental, est éloigné de la pratique où nous possédons un échantillon de données, c'est-à-dire un ensemble fini de données d'apprentissage, et non une séquence infinie. C'est pourquoi le cadre de l'*identification à la limite par données fixées* a été introduit [Gol78]. Il repose sur la définition d'un *échantillon caractéristique* :

Définition 1.7 (Echantillon caractéristique)

Soit CS un ensemble de données d'apprentissage d'un langage L . CS est un

échantillon caractéristique de L pour l'algorithme ϕ si pour tout ensemble de données S , $CS \subseteq S \Rightarrow \mathcal{L}(\phi(S)) = L$.

Moins formellement, un échantillon est caractéristique d'un langage L pour un algorithme ϕ si, chaque fois qu'il est contenu dans les données d'apprentissage, l'algorithme retourne une représentation équivalente à celle qu'il cherche. Ce qui permet de définir le paradigme suivant :

Définition 1.8 (Identification à la limite par données fixées)

Une classe de langages \mathbb{L} est identifiable à la limite par données fixées à partir d'échantillons de type $\mathbb{E}_{\mathbb{L}}$ s'il existe un algorithme ϕ tel que pour tout langage $L \in \mathbb{L}$ il existe un échantillon caractéristique CS pour ϕ .

Les différents types d'échantillons correspondent aux différents types de présentations introduits précédemment. On parlera ainsi d'échantillons d'exemples positifs et d'échantillons d'exemples positifs et négatifs. Dans le second cas, au lieu de représenter les données par une paire (exemple, étiquette), comme c'est le cas pour les informants, il est fréquent de définir l'échantillon comme un couple $\langle S_+, S_- \rangle$ formé de deux ensembles, S_+ contenant les exemples positifs, S_- les exemples négatifs.

Le comportement d'un algorithme défini dans le premier cadre d'identification à la limite présenté ici est souvent, pour ne pas dire toujours, le suivant : à chaque instant il possède une hypothèse courante, qu'il ne modifie que quand une nouvelle donnée vient la contredire (on parle alors d'*algorithmes consistants*). Dans ce cas, il infère une nouvelle hypothèse à partir de l'ensemble des données qu'il a reçu jusqu'à présent, sans tenir compte de l'ordre dans lequel elles lui ont été fournies. Intuitivement, cela revient à un processus d'apprentissage à l'aide d'un échantillon, d'un ensemble de données. Plus formellement, Gold a prouvé [Gol78] que l'identification incrémentale à la limite est équivalente à l'identification à la limite par données fixées². Les deux paradigmes expriment donc la même notion d'apprentissage.

Le problème des définitions précédentes est leur absence de contrainte de polynomialité : une classe de langages pourrait être identifiable à la limite, mais demander un nombre gigantesque de données d'apprentissage (la taille de l'ensemble caractéristique pourrait être déraisonnablement grande). Par conséquent, un résultat d'apprentissage dans ces paradigmes peut être inutile en pratique. C'est pourquoi ils ont été étendus. Le premier a donné naissance [Pit89] à celui qui suit :

Définition 1.9 (Identification polynomiale à la limite)

² En fait, la preuve initiale ne concerne qu'un sens de l'équivalence. Mais, bien que cela n'ait jamais été publié, la réciproque est aussi vraie.

Une classe de langages \mathbb{L} est identifiable polynomialement à la limite à partir des présentations de type $\mathbb{P}_{\mathbb{L}}$ et en utilisant la classe de représentations \mathbb{R} , s'il existe un algorithme ϕ tel que, pour tout langage $L \in \mathbb{L}$ et pour toute présentation P de L de type $\mathbb{P}_{\mathbb{L}}$:

- le temps d'exécution de ϕ est polynomial dans la taille de P_n pour tout n ,
- le nombre maximum d'erreurs de prédiction que fait l'algorithme est polynomial, c'est-à-dire qu'il existe un polynôme $p(\cdot)$ tel que $|\{n \in \mathbb{N} : \mathcal{L}(\phi(P_n)) \neq L\}| < p(\|r_L\|)$, où r_L est la plus petite représentation de L dans \mathbb{R} .

Cette définition nécessite un codage, ou une description, raisonnable des représentations des langages (voir section 1.1.2 pour plus de détails).

Il est à noter que dans la définition précédente (mais aussi la suivante) la plus petite représentation d'un langage n'a pas besoin d'être calculable : seule son existence nous intéresse. Dans la formalisation de l'enseignement développée par Matthias et Goldman [GM96], cela correspond à des problèmes "semi-poly-enseignables", c'est-à-dire que l'apprenant fonctionne de manière polynomiale alors que l'enseignant peut nécessiter d'utiliser des processus exponentiels. (A contrario, un problème est "enseignable" si le couple enseignant/enseigné a un comportement entièrement polynomial.)

Pour en revenir à l'identification polynomiale, aussi séduisant que soit ce paradigme, il est malheureusement bien souvent trop restrictif : même les langages réguliers ne sont pas apprenables dans ce cadre à partir d'informants [Pit89]. C'est pourquoi un troisième paradigme d'identification à la limite a été introduit. Il repose sur celui d'identification à la limite par données fixées, auquel deux contraintes de polynomialité ont été ajoutées [dlH97].

Définition 1.10 (Identification en temps et données polynomiaux)

On dit qu'une classe de langages \mathbb{L} est identifiable à la limite en temps et données polynomiaux à partir d'échantillons de type $\mathbb{E}_{\mathbb{L}}$ et en utilisant la classe de représentations \mathbb{R} , s'il existe un algorithme ϕ et deux polynômes $p(\cdot)$ et $q(\cdot)$ tels que :

1. Étant donné un échantillon S de taille m pour $L \in \mathbb{L}$, ϕ retourne en un temps $\mathcal{O}(p(m))$ une hypothèse $H \in \mathbb{R}$ cohérente³ avec S ;
2. Pour toute représentation $R \in \mathbb{R}$ de taille k d'un langage $L \in \mathbb{L}$, il existe un échantillon caractéristique CS de taille $\mathcal{O}(q(k))$

Il est à noter qu'à cause du peu de résultats positifs dans le cadre de la définition 1.9, on parle fréquemment d'identification polynomiale à la limite concernant ce dernier paradigme.

³ Une hypothèse est *cohérente* avec un échantillon S si elle n'est en contradiction avec aucune des données de S . C'est l'application directe du principe inductif ERM.

Nous étudierons dans le chapitre suivant les résultats d'apprenabilité sur les langages hors-contextes dans le cadre de l'identification à la limite avec contraintes de polynomialité. C'est en effet dans le cadre de l'identification à la limite en temps et données polynomiaux que nous avons choisi de travailler. Nous avons conscience des limitations d'une telle approche : même dans le cadre de l'identification en temps et données polynomiaux, rien n'est dit sur le comportement de l'algorithme quand l'échantillon caractéristique n'est pas présent dans les données. D'un autre côté, c'est le seul paradigme d'apprentissage dans lequel des résultats positifs non-triviaux ont été démontrés concernant les langages hors-contextes. De plus, il peut être vu comme une condition nécessaire que doit satisfaire l'idée constituante d'un processus d'inférence grammaticale : à partir d'un algorithme capable d'identifier à la limite une classe de langages, on peut greffer des heuristiques permettant ainsi d'obtenir de bons résultats en pratique (où rien ne permet d'affirmer *a priori* que l'ensemble caractéristique figure dans les données). C'est ainsi que, à partir de l'algorithme RPNI [OG92], pour lequel un résultat d'identification en temps et données polynomiaux a été démontré, l'algorithme EDSM [LPP98] a été obtenu. Ce dernier algorithme est devenu la référence en ce qui concerne l'apprentissage des langages réguliers. L'identification à la limite peut ainsi être vue comme une preuve de "concept", un moyen de prouver la pertinence d'une idée vis-à-vis de l'apprentissage : si les données contiennent suffisamment d'information, alors cette idée permet d'apprendre.

1.3 Pourquoi apprendre les langages hors-contextes ?

Bien que son acte de naissance date des travaux de Gold dans les années 60, l'inférence grammaticale ne s'est, jusqu'à récemment, que très peu intéressée aux langages hors-contextes. Par exemple, il n'existe à notre connaissance qu'un seul article résumant l'avancée des recherches dans cette voie, publié sous la forme d'un rapport technique [Lee96] (un deuxième va bientôt être publié). Dans les années 80, l'école japonaise a été presque la seule à obtenir quelques résultats [Tak88, Ish89, Sak87, Yok89], dont les plus intéressants seront développés dans le prochain chapitre. L'attention des chercheurs de la communauté s'est principalement concentrée sur la classe des langages réguliers, et il a fallu attendre la fin des années 90 et le début du nouveau siècle pour que l'apprentissage des langages hors-contextes devienne d'actualité. C'est ainsi que, suite à un workshop à la conférence ECML en 2003 [dlHAVZO03], il a été décidé de créer la compétition OMPHALOS [SCvZ04].

Les résultats positifs obtenus sur la classe des langages réguliers ne sont pas

étrangers au renouveau de l'intérêt concernant les langages hors-contextes. En effet, le problème étant globalement résolu pour la classe de langages la plus simple de la hiérarchie de Chomsky, le nouveau challenge devient naturellement la classe de difficulté suivante de la hiérarchie.

D'autre part, l'utilisation des langages réguliers sur des problèmes pratiques a mis à jour leurs limitations. Par exemple, les structures du code génétique ne sont pas régulières, et une étude de ces structures nécessitera l'utilisation de relations hors-contextes [SBH⁺94, AM97, JLP01, SB02]. En ce qui concerne les langues naturelles, cela fait déjà quelques années que les langages réguliers ne sont plus d'actualité. Même les langages hors-contextes sont mis à mal par cette communauté [Shi85]. Toutefois, si certaines structures ne répondent pas aux critères hors-contextes, la grande majorité en font partie, et cela est d'autant plus vrai si on s'intéresse à la langue parlée : les structures sensibles aux contextes mises en avant pour réfuter la conjecture considérant les langues naturelles comme des langages hors-contextes appartiennent, à une exception près [Rad91], à la langue écrite.

D'autres champs d'applications nécessitent des langages et des structures hors-contextes. C'est le cas évidemment des langages de balises, comme par exemple XML et de ses technologies associées [Fer01, ASA01, Chi01]. Mais on retrouve des langages hors-contextes dans des domaines aussi variés que la compression de textes [NMW97] ou le traitement automatique de la parole [Bak79, Sto95].

2 *Sur la difficulté d'apprentissage des grammaires hors-contextes*

Dans le chapitre précédent, nous avons déjà mis en avant quelques résultats négatifs d'apprenabilité des langages hors-contextes. Malheureusement, ce ne sont pas les seuls et nous en présenterons d'autres dans ce chapitre. Il nous a donc semblé intéressant de réfléchir aux raisons rendant plus difficile l'apprentissage de cette classe, comparativement à la classe des langages réguliers.

2.1 D'où vient cette difficulté?

Cette difficulté est liée à plusieurs paramètres allant des structures algébriques inhérentes à ces langages aux propriétés structurelles des grammaires les représentant. Les réflexions qui suivent n'ont pas la prétention de constituer une liste exhaustive; elles constituent des explications permettant de mieux appréhender la problématique de l'apprentissage des langages hors-contextes.

2.1.1 Opérabilité

Lorsque l'on s'intéresse à la classe des langages réguliers, l'apprentissage peut s'appuyer sur des propriétés algébriques fortes. Ainsi, l'union, la concaténation et l'intersection de deux langages réguliers sont des langages réguliers. De la même manière, le complémentaire d'un langage de cette classe est un langage régulier (il suffit de prendre l'automate déterministe canonique complet du langage et d'inverser les états finaux et les états non-finaux pour obtenir une représentation du complémentaire du langage).

Concernant la classe des langages hors-contextes, la stabilité des opérations algébriques n'est pas toujours assurée. En effet, si l'union et la concaténation de deux langages hors-contextes correspondent à des langages de cette classe,

ce n'est pas le cas de leur intersection. En conséquence, le complémentaire d'un langage hors-contexte n'est pas forcément hors-contexte. Cette dernière propriété a des conséquences en apprentissage. Si, dans le cas des réguliers, l'utilisation d'exemples négatifs était pertinente et aisément définissable, car, ils possédaient une structuration du même type que celle que l'on cherchait à apprendre, ce n'est donc plus le cas quand on cherche à apprendre des langages hors-contextes. L'information contenue dans les exemples négatifs n'a donc pas la même portée.

Il devient même difficile de définir quelle pourrait être l'utilisation de ces contre-exemples. Intuitivement, les exemples négatifs qui semblent les plus utiles à l'inférence sont ceux de la frontière du langage à identifier, c'est-à-dire ceux qui sont très proches des mots du langage. Dans le cas d'un langage régulier, cette frontière est régulière : dans l'automate déterministe canonique complet représentant le langage, elle correspond aux mots atteignant les états non-finaux et non-puits. Si, par exemple, nous nous plaçons dans le paradigme d'identification à la limite en temps et données polynomiaux à partir d'exemples positifs et négatifs, la définition d'un échantillon caractéristique (voir la définition 1.7) est ainsi facilitée. Notre propos est que la difficulté est plus importante pour définir ces contre-exemples utiles pour les langages hors-contextes : les exemples négatifs d'un éventuel échantillon caractéristique doivent être définis en s'appuyant sur une structure qui peut très bien ne pas être hors-contexte.

Ce qui précède a pour conséquence naturelle d'essayer d'apprendre les langages hors-contextes à partir d'exemples positifs seulement. Or le résultat suivant nous en empêche [Gol67] :

Théorème 2.1 *Aucune classe superfinie de langages formels n'est identifiable à la limite à partir d'exemples positifs seulement.*

Une classe superfinie est constituée de tous les langages finis et d'au moins un langage infini. C'est trivialement le cas de la classe des langages hors-contextes (et des réguliers). Intuitivement, le théorème se comprend bien : l'ensemble des exemples vus par l'algorithme à un instant donné forme un langage fini. Il n'est alors pas possible de différencier ce langage d'un langage infini le contenant.

Ainsi, l'utilisation d'exemples négatifs, ou d'autres formes d'information complémentaire aux exemples positifs, est nécessaire à l'inférence des langages hors-contextes. Mais cette induction sera rendue plus difficile par la faiblesse des propriétés algébriques sous-jacentes à la classe.

2.1.2 Indécidabilité

Nous l'avons déjà explicité dans le chapitre précédent, mais nous préférons y revenir ici, car cela constitue une des principales difficultés de l'inférence grammaticale des langages hors-contextes : étant données deux grammaires algébriques,

on ne peut pas décider si elles engendrent le même langage. Comme nous nous plaçons dans le cadre de l'identification à la limite, cela a des conséquences sur l'apprentissage. Dans le cas de l'inférence de langages réguliers à l'aide d'automates, il existe un représentant canonique calculable pour chaque langage. On peut donc réduire cet apprentissage à l'identification de ce représentant (c'est ce que fait, par exemple, l'algorithme RPNI).

Cette approche n'est pas possible pour les langages hors-contextes. Il en résulte un flou, un degré d'incertitude plus élevé, dans l'inférence. Pour un langage hors-contexte donné, il existe un nombre potentiellement infini de solutions exactes, c'est-à-dire de grammaires l'engendrant, avec aucune comparaison possible de pertinence entre elles. Même si l'on réduit le nombre de solutions par des restrictions relevant du bon sens, par exemple en interdisant les productions de type $N \rightarrow M$, où N et M sont des non-terminaux, le nombre de grammaires représentant un langage hors-contexte reste infini. Par exemple, si l'on s'intéresse au langage $\{a^n b^n : n > 0\}$, il peut être représenté, entre autres, par chaque élément de la famille de grammaires $\mathcal{G} = \cup_{i \in \mathbb{N}} G_i$ avec pour tout $i \in \mathbb{N}$, $G_i = \langle \{a, b\}, \{N_j : 0 \leq j \leq i\}, P_i, N_0 \rangle$ où $P_i = \{N_j \rightarrow aN_{j+1}b : 0 \leq j < i\} \cup \{N_j \rightarrow ab : 0 \leq j \leq i\} \cup \{N_i \rightarrow aN_0b\}$. \mathcal{G} est une famille infinie de grammaires.

Chercher à identifier une grammaire particulière à partir de données d'apprentissage n'a pas de sens quand une infinité de solutions exactes existent. L'identification porte donc uniquement sur le langage et non sur sa représentation, ce qui introduit forcément une difficulté supplémentaire.

C'est à cause de l'indécidabilité de l'équivalence de deux grammaires hors-contextes que le résultat négatif suivant a été obtenu [dlH97]:

Théorème 2.2 *La classe des langages hors-contextes n'est pas identifiable à la limite en temps et données polynomiaux à partir d'échantillons formés d'exemples positifs et négatifs.*

La preuve de ce théorème repose sur le fait que si l'équivalence de deux éléments d'une classe de représentation est indécidable, alors il existe forcément deux représentations R_1 et R_2 dans la classe, de taille inférieure à n , telles que le plus petit mot appartenant à la différence symétrique $\mathcal{L}(R_1) \oplus \mathcal{L}(R_2)$ est de longueur exponentielle en n . Quel que soit l'algorithme utilisé, la taille de l'échantillon caractéristique permettant de différencier R_1 de R_2 est alors forcément exponentielle dans la taille de la cible.

Ce théorème annihile l'espoir d'apprendre l'intégralité de la classe à l'aide de représentations dont le problème de l'équivalence est indécidable. Toutefois il existe toujours des possibilités pour s'attaquer à l'apprentissage de langages hors-contextes, que ce soit, par exemple, en changeant de représentation ou en se restreignant à des sous-classes.

Si la décidabilité de l'équivalence de deux grammaires hors-contextes quelconques est impossible, il existe une sous-classe non-triviale pour laquelle un résultat positif a été obtenu par Géraud Sénizergues [Sén97] :

Propriété 2.1 *L'équivalence de deux automates à pile déterministes est décidable.*

Un automate à pile est *déterministe* si pour chaque combinaison d'états, de lettres et de sommets de pile, une unique action est définie. Nous avons déjà dit que la classe des automates à pile forme une classe de représentation des langages hors-contextes, au même titre que les grammaires algébriques. Il existe en outre une fonction injective allant des automates à pile dans la classe des grammaires hors-contextes. Il s'ensuit que l'on peut définir la classe des grammaires hors-contextes déterministes comme celle des grammaires correspondant aux langages pouvant être représentés par des automates à pile déterministes. La classe de langages correspondante est strictement incluse dans celle des langages hors-contextes. La propriété de fermeture suivante est à noter : le complémentaire d'un langage déterministe est un langage déterministe. Nous détaillerons dans la suite des travaux d'inférence se restreignant à cette sous-classe.

2.1.3 Ambiguïté et non-déterminisme

Nous avons vu que la classe des langages hors-contextes est définie, dans la hiérarchie de Chomsky, par une propriété syntaxique sur la forme des règles. Nous avons aussi vu que cette propriété n'est pas assez restrictive pour correspondre entièrement à la classe des automates à pile déterministes : certains langages sont intrinsèquement non-déterministes. Cela n'est pas le cas pour les langages réguliers, où les automates finis déterministes caractérisent l'ensemble de la classe. C'est d'ailleurs en se restreignant à cette classe de machines abstraites qu'il a été possible d'identifier à la limite, en temps et données polynomiaux et à partir d'exemples positifs et négatifs, les langages réguliers.

Ce problème de déterminisme (ou plutôt de non-déterminisme) se retrouve dans l'autre représentation usuelle des langages hors-contextes. En effet, certaines grammaires hors-contextes souffrent du même mal : on les appelle les grammaires *ambiguës*. Dans une telle grammaire, les mots peuvent être engendrés par des dérivations structurellement différentes, c'est-à-dire qu'à un mot correspond plusieurs arbres de dérivation. Comme pour les automates à pile, il existe des langages admettant des grammaires ambiguës et non-ambiguës mais aussi des *langages inhéremment ambiguës*, c'est-à-dire qui ne sont engendrés que par des grammaires de ce type. On parle *a contrario* de la classe des langages hors-contextes non-ambigus. Nous revenons sur ces sous-classes dans la section 2.2.2.

Un langage déterministe est non-ambigu mais l'inverse n'est pas vrai : par exemple le langage des palindromes $\{ww^R : w \in \{a,b\}^*\}$ est non ambigu mais pas déterministe. Cette hétérogénéité rend bien compte des difficultés de l'inférence de la classe entière : la classe des langages hors-contextes regroupe des langages aux caractéristiques très différentes. C'est aussi une raison expliquant le peu de comparaisons de résultats d'apprenabilité : lorsque l'on s'intéresse à une sous-classe définie par un type de représentations spécifique (par exemple, les langages représentables par des automates à pile déterministes) le résultat obtenu sera difficilement comparable avec ceux éventuellement obtenus à partir de grammaires non ambiguës.

Vis-à-vis de l'apprentissage, ces non-déterminismes posent évidemment problème : l'utilisation de représentations non-déterministes induit une perte de contrôle lors de l'inférence. En effet, si l'on construit une grammaire pour qu'elle engendre un mot d'une certaine façon, mais que finalement ce mot peut être obtenu différemment, il se peut, par exemple, que d'autres mots, ceux-ci non-désirés, soient engendrés par la grammaire. D'autre part, le déterminisme est souvent utilisé comme critère de décision permettant de guider l'apprentissage : si on relâche cette contrainte, le nombre d'hypothèses cohérentes avec un ensemble de données augmente considérablement.

2.1.4 Expansivité

Le Théorème 2.2 statue que la classe des grammaires hors-contextes n'est pas identifiable à la limite en temps et données polynomiaux, à partir d'exemples positifs et négatifs, en raison de l'indécidabilité de l'équivalence de deux grammaires hors-contextes. Mais ce n'est pas la seule raison qui peut rendre une sous-classe non apprenable dans ce paradigme. Un exemple classique est celui du problème de l'*expansivité*. Dans [dlH97], une sous-classe souffrant de ce problème est exhibée. Il s'agit de la classe des *grammaires simples déterministes* :

Définition 2.1 (Grammaire simple déterministe)

Une grammaire $G = \langle \Sigma, V, P, A \rangle$ est simple déterministe si toutes les dérivations $N \rightarrow \alpha$ sont de la forme $\alpha \in \Sigma \cup \Sigma V \cup \Sigma V^2$ et, pour toute lettre $a \in \Sigma$, si $N \rightarrow a\alpha$ et $N \rightarrow a\beta$, alors $\alpha = \beta$.

L'équivalence de deux grammaires simples déterministes est décidable, puisque ce sont des grammaires déterministes, mais la classe de langages correspondante n'est pas identifiable à la limite en temps et données polynomiaux à partir d'exemples positifs et négatifs. La preuve repose sur l'exemple suivant : $G_n = \langle \{a\}, \{N_i : i \leq n\}, P, N_0 \rangle$, où $P = \{N_i \rightarrow aN_{i+1}N_{i+1} : i < n\} \cup \{N_n \rightarrow a\}$. Cette grammaire simple déterministe engendre le langage réduit au mot a^k avec

$k = 2^{n+1} - 1$. Le nombre de pas de production nécessaires pour obtenir le mot à partir de l'axiome est exponentiel dans la taille de la grammaire.

Une conséquence sur l'apprentissage est que la taille de l'ensemble caractéristique permettant de distinguer $\mathcal{L}(G_n)$ et $\{\lambda\}$ est alors exponentielle dans la taille de la plus petite représentation du langage (puisqu'il doit contenir $a^k, k = 2^{n+1} - 1$). Ceci rend l'identification à la limite en temps et données polynomiaux impossible. Il est à noter que le même problème se pose pour le cadre d'apprentissage PAC, où l'échantillon doit aussi être polynomial dans la taille de la représentation du langage.

Cet exemple est troublant : apprendre un langage formé d'un seul mot à partir d'exemple(s) ne paraît pas compliqué. Le problème semble venir des paradigmes d'apprentissage. La contrainte de polynomialité a été introduite dans un but : si pour apprendre un langage (ou tout autre concept) il est nécessaire d'avoir un échantillon d'apprentissage déraisonnablement grand, alors ce langage n'est pas apprenable. Mais cette contrainte de polynomialité porte sur la taille de la représentation cible et c'est là que se pose le problème de l'expansivité. D'autant plus que cela est en contradiction avec un des principes inductifs présentés précédemment : celui de la compression de l'information, ou principe MDL. Si "apprendre" c'est être capable de représenter de manière synthétique les exemples d'apprentissage, où est le problème quand on infère une représentation exponentiellement plus petite que les données ? Nous n'avons pas de solutions à proposer, mais les questions soulevées par ce problème dans les paradigmes d'apprentissage doivent être prises en compte dans les résultats d'apprenabilité.

2.1.5 Générativité

Même s'il est possible de tester polynomialement l'appartenance d'un mot au langage généré par une grammaire algébrique, cette classe de représentation est avant tout générative. Ainsi, pour obtenir un mot à partir de l'axiome, il est nécessaire d'effectuer plusieurs pas de dérivation et, à chaque étape, nous ne possédons qu'une forme sententielle, peu informative sur les mots du langage. Dans le cas des automates finis déterministes, tester si un mot appartient au langage se fait en un temps linéaire dans la taille du mot (alors que pour une grammaire hors-contexte quelconque la complexité est cubique) et, à chaque étape, on peut se retrouver dans un état final, ce qui nous permet de connaître quel préfixe du mot fait partie du langage. Ainsi un processus d'apprentissage de langages réguliers peut se servir des relations préfixales (et même suffixales) comme indicateur pertinent de la structure du langage. Or ce n'est pas possible pour un langage hors-contexte : deux mots tels que l'un est préfixe de l'autre peuvent très bien n'avoir aucun pas de dérivation en commun.

En ce qui concerne l'apprentissage, si nous essayons d'inférer une grammaire

hors-contexte par un processus ascendant, c'est-à-dire retrouver les règles à partir d'un ensemble de mots du langage, après chaque inférence d'une règle, ce que nous possédons n'est pas une grammaire tant que l'axiome n'a pas été trouvé. Par exemple, si le mot correspondant à la dérivation la plus courte a besoin de dix pas de dérivation, il nous faudra inférer neuf règles, sans être capable de tester efficacement leur pertinence, avant de posséder une hypothèse correspondant à une grammaire. Ainsi, il est difficile, dans une telle approche, de tester la pertinence d'une règle lors de son inférence.

L'autre façon de s'attaquer à l'apprentissage est d'adopter une attitude descendante, consistant à partir de la grammaire n'engendrant que les exemples de mots du langage contenus dans les données et à la généraliser. Mais le processus de généralisation, revenant à créer de nouvelles règles ou à regrouper des non-terminaux se heurte au même problème : il est possible que la règle inférée ne permette pas à la grammaire d'engendrer un langage plus grand que le précédent (ou de se rapprocher de la cible). On est donc dans l'obligation soit d'inférer des règles "à l'aveugle", sans pouvoir être guidé directement par les données, soit de s'assurer d'une croissance monotone du langage en inférant ou en modifiant plusieurs règles en même temps, ce qui est forcément plus difficile.

Lorsque l'on travaille avec des automates finis déterministes, les processus d'inférence commencent avec l'automate reconnaissant uniquement les exemples positifs, puis ils fusionnent des états, généralisant ainsi le langage reconnu par l'automate (propriété assurée par le théorème de Myhill-Nerode[A.N58]). Il est possible ensuite, si l'on dispose d'exemples négatifs et après une phase de détermination, de tester si le nouvel automate est cohérent avec les données. Comme toute fusion d'état généralise le langage reconnu par l'automate, il est possible de tester individuellement la pertinence de chaque étape.

2.1.6 Indivisibilité

Dans une grammaire, le langage représenté dépend fortement de l'ensemble des règles et des non-terminaux de la grammaire. Si un processus d'apprentissage n'est capable d'inférer qu'une partie de la cible qu'il cherche à apprendre, le langage correspondant aura très peu de liens avec le langage cible. Imaginons un mot qui nécessite plusieurs pas de dérivation à partir de l'axiome pour être généré. Il suffit qu'une des règles utilisées dans sa dérivation soit absente pour que ce mot ne fasse plus partie du langage. Ainsi, une grammaire forme un tout que l'on ne peut diviser.

Du point de vue de l'apprentissage, cela induit que tant que l'intégralité des règles n'a pas été découverte, ce dont nous disposons peut générer un langage extrêmement différent du langage cible. Par exemple, la grammaire $G = \langle \{a,b\}, \{S\}, \{S \rightarrow aSa, S \rightarrow bSb, S \rightarrow \lambda\}, S \rangle$ engendre le langage des palindromes

de longueur paire. Si on enlève la règle $S \rightarrow \lambda$ le langage engendré est vide, alors que si c'est la règle $S \rightarrow aSa$ (*resp.* $S \rightarrow bSb$) qui est absente, la grammaire représente alors le langage régulier $(bb)^*$ (*resp.* le langage $(aa)^*$).

Les processus d'apprentissage suivent, dans leur grande majorité, un processus identique : chaque étape généralise le langage représenté par l'hypothèse courante, puis, après une éventuelle phase de déterminisation, teste la cohérence et la pertinence de ces modifications. Il est difficile d'adapter cette approche aux grammaires : ce n'est pas parce qu'on ajoute une ou plusieurs règles que le nouveau langage généralise le précédent. En fait, il n'a pas été mis à jour de d'opérateur généralisant garantissant une croissance monotone du langage d'une grammaire. Et le fait qu'une grammaire forme un tout tel que l'absence d'un unique élément change complètement le langage représenté tend à montrer qu'un tel opérateur peut très bien ne pas exister.

2.1.7 Structuralité

Un point est souvent mis en avant lorsqu'il s'agit de comparer les résultats d'apprentissage obtenus en inférence grammaticale avec ceux des autres domaines de l'apprentissage automatique. Il consiste à dire qu'un automate fini, ou une grammaire, apportent plus d'informations sur le langage que, par exemple, un classifieur. Nous avons vu par exemple que les automates peuvent être exprimés graphiquement. De la même manière, une grammaire n'est pas seulement un *parser* répondant à la question "ce mot appartient-il au langage?", mais elle apporte souvent une véritable explication sur la structure du langage (ce qui est sans doute moins vrai quand on utilise une des formes normales précédemment introduites). Quand on cherche à apprendre des langages hors-contextes à l'aide de grammaires, il nous faut donc apprendre le langage en lui-même mais aussi sa structuration.

Cette double difficulté se paie forcément lors de l'inférence : l'espace des hypothèses, déjà important quand on considère les langages, l'est encore plus si l'on rajoute le problème de la structure. C'est encore plus vrai lorsqu'au lieu de s'intéresser aux grammaires classiques, notre attention se porte sur des grammaires d'arbres ou de graphes, où la structuration joue un rôle prédominant. En ce qui concerne les langages réguliers, la difficulté a encore une fois été détournée en réduisant l'espace des hypothèses aux représentants canonique de chaque langage (ce qui revient à dire que chaque langage correspond à une unique structuration). Nous avons déjà longuement expliqué que l'inexistence d'une représentation canonique calculable pour chaque langage hors-contexte empêchait une telle restriction de l'espace des hypothèses. Il est toutefois possible de se restreindre à une forme normale, mais on perd alors l'explication structurelle du langage. . .

Nous verrons par la suite que si l'information structurelle est contenue dans les

données alors il devient possible d'identifier l'ensemble de la classe des langages hors-contextes (voir Section 2.2.1). Mais si cette information n'est pas contenue dans les données, le choix d'une représentation ne rendant pas compte de la structuration du langage, comme par exemple les systèmes de réécriture, peut permettre d'obtenir des résultats d'apprentissage jouissant d'une plus grande portée que ceux obtenus avec des grammaires.

2.2 Quelles solutions?

Tous les travaux en inférence grammaticale cherchant à apprendre des langages hors-contextes utilisent des approches contournant les problèmes développés précédemment, même si bien souvent ces difficultés et les idées pour les gérer ne sont pas explicitées. En fait, c'est même une obligation : il n'est pas envisageable d'apprendre des langages algébriques sans tenir compte des limitations d'apprentissage inhérentes à cette classe de langages.

2.2.1 Données structurées *a priori*

Nous avons vu qu'il n'est pas possible d'identifier à la limite en temps et données polynomiaux l'ensemble de la classe des langages hors-contextes, que ce soit à partir d'exemples positifs ou que ce soit à partir d'exemples positifs et négatifs. Cependant, si l'on enrichit les données d'apprentissage avec des informations supplémentaires, il est possible d'apprendre l'ensemble de la classe. Ainsi, il est possible de monter dans la hiérarchie de Chomsky en utilisant des structures plus complexes que de simples mots. Une des idées principales dans cette approche est d'utiliser des données contenant l'information structurelle sur le langage. Cela peut se faire en utilisant, en guise d'exemples, des squelettes.

Définition 2.2 (Squelette)

Un squelette d'un mot m vis-à-vis d'une grammaire G est un arbre de dérivation de m dans G dont tous les noeuds internes portent la même étiquette.

Une présentation sous forme de squelettes d'un langage L pour une grammaire le représentant G est une séquence infinie des squelettes de mots de L vis-à-vis de G telle que toute les structurations des mots du langage dans G apparaissent au moins une fois. Le résultat de Gold sur les classes superfinies s'appliquent aux présentations d'éléments ainsi structurés.

Pour contourner ce résultat négatif, Sakakibara introduit un nouveau type de grammaire hors-contexte [Sak92] :

Définition 2.3 (Grammaire réversible)

Une grammaire hors-contexte $G = \langle \Sigma, V, P, A \rangle$ est réversible si :

- $A \rightarrow \alpha \in P$ et $B \rightarrow \alpha \in P$ implique $A = B$,
- $A \rightarrow \alpha B \beta \in P$ et $A \rightarrow \alpha C \beta \in P$ implique $B = C$.

Sakakibara montre que les grammaires réversibles constituent une forme normale des langages hors-contextes. Ainsi, toute grammaire hors-contexte peut être transformée en grammaire réversible, mais la taille de la grammaire réversible peut-être exponentiellement plus grande que celle de la grammaire initiale.

On dit d'un squelette qu'il est réversible s'il a été obtenu à partir de l'arbre de dérivation d'un mot dans une grammaire réversible. L'algorithme RT (qui sera détaillé dans le chapitre 3) identifie à la limite en temps et données polynomiaux la classe des langages hors-contextes à partir d'exemples sous forme de squelettes réversibles. Le principe de l'algorithme est proche de celui d'Angluin pour les langages (réguliers) 0-réversibles [Ang82]. Il consiste à construire l'automate d'arbres reconnaissant uniquement les squelettes du langage, puis à fusionner les noeuds ne respectant pas les contraintes de réversibilité. Une grammaire hors-contexte est ensuite construite à partir de l'automate.

Ce résultat peut sembler contradictoire : il n'est pas possible d'identifier à la limite la classe des langages hors-contextes à partir de squelettes, mais l'algorithme RT identifie cette classe à la limite en temps et données polynomiaux à partir de squelettes réversibles. En fait, dans le premier cas nous cherchons à identifier une des grammaires qui correspondent à un ensemble de squelettes, alors que dans le second cas il nous faut des squelettes qui se conforment à une grammaire cible. Ce dernier point réduit drastiquement l'espace des hypothèses.

Plusieurs sous-classes des langages hors-contextes sont apprenables à partir de squelettes de mots du langage [BM04, Fer02, Yok91, Ish89]. L'idée est de transposer les résultats obtenus sur les réguliers aux hors-contextes en passant par les langages d'arbres. Un exemple caractéristique de cette approche est celui des langages d'arbres k -testables [KS94].

D'autres types de données d'apprentissage ont été étudiés suite à ces résultats. La classe est identifiable à partir de squelettes positifs et négatifs, de squelettes positifs et d'exemples négatifs [GO93]. Mais les contraintes sur la forme des squelettes, ainsi que l'hypothèse de leur existence dans des données réduisent la portée de ces résultats : il est difficile d'imaginer un problème du monde réel où les exemples auraient une structuration réversible; quant aux squelettes négatifs, il est difficile d'imaginer des données en comportant (rappelons que le complémentaire d'un langage hors-contexte n'est pas forcément hors-contexte).

2.2.2 Restrictions à des sous-classes

Comme l'ensemble de la classe n'est pas identifiable à la limite, de nombreux travaux ont consisté à se restreindre à des sous-classes ne souffrant pas de certaines des difficultés précédemment répertoriées.

Linéarité

Le problème de l'expansivité ayant été identifié, l'attention des chercheurs de la communauté s'est concentrée sur une sous-classe de langages hors-contextes ne souffrant pas de ce problème :

Définition 2.4 (Langage linéaire)

Une grammaire hors-contexte $G = \langle \Sigma, V, P, A \rangle$ est linéaire si toutes ses productions sont de la forme $N \rightarrow w_1 M w_2$, ou $N \rightarrow w_1$, avec $w_1, w_2 \in \Sigma^*$ et $N, M \in V$. Un langage hors-contexte est linéaire s'il peut être représenté par une grammaire hors-contexte linéaire.

Il est à noter que cette classe n'est pas trop restrictive : tous les langages réguliers, et la plupart des langages hors-contextes déjà présentés dans ce document sont des langages linéaires. Par exemple, le langage des palindromes est linéaire car il peut être représenté par la grammaire linéaire $\langle \{a, b\}, \{S\}, \{S \rightarrow aSa, S \rightarrow bSb, S \rightarrow a, S \rightarrow b, S \rightarrow \lambda\}, S \rangle$. Par contre, le langage de Dyck n'est pas linéaire.

L'équivalence de deux grammaires hors-contextes linéaires n'étant pas décidable, cette classe n'est pas identifiable à la limite en temps et données polynomiaux à partir d'exemples positifs et négatifs [dlH97]. Toutefois, les propriétés de ces langages sont suffisamment intéressantes pour que des sous-classes aient été étudiées, certains auteurs considérant même que la linéarité est une condition nécessaire à l'apprentissage [dlHO02].

La principale est celle des *langages linéaires équilibrés* où les productions sont de la forme $N \rightarrow a_1 M a_2$, ou $N \rightarrow a_1$, ou $N \rightarrow \lambda$, avec $a_1, a_2 \in \Sigma$ et $N, M \in V$. Cette classe contient tous les langages réguliers, et des langages hors-contextes comme $\{a^n b^n\}$ ou le langage des palindromes. Différents algorithmes ont été étudiés pour apprendre ces langages [Tak88, Tak94, SG94, Mäk96, KMT97]. Ils reposent tous sur la même approche : elle consiste à se ramener à l'apprentissage de langages réguliers en transformant une règle $N \rightarrow a_1 M a_2$ en $N \rightarrow [a_1 a_2] M$. Cette transformation porte le nom de *réduction régulière*. Un mot $a_1 a_2 \dots a_n$ appartient au langage engendré par une grammaire linéaire équilibrée si et seulement si $\langle a_1 a_n \rangle \langle a_2 a_{n-1} \rangle \dots \langle a_{n/2} a_{n/2+1} \rangle$ appartient à sa réduction régulière. Une fois les données d'apprentissage transformées, toutes les méthodes permettant d'inférer des langages réguliers peuvent être utilisées. La classe des langages linéaires équilibrés est donc identifiable à la limite en temps et données polynomiaux à partir

d'exemples positifs et négatifs. De la même manière, plusieurs sous-classes de ces langages sont apprenables à partir d'exemples positifs seulement.

Déterminisme

Comme nous l'avons précédemment vu, décider si deux grammaires hors-contextes quelconques engendrent le même langage n'est pas possible. Mais il existe une sous-classe, les langages hors-contextes déterministes, où le problème devient décidable. C'est pourquoi plusieurs travaux se sont concentrés sur cette classe, aidés en cela par le fait que le complémentaire d'un langage déterministe est un langage déterministe (et que par conséquent l'utilisation d'exemples négatifs se justifie plus facilement que dans le cas général).

Mais les langages déterministes, quand ils sont représentés par des grammaires, souffrent du problème de l'expansivité, ce qui nous empêche d'espérer un résultat d'identification en temps et en données polynomiaux. C'est pourquoi une sous-classe, permettant de ne pas pâtir du problème de l'équivalence ni de celui de l'expansivité, a été étudiée : il s'agit des langages *linéaires déterministes*. Une grammaire $\langle \Sigma, V, P, A \rangle$ est linéaire déterministe si ses productions sont de la forme $N \rightarrow \lambda$ ou $N \rightarrow aMw_1$, avec $a \in \Sigma$, $N, M \in V$ et $w_1 \in \Sigma^*$, et si $N \rightarrow a\alpha \in P$ et $N \rightarrow a\beta \in P$ alors $\alpha = \beta$. Une forme canonique calculable peut être définie pour les grammaires linéaires déterministes, ce qui permet un algorithme incrémental par niveau, identifiant à la limite la classe des langages linéaires déterministes en temps et en données polynomiaux, à partir d'exemples positifs et négatifs [dlHO02].

D'autres sous-classes des langages hors-contextes déterministes ont été étudiées, comme par exemple les langages simples déterministes [Ish95]. Le résultat le plus intéressant concerne la classe des *langages très simples déterministes*. Cette classe est caractérisée par les *grammaires très simples déterministes* $\langle \Sigma, V, P, A \rangle$ dont les productions sont de la forme $N \rightarrow a\alpha$, où $a \in \Sigma$ et $\alpha \in V^*$, et pour toute lettre a de l'alphabet Σ , il existe un unique non terminal N et une unique règle $N \rightarrow a\alpha$. Cela correspond aux grammaires hors-contextes sous forme normale de Greibach, avec la contrainte que chaque lettre de l'alphabet ne doit apparaître que dans une seule règle de production. Ces langages peuvent être considérés comme des super-déterministes à cause des propriétés suivantes : pour tout $\alpha, \beta \in V^+$, pour tout $w \in \Sigma^+$,

- Si $A \xRightarrow{*} w\alpha$ et $A \xRightarrow{*} w\beta$ alors $\alpha = \beta$ (déterminisme descendant);
- Si $\alpha \xRightarrow{*} w$ et $\beta \xRightarrow{*} w$ alors $\alpha = \beta$ (déterminisme ascendant).

Dans [Yok03], il est démontré que la classe des langages très simples déterministes est identifiable polynomialement à la limite à partir d'exemples positifs seulement. Malheureusement, le temps d'exécution de l'algorithme, bien que polynomial dans la taille des exemples, est exponentiel dans la taille de l'alphabet. Or, de par leur

définition, la taille des grammaires très simples est liée à la taille de leur alphabet. En dépit de cela, c'est à notre connaissance la plus importante classe de langages hors-contextes apprenable à partir d'exemples positifs seulement.

2.2.3 Changement de représentation

Nous avons vu que, malgré les résultats négatifs forts sur l'ensemble de la classe des langages hors-contextes, il demeure possible d'obtenir des résultats positifs d'apprentissage en se restreignant à des sous-classes. Ces sous-classes sont définies par des grammaires sur lesquelles des contraintes, principalement syntaxiques, ont été ajoutées.

La plupart des difficultés mises en évidence précédemment proviennent de la classe de représentation choisie pour les langages hors-contextes : les grammaires hors-contextes. Une solution pour s'attaquer à l'apprentissage des langages hors-contextes est donc de choisir un autre formalisme de représentation des langages qui souffrira certainement d'autres problèmes, mais pourrait nous permettre d'apprendre des langages pour qui, jusqu'à présent, aucun résultat d'apprenabilité n'était possible.

Si l'on considère, à juste titre, que les grammaires d'arbres et les automates d'arbres sont trop proches des grammaires usuelles pour s'affranchir de leurs limitations, très peu de travaux ont cherché à changer de représentation. Une exception notable est celle des *grammaires pures*. Dans ce formalisme, aucune distinction n'est faite entre l'ensemble des non-terminaux et l'ensemble des lettres, ce qui revient à confondre les formes sententielles et les mots. Ainsi on réécrit un bloc de symboles par un autre, à partir d'un axiome. Cela reste, par conséquent, une classe de représentations génératrices.

Par exemple, le langage $\{a^n b^n : n > 0\}$ peut être représenté par la grammaire pure $G = \langle \{a, b\}, \{ab \rightarrow aabb\}, ab \rangle$ où $ab \rightarrow aabb$ est l'unique production et ab l'axiome.

L'apprentissage de ces grammaires est toujours un problème ouvert [KMT00]. Mais les propriétés de ce formalisme sont intéressantes et permettent de contourner certains problèmes mis en avant pour les grammaires classiques. Ainsi, pour obtenir un mot du langage, il suffit d'appliquer une seule règle sur un autre mot du langage. Dans le cas d'une grammaire algébrique usuelle, l'obtention d'un mot nécessite une dérivation de l'axiome jusqu'au mot et, à chaque étape, on a une forme sententielle qui n'est d'aucune utilité vis-à-vis du langage. Cette caractéristique des grammaires pures permet l'inférence de règles à partir de peu de données (deux mots du langage qui diffèrent d'un unique facteur forment un bon indice pour l'inférence d'une règle).

Dans la deuxième partie de ce document, nous présenterons des travaux basés sur un changement de représentation des langages hors-contextes. Le formalisme

utilisé, celui des systèmes de réécriture de mots, est très proche de celui des grammaires pures. La différence principale est que les règles forment un système reconnaisseur et non un générateur, ce qui permet de s'affranchir des problèmes liés à la générativité des grammaires présentés précédemment.

2.3 Conclusion

Dans ce chapitre, nous avons répertorié plusieurs raisons expliquant pourquoi il est difficile d'apprendre des langages hors-contextes. Nous avons aussi vu comment ces difficultés étaient gérées, ou contournées, par les meilleurs algorithmes de l'état de l'art. Cette tentative d'inventaire des problèmes et des solutions est, à notre connaissance une première. Elle nous offre une grille de lecture indispensable des algorithmes d'inférence de langages algébriques et permet ainsi de mieux saisir les idées et les approches ayant obtenu des résultats positifs d'apprentissage.

Nous avons vu comment l'ambiguïté, le déterminisme et la linéarité semblent jouer un rôle primordial dans l'inférence. Il est à noter que la réversibilité de Sakakibara correspond aussi à des contraintes de déterminisme. En outre, la plupart des difficultés mises à jour dans ce qui précède sont liées à l'utilisation de grammaires comme représentations des langages algébriques : changer de formalisme pour représenter ces langages semble être une solution pertinente pouvant permettre l'inférence de langages pour lesquels aucun algorithme n'existe.

La partie suivante regroupe trois contributions à l'inférence grammaticale de langages hors-contextes que nous avons développées lors de nos trois années de thèse. La première est une tentative, cependant infructueuse, de structuration *a priori* des données, à l'aide d'un algorithme de compression de textes. La seconde propose l'utilisation d'une autre représentation des langages : les systèmes de réécriture de mots. Un algorithme est étudié et un résultat d'identification à la limite obtenu. Le troisième et dernier chapitre de cette partie présente un travail consistant à se restreindre à une sous-classe des langages algébriques. Mais, contrairement à ce qui est usuellement fait dans un tel cas en inférence grammaticale, cette sous-classe est définie par une contrainte portant directement sur les langages et non sur une de leurs représentations. Un résultat d'identification et une application à un problème des langues naturelles sont explicités.

Troisième partie

Nos contributions

3 *Structuration a priori des données*

3.1 Motivation

Nous avons vu, dans la partie précédente, qu'il existe un algorithme, nommé RT [Sak92], capable d'identifier à la limite en temps et données polynomiaux l'ensemble de la classe des langages hors-contextes. L'inconvénient de cet algorithme est de nécessiter des données structurées en entrée. En effet, les exemples d'apprentissage doivent lui être fournis sous la forme de squelettes, c'est-à-dire d'arbres ordonnés dont les noeuds internes portent tous la même étiquette, et dont les feuilles composent le mot représenté. En fait, cette structure d'arbre est équivalente à une structuration réalisée à l'aide de parenthèses : puisque les noeuds internes d'un squelette sont identiquement nommés, ce qui importe est comment sont structurellement regroupées les lettres du mot. En outre, n'importe quel type de squelettes n'est pas suffisant pour permettre l'induction : ils doivent correspondre à une structuration *réversible*. Malheureusement il est très rare de disposer de données d'apprentissage structurées, et encore plus rare que cette structuration réponde aux contraintes souhaitées par l'algorithme de Sakakibara.

L'idée ayant donné naissance au travail présenté dans ce chapitre est très naturelle : puisque l'on peut apprendre l'ensemble de la classe à partir d'exemples suivant une certaine structuration, mais qu'elle ne peut être raisonnablement présente dans les données, pourquoi ne pas chercher à structurer *a priori* les données ?

Pour réaliser cela, nous allons nous servir d'un algorithme de compression de données, appelé SEQUITUR [NMW97], qui, à partir d'un texte long, comme un roman par exemple, construit puis code une grammaire hors-contexte générant exactement le texte initial. En comprimant la grammaire ainsi obtenu avec les outils usuels de compression de texte, il est possible d'obtenir des taux meilleurs que ceux de l'état de l'art. En ce qui concerne notre problème d'apprentissage, nous verrons que la grammaire que retourne SEQUITUR est une grammaire réversible. Il est important de préciser que si SEQUITUR utilise des grammaires, ce n'est pas

un algorithme d'apprentissage dans le sens que nous l'entendons : il ne cherche pas à apprendre un langage à partir d'exemples de mots, mais à structurer un mot donné.

Dans la première section de ce chapitre nous détaillerons l'algorithme SEQUITUR et les propriétés des grammaires qu'il construit. Puis, dans la section suivante, nous verrons comment adapter l'algorithme pour qu'il puisse être utilisé non pas sur un seul exemple, mais sur un ensemble de mots d'un langage, afin de répondre à notre problématique. Nos expérimentations et les discussions qui s'en suivent feront l'objet d'une troisième section.

3.2 L'algorithme SEQUITUR

Craig Nevill-Manning et Ian Witten ont introduit en 1996 l'algorithme SEQUITUR [NMWO96]. Le but était d'utiliser une approche grammaticale pour compresser du texte. Le principe est très simple : chaque fois qu'un motif de deux lettres successives apparaît plusieurs fois, SEQUITUR remplace ses occurrences par un nouveau non-terminal dont l'unique production est le motif en question. Par récursion, ce motif peut contenir des non-terminaux précédemment générés. Si la création d'un nouveau non-terminal implique qu'un ancien non-terminal n'est plus utilisé qu'une fois (ce qui arrive, par exemple, lorsque deux motifs se chevauchent), alors ce dernier est détruit et remplacé par le motif qu'il engendrait. Une fois l'ensemble du texte ainsi transformé, la grammaire est codée sous forme de texte, puis compressée par un algorithme usuel de compression (de type *gzip*). Le fichier résultant est plus petit que ce qu'est capable de faire les méthodes classiques de compression de texte. Dans un deuxième article [NMW97], plus complet, l'influence de la taille du motif et la complexité de l'algorithme sont étudiées. En sus de l'apport de ce travail dans le domaine de la compression de données, c'est aussi un formidable éclairage de l'intérêt que peut avoir l'inférence grammaticale pour d'autres domaines.

3.2.1 Fonctionnement

Un pseudo-code simplifié de SEQUITUR est présenté par l'algorithme 3.1.

L'algorithme est incrémental : il lit le mot de gauche à droite, concaténant chaque nouvelle lettre à l'unique règle de production de l'axiome S . Il vérifie ensuite si le motif formé des deux dernières lettres lues a été rencontré dans ce qu'il connaît déjà du mot : l'ensemble H du pseudo-code 3.1 sert à stocker l'ensemble des motifs déjà rencontrés (en pratique, l'utilisation d'une table de hachage permet d'optimiser ce stockage). Il peut se produire trois cas différents :

- Le motif apparaît déjà dans H et il a déjà été vu plus d'une fois. Ceci

Algorithme 3.1 : SEQUITUR (taille des motifs égale à 2)**Données** : Un texte w (*i.e.* un long mot)**Résultat** : une grammaire hors-contexte $G = \langle \Sigma, V, P, S \rangle$ telle que $\mathcal{L}(G) = \{w\}$ **début**

```

lettre_courante ← prochaine_lettre(w);
V ← {S}; P = {S → lettre_courante}; Σ ← {lettre_courante};
H ← ∅;
tant que La lecture de w n'est pas finie faire
  dernière_lettre ← lettre_courante ;
  lettre_courante ← prochaine_lettre(w);
  Σ ← Σ ∪ {lettre_courante};
  concaténer_lettre(production(S), lettre_courante);
  si existe_motif(H, dernière_lettre lettre_courante) alors
    si existe_une_règle(N → dernière_lettre lettre_courante, P)
      alors
        P ← remplacer_motif(P, dernière_lettre lettre_courante, N);
      sinon
        N ← nouveau_non_terminal();
        V ← V ∪ {N};
        P ← remplacer_motif(P, dernière_lettre lettre_courante, N);
        P ← P ∪ {N → dernière_lettre lettre_courante};
        nettoyer(P)
    sinon
      ajouter(H, dernière_lettre lettre_courante)
retourner ⟨Σ, V, P, S⟩

```

fin

implique qu'un non-terminal le générant existe déjà dans la grammaire. Dans ce cas, il suffit de remplacer cette dernière occurrence du motif par le non-terminal N en question. À ce moment, un nouveau motif doit être récursivement vérifié: celui formé de l'antépénultième lettre lue (ou le non-terminal l'ayant remplacé) et de N . Pour des raisons de clarté, nous n'avons pas surchargé le pseudo-code avec cette étape.

- Le motif existe déjà dans H mais il n'a été rencontré qu'une seule fois. Il nous faut alors créer un nouveau non-terminal le générant, et remplacer ses occurrences, dans la règle de l'axiome, par ce non-terminal.
- C'est la première fois que ce motif est rencontré, il nous suffit alors de le stocker dans H .

Lorsqu'une nouvelle règle est créée, il est possible qu'elle rende caduque une règle précédemment inférée : une production précédemment inférée, bien qu'utilisée plusieurs fois, peut désormais n'apparaître que dans une seule partie droite de règle. À ce moment, plus rien ne justifie son existence (vis-à-vis du but de compression). Cette règle doit être enlevée de la grammaire, ce qui est le rôle de la fonction `nettoyer()`.

3.2.2 Exemples

Pour clarifier le fonctionnement de l'algorithme, nous détaillons une exécution sur un exemple, ce qui nous permet aussi de donner quelques détails passés sous silence jusqu'à présent.

Supposons que SEQUITUR reçoive en entrée le mot *aabaab*.

Après avoir lu les 4 premières lettres sans avoir trouver de motif se répétant, l'unique règle de la grammaire est $S \rightarrow aaba$.

La lettre suivante étant *a*, le motif *aa* possède deux occurrences : le non-terminal *A* est créé, ainsi que la règle $A \rightarrow aa$. Après remplacement, la règle initiale devient $S \rightarrow AbA$.

La lettre suivante est *b* ce qui donne la règle $S \rightarrow AbAb$. Le motif *Ab* apparaissant deux fois, il est remplacé par un non-terminal *N* dont l'unique règle est $N \rightarrow Ab$. La règle de l'axiome devient $S \rightarrow NN$.

Le non-terminal *A* n'apparaissant qu'une seule fois, la phase de nettoyage le détruit : la grammaire finale est $G = \langle \{a,b\}, \{S,N\}, \{S \rightarrow NN, N \rightarrow aab\}, s \rangle$. Il est facile de vérifier que $\mathcal{L}(G) = \{aabaab\}$.

L'exemple suivant est disponible sur la page internet de l'algorithme SEQUITUR : <http://sequitur.info/example.html>.

Le texte à compresser est composé de deux phrases en anglais (voir ci-dessous) séparées par un saut de ligne. Le caractère répétitif des mots permet de comprendre le type de structurations obtenues sans être obligé d'utiliser un long texte :

*pease porridge hot, pease porridge cold, pease porridge in the pot, nine days old.
some like it hot, some like it cold, some like it in the pot, nine days old.*

La grammaire retournée par l'algorithme à partir de ces deux phrases est $G = \langle \{a,c,d,e,g,h,i,k,l,m,n,o,p,r,s,t,y\}, V, P, S \rangle$ avec $V = \{S, N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, N_9, N_{10}, N_{11}, N_{12}\}$ et $P = \{$

$S \rightarrow N_1 N_2 N_3 N_4 N_3 N_5$ R $N_6 N_2 N_7 N_4 N_7 N_5$

$N_1 \rightarrow pease N_8 rridg N_9$

$N_2 \rightarrow hot$

$N_3 \rightarrow N_{10} N_{11}$

$N_4 \rightarrow c N_{11}$

$$N_5 \rightarrow N_{12} \text{-} th N_8 t N_{10} n N_{12} N_9 \text{days-} N_{11}.$$

$$N_6 \rightarrow som N_9 lik N_9 it \text{-}$$

$$N_7 \rightarrow N_{10} N_6$$

$$N_8 \rightarrow N_9 po$$

$$N_9 \rightarrow e \text{-}$$

$$N_{10} \rightarrow , \text{-}$$

$$N_{11} \rightarrow old$$

$$N_{12} \rightarrow in \}$$

R note le retours chariot, - un espace. Les mots anglais ou groupes de mots apparaissant au moins deux fois ont presque tous leur propre non-terminal: N_1 dérive “pease porridge”, N_2 “hot”, N_4 “cold”...

3.2.3 Propriétés

Si l'on considère que stocker et récupérer l'information dans une table de hachage se réalise en temps constant, alors l'algorithme a une complexité temporelle linéaire dans la taille du mot. C'est ce point qui a le plus fait débat dans la communauté de la compression de données [LS02]: bien que cela soit couramment utilisé, ce ne peut être valable que si on connaît *a priori* la taille maximale de la table. Or cette borne maximale est quadratique... Quoiqu'il en soit, il est toujours possible d'utiliser un arbre équilibré, ce qui permet d'assurer une complexité globale quadratique pour SEQUITUR. Si cette borne semble trop importante pour faire de SEQUITUR un algorithme de compression de données universellement reconnu, elle est acceptable en inférence grammaticale.

Nous allons maintenant expliquer pourquoi la grammaire construite par l'algorithme est réversible. Pour cela nous commençons par rappeler la définition suivante :

Définition 3.1 (Grammaire réversible)

Une grammaire hors-contexte $G = \langle \Sigma, V, P, A \rangle$ est réversible si :

1. $A \rightarrow \alpha \in P$ et $B \rightarrow \alpha \in P$ implique $A = B$,
2. $A \rightarrow \alpha B \beta \in P$ et $A \rightarrow \alpha C \beta \in P$ implique $B = C$.

Nous obtenons immédiatement le résultat suivant :

Propriété 3.1 *Les grammaires en sortie de SEQUITUR sont réversibles.*

Preuve : Le premier point de la définition 3.1 est forcément assuré par les grammaires en sortie de Sequitur, par construction, puisqu'une règle est créée si et seulement si aucune autre règle ayant la même partie droite existe. Quant au deuxième point, il est aussi trivial que les grammaires en sortie de Sequitur le vérifient : chaque non terminal apparait en partie gauche d'une unique règle. \square

3.3 L'algorithme RT

3.3.1 Fonctionnement

L'algorithme RT a été proposé au début des années 90 par Sakakibara [Sak92]. À partir de squelettes de mots du langage il construit une grammaire, sous forme normale réversible, engendrant un langage contenant tous les exemples d'apprentissage et, pour chaque mot, conservant la structuration décrite par le squelette. Nous commençons par rappeler la définition de squelette d'un mot.

Définition 3.2 (Squelette d'un mot)

Un squelette d'un mot m est un arbre ordonné dont tous les noeuds internes portent la même étiquette et dont les feuilles sont les lettres du mot, ordonnées de gauche à droite.

Si le squelette d'un mot est défini par rapport à une grammaire, il correspond alors à l'arbre de dérivation de ce mot dans cette grammaire dont on aurait remplacé les étiquettes des noeuds internes par une même étiquette. Un exemple est détaillé dans la figure 3.1. On parlera ainsi de *squelette réversible* si la structuration provient d'une grammaire réversible (voire définition 3.1).

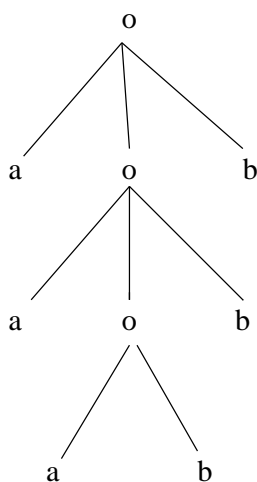


FIG. 3.1 – Le squelette du mot $aaabbb$ par rapport à la grammaire $\langle \{a,b\}, \{S\}, \{S \rightarrow ab, S \rightarrow aSb\}, S \rangle$.

Nous pouvons maintenant détailler le fonctionnement de l'algorithme RT (voir le pseudo-code 3.2).

La fonction `creer_automate_d_arbre()` retourne l'automate d'arbres reconnaissant l'ensemble des squelettes de S et seulement cet ensemble. L'algorithme

Algorithme 3.2 : RT

Données : Un ensemble S de squelettes réversibles**Résultat** : une grammaire hors-contexte réversible G **début** $A \leftarrow \text{creer_automate_d_arbre}(S);$ **pour chaque** *Paire d'états* (q_1, q_2) **de** A **faire** **si** q_1 *et* q_2 *ne respectent pas les conditions de réversibilité* **alors** $A \leftarrow \text{fusionner}(A, q_1, q_2)$ $G \leftarrow \text{genere_grammaire}(A);$ **retourner** G **fin**

teste ensuite si chaque paire d'état respecte ou non les contraintes de réversibilité. Nous ne détaillons pas ici la procédure permettant de passer d'un automate d'arbres réversible à une grammaire hors-contexte. Toutefois, une conséquence de ce processus est que pour tester la réversibilité il suffit de s'intéresser aux transitions utilisant les deux états : si deux transitions ne diffèrent que par ces deux états, alors il faut les fusionner. Cela revient à trouver la plus petite partition des états de l'automate initial respectant la réversibilité.

3.3.2 Propriétés

Dans [Sak92], Sakakibara montre que le temps d'exécution de l'algorithme RT est cubique dans la taille des squelettes en entrée.

D'autre part, il propose la construction d'un échantillon caractéristique à partir de l'automate d'arbres cible. La taille de cet échantillon est clairement polynomial dans la taille de la représentation cible. On peut donc en conclure que RT identifie à la limite en temps et en données polynomiaux la classe des langages hors-contextes représentés par des grammaires réversibles.

Nous aimerions attirer l'attention du lecteur sur un point. La taille de l'échantillon caractéristique est polynomial dans la taille de la grammaire cible, qui est, dans la preuve de Sakakibara, sous forme normale réversible. Or nous savons que la plus petite grammaire réversible représentant un langage hors-contexte peut être de taille exponentielle dans la taille de la plus petite grammaire hors-contexte représentant ce langage. Nous aurons dans un tel cas un échantillon caractéristique exponentiellement plus grand que la représentation du langage. Ce problème est lié à la définition du paradigme d'apprentissage qui stipule l'existence d'une cible sans précision supplémentaire et impose une contrainte de polynomialité dépendant de cette cible.

3.4 Modifications de SEQUITUR

Les auteurs de SEQUITUR ont tenté d'utiliser leur approche pour apprendre des langages [MW97]. Malheureusement leur processus se contentait de fusionner des non-terminaux, ce qui introduisait du non-déterminisme qu'ils ne réussirent pas à gérer. Contrairement à ce qu'ils ont tenté de faire, nos modifications n'ont pas pour but d'apprendre un langage en généralisant les règles d'une grammaire, mais à structurer des données afin d'utiliser *par la suite* un algorithme qui met en place un processus de généralisation.

La première modification que nous avons apporté à l'algorithme cherchait à le rendre capable de gérer non pas un mot mais un ensemble d'exemples d'un langage cible. Pour cela, nous avons dans un premier temps autorisé l'axiome à posséder plusieurs productions, une par exemple de l'ensemble d'apprentissage. L'algorithme ainsi modifié revient à créer une nouvelle règle à chaque nouveau mot, puis pour chaque exemple, à utiliser la version de SEQUITUR précédemment détaillée.

Le problème principal qui est alors apparu est que nos grammaires ainsi construites peuvent ne pas être réversibles. Ainsi, dans ce nouvel algorithme, l'axiome possède plusieurs dérivations, ce qui ne permet pas de garantir le deuxième point de la définition 3.1. En effet, la preuve de la propriété 3.1 reposait sur le fait qu'à un non-terminal correspondait une unique production. Comme ce n'était plus vérifié, et les expérimentations l'ont confirmé, la réversibilité n'était plus assurée.

Pour conserver des grammaires réversibles en sortie de l'algorithme nous avons mis en place le processus suivant. Lorsqu'un nouvel exemple est considéré, nous le lisons lettre par lettre, comme dans la version initiale. Mais au lieu de concaténer la nouvelle lettre, nous testons si elle est déjà reconnue à cette position par la grammaire courante. Ainsi, tant que le préfixe courant du nouveau mot w est aussi un préfixe d'un mot w' déjà traité, aucune création de non-terminal ou de règle n'a lieu. À partir du moment où la lettre courante ne peut être générée par la grammaire, un nouveau non-terminal N est créé. Il est inséré dans la grammaire à la place de la lettre du préfixe de w' qui le différencie celui de w . La première des deux productions de N correspond à la fin de la dérivation de w' . Quant à la deuxième production de ce non-terminal, elle commence par la lettre de w qui a conduit à la création de N . La fin de cette règle est construite à l'aide de la version initiale de SEQUITUR.

Grâce à cette construction, la réversibilité est assurée : chaque non-terminal possédant deux productions (voir plus si plusieurs mots partagent exactement le même préfixe) a des règles dont le premier élément est différent, ce qui permet d'assurer le deuxième point de la définition d'une grammaire réversible.

3.5 Expérimentations

Pour des raisons que nous expliciterons dans la section suivante, nous avons testé notre approche uniquement sur des données synthétiques, construites par nos soins à la main. Afin d'expliquer le comportement de l'algorithme, nous avons choisi de détailler ici la forme des structurations quand les données d'apprentissage sont des exemples du langage $\{a^n b^n : n > 0\}$.

3.5.1 Résultats

Plusieurs expérimentations ont été menées sans être totalement concluantes. Avant de donner des explications *a posteriori* sur les raisons de cet échec, nous détaillons ici les différentes approches qui ont été testées.

Si l'on considère le langage cible $L = \{a^n b^n : n > 0\}$, la structuration réversible permettant un apprentissage de ce langage par l'algorithme est, par exemple, celle de la grammaire $\langle \{a,b\}, \{S\}, \{S \rightarrow aSb, S \rightarrow ab\}, S \rangle$. Ainsi, le mot *aaaabbbb* doit être structuré ainsi $[a[a[a[ab]b]b]b]$. Notons que des structurations comme $[aa[a[ab]b]bb]$ ou $[a[aa[ab]bb]b]$ sont aussi acceptables, dans le sens où elles permettent aussi à l'algorithme de Sakakibara d'apprendre le langage L . Par contre, si les structurations ne suivent pas l'une de ces formes, l'algorithme RT retourne un langage sans lien avec la cible : dans certains cas par exemple, l'automate d'arbres initial ne brise aucune contrainte de réversibilité et le langage retourné est donc seulement celui des exemples.

Devant ce type de résultats, nous ne présentons pas ici une étude détaillée, avec courbes et tableaux à l'appui, qui ne permettrait que de se rendre compte que, dans certains cas, l'erreur en généralisation est de 100%. . . C'est pourquoi nous avons choisi de détailler le comportement de l'algorithme sur un exemple simple, le langage $\{a^n b^n : n > 0\}$.

La première série d'expériences est la plus simple : elle consiste à utiliser l'algorithme précédemment décrit sans aucun pré-traitement des données. Dans ce cas, aucune grammaire correcte n'a pu être obtenue. Ainsi, si l'on cherche à apprendre le langage L et que le premier mot de l'ensemble d'apprentissage est *aaaabbbb*, l'algorithme le structure ainsi : $[aa][aa][bb][bb]$. Puis, si le second mot est plus petit, par exemple *aaabbb*, la structuration devient $[aa][a[a]][bb][bb]$. En fait, le facteur *ab*, dont les lettres devraient être ensemble pour obtenir une structuration permettant l'apprentissage, est généré dès le début par deux non-terminaux distincts, ce qui nous empêche d'obtenir un type de structuration correspondant à nos attentes.

Lors de ces tests, nous nous sommes rendu compte que l'ordre dans lequel étaient traités les exemples avait une grande importance sur la pertinence du résultat. Par exemple, nous avons vu précédemment que si le premier mot est

$aaaabbbb$, le facteur ab central est divisé en deux. Or, comme nous le verrons dans la suite, cela peut ne pas se produire si un autre mot du langage est présenté en premier à l'algorithme.

C'est pourquoi nous avons mené une seconde expérience, en ordonnant les données d'apprentissage par taille croissante, puis par l'ordre lexicographique. Ainsi, sur l'exemple du langage L , si le premier mot dont l'algorithme doit s'occuper est ab , sa structuration est, dans un premier temps, $[ab]$. Puis, si le mot suivant est $aabb$, l'unique facteur apparaissant deux fois est ab , ce qui autorise une structuration correcte. Par contre, si le deuxième exemple est $aaaabbbb$, alors la structuration sera $[aa][aa][bb][bb]$ puisque la redondance du facteur aa est alors détectée avant celle du facteur ab .

Ainsi, le seul cas qui nous permet d'obtenir une structuration pertinente vis-à-vis du langage cible, est celui où les données sont triées par taille croissante et dont tous les mots plus petits qu'une certaine taille sont présents dans l'échantillon d'apprentissage. Ce qui n'est certainement pas réaliste. D'autant plus que si le nombre d'exemples d'apprentissage augmente, la structuration se complexifie et l'algorithme de Sakakibara n'est plus capable d'apprendre le langage : déjà, dans l'exemple ci-dessus, après avoir lu les mots ab et $aabb$, les structurations sont $[a[b]]$ et $[a[[ab]b]]$, ce qui n'est pas optimal.

3.5.2 Discussion

Devant des résultats comme ceux précédemment explicités, il est important de comprendre en quoi l'approche proposée ne permet pas de structurer correctement les exemples.

La première explication que nous pouvons donner repose sur le principe même de l'algorithme SEQUITUR. C'est un algorithme dont le but est de compresser un texte à l'aide d'une grammaire. Un non-terminal est utilisé pour générer un motif quand celui-ci dépasse une certaine fréquence d'apparition (deux occurrences dans les exemples détaillés ci-dessus). Or la fréquence d'apparition d'un motif n'est pas une mesure permettant de savoir si ce motif est un constituant, c'est-à-dire s'il est généré directement par un non-terminal (voir la définition 1.3). En effet, si l'on s'intéresse à l'exemple précédent, le langage $L = \{a^n b^n : n > 0\}$, le constituant principal du langage est le facteur ab . Or, dans un mot $a^k b^k$ du langage, pour $k > 4$, ce facteur n'apparaît qu'une seule fois alors que les facteurs de même taille aa et bb apparaissent $k/2$ fois. Ceci montre bien que la fréquence d'apparition d'un facteur n'est pas liée à la façon dont il est généré par la grammaire, ni à son rôle dans la construction des règles de cette grammaire.

Une autre raison peut expliquer l'échec de notre algorithme et la difficulté de toute approche cherchant à structurer *a priori* les données pour utiliser ensuite l'algorithme de Sakakibara : la structuration doit être celle de la cible. Ainsi, il

ne suffit pas d'avoir une structure réversible, il nous faut identifier une de celles qui correspondent au langage. En effet, pour le langage $L = \{a^n b^n : n > 0\}$, si plusieurs grammaires réversibles peuvent le représenter, elles ont des caractéristiques communes : toutes les grammaires de la famille $\mathbb{G} = \cup_i G_i$, avec pour tout $i > 0$, $G_i = \langle \{a,b\}, \{S\}, P_i, S \rangle$ où $P_i = \{S \rightarrow a^j S b^j : j \leq i\} \cup \{S \rightarrow ab\}$, engendrent le langage L et sont réversibles. Si les exemples sont structurés par l'une des grammaires de \mathbb{G} , l'algorithme de Sakakibara apprendra correctement le langage. Mais toute autre structure échouera. En fait, structurer correctement des exemples de mots du langage revient à apprendre une des structures correctes du langage, ce qui est presque aussi difficile que d'apprendre le langage lui-même (on a déjà dit que l'une des difficultés de l'inférence des langages hors-contextes était cette dualité entre le langage et sa structure). De plus, le résultat statuant qu'aucune classe de langages superfinie ne peut être identifiée à la limite à partir d'exemples positifs seulement tend à conforter l'idée qu'il n'est pas possible de trouver la structure à partir de ce même type d'exemples.

3.6 Conclusion

L'idée de structurer *a priori* les exemples d'apprentissage pour utiliser ensuite l'algorithme RT de Sakakibara est tentante. L'algorithme de compression de données SEQUITUR possède plusieurs qualités pour être un prétendant pour ce type d'approches. Malheureusement, nous avons vu que cette voie est une impasse, et notre intuition est qu'il est aussi difficile d'apprendre une grammaire représentant un langage à partir d'exemples de mots de ce langage que de trouver la structuration de ces exemples correspondant à la grammaire cible. En particulier, les problèmes de générativité et d'indivisibilité se posent de la même façon.

Il est possible qu'à l'avenir un algorithme capable de structurer *a priori* les données soit mis en évidence. Mais nos résultats ainsi que l'expérience acquise pendant ces travaux, ne nous rendent pas optimistes quant à l'aboutissement d'une telle démarche. Si nous avons choisi de détailler ces recherches, pourtant globalement infructueuses, dans cette thèse, c'est principalement pour rendre compte de la difficulté d'une telle approche, et prévenir ainsi les chercheurs qui seraient aussi tentés par cette voie.

4

Changement de représentation

4.1 Motivation

Dans la première partie de ce document, nous avons explicité plusieurs facteurs rendant difficile l'inférence de grammaires hors-contextes. Notre but étant d'apprendre des langages hors-contextes, une voie possible pour contourner ces problèmes est de changer la représentation des langages. Nous avons exploré une telle alternative en choisissant de travailler avec des systèmes de réécriture de mots. Ce formalisme a été introduit en 1910 par Axel Thue [Thu14, ST00] et ses extensions aux termes et aux arbres ont été, et sont encore, étudiés en apprentissage [Rao04, TN90, LG90]. Lorsque nous avons commencé à étudier les systèmes de réécriture de mots (SRMs) en 2003, c'était, à notre connaissance, la première fois que ce formalisme était utilisé pour faire de l'apprentissage.

Réécrire un mot consiste à remplacer des facteurs par d'autres, et ce autant que possible, en suivant des lois appelées *règles de réécriture*. Par exemple, considérons les mots sur l'alphabet $\Sigma = \{a,b\}$ et la règle $ab \vdash \lambda$. Utiliser cette règle revient à remplacer des facteurs ab par le mot vide, c'est-à-dire par effacer des facteurs ab . Ainsi, le mot $abaabbab$ peut se réécrire comme suit :

$$abaabbab \vdash abaabb \vdash abab \vdash ab \vdash \lambda$$

D'autres *dérivations* sont possibles (on peut, par exemple, effacer en premier le facteur ab en tête du mot), mais elles finiront toutes par réécrire $abaabbab$ en λ . Nous étudierons dans la suite cette propriété.

Pour en revenir à l'exemple, il est assez intuitif qu'un mot se réécrira en λ si et seulement si il appartient au langage de parenthèses, c'est-à-dire au langage de Dyck. Plus précisément, le langage de Dyck sur une paire de parenthèses (a,b) est entièrement caractérisé par cette simple règle de réécriture et par le mot vide, qui est atteint par réécriture à partir de n'importe quel autre mot du langage. Cette propriété a été énoncée pour la première fois par Nivat [Niv70], ce qui a

été le point de départ d'un grand nombre de travaux en théorie des langages au cours des trente dernières années.

Dans ce chapitre, nous commencerons par définir un nouveau type de SRM, appelé *système délimité de réécriture de mots (SDRM)*. Puis nous étudierons les différentes propriétés que doivent suivre de tels systèmes pour représenter efficacement des langages formels. Nous verrons ensuite comment faire en sorte que ces propriétés soient respectées. Nous détaillerons par la suite un algorithme capable d'apprendre de tels systèmes. Ses propriétés de convergence et la classe de langages apprise seront ensuite étudiées. Avant de conclure, nous développerons quelques expérimentations que nous avons menées.

4.2 Définir des langages avec des systèmes de réécriture

Comme nous l'avons explicité dans la section précédente, les systèmes de réécriture de mots (SRMs) sont habituellement définis par des ensembles de règles de réécriture. Ces règles permettent de remplacer des facteurs d'un mot par d'autres facteurs. Néanmoins, ce mécanisme offre peu de flexibilité : une règle que l'on aimerait utiliser au début des mots (ou à la fin) va pouvoir s'appliquer aussi au milieu du mot et avoir ainsi des effets indésirables. C'est pourquoi nous introduisons deux nouveaux symboles $\$$ et \mathcal{L} , appelés délimiteurs, qui ne font pas partie de l'alphabet Σ et qui marquent respectivement le début et la fin des mots. Nous considérons donc des mots de l'ensemble $\$\Sigma^*\mathcal{L}$. Quant aux règles de réécriture, elles sont *partiellement* marquées, et ainsi définies sur $\overline{\Sigma^*} = (\lambda + \$)\Sigma^*(\lambda + \mathcal{L})$. Leur forme contraindra par conséquent leur utilisation : soit en début de mot (les règles commençant par un $\$$), soit à la fin du mot (les règles finissant par \mathcal{L}), soit à l'intérieur du mot (les règles sans délimiteurs), soit sur un unique mot (les règles commençant par $\$$ et finissant par \mathcal{L}).

Cette approche est à mi-chemin entre les systèmes de réécritures de mots usuels et les *systèmes de réécriture avec variables* [MNO88] où le nombre de "variables" (ici les délimiteurs) et leur utilisation ne sont pas fixés.

Définition 4.1 (Système délimité de réécriture de mots)

- Une règle de réécriture R est une paire ordonnée (l,r) , généralement notée $l \vdash r$. l s'appelle la partie gauche, r la partie droite.
- On dit que $R = l \vdash r$ est une règle de réécriture délimitée si l et r satisfont une des quatre contraintes suivantes :
 1. $l, r \in \$\Sigma^*$, ou

2. $l, r \in \$\Sigma^* \mathcal{L}$, ou
3. $l, r \in \Sigma^* \mathcal{L}$, ou
4. $l, r \in \Sigma^*$.

Les règles de type 1 et 2 sont appelées des $\$$ -règles.

- Un système délimité de réécriture de mots (SDRM) est un ensemble fini de règles de réécriture délimitées.

Par exemple, $\$ab \vdash \abb , $ab \vdash \lambda$ et $\$ab\mathcal{L} \vdash \$abb\mathcal{L}$ sont des règles de réécriture délimitées, mais pas $ab \vdash \$$ ($ab \in \Sigma^*$ et $\$ \in \Σ^*) ou $\$ab\mathcal{L} \vdash a\$b\mathcal{L}$.

On note $|\mathcal{R}|$ le nombre de règles d'un SDRM \mathcal{R} et $\|\mathcal{R}\|$ la somme des longueurs des règles de \mathcal{R} : $\|\mathcal{R}\| = \sum_{(l \vdash r) \in \mathcal{R}} |lr|$.

Étant donné un SDRM \mathcal{R} et deux mots w_1 et w_2 dans $\overline{\Sigma^*}$, on dit que w_1 se réécrit en un pas en w_2 , noté $w_1 \vdash_{\mathcal{R}} w_2$ ou simplement $w_1 \vdash w_2$, ssi il existe une règle $(l \vdash r) \in \mathcal{R}$ et deux mots $u, v \in \overline{\Sigma^*}$ tels que $w_1 = ulv$ et $w_2 = urv$. Un mot w est réductible s'il existe w' tel que $w \vdash w'$; il est irréductible sinon.

La propriété suivante est immédiate :

Propriété 4.1 $\$\Sigma^* \mathcal{L}$ est stable par réécriture : Si $w_1 \in \$\Sigma^* \mathcal{L}$ et $w_1 \vdash w_2$ alors $w_2 \in \$\Sigma^* \mathcal{L}$, c'est-à-dire qu'aucun $\$$ ou \mathcal{L} ne peut apparaître, se déplacer ou disparaître dans un mot par réécriture.

On note $\vdash_{\mathcal{R}}^*$ (ou simplement \vdash^*) la fermeture réflexive et transitive de $\vdash_{\mathcal{R}}$. On dit que w_1 se réduit en w_2 ou que w_2 est dérivable à partir de w_1 ssi $w_1 \vdash^* w_2$.

Le langage induit par un système délimité de réécriture de mots est défini de manière analogue au langage induit par un système de réécriture de mots usuel :

Définition 4.2 (Langage induit par un SDRM)

Soit \mathcal{R} un SDRM et $e \in \Sigma^*$ un mot irréductible dans ce système. Le langage $\mathcal{L}(\mathcal{R}, e)$ est l'ensemble des mots se réduisant en e grâce aux règles de \mathcal{R} : $\mathcal{L}(\mathcal{R}, e) = \{w \in \Sigma^* \mid \$w\mathcal{L} \vdash_{\mathcal{R}}^* \$e\mathcal{L}\}$.

Pour décider si un mot appartient ou non à $\mathcal{L}(\mathcal{R}, e)$ il suffit de vérifier si ce mot peut se réduire en e . Cela pose deux problèmes principaux : à l'aide d'un SDRM quelconque, un mot peut se dériver en un grand nombre de mots irréductibles différents, et chacune de ces dérivations peut être de longueur exponentielle dans la taille du mot. Avant de nous intéresser à ces problèmes, ce qui sera le sujet de la section suivante, nous développons quelques exemples.

Exemple 4.1 Soit $\Sigma = \{a, b\}$ l'alphabet des langages.

- $\mathcal{L}(\{ab \vdash \lambda\}, \lambda)$ représente le langage de Dyck. En effet, puisque l'unique règle efface les facteurs ab , nous obtenons l'exemple de dérivation suivant :

$$\$aabbab\mathcal{L} \vdash \$aabb\mathcal{L} \vdash \$ab\mathcal{L} \vdash \$\mathcal{L}$$

- $\mathcal{L}(\{ab \vdash \lambda; ba \vdash \lambda\}, \lambda)$ est le langage algébrique $\{w \in \Sigma^* : |w|_a = |w|_b\}$, car chaque pas de dérivation efface un a et un b .
- $\mathcal{L}(\{aabb \vdash ab; \$ab\mathcal{L} \vdash \$\mathcal{L}\}, \lambda) = \{a^n b^n : n \geq 0\}$. Par exemple,

$$\$aaaabbbb\mathcal{L} \vdash \$aaabbb\mathcal{L} \vdash \$aabb\mathcal{L} \vdash \$ab\mathcal{L} \vdash \$\mathcal{L}$$

Il est à noter que la règle $\$ab\mathcal{L} \vdash \\mathcal{L} est nécessaire pour obtenir λ , ce qui ne serait pas possible sans l'aide de délimiteurs.

- $\mathcal{L}(\{\$ab \vdash \$\}, \lambda)$ est le langage régulier $(ab)^*$. En effet,

$$\$ababab\mathcal{L} \vdash \$abab\mathcal{L} \vdash \$ab\mathcal{L} \vdash \$\mathcal{L}$$

Nous verrons dans le théorème 4.2 comment tous les langages réguliers peuvent être efficacement représentés par un SDRM.

4.3 Représenter efficacement des langages par des SDRMs

Dans cette section, nous allons introduire deux contraintes sur les règles des SDRMs permettant d'obtenir des systèmes représentant efficacement des langages. Ces deux contraintes permettent à nos systèmes de satisfaire deux propriétés (usuelles) des systèmes de réécriture : la propriété de terminaison (sous-section 4.3.1) et la propriété de confluence (sous-section 4.3.2). Enfin, nous montrerons que ces contraintes ne sont pas trop restrictives : la classe de langages représentables contient l'ensemble des langages réguliers et les principaux langages hors-contextes régulièrement étudiés dans la littérature.

4.3.1 La propriété de terminaison

Comme cela a déjà été introduit, un mot w appartient au langage $\mathcal{L}(\mathcal{R}, e)$ ssi w se réécrit en e en utilisant les règles de \mathcal{R} . Toutefois cette définition est, dans le cas général, trop peu contraignante pour être efficace et soulève par conséquent de multiples difficultés. La première est qu'il est facile d'imaginer des SDRMs dans lesquels un mot peut se réécrire indéfiniment. Le SDRM $\mathcal{R} = \{a \vdash b; b \vdash a; c \vdash cc\}$ est un bon exemple puisqu'il permet les dérivations suivantes :

$$\begin{aligned} aa \vdash ba \vdash aa \vdash ba \vdash aa \dots \\ aca \vdash acca \vdash accca \vdash acccca \vdash accccca \vdash \dots \end{aligned}$$

La terminaison des dérivations induites par un SRM est indécidable dans le cas général, et la décidabilité de cette propriété pour des sous-classes de SRMs est

toujours un thème de recherche (voir par exemple [MG05]). En ce qui concerne les SDRMs, il n'y a pas de raison pour que ce soit plus simple que dans le cas général.

D'un autre côté, même si toutes les dérivations d'un SDRM sont finies, elles peuvent être de taille exponentielle dans la longueur du mot à réécrire, et, par conséquent, inefficacement calculables. Par exemple, considérons le SDRM suivant :

$$\mathcal{R} = \left\{ \begin{array}{l} 1\mathcal{L} \vdash 0\mathcal{L}, \quad 0\mathcal{L} \vdash c1d\mathcal{L}, \\ 0c \vdash c1, \quad 1c \vdash 0d, \\ d1 \vdash 1d, \quad dd \vdash \lambda \end{array} \right\}$$

Toutes les dérivations induites par \mathcal{R} sont finies. En effet, si l'on suppose que $d > 1 > 0 > c$, la partie gauche l de chaque règle $l \vdash r$ est plus grande, suivant l'ordre hiérarchique, que sa partie droite r , ce qui en fait un système dont les dérivations se terminent toujours [DJ90] (on parle alors de système noëthérien). Toutefois, si l'on cherche à réécrire $\underbrace{\$11\dots 1}_n\mathcal{L} = \$1^n\mathcal{L}$ en $\$0^n\mathcal{L}$, alors tous les codages en base 2 des entiers positifs entre $2^n - 1$ et 0 seront dérivés une fois pendant la réduction de $\$1^n\mathcal{L}$. La dérivation correspondante aura par conséquent une taille exponentielle dans la taille du mot en entrée. Par exemple, pour $n = 4$, nous obtenons la dérivation suivante :

$$\begin{array}{l} \$1111\mathcal{L} \\ \vdash \$1110\mathcal{L} \\ \vdash \$111\underline{c}1d\mathcal{L} \vdash \$110\underline{d}1d\mathcal{L} \vdash \$1101\underline{d}d\mathcal{L} \vdash \$1101\mathcal{L} \\ \vdash \$1100\mathcal{L} \\ \vdash \$110\underline{c}1d\mathcal{L} \vdash \$11\underline{c}11d\mathcal{L} \vdash \$10\underline{d}11d\mathcal{L} \vdash \$101\underline{d}1d\mathcal{L} \vdash \$1011\underline{d}d\mathcal{L} \vdash \$1011\mathcal{L} \\ \vdash \$1010\mathcal{L} \\ \vdash^* \$1001\mathcal{L} \\ \vdash \$1000\mathcal{L} \\ \vdash^* \$0111\mathcal{L} \dots \\ \vdash^* \$0000\mathcal{L}. \end{array}$$

Pour empêcher de telles dérivations, nous avons besoin d'une extension de l'ordre hiérarchique \triangleleft à $\overline{\Sigma^*}$. Pour ce faire, nous définissons l'ordre hiérarchique étendu, noté \prec , comme suit :

$$\forall w_1, w_2 \in \Sigma^*, \text{ si } w_1 \triangleleft w_2 \text{ alors } w_1 \prec \$w_1 \prec w_1\mathcal{L} \prec \$w_1\mathcal{L} \prec w_2.$$

Si $a < b$ alors $\lambda \triangleleft a \triangleleft b \triangleleft aa \triangleleft ab \triangleleft ba \triangleleft bb \triangleleft aaa \triangleleft \dots$, ce qui implique $\lambda \prec \$ \prec \mathcal{L} \prec \$\mathcal{L} \prec a \prec \$a \prec a\mathcal{L} \prec \$a\mathcal{L} \prec b \prec \dots$. Notons que \prec donne à $\overline{\Sigma^*}$ la structure d'un ensemble bien ordonné, c'est-à-dire que chaque sous-ensemble de $\overline{\Sigma^*}$ admet un minimum.

La définition suivante, un peu technique, permet de définir des SDRMs dont les dérivations sont finies et de taille polynomiale.

Définition 4.3 (SDRM hybride)

- Une règle délimitée $R = l \vdash r$ est hybride si :
 1. R est une $\$$ -règle et $r \prec l$, ou
 2. R n'est pas $\$$ -règle et $|r| < |l|$.
- Un SDRM est hybride si toutes ses règles sont hybrides.

Par exemple, les règles $bb \vdash a$ et $\$bb \vdash \aa sont hybrides, mais pas $bb \vdash aa$. Comme cette propriété est purement syntaxique, il est immédiat de la vérifier pour une règle.

On obtient le résultat suivant :

Théorème 4.1 *Toutes les dérivations induites par un SDRM hybride \mathcal{R} sont finies. De plus, chaque dérivation d'un mot w demande au plus $|w| \times |\mathcal{R}|$ pas de réécriture.*

Preuve : Soit $w_1 \vdash w_2$ un pas de réécriture. Il existe une règle hybride $l \vdash r$ et deux mots $u, v \in \overline{\Sigma}^*$ tels que $w_1 = ulv$ et $w_2 = urv$. Notons que si $|r| < |l|$ alors $r \prec l$ et si $r \prec l$ alors $w_2 \prec w_1$. Il s'en suit que si on considère une dérivation $u_0 \vdash u_1 \vdash u_2 \vdash \dots$ alors $u_0 \succ u_1 \succ u_2 \succ \dots$. Comme $\overline{\Sigma}^*$ est un ensemble bien ordonné par \prec , il n'existe pas de chaîne infinie et strictement décroissante $u_0 \succ u_1 \succ u_2 \succ \dots$, ce qui assure que toutes les dérivations sont finies.

Il nous reste à montrer que les dérivations sont de longueur polynomiale, ce qui se fait par induction. Soit $n \geq 0$. Supposons que pour tout w' tel que $|w'| < n$, la longueur des dérivations de w' est au plus $|w'| \times |\mathcal{R}|$. Soit w un mot de taille n . Le nombre de pas de dérivation qui préservent la taille de w est au plus $|\mathcal{R}|$: toutes les règles conservant la taille sont de la forme $\$l \vdash \r , avec $|r| = |l|$ et $l \succ r$. En outre, elles ne peuvent s'appliquer qu'une fois chacune. En effet, dans le cas contraire, il existerait une dérivation $\$lu \vdash \$ru \vdash^* \$lv$ avec $|u| = |v|$ (puisque la taille est conservée). Comme $\$ru \vdash^* \lv et $|u| = |v|$, cela implique $r \succeq l$ ce qui est impossible. Il n'y a donc que $|\mathcal{R}|$ pas de dérivation possibles qui conservent la taille, la règle suivante induit forcément un mot w' tel que $|w'| < n$. Par l'hypothèse d'induction, la taille d'une dérivation commençant par w est d'au plus $|\mathcal{R}| + |w'| \times |\mathcal{R}| \leq |w| \times |\mathcal{R}|$ \square

Notons qu'un SDRM dont toutes les règles seraient orientées uniquement par l'ordre hiérarchique étendu ne donnerait pas forcément des SDRMs dont les dérivations sont efficaces : le système est alors noëthérien mais les dérivations ne sont pas forcément polynomiales dans la taille du mot.

4.3.2 La propriété de confluence

Un SDRM hybride induit des dérivations finies et linéaires (voir théorème 4.1). Mais la polynomialité des dérivations n'est pas suffisante pour tester efficacement si un mot appartient au langage induit par un SDRM. En effet, dans un SDRM hybride quelconque, un mot peut se dériver en un nombre important de mots irréductibles. Par conséquent, répondre à la question " $w \in \mathcal{L}(\mathcal{R}, e)$?" nécessite de calculer *toutes* les dérivations de w et de tester ensuite si l'une d'elle conduit à e . En d'autres termes, un tel SDRM est non déterministe et donc inefficace. Une manière usuelle de contourner ce problème est d'imposer aux systèmes d'être *confluents* [CR36].

Définition 4.4 (Confluence)

Un SDRM est *confluent* si pour tous mots $w_1, u_1, u_2 \in \overline{\Sigma}^*$ tels que $w_1 \vdash^* u_1$ et $w_1 \vdash^* u_2$, il existe un mot w_2 tel que $u_1 \vdash^* w_2$ et $u_2 \vdash^* w_2$ (voir la figure 4.1).

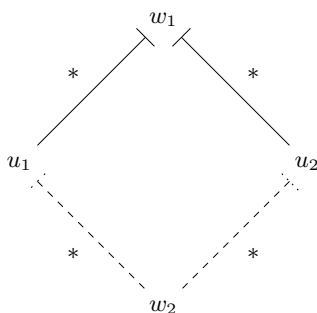


FIG. 4.1 – La confluence porte aussi le nom de "lemme du diamant".

Dans la définition précédente, si u_1 et u_2 sont des mots irréductibles alors $u_1 = u_2 (= w_2)$. Donc, pour tout mot w il n'existe qu'un mot irréductible pouvant être obtenu à partir de w dans un SDRM confluent. Malheureusement la confluence n'est pas décidable dans le cas général; nous l'assurons en restreignant notre étude à des SDRM particuliers. Cette contrainte est beaucoup plus restrictive que ce qui est nécessaire pour assurer la confluence, mais elle a l'intérêt d'être facilement et rapidement vérifiable.

Définition 4.5 (SDRM PNC)

- Deux règles délimitées $R_1 = l_1 \vdash r_1$ et $R_2 = l_2 \vdash r_2$ sont presque non-chevauchantes (PNC) si les cas suivants sont vérifiés:
 1. Si $l_1 = l_2$ alors $r_1 = r_2$

2. Si l_1 est strictement inclus dans l_2 (voir figure 4.2 (a)), c'est-à-dire si $\exists u, v \in \overline{\Sigma^*}, uv \neq \lambda$ tels que $ul_1v = l_2$ alors $ur_1v = r_2$
 3. Si l_2 est strictement inclus dans l_1 , c'est-à-dire si $\exists u, v \in \overline{\Sigma^*}, uv \neq \lambda$ tels que $ul_2v = l_1$ alors $ur_2v = r_1$
 4. Si un suffixe strict de l_1 est un préfixe strict de l_2 (voir figure 4.2 (b)), c'est-à-dire si $\exists u, v \in \overline{\Sigma^*}, l_1u = vl_2, 0 < |v| < |l_1|$, alors $r_1u = vr_2$
 5. Si un suffixe strict de l_2 est un préfixe strict de l_1 , c'est-à-dire si $\exists u, v \in \overline{\Sigma^*}, l_2u = vl_1, 0 < |v| < |l_2|$, alors $r_2u = vr_1$
 6. Si aucun chevauchement ne se produit, c'est-à-dire $l_i = uv$ et $l_j = vw$ implique $v = \lambda$, pour $i, j \in \{1, 2\}$.
- Un SDRM est presque non chevauchant (PNC) si les règles qui le constituent sont presque non chevauchantes deux à deux.

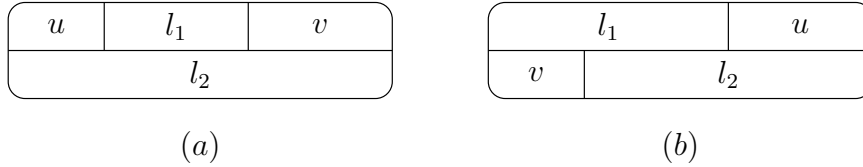


FIG. 4.2 – Deux règles se chevauchent si leur partie gauche se chevauchent; la partie (a) de la figure correspond au cas 2 (et symétriquement au cas 3); la partie (b) illustre le cas 4 (et symétriquement le cas 5).

Moins formellement, cette propriété a le but suivant : si deux règles peuvent s'appliquer sur le même facteur d'un mot, alors utiliser l'une ou l'autre dérivera le même mot. Cette condition est souvent utilisée dans les travaux sur la réécriture de termes (voir par exemple [O'D77]).

Exemple 4.2

- Les règles $R_1 = ab \vdash \lambda$ et $R_2 = ba \vdash \lambda$ sont presque non chevauchantes car aba se réduit en a avec les deux règles et que bab se réduit en b avec chacune des règles.
- Par contre, les règles $R_3 = ab \vdash a$ et $R_4 = ba \vdash a$ ne sont pas PNC; en effet, si aba se réécrit en aa avec les deux règles, ce n'est pas le cas de bab qui se réduit en ba avec R_3 et en ab avec R_4 .
- La règle $R_5 = aa \vdash b$ forme à elle seule un SDRM qui n'est pas PNC car aaa peut être réécrit en ab et en ba en utilisant R_5 .

On obtient le résultat suivant :

Propriété 4.2 *Soit \mathcal{R} un SDRM presque non chevauchant et w un mot de $\Sigma^* \mathcal{L}$. Si il existe w_1 et w_2 dans $\Sigma^* \mathcal{L}$ tels que $w_1 \neq w_2$, $w \vdash_{\mathcal{R}}^* w_1$ et $w \vdash_{\mathcal{R}}^* w_2$ alors il existe $z \in \Sigma^* \mathcal{L}$ tel que $w_1 \vdash_{\mathcal{R}}^* z$ et $w_2 \vdash_{\mathcal{R}}^* z$. En d'autres termes, un SDRM PNC est confluent.*

Preuve : Soit \mathcal{R} un SDRM presque non chevauchant. Nous notons $w_1 \vdash_{\epsilon} w_2$ si $w_1 \vdash_{\mathcal{R}} w_2$ ou si $w_1 = w_2$. Nous allons montrer que $\vdash_{\mathcal{R}}$ est sous-commutative [Klo92] : pour tout w, u_1, u_2 , si $w \vdash_{\mathcal{R}} u_1$ et $w \vdash_{\mathcal{R}} u_2$ alors il existe w' tel que $u_1 \vdash_{\epsilon} w'$ et $u_2 \vdash_{\epsilon} w'$.

Supposons que $w \vdash_{\mathcal{R}} u_1$ à l'aide d'une règle $R_1 = l_1 \vdash r_1$ et que $w \vdash_{\mathcal{R}} u_2$ à l'aide d'une règle $R_2 = l_2 \vdash r_2$. Si les deux pas de réécriture sont indépendants (c'est-à-dire sans chevauchement au niveau des règles), alors il existe x, y, z tels que $w = xl_1yl_2z$, $u_1 = xr_1yl_2z$ et $u_2 = xl_1yr_2z$. Dans ce cas $u_1 \vdash_{\mathcal{R}} w'$ et $u_2 \vdash_{\mathcal{R}} w'$, avec $w' = xr_1yr_2z$. Si les pas de réécriture ne sont pas indépendants, alors R_1 chevauche R_2 (ou vice versa) et, comme le système est PNC, $u_1 = u_2$.

Il est possible, sans difficulté, de généraliser cette propriété par un raisonnement par induction : si $w \vdash_{\mathcal{R}}^* u_1$ et $w \vdash_{\mathcal{R}}^* u_2$, alors il existe w' tel que $u_1 \vdash_{\epsilon}^* w'$ et $u_2 \vdash_{\epsilon}^* w'$, ou \vdash_{ϵ}^* est la fermeture transitive de \vdash_{ϵ} . Finalement, comme $u_1 \vdash_{\epsilon}^* w'$ et $u_2 \vdash_{\epsilon}^* w'$ on en déduit que $u_1 \vdash_{\mathcal{R}}^* w'$ et $u_2 \vdash_{\mathcal{R}}^* w'$. \square

4.3.3 Sur les SDRMs hybrides et PNCs

Dans le reste de ce chapitre, nous ne considérons que des SDRMs hybrides et PNCs. Cela implique les propriétés suivantes :

- Pour chaque mot w , il n'existe qu'un unique mot irréductible pouvant être obtenu par réécriture à partir de w . Ce mot irréductible est appelé *forme normale de w* et est noté $w \downarrow$ (ou $w \downarrow_{\mathcal{R}}$ quand il y a une ambiguïté sur le SDRM \mathcal{R}).
- Comme ces systèmes sont noéthériens, aucune dérivation ne peut être prolongée indéfiniment. De plus, quelle que soit la dérivation envisagée, chaque pas de réécriture induit un mot inéluctablement plus proche de $w \downarrow$.

Une conséquence importante des points qui précèdent est que l'on dispose maintenant d'un algorithme simple et rapide pour tester l'appartenance d'un mot w à un langage $\mathcal{L}(\mathcal{R}, e)$: il suffit de (i) calculer la forme normale $w \downarrow$ de w (ce qui se fait en temps polynomial dans la taille de \mathcal{R} et de w) et (ii) tester si $w \downarrow$ et e sont syntaxiquement équivalents.

Les deux contraintes que l'on a ajoutées à nos SDRMs peuvent paraître trop restrictives vis-à-vis de la classe de langages représentables. Il n'en est rien : il est facile de vérifier, par exemple, que tous les exemples de langages détaillés à la

fin de la section 4.2 correspondent à des SDRMs hybrides et PNCs. De plus, le résultat suivant établit que tous les langages réguliers sont représentables par des SDRMs hybrides et PNCs :

Théorème 4.2 *Pour chaque langage régulier L , il existe un SDRM hybride et PNC \mathcal{R} et un mot e irréductible dans \mathcal{R} tels que $L = \mathcal{L}(\mathcal{R}, e)$.*

Preuve : On peut prouver ce théorème en montrant qu'il existe une équivalence entre (1) les SDRMs hybrides et PNCs ne comportant que des $\$$ -règles et (2) les *grammaires préfixes* [FJ94] qui forment une classe de représentation des langages réguliers. Une telle preuve se trouve dans [EdIHJ04].

Nous avons préféré donner ici une preuve directe, s'appuyant sur la représentation des langages réguliers par les automates finis déterministes.

Soit $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ l'automate canonique minimal représentant le langage régulier L . Pour chaque état q de l'automate, nous définissons le plus petit mot w_q qui permet d'atteindre cet état à partir de l'état initial, c'est-à-dire $w_q \in \Sigma^*$ tel que $\delta(q_0, w_q) = q$ et $\forall w' \triangleleft w_q, \delta(q_0, w') \neq q$. Soit e le plus petit mot, vis-à-vis de \triangleleft , permettant d'atteindre un état final : $e = \min_{q \in F}^{\triangleleft} w_q$. Soit $\mathcal{R}_1 = \{ \$w_q x \vdash \$w_{q'} : q, q' \in Q, x \in \Sigma, \delta(q, x) = q', w_{q'} \neq w_q x \}$ et $\mathcal{R}_2 = \{ \$w_q \mathcal{L} \vdash \$e \mathcal{L} : q \in F, w_q \neq e \}$ et $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. Par induction, il est facile de montrer que $\mathcal{L}(\mathcal{R}, e) = L$ car $\forall w \in \Sigma^*, \delta(q_0, w) = q \iff \$w \vdash^* \$w_q$. Un exemple de cette construction est donné à la fin de cette preuve.

Il nous reste à prouver que \mathcal{R} est un SDRM hybride et PNC. Dans un premier temps, montrons que \mathcal{R} est hybride, c'est-à-dire que pour toute règle $\$l \vdash \r de \mathcal{R}_1 , $l \triangleleft r$. Par construction, si il existe une règle $\$w_q x \vdash \$w_{q'}$ alors $\delta(q_0, w_q x) = q'$; comme $w_{q'}$ est le plus petit mot atteignant l'état q' , on a $w_{q'} \prec \$w_q x$. En ce qui concerne les règles de \mathcal{R}_2 , comme $e = \min_{q \in F}^{\triangleleft} w_q$, ses règles sont forcément hybrides.

Il nous faut prouver maintenant que \mathcal{R} est presque non chevauchant. Considérons deux règles $R_1 = \$w_q x \vdash \w_p et $R_2 = \$w_{q'} y \vdash \$w_{p'}$ de \mathcal{R}_1 . La seule possibilité de chevauchement est que l'une des parties gauches, disons celle de R_2 , soit contenue dans l'autre (celle de R_1). Formellement, il existe $m \in \Sigma^*$ tel que $w_{q'} y m x = w_q$. Il y a deux cas :

- Soit $m = \lambda$ et alors $w_{q'} y = w_q$, ce qui implique que $\delta(q', y) = q$. Or, par construction de R_2 , $\delta(q', y) = p'$ et alors $p' = q$ (car l'automate est déterministe), et donc $w_{p'} = w_q$. Ce qui implique $R_2 = \$w_{q'} y \vdash \w_q avec $w_{q'} y = w_q$, ce qui est en contradiction avec la définition des règles.
- Soit $m \neq \lambda$ et alors $w_{q'} y m = w_q$. On en déduit que $q = \delta(q_0, w_q) = \delta(q_0, w_{q'} y m) = \delta(q', y m) = \delta(p', m) = \delta(q_0, w_{p'} m)$. D'autre part, la définition de R_2 implique $\$w_{p'} \prec \$w_{q'} y$ ce qui conduit à $\$w_{p'} m \prec \$w_{q'} y m = \$w_q$. Donc, on obtient $\delta(q_0, w_{p'} m) = q$ et $\$w_{p'} m \prec \w_q , ce qui est impossible car w_q est le plus petit mot qui atteint l'état q .

En ce qui concerne les règles de \mathcal{R}_2 , aucune ne peut chevaucher une autre règle de \mathcal{R}_2 car leur partie gauche sont délimitées des deux cotés. La seule possibilité de chevauchement concerne donc une règle de \mathcal{R}_2 avec une règle de \mathcal{R}_1 . Or ceci est impossible pour la même raison que le deuxième item ci-dessus. \square

Exemple 4.3 Soit A l'automate fini déterministe minimal reconnaissant le langage $L = (a + b)^*a(a + b)$ (voir figure 4.3). A possède quatre états $\{0,1,2,3\}$, 0 est l'état initial et 2 et 3 sont les états finaux. En outre, $\delta(0,b) = \delta(3,b) = 0$, $\delta(0,a) = \delta(3,a) = 1$, $\delta(1,a) = \delta(2,a) = 2$, $\delta(1,b) = \delta(2,b) = 3$. La construction ci-dessus donne : $w_0 = \lambda, w_1 = a, w_2 = aa$ et $w_3 = ab$. On a aussi $e = aa$, $\mathcal{R}_1 = \{\$b \vdash \$; \$aaa \vdash \$aa; \$aab \vdash \$ab; \$aba \vdash \$a; \$abb \vdash \$\}$ et $\mathcal{R}_2 = \{\$ab\mathcal{L} \vdash \$aa\mathcal{L}\}$. Il est rapide de vérifier que $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ est un SDRM hybride et PNC tel que $\mathcal{L}(\mathcal{R}, aa) = L$.

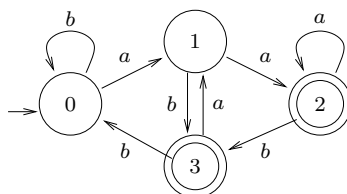


FIG. 4.3 – L 'automate fini déterministe minimal reconnaissant le langage $(a + b)^*a(a + b)$.

4.4 LARS: fonctionnement et exemples

Nous nous intéressons maintenant à l'apprentissage des systèmes délimités de réécriture de mots. C'est l'objet d'un algorithme, LARS (voir l'algorithme 4.1), qui à partir d'un échantillon d'exemples positifs et négatifs d'un langage cible, retourne un SDRM cohérent avec les données. Une implémentation de cet algorithme est disponible en ligne à l'adresse

<http://eurise.univ-st-etienne.fr/~eyraud/LARS>.

LARS est un algorithme glouton qui énumère les règles dans l'ordre hiérarchique étendu \prec . Les parties gauches et droites des règles sont prises dans l'ensemble trié des facteurs des exemples positifs (nommé F dans le pseudo-code). Cette manière de faire réduit drastiquement l'espace de recherche sans remettre en question les capacités d'apprentissage de l'algorithme (du moins dans le cadre de l'identification à la limite en temps et données polynomiaux).

Un facteur ne peut être utilisé en partie gauche que si aucune règle précédemment inférée ne l'a fait disparaître de l'ensemble d'apprentissage (fonction *apparaît()*): dans le cas contraire la règle n'aurait aucun effet direct sur

Algorithme 4.1 : LARS (Learning Algorithm for Rewriting Systems)**Données** : un échantillon d'exemples positifs et négatifs $S = \langle S_+, S_- \rangle$ **Résultat** : $\langle \mathcal{R}, e \rangle$ où \mathcal{R} est un SDRM hybride et PNC et e un mot irréductible

début

```

 $\mathcal{R} \leftarrow \emptyset; I_+ \leftarrow S_+; I_- \leftarrow S_-;$ 
 $F \leftarrow \text{trier}_{\preceq} \{v \in \overline{\Sigma}^* : \exists u, w \in \overline{\Sigma}^*, uvw \in I_+\};$ 
pour  $i = 1$  à  $|F|$  faire
  si  $\text{apparaît}(F[i], I_+)$  alors
    pour  $j = 0$  à  $i - 1$  faire
       $\mathcal{S} \leftarrow \mathcal{R} \cup \{F[i] \vdash F[j]\};$ 
      si  $\text{est\_SDRM}(\mathcal{S})$  et  $\text{est\_hybride}(\mathcal{S})$  et  $\text{est\_PNC}(\mathcal{S})$  alors
         $E_+ \leftarrow \text{normaliser}(I_+, \mathcal{S}); E_- \leftarrow \text{normaliser}(I_-, \mathcal{S});$ 
        si  $E_+ \cap E_- = \emptyset$  alors
           $\mathcal{R} \leftarrow \mathcal{S}; I_+ \leftarrow E_+; I_- \leftarrow E_-;$ 
     $e \leftarrow \min_{\preceq} I_+;$ 
  pour chaque  $w \in I_+$  faire
    si  $w \neq e$  alors  $\mathcal{R} \leftarrow \mathcal{R} \cup \{w \vdash e\};$ 
retourner  $\langle \mathcal{R}, e \rangle;$ 

```

fin

les exemples mais pourrait (1) empêcher l'inférence de règles importantes (par la contrainte PNC), et (2) être incohérente avec le langage (même si elle ne l'est pas avec l'échantillon). Ensuite, comme l'indique leur nom, les fonctions $\text{est_SDRM}()$, $\text{est_hybride}()$ et $\text{est_PNC}()$ vérifient que la règle candidate forme bien un SDRM hybride et PNC avec celles déjà inférées. Enfin, la fonction $\text{normaliser}()$ calcule les formes normales de l'échantillon d'apprentissage courant : $\text{normaliser}(X, \mathcal{S}) = \{w \downarrow_{\mathcal{S}} : w \in X\}$. Une règle est définitivement acceptée si elle ne crée pas d'incohérence : aucun mot positif ne se réécrit en un mot négatif (et vice versa).

Avant de clarifier le fonctionnement de LARS sur un exemple, nous établissons le résultat suivant :

Théorème 4.3 À partir d'un échantillon d'exemples positifs et négatifs $S = \langle S_+, S_- \rangle$ de taille m , LARS retourne, en un temps polynomial en m , un SDRM hybride et PNC \mathcal{R} et un mot irréductible e tels que $S_+ \subseteq \mathcal{L}(\mathcal{R}, e)$ et $S_- \cap \mathcal{L}(\mathcal{R}, e) = \emptyset$.

Preuve : La polynomialité et la preuve de terminaison sont immédiates (le nombre de facteurs dans F est un polynôme du nombre d'exemples positifs). Les règles telles

que $S_- \cap \mathcal{L}(\mathcal{R}, e) = \emptyset$ sont rejetées suite au processus de normalisation. À chaque tour de boucle I_+ contient les formes normales courantes des exemples positifs et la boucle finale “pour chaque” s’assure que $S_+ \subseteq \mathcal{L}(\mathcal{R}, e)$. Cette dernière boucle infère bien des règles PNCs : leurs parties gauches sont constituées de formes normales et comme elles sont marquées par un \$ et un \mathcal{L} , aucun chevauchement ne peut se produire entre elles et avec les précédentes.

□

Le tableau 4.1 représente l’exécution de LARS sur l’échantillon $S = \langle S_+, S_- \rangle$ avec $S_+ = \{\$b\mathcal{L}, \$abb\mathcal{L}, \$abaabbb\mathcal{L}, \$ababb\mathcal{L}\}$ et $S_- = \{\$\lambda\mathcal{L}, \$a\mathcal{L}, \$ab\mathcal{L}, \$aa\mathcal{L}, \$baab\mathcal{L}, \$bab\mathcal{L}, \$aab\mathcal{L}, \$abab\mathcal{L}, \$aabb\mathcal{L}\}$. Cet échantillon correspond au langage de Lukasiewicz qui est décrit par la grammaire $\langle \{a, b\}, \{S\}, \{S \rightarrow aSS; S \rightarrow b\}, S \rangle$.

Comme ce tableau n’est pas forcément facile à comprendre et, somme toute, peu informatif sur les raisons de rejet d’une règle, nous développons ci-dessous l’exemple d’exécution que le tableau résume.

La première étape consiste à construire l’ensemble trié F des facteurs des mots de S_+ .

LARS commence par chercher des règles dont la partie gauche est λ . Or aucune règle hybride ne peut être construite à partir de ce facteur. Pour la même raison, les facteurs \$ et \mathcal{L} ne donnent lieu à aucune inférence.

Le premier facteur pertinent pour être une partie gauche de règle est le facteur a . Comme $I_+ = S_+$ à ce niveau du processus, ce facteur apparaît dans I_+ . La seule règle hybride pouvant être construite est $a \vdash \lambda$, mais elle est rejetée car elle réécrit l’exemple négatif $\$ab\mathcal{L}$ en $\$b\mathcal{L}$ qui est un exemple positif (et qui par conséquent appartient, après normalisation, à E_+ et E_-).

Le facteur candidat suivant est $\$a$ et il ne peut être utilisé que dans la règle $\$a \vdash \lambda$, qui est rejetée car elle induit la même incohérence que la règle précédente.

Comme les facteurs $a\mathcal{L}$ et $\$a\mathcal{L}$ n’apparaissent pas dans l’échantillon d’apprentissage, le facteur suivant est b . La première règle candidate est $b \vdash \lambda$, qui est rejetée car elle réécrit l’exemple positif $\$b\mathcal{L}$ en $\$\mathcal{L}$, qui est un des exemples négatifs. La seconde règle est $b \vdash a$ qui est une non-\$-règle ne réduisant pas la taille, et ne satisfaisant donc pas la contrainte d’hybridité.

Les facteurs $\$b, b\mathcal{L}, \$b\mathcal{L}$ apparaissent dans I_+ mais les règles les utilisant en partie gauche induisent des incohérences avec les données (l’exemple décrit pour la règle $b \vdash \lambda$ peut être réutilisé ici).

Ensuite, LARS s’occupe des règles dont la partie gauche est aa . La règle $aa \vdash \lambda$ réduit l’exemple négatif $\$aab\mathcal{L}$ en $\$b\mathcal{L}$ qui est positif. De la même manière, la règle $aa \vdash a$ (resp. $aa \vdash b$) réécrit $\$aabb\mathcal{L}$ en $\$abb\mathcal{L}$ (resp. $\$aa\mathcal{L}$ en $\$b\mathcal{L}$) et est donc rejetées.

Le facteur suivant de F qui apparaît dans I_+ est ab . La règle $ab \vdash \lambda$ n’est pas acceptée car elle réécrit, par exemple, le négatif $\$bab\mathcal{L}$ dans le mot du langage $\$b\mathcal{L}$. Pour une raison similaire, la règle $ab \vdash a$ (qui réécrit à la fois l’exemple positif

$\mathbf{F}[i]$	apparaît	règle courante	hybride et PNC	$E_+ \cap E_- = \emptyset$	I_+	I_-
a	oui	$a \vdash \lambda$	oui	non	S_+	S_-
$\$a$	oui	$\$a \vdash \$$	oui	non		
b	oui	$b \vdash \lambda$ $b \vdash a$	oui oui	non non		
$\$b$	oui	$\$b \vdash \$$ $\$b \vdash \a	oui oui	non non		
$b\mathcal{L}$	oui	$b\mathcal{L} \vdash \mathcal{L}$ $b\mathcal{L} \vdash a\mathcal{L}$	oui oui	non non		
$\$b\mathcal{L}$	oui	$\$b\mathcal{L} \vdash \\mathcal{L} $\$b\mathcal{L} \vdash \$a\mathcal{L}$	oui oui	non non		
aa	oui	$aa \vdash \lambda$ $aa \vdash a$ $aa \vdash b$	oui oui non	non non -		
$\$aa$	oui	$\$aa \vdash \$$ $\$aa \vdash \a $\$aa \vdash \b	oui oui oui	non non non		
ab	oui	$ab \vdash \lambda$ $ab \vdash a$ $ab \vdash b$ $ab \vdash aa$	oui oui oui non	non non non -		
$\$ab$	oui	$\$ab \vdash \$$	oui	oui		
ba	non	-	-	-		
bb	oui	$bb \vdash \lambda$ $bb \vdash a$ $bb \vdash b$ $bb \vdash aa$ $bb \vdash ab$ $bb \vdash ba$	oui non oui non non non	non - non - - -		
aab	oui	$aab \vdash \lambda$ $aab \vdash a$	oui oui	non oui	$\{\$b\mathcal{L}\}$	$\{\$ \mathcal{L}, \$a\mathcal{L}, \$aa\mathcal{L}, \$ba\mathcal{L}\}$

TAB. 4.1 – Exemple d'exécution de LARS. La 1^{ière} colonne contient les facteurs des exemples, classés verticalement dans l'ordre dans lequel LARS les utilise. La 3^{ième} contient la règle en cours d'évaluation; Les 2 règles inférées sont en gras.

$\$abb\mathcal{L}$ et l'exemple négatif $\$ab\mathcal{L}$ en $\$a\mathcal{L}$) et la règle $ab \vdash b$ ($\$ab\mathcal{L}$ appartient à I_- et $\$b\mathcal{L}$ à I_+) sont rejetées par l'algorithme. $ab \vdash aa$ n'est pas une règle hybride.

Le candidat suivant est le facteur $\$ab$. La règle $\$ab \vdash \λ est acceptée car elle vérifie toutes les conditions. Après normalisation, on obtient $I_+ = \{\$b\mathcal{L}, \$aabb\mathcal{L}\}$ et $I_- = \{\$\lambda\mathcal{L}, \$a\mathcal{L}, \$aa\mathcal{L}, \$baab\mathcal{L}, \$bab\mathcal{L}, \$aab\mathcal{L}, \$aabb\mathcal{L}\}$. La règle est donc ajoutée à \mathcal{R} qui devient $\{\$ab \vdash \$\lambda\}$.

Toutes les autres règles possibles avec $\$ab$ en partie gauche sont rejetées car elles ne peuvent former un SDRM PNC avec \mathcal{R} : leur partie gauche est égale à la règle de \mathcal{R} mais pas leur partie droite.

Le facteur suivant dans F est ba mais il n'apparaît plus dans I_+ suite à la normalisation.

Le facteur bb apparaît toujours dans I_+ mais aucune règle PNC ne peut être inférée avec ce facteur en partie gauche. En effet, supposons que nous sommes en train d'étudier une règle $bb \vdash r$, pour un r quelconque. Le facteur $\$abb$ se réécrit alors en $\$ar$ et en $\$b$ avec la règle précédemment inférée. Il n'est en aucun cas possible que ces deux mots soient égaux (comme le voudrait la contrainte PNC).

Le facteur suivant est aab et il apparaît toujours dans I_+ . La règle $aab \vdash \lambda$ est rejetée car elle n'est pas cohérente, alors que la règle $aab \vdash a$ est acceptée et par conséquent ajoutée à \mathcal{R} . Le processus de normalisation aboutit à $I_+ = \{\$b\mathcal{L}\}$ et $I_- = \{\$\lambda\mathcal{L}, \$a\mathcal{L}, \$aa\mathcal{L}, \$ba\mathcal{L}\}$.

Comme il ne reste plus qu'un mot dans I_+ , LARS n'inférera plus aucune autre règle. Il rend donc le système $\langle \{\$ab \vdash \$\lambda, aab \vdash a\}, \$b\mathcal{L} \rangle$. Même si cela n'est pas forcément trivial, on peut montrer que ce système induit bien le langage cible.

4.5 Apprendre des SDRMs hybrides et PNCs

Dans cette section nous étudions les capacités d'apprentissage de LARS. Nous démontrons dans un premier temps un résultat d'identification pour une classe particulière de SDRMs hybrides et PNCs : les SDRMs clos. Dans un deuxième temps, nous montrons que l'algorithme est capable de rendre des SDRMs corrects pour des langages n'admettant pas de SDRM clos. Nous discutons ensuite la position de la classe de langages apprenables dans la hiérarchie de Chomsky.

4.5.1 Un résultat d'identification

LARS est un algorithme glouton qui infère incrémentalement un SDRM. Toutefois, il ne considère pas toutes les règles hybrides : il ne s'occupe que de celles dont les parties gauches (et droites) sont des facteurs des mots de l'échantillon d'apprentissage. On dira de telles règles qu'elles sont *applicables* au langage. De

plus, une règle n'est acceptée que si elle ne réduit aucun exemple positif en un exemple négatif (et vice versa). De telles règles sont dites *cohérentes*.

Définition 4.6 (Règle applicable et cohérente)

Soit $L \subseteq \Sigma^*$ un langage et $R = l \vdash r$ une règle de réécriture délimitée. On dit que :

- R est applicable à L s'il existe $w \in \$L\mathcal{L}$ et $u, v \in \overline{\Sigma^*}$ tel que $w = ulv$;
- R est cohérente par rapport à L si $\forall u, v \in \overline{\Sigma^*}$, $(ulv \in \$L\mathcal{L} \Leftrightarrow urv \in \$L\mathcal{L})$.

Par exemple, si $L = \{a^n b^n \mid n \geq 0\}$, $bba \vdash ba$ n'est pas applicable (aucun facteur d'un mot de L ne contient un b suivi d'un a). Cette règle est pourtant cohérente (elle ne réécrit que des mots négatifs en d'autres mots négatifs), mais il n'est pas pertinent de l'utiliser puisqu'elle ne donne aucune information sur le langage. D'autre part, $ab \vdash a$ n'est pas cohérente par rapport à L car elle réduit, par exemple, le mot $\$aabb\mathcal{L} \in \$L\mathcal{L}$ en $aab \notin \$L\mathcal{L}$. En fait une règle est cohérente par rapport à L si $\$L\mathcal{L}$ et $\Sigma^* \mathcal{L} \setminus \$L\mathcal{L}$ sont stables par réécriture avec cette règle.

Toutes les règles hybrides pouvant être utilisées pour représenter un langage L sont nécessairement cohérentes (et applicables car les règles non applicables réécrivent des mots ne faisant pas partie du langage). Mais, dans le cas général, l'ensemble de ces règles ne constitue pas un SDRM PNC : LARS ne les inférera pas toutes (et, par conséquent, peut ne pas réussir à apprendre le langage). C'est pourquoi nous introduisons la définition suivante :

Définition 4.7 (SDRM clos)

Soit $L = \mathcal{L}(\mathcal{R}, e)$ un langage et R_{max} la plus grande règle de \mathcal{R}^1 . \mathcal{R} est clos si :

- \mathcal{R} est un SDRM hybride et PNC, et
- Pour toute règle R , si :
 1. $R \prec R_{max}$ et
 2. R est applicable à L et
 3. R est cohérente par rapport à L ,
 alors R appartient à \mathcal{R} .

Un langage est clos s'il peut être représenté par un SDRM clos.

Cette propriété est probablement indécidable car tester si une règle est cohérente ne semble pas l'être. Néanmoins, elle nous permet d'obtenir le résultat suivant :

Théorème 4.4 *Étant donné un langage $L = \mathcal{L}(\mathcal{T}, e)$ tel que \mathcal{T} soit un SDRM clos, il existe un ensemble caractéristique $CS = \langle CS_+, CS_- \rangle$ tel que, à partir de $S = \langle S_+, S_- \rangle$, avec $CS_+ \subseteq S_+$ et $CS_- \subseteq S_-$, l'algorithme LARS retourne un SDRM hybride et PNC \mathcal{R} et un mot irréductible e tel que $\mathcal{L}(\mathcal{R}, e) = \mathcal{L}(\mathcal{T}, e)$.*

¹ \preceq est étendu aux paires de mots : $(u_1, u_2) \preceq (v_1, v_2)$ si $u_1 \prec v_1$ ou $(u_1 = v_1$ et $u_2 \prec v_2)$

Il est à noter que l'échantillon caractéristique peut ne pas être polynomial comme requis dans [dH97].

Pour prouver ce théorème, nous commençons par définir un échantillon caractéristique pour un langage clos :

Définition 4.8 (Échantillon caractéristique pour LARS)

Soit $L = \mathcal{L}(\mathcal{T}, e)$ le langage cible. \mathcal{T} est un SDRM clos et R_{max} est sa plus grande règle par rapport à \preceq . On définit l'ensemble caractéristique $CS = \langle CS_+, CS_- \rangle$ ainsi :

1. $\$e\mathcal{L} \in CS_+$.
2. Pour chaque règle $R = l \vdash r \in \mathcal{T}$, il existe deux mots $ulv, u'rv' \in \$L\mathcal{L} \cap CS_+$ pour des $u, v, u', v' \in \overline{\Sigma^*}$.
3. Pour chaque règle hybride $R = l \vdash r$ telle que $R \prec R_{max}$ et $R \notin \mathcal{T}$ (R n'est donc pas cohérente puisque \mathcal{T} est clos), il existe $u, v \in \overline{\Sigma^*}$ tels que :
 - $ulv \in (\$ \Sigma^* \mathcal{L} \setminus \$L\mathcal{L}) \cap CS_-$ et $urv \in \$L\mathcal{L} \cap CS_+$, ou
 - $urv \in (\$ \Sigma^* \mathcal{L} \setminus \$L\mathcal{L}) \cap CS_-$ et $ulv \in \$L\mathcal{L} \cap CS_+$.
4. Pour chaque sous-système \mathcal{R} de \mathcal{T} (i.e. $\mathcal{R} \subseteq \mathcal{T}$) et pour chaque règle $R = l \vdash r$ tels que $\mathcal{L}(\mathcal{R}, e) \neq \mathcal{L}(\mathcal{T}, e)$ et $\mathcal{L}(\mathcal{R} \cup \{R\}, e) = \mathcal{L}(\mathcal{T}, e)$, il existe $w \in \$L\mathcal{L} \setminus \$\mathcal{L}(\mathcal{R}, e)\mathcal{L}$ tel que $w = ulv$ (car R est applicable), w est en forme normale par rapport à \mathcal{R} et $w \in CS_+$.

Avant de donner la preuve du théorème 4.4, nous aimerions discuter informellement de la définition 4.8. Les deux premiers points permettent de s'assurer que les données d'apprentissage contiennent tous les éléments nécessaires à l'inférence : le plus petit mot du langage, ainsi que toutes les parties gauches et droites des règles de la cible apparaissent dans l'ensemble des exemples positifs. Comme l'algorithme ne s'occupe que des règles qu'il peut former à partir de facteurs d'exemples positifs, ces conditions sont nécessaires à l'identification. Elles correspondent à celles requises dans la grande majorité des algorithmes d'inférence grammaticale, quand le cadre d'apprentissage est l'identification à la limite (voir par exemple [OG92]).

Le troisième point concerne les règles hybrides qui ne sont pas cohérentes par rapport au langage. Pour rejeter une telle règle $l \vdash r$, il est nécessaire d'avoir deux mots ulv et urv dans l'ensemble d'apprentissage, l'un positif, l'autre négatif. Même si la cohérence est une relation d'équivalence, il est suffisant, pour rejeter une telle règle, de posséder dans l'échantillon un exemple positif qui se réécrit en un négatif, ou un négatif qui se réécrit en un positif. Le problème du premier cas est qu'il peut avoir comme conséquence une augmentation inutile de la taille de l'ensemble F . Mais il n'est pas toujours possible de trouver un exemple négatif se réécrivant en un positif. Comme de toute façon au moins un

des deux cas se produit (la règle n'est pas cohérente), cela nous permet d'assurer que l'échantillon caractéristique contient l'information permettant de rejeter les règles non cohérentes.

Le dernier point est plus technique : son but est d'empêcher que des facteurs importants disparaissent : si une règle est telle que sans elle il n'est pas possible d'induire le langage cible, il faut que son facteur gauche apparaisse encore dans I_+ au moment où LARS va s'en occuper.

Il est important de noter que la définition précédente (ainsi que la preuve suivante) montre l'existence d'un échantillon caractéristique et non sa construction : le dernier point requiert la comparaison entre les langages engendrés par deux systèmes de réécriture ce qui est, à notre connaissance, non décidable. Toutefois, ceci n'est pas un problème d'un point de vue théorique. En effet, il a été montré dans [dlH97] que l'identification à la limite en temps et données polynomiaux (voir la définition 1.10) est équivalente à la "semi-poly enseignabilité", comme défini par Goldman et Mathias [GM96]. Cette dernière notion nécessite l'existence d'un échantillon caractéristique, mais pas sa constructibilité.

Preuve : Soit $L = \mathcal{L}(\mathcal{T}, e)$ le langage cible. \mathcal{T} est un SDRM clos et R_{max} est sa plus grande règle par rapport à \preceq . Soit $CS = \langle CS_+, CS_- \rangle$ l'ensemble caractéristique de \mathcal{T} .

Il nous faut prouver qu'à partir d'un échantillon d'apprentissage $S = \langle S_+, S_- \rangle$ tel que $S_+ \supseteq CS_+$ et $S_- \supseteq CS_-$, Lars retourne un système équivalent à \mathcal{T} . Par construction de l'ensemble caractéristique, F contient toutes les parties gauches et parties droites des règles de la cible (Cas 2 de la définition 4.8). Nous supposons maintenant que Lars a été exécuté pendant un certain temps. Soit \mathcal{R} le SDRM hybride et PNC courant. Comme \mathcal{T} est clos, $\mathcal{R} \subseteq \mathcal{T}$.

Soit $R = l \vdash r$ la prochaine règle à être testée (c'est-à-dire l apparaît toujours dans I_+). Nous supposons que $\mathcal{R} \cup \{R\}$ est un SDRM hybride et PNC. Dans le cas contraire, l'algorithme rejette la règle, ce qui n'est pas un problème puisque alors $\mathcal{T} \cup \{R\}$ n'est pas PNC et donc $R \notin \mathcal{T}$ (car \mathcal{T} est clos). Deux cas peuvent se produire :

1. Si R est incohérente, il existe $m \in (\Sigma^* \setminus L) \cap S_-$ et $m' \in L \cap S_+$ tels que $m \vdash_{\{R\}} m'$ (Cas 3 de la définition 4.8). Par construction de I_+ et I_- , il existe $u = m \downarrow \in I_-$ et $u' = m' \downarrow \in I_+$. Comme $\mathcal{R} \cup \{R\}$ est PNC, il est aussi confluent ce qui implique qu'il existe un mot z tel que $u \vdash_{\mathcal{R} \cup \{R\}} z$ et $u' \vdash_{\mathcal{R} \cup \{R\}} z$. Donc $z \in E_+ \cap E_-$ et la règle est rejetée.
2. Si R est cohérente alors considérons le système $\mathcal{S} = \mathcal{R} \cup \{T \in \mathcal{T} \mid R \prec T\}$. \mathcal{S} est le système formé des règles inférées jusqu'à présent par Lars et des règles de la cible qui sont plus grandes que la règle courante (donc les règles applicables et cohérentes qui n'ont pas encore été traitées par l'algorithme). Deux sous-cas

peuvent se produire :

- (a) $\mathcal{L}(\mathcal{S}, e) = L$ et alors R n'est pas nécessaire pour induire L . Lars infère cette règle car il n'y a aucun moyen de faire autrement (mais il aurait pu ne pas la garder et ça n'aurait eu aucune conséquence).
- (b) $\mathcal{L}(\mathcal{S}, e) \neq L$ et alors $\mathcal{L}(\mathcal{S} \cup \{R\}, e) = L$. Par le Cas 4 de la définition 4.8, il existe un mot $w \in L \setminus \mathcal{L}(\mathcal{S}, e)$ dans S_+ , en forme normale par rapport à \mathcal{S} . Comme $\mathcal{R} \subseteq \mathcal{S}$, il est clair que $w \in I_+$. Comme w se réécrit avec la règle R , R sera bien détectée et inférée par Lars.

À la fin de l'exécution de Lars, le SDRM retourné contient toutes les règles de \mathcal{T} à l'exception de celles dont la partie gauche n'apparaissait plus dans I_+ au moment où Lars les a considérées. Néanmoins, ces règles n'étaient pas nécessaires pour identifier L car sinon le Cas 4 de la définition de l'ensemble caractéristique aurait permis leur inférence. Pour finir, notons que e est l'unique élément de I_+ à la fin de l'exécution et que $\mathcal{L}(\mathcal{R}, e) = L$. \square

4.5.2 Sur la classe apprise et sa position dans la hiérarchie de Chomsky

Dans ce qui suit, nous précisons la position de la classe de langages apprenable par LARS dans la hiérarchie de Chomsky. La difficulté principale est que cette hiérarchie (ainsi que les sous classes étudiées dans la littérature) est définie par des contraintes syntaxiques sur les règles de production d'une grammaire. Or le lien entre grammaires et systèmes de réécriture de mots est difficile à établir. C'est même le problème principal qui se pose lorsque l'on change la représentation des langages : il est difficile de comparer les résultats ainsi obtenus avec ceux déjà mis à jour. Toutefois, à l'aide de contre-exemples, il est possible de déterminer les contours de la classe de langages apprise.

Les résultats de cette section sont résumés par la figure 4.4.

Nous avons obtenu un résultat d'identification pour la classe des langages clos. Nous verrons que LARS est aussi capable d'apprendre des langages qui ne sont pas clos.

Bien que savoir si un langage peut être induit par un SDRM clos soit indécidable, il n'est pas toujours difficile de le vérifier à la main sur des exemples. Ainsi, il est facile de montrer que $\langle \{ab \vdash \lambda, ba \vdash \lambda\}, \lambda \rangle$, qui induit le langage $\{w \in \{a, b\}^* : |w|_a = |w|_b\}$, est un SDRM clos. En fait, la plupart des langages hors-contextes étudiés dans la littérature admettent un SDRM clos, comme nous le verrons dans la section suivante. Cette propriété ne semble donc pas trop restrictive.

D'autre part, certains langages non-clos sont inférables par LARS : par exemple, le langage régulier $L_0 = b^*(ab)^+a^*$ est induit par le SDRM hybride et PNC

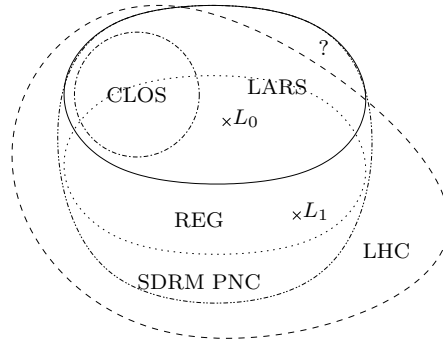


FIG. 4.4 – LARS et la hiérarchie de Chomsky. LARS dénote la classe de langages apprenable par l’algorithme LARS et SDRM PNC la classe de langages représentables à l’aide de SDRMs hybrides et PNCs.

$\langle \mathcal{R}, ab \rangle$ avec $\mathcal{R} = \{ \$b \vdash \$, a\mathcal{L} \vdash \mathcal{L}, abab \vdash ab \}$. $R = aa\mathcal{L} \vdash \mathcal{L}$ est cohérente et applicable à L_0 et $R \prec (abab \vdash ab)$, mais $\mathcal{R} \cup \{R\}$ n’est pas PNC. \mathcal{R} n’est donc pas clos, pourtant LARS est capable de l’inférer ! Pour mieux comprendre ce qui se passe, il faut noter que tous les SDRMs représentant L_0 ont besoin d’au moins une règle traitant les facteurs ab (donc d’une règle dont la partie gauche est au moins aussi grande que ab). Par conséquent les deux règles R et $a\mathcal{L} \vdash \mathcal{L}$ seront toujours plus petites que R_{max} . Or ces deux règles, bien que cohérentes et applicables, ne sont pas PNCs. Ce qui implique que L_0 n’est pas un langage clos, même si LARS est capable de l’apprendre. En fait, R est inutile pour représenter le langage : elle efface les paires de a à la fin des mots, alors que $a\mathcal{L} \vdash \mathcal{L}$ efface déjà tous les suffixes composés uniquement de a . Cet exemple montre que les langages clos sont strictement contenus dans la classe des langages apprenables par LARS.

Ensuite, s’il est pertinent de penser que les langages apprenables par LARS sont tous hors-contextes (les structures purement sensibles aux contextes ne semblent pas être représentables par des SDRMs [MNO88]), les langages réguliers ne sont malheureusement pas tous apprenables par l’algorithme (et ce même si ces langages sont représentables par des SDRM hybrides et PNCs, comme le prouve le théorème 4.2). Par exemple, considérons le langage $L_1 = \{w \in \{a,b\}^* \mid |w|_a \bmod 3 = |w|_b \bmod 3\}$ (voir figure 4.5). L_1 est induit par $e = \lambda$ et, par exemple, l’un des SDRMs hybrides suivants :

$$\begin{aligned} \mathcal{R}_1 &= \{aa \vdash b; ab \vdash \lambda; ba \vdash \lambda; bb \vdash a\} \\ \mathcal{R}_2 &= \{\$aa \vdash \$b; \$ab \vdash \$; \$ba \vdash \$; \$bb \vdash \$a\} \end{aligned}$$

\mathcal{R}_2 est PNC alors que \mathcal{R}_1 est confluent mais pas PNC : la règle $aa \vdash b$ n’est pas PNC avec elle-même. Cette règle étant la plus petite cohérente et applicable à

L_1 , ce langage ne peut être clos. Sur un ensemble d'apprentissage suffisamment grand pour rejeter toutes les règles non cohérentes, l'algorithme commence par inférer le système suivant :

$$\mathcal{R} = \{\$aa \vdash \$b; \$ab \vdash \$; ba \vdash \lambda\}.$$

\mathcal{R} est un SDRM hybride et PNC, mais il n'engendre pas L , en particulier parce que beaucoup de facteurs bb apparaissent toujours dans l'échantillon courant. Or aucune règle ne peut réduire ces facteurs et être PNC avec les trois premières règles. Ce qui implique qu'aucun sur-système de \mathcal{R} ne peut induire L_1 . Comme le SDRM \mathcal{R} sera *toujours* inféré par LARS si les données sont en nombre suffisant pour rejeter les règles incohérentes, L est un langage régulier qui ne peut pas être appris par LARS.

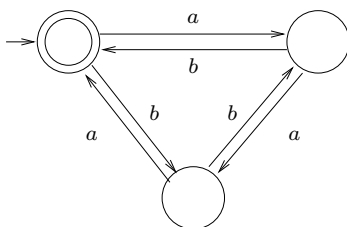


FIG. 4.5 – L'automate fini déterministe minimal reconnaissant $L_1 = \{w \in \{a,b\}^* : |w|_a \bmod 3 = |w|_b \bmod 3\}$.

4.6 Résultats expérimentaux

Nous avons donné dans la section précédente les contours de la classe de langages apprenable par l'algorithme LARS, en montrant qu'elle tenait une place orthogonale dans la hiérarchie de Chomsky. Même si nous n'avons pas pu le démontrer formellement, notre intuition est que cette classe n'est pas comparable avec celles déjà étudiées, comme les grammaires algébriques linéaires ou déterministes. Devant cette difficulté d'appréhension des capacités d'apprenant de LARS il nous a semblé utile de détailler ici quelques expériences menées, dans le but avoué de montrer que la classe est suffisamment large pour être intéressante.

Conscients que les expérimentations développées concernent des langages jouets, loin d'une éventuelle application concrète, nous commençons dans un premier temps par définir le rôle des expérimentations en inférence grammaticale. Puis, dans un deuxième temps, nous donnerons des exemples de langages que LARS est capable d'apprendre, en détaillant les SDRMs qu'il retourne et en donnant des indications sur l'ensemble d'apprentissage.

4.6.1 Expérimentations en inférence grammaticale

Les expérimentations en inférence grammaticale peuvent être divisées en trois grandes familles :

- Les compétitions à grande échelle sur des jeux de données largement distribués;
- L’apprentissage sur des données du monde réel;
- L’identification de langages sur des données synthétiques correspondant à des exemples “jouets”.

Le premier point correspond à des compétitions comme ABBADINGO [LPP98] ou OMPHALOS [SCvZ04]. Dans de tels cas, l’échantillon d’apprentissage, mais aussi la taille de l’alphabet ou des grammaires, correspondent aux limites de l’état de l’art. Dans le cas de ABBADINGO, un tiers des problèmes n’ont toujours pas été résolus, sept ans après leur introduction ! Pour prétendre résoudre de tels challenges, il est nécessaire de mettre en place des heuristiques puissantes, bien souvent liées à la manière dont les données ont été générées (voir par exemple l’article [Cla06a] sur comment a été gagné la compétition OMPHALOS). Cette approche oblige les chercheurs à pousser leurs algorithmes au maximum de leurs possibilités afin d’obtenir des taux de classification optimaux. Ce faisant, de nombreuses problématiques primordiales sont abandonnées : la correction et la convergence des algorithmes ne sont pas étudiées; le résultat final est bien souvent une “machine à gaz” complètement inintelligible, y compris pour ses géniteurs qui bien souvent ne peuvent pas expliquer pourquoi leur approche marche; etc.

En ce qui concerne le second point, la situation n’a pas évolué depuis les années soixante-dix : s’il n’existe pas de langage cible à apprendre et/ou si les données contiennent des erreurs (ce que l’on appelle du bruit), le problème n’est pour l’instant pas résolu par des méthodes d’inférence grammaticale déterministe. Une approche stochastique est mieux adaptée à ce type de problèmes, mais de nombreuses nouvelles difficultés se posent quand le but est d’apprendre une distribution au lieu d’un langage (par exemple, en plus de la structuration et du contenu du langage, il faut aussi apprendre une loi de probabilité sur les mots de ce langage).

Enfin, lorsqu’une nouvelle idée algorithmique est présentée, il est raisonnable de montrer sa pertinence par des expérimentations. Dans le cas de l’identification à la limite à l’aide d’un échantillon caractéristique, le résultat repose sur la présence de certaines informations dans les données. C’est pourquoi la plupart des nouveaux algorithmes sont présentés accompagnés d’expériences sur des petits langages, à l’aide de jeux de données facilement intelligibles : le but est de montrer que les contraintes sur l’échantillon d’apprentissage ne sont pas déraisonnables. Dans ce cas, les expériences n’ont donc pas le même objectif : elles permettent d’appréhender ce qui se passe et non pas de montrer que l’algorithme

a de bonnes performances sur des ensembles de données générés aléatoirement.

C'est dans ce dernier cas que nous avons choisi de tester LARS. Nous avons par conséquent fait des expériences sur des exemples jouets mais qui sont considérés comme difficilement apprenables par plusieurs auteurs (voir par exemple [SK99] et [NM05]).

Nous avons quand même testé notre algorithme sur les jeux de données de la compétition d'apprentissage de langages hors-contextes OMPHALOS et les résultats ont été décevants. Nous voyons deux explications principales à cela : la première est que LARS est un algorithme glouton qui requiert certaines caractéristiques à l'échantillon d'apprentissage; la seconde est qu'il n'est pas sûr du tout que les langages de cette compétition admettent un SDRM hybride et PNC. Concernant le premier point, l'attitude globale de l'algorithme permet de prouver sa pertinence : sa décision de garder une règle repose sur le fait qu'*il n'y a pas de raison de la rejeter*. Un point de vue pragmatique, mieux approprié à ce genre de problèmes, serait d'inférer une règle car *cela semble être une idée pertinente*. Pour faire un parallèle avec les langages réguliers, l'algorithme EDMSM ayant gagné la compétition ABBADINGO se base sur l'algorithme RPNI pour lequel un résultat d'identification a été prouvé. Mais ce deuxième algorithme repose sur une approche identique à celle de LARS en ce qui concerne son processus de généralisation, ce qui n'en faisait pas un bon candidat pour la compétition. EDMSM a repris les principes de RPNI en introduisant une heuristique permettant de choisir en priorité la généralisation la plus pertinente à chaque étape : au lieu de se fixer un ordre sur ces généralisations et de les accepter au fur et à mesure si rien ne les contredisait, EDMSM effectue en parallèle toutes les généralisations possibles, puis affecte à chacune un score, permettant ainsi de ne garder que celle qui semble la plus appropriée. Pour en revenir à LARS, cela reviendrait à affecter un score à toutes les règles qu'il est possible d'inférer, de sélectionner la première cohérente, puis de recommencer le processus jusqu'à obtenir un système permettant d'induire l'ensemble des exemples positifs. Il serait intéressant de développer une telle approche, bien souvent appelée "méthode dirigée par les données", même si le type et la portée des résultats ainsi obtenus seraient radicalement différents.

Pour fixer les idées, nous développons, à titre d'exemple, le comportement de LARS sur le premier problème de la compétition OMPHALOS (composé de 266 exemples positifs et de 535 exemples négatifs). L'alphabet est $\Sigma_1 = \{a,b,c,d,e,f\}$. L'algorithme commence par inférer les règles triviales suivantes : $\mathcal{R}_{current} = \{\$b \vdash \$, c \vdash \lambda, f \vdash \lambda, be \vdash e, db \vdash d, dde \vdash ed, bbde \vdash bde\}$. La deuxième et la troisième règles effacent complètement deux lettres de l'alphabet. Les autres réduisent, entre autre, considérablement le nombre d'occurrences de la lettre b . Nous ne connaissons pas le langage cible, mais il ne semble pas pertinent de diviser par deux la taille de l'alphabet pour apprendre un langage. Mais comme aucun exemple négatif ne permet de rejeter ces règles probablement incohérentes

par rapport au langage, l'algorithme les accepte. Une des conséquences est que l'ensemble courant d'apprentissage I_+ contient 215 exemples positifs trop proches des exemples négatifs. Ainsi, aucune autre règle intéressante ne peut être inférée : l'algorithme ajoute 214 règles de la forme $\$w\mathcal{L} \vdash \$e\mathcal{L}$, réécrivant chaque exemple positif restant w dans le plus petit e .

4.6.2 Comportement de LARS sur des petits langages

Dans cette section, nous présentons des langages spécifiques pour lesquels des SDRMs existent et sur lesquels LARS a été testé. Dans chaque cas nous décrivons la tâche et la taille de l'échantillon d'apprentissage sur lequel l'algorithme a été utilisé. Aucun temps d'exécution n'est donné puisqu'il n'excède jamais la seconde : les échantillons et les systèmes sont tous très petits.

Le langage de Dyck.

Le langage de parenthèses est un classique de la théorie des langages. Il peut être représenté par le système $\langle \{ab \vdash \lambda\}, \lambda \rangle$ ou par la grammaire algébrique $\langle \{a,b\}, \{S\}, P, S \rangle$ où $P = \{S \rightarrow aSbS; S \rightarrow \lambda\}$. Ce langage est appris dans [SK99] à partir de l'ensemble des exemples positifs de taille inférieure à 10 et de tous les exemples négatifs de taille inférieure à 20. Dans [NM05], les auteurs parviennent à l'apprendre à partir de l'ensemble des exemples et contre-exemples d'une certaine taille, typiquement de cinq à sept. L'algorithme LARS infère le système correct mais à partir d'un échantillon d'apprentissage bien plus petit (environ une vingtaine d'éléments). D'autre part, [PPK⁺04] ont testé, sans succès, leur système E-GRIDS sur ce langage, mais avec des exemples positifs seulement. Notre algorithme a aussi été capable d'apprendre ce langage avec un nombre de paires de parenthèses plus important, à l'aide d'échantillons de taille raisonnable.

Le langage $\{a^n b^n : n \in \mathbb{N}\}$.

Le langage $\{a^n b^n : n \in \mathbb{N}\}$ est souvent utilisé comme exemple de langage hors-contexte non régulier. Le SDRM correspondant est $\langle \{aabb \vdash ab; \$ab\mathcal{L} \vdash \$\lambda\mathcal{L}\}, \lambda \rangle$. Des variantes de ce langage sont, par exemple, $\{a^m b^n c^m : m, n \in \mathbb{N}\}$, étudiées dans [SK99], ou $\{a^m b^n : 1 \leq m \leq n\}$ [NM05]. Dans tous les cas, LARS a appris un système correct à partir d'un échantillon de moins de 20 exemples, ce qui est beaucoup moins que ce que nécessitent les autres méthodes.

Les langages réguliers.

Nous avons testé notre algorithme sur les jeux de données disponibles pour les langages réguliers. Ceux de la compétition ABBADINGO sont trop difficiles pour

LARS (les raisons sont les mêmes que celles développées à la fin de la section 4.6.1). Par contre, les résultats ont été meilleurs quand nous nous sommes intéressés à des tâches d'inférence plus anciennes [Dup94]. Ces jeux de données correspondent à de plus petits automates, c'est-à-dire à des systèmes de réécriture contenant une à six règles. Sur la plupart des échantillons, l'algorithme a retourné un système correct, mais quand ce ne fut pas le cas, le langage inféré avait peu à voir avec la cible.

D'autres langages et propriétés

Le langage $\{a^p b^q : p \geq 0, q \geq 0, p \neq q\}$ n'est pas apprenable avec les principaux algorithmes de l'état de l'art. Pourtant, LARS retourne le système correct $\langle \{\$b\mathcal{L} \vdash \$a\mathcal{L}; aa\mathcal{L} \vdash a\mathcal{L}; \$bb \vdash \$b; aab\mathcal{L} \vdash a\mathcal{L}; \$abb \vdash \$b; aabb \vdash ab\}, a \rangle$ à partir d'un échantillon d'apprentissage de moins de 20 éléments. Il est à noter que ce langage n'est pas représentable par un SRM sans délimiteur.

Les langages $\{w \in \{a,b\}^* : |w|_a = |w|_b\}$ et $\{w \in \{a,b\}^* : 2|w|_a = |w|_b\}$ sont utilisés dans [NM05]. Dans les deux cas, notre algorithme est capable d'inférer un SDRM les représentant à l'aide de moins de trente exemples.

Le langage de Lukasiewicz est généré par la grammaire $\langle \{a,b\}, \{S\}, P, S \rangle$ où $P = \{S \rightarrow aSS; S \rightarrow b\}$. Il correspond à l'ensemble des mots contenant un b de plus que de a mais dont tous les préfixes stricts contiennent au moins autant de a que de b . Nous nous attendions à inférer le système $\langle \{abb \vdash b\}, b \rangle$, traduction directe des règles de la grammaire, mais l'algorithme retourne $\langle \{\$ab \vdash \$\lambda; aab \vdash a\}, b \rangle$, qui est correct (et même clos).

Le langage $\{a^m b^m c^n d^n : m, n \geq 0\}$ n'est pas linéaire (tout comme les langages de Dyck et de Lukasiewicz), et est reconnu par le système $\langle \{aabb \vdash ab; ccdd \vdash cd, \$abcd\mathcal{L} \vdash \$\mathcal{L}, \$ab\mathcal{L} \vdash \$\mathcal{L}, \$cd\mathcal{L} \vdash \$\mathcal{L}\}, \lambda \rangle$ que LARS apprend à partir d'un échantillon de taille inférieure à 20.

Par contre, le langage des palindromes ($\{w : w = w^R\}$) ne peut être représenté par un SDRM, à moins d'utiliser un caractère spécial pour marquer le milieu des mots (dans ce cas, une dizaine de données d'apprentissage suffisent à LARS pour l'apprendre).

Enfin, le système $\langle \{ab^k \vdash b\}, b \rangle$, correspondant au langage de Lukasiewicz étendu, nécessite un échantillon d'apprentissage de taille exponentielle.

4.7 Conclusion et perspectives

Dans ce travail, nous nous sommes attaqué à l'apprentissage de langages à l'aide de systèmes de réécriture. Nous avons défini les systèmes délimités de réécriture de mots hybrides et presque non-chevauchants et montré qu'ils consti-

tuaiement une classe intéressante de représentations de langages. Puis nous avons proposé un algorithme glouton permettant un résultat d'identification sur une sous-classe de ces systèmes. Ce résultat est incomplet puisque la polynomialité de l'ensemble caractéristique n'est pas assurée. Toutefois, nous avons déjà eu l'occasion de mettre en doute la pertinence de cette contrainte (voir dans le chapitre 2 les sections "expansivité" et "linéarité"). Enfin, nous avons étudié la place de la classe apprise dans la hiérarchie de Chomsky et montré sur des exemples que le nombre de données nécessaires à l'apprentissage était inférieur à celui des autres algorithmes capables de les apprendre.

La simplicité relative de notre approche, si elle permet d'obtenir des résultats positifs, possède ses limites : nous n'avons pu apprendre, par exemple, aucun des difficiles problèmes de la compétition OMPHALOS. Il est certainement possible d'améliorer les performances de notre algorithme sur ce type de problèmes en essayant de réduire l'espace de ses hypothèses, en se restreignant, par exemple, aux systèmes de réécriture monadiques [BO93] ou basiques [Sén98]. Il est aussi possible d'utiliser d'autres types de systèmes de réécriture, comme par exemple les langages CR [MNO88].

La principale difficulté rencontrée, tant pour les expérimentations que pour le théorème d'identification, est liée à la propriété PNC. D'un point de vue empirique, l'inférence d'une mauvaise règle peut, à cause de cette contrainte, empêcher la découverte d'une règle nécessaire pour représenter le langage. D'autre part, la restriction à la classe des SDRMs clos pour le résultat d'apprentissage est aussi une conséquence directe de cette contrainte (voir, par exemple, les langages non-clos que l'algorithme arrive à apprendre).

C'est fort de ces constatations que nous avons commencé un travail où la contrainte PNC n'est plus requise. Pour s'assurer qu'un mot ne peut se réécrire qu'en un seul mot irréductible, nous avons choisi de mettre en place une *stratégie de réécriture*. Au moment où sont écrites ces lignes, il est trop tôt pour déterminer si cette nouvelle approche permettra d'obtenir de meilleurs résultats.

5 *Restriction à une sous-classe*

5.1 Motivation

Nous avons vu dans la première partie de ce document qu’une alternative pour résoudre les problèmes théoriques sur l’apprentissage des langages hors-contextes est de se concentrer sur des sous-classes de ces langages. Cette approche se justifie d’autant plus lorsque nous ne disposons que d’exemples de mots du langage pour apprendre : aucune classe contenant tous les langages finis et au moins un langage infini ne peut être identifiée à la limite à partir d’exemples positifs seulement, et ce même si on ne tient compte d’aucune contrainte de complexité [Gol67] (voir le théorème 2.1)! Pourtant, dans nombre de domaines, il n’est pas possible de disposer d’autres informations, comme des exemples négatifs ou une structuration des exemples positifs : par exemple, qu’est-ce qu’un exemple négatif de gène ? Une séquence d’ADN non codante est-elle réellement informative de ce qu’est un gène ? Cette problématique de l’apprentissage à partir de données positives seulement est une pierre angulaire du domaine des langues naturelles : lorsqu’un enfant acquiert sa langue maternelle, il ne dispose que d’exemples d’éléments du langage (personne ne dit à son enfant : ”ce que je vais dire n’est pas français, écoute bien, ça va t’aider”). Pourtant, un enfant apprend très rapidement à parler, au regard de la difficulté de cet apprentissage : au bout d’un an il est capable de comprendre des phrases simples, et il commence à parler en même temps qu’il fait ses premiers pas ! Nous reviendrons à la fin de ce chapitre sur cette problématique de la modélisation de l’acquisition d’une langue par un enfant.

Pour en revenir à l’inférence grammaticale, la principale sous-classe des langages algébriques identifiable polynomialement à la limite à partir d’exemples positifs seulement est celle, déjà introduite dans la section 2.2.2, des langages *très simples* [Yok03]. La définition de cette classe repose, comme bien souvent, sur des contraintes sur les règles des grammaires. Notre approche est différente : nous définissons des contraintes sur les langages eux-mêmes, indépendamment de leur représentation. Ce travail repose sur une des idées les plus anciennes en inférence grammaticale : la *substituabilité*, introduite par Harris [Har54]. Ce travail a donné naissance à la linguistique distributionnelle. Toutefois, même si de nom-

breux algorithmes ont utilisé implicitement cette notion, elle n'a jamais donné lieu à une réelle formalisation. Dans ce chapitre, nous proposons une formalisation explicite de cette notion, ce qui nous permet de définir une classe particulière de langages, les *langages substituables*, pour lesquels nous donnons des algorithmes capables de les identifier à la limite en temps et données polynomiaux.

La clef pour comprendre l'idée de Harris, est de s'intéresser aux paires de mots (u, v) et de tester s'ils apparaissent dans le même contexte dans un langage cible L , c'est-à-dire si il existe l, r tels que lur et lvr appartiennent tous deux au langage. On peut alors conjecturer sans prendre trop de risques que les deux facteurs u et v ont été générés par le même non terminal (si notre but est d'apprendre des grammaires). La description informelle qui précède porte en elle une ambiguïté : les mots u et v doivent-ils apparaître toujours dans les mêmes contextes ? Ou partagent-ils seulement quelques contextes ? On peut écrire le premier critère comme suit¹ :

$$\forall l, r \text{ } lur \in L \text{ si et seulement si } lvr \in L.$$

Le second critère, plus faible, s'écrit :

$$\exists l, r \text{ } lur \in L \text{ et } lvr \in L.$$

Le problème est que pour tirer des conclusions sur la structure du langage, nous avons besoin de la première propriété, mais tout ce que l'on peut vérifier sur un échantillon de données est la seconde propriété. En effet, si le langage est infini, on ne peut tester le premier critère à partir d'un échantillon infini. Notre idée est de se restreindre à la classe de langages pour laquelle le second critère implique le premier : si deux facteurs de mots du langage apparaissent au moins une fois dans le même contexte, alors ils apparaîtront toujours dans les mêmes contextes. Nous appelons les langages répondant à cette propriété les *langages substituables*. Cette contrainte est assez naïve et peu sembler trop restrictive, mais nous verrons dans la suite que nombre de langages hors-contextes la satisfont.

Dans la section suivante nous définirons formellement la classe de langages qui nous intéresse. Puis nous expliciterons un principe algorithmique général pour l'apprendre et donnerons deux algorithmes l'utilisant, selon que l'on représente les langages par des grammaires ou par des systèmes de réécriture. Dans les deux cas, un résultat d'identification sera prouvé. Ensuite nous étudierons les caractéristiques de ces langages et leurs liens avec d'autres classes étudiées dans la littérature. Enfin, avant de conclure, nous verrons comment cette approche permet de répondre à une interrogation récurrente dans le domaine de l'acquisition de la langue par les enfants.

¹ Nous définirons plus formellement nos notations dans la suite. Nous faisons appel pour le moment à l'intuition du lecteur.

5.2 Substituabilité: définitions et exemples

Nous commençons par introduire nos deux notions de substituabilité. La première, qui correspond à la notion de substituabilité forte, est usuellement utilisée en théorie des langages.

Définition 5.1 (Congruence syntaxique)

Deux mots u et v sont syntaxiquement congruents par rapport à un langage L , noté $u \equiv_L v$, si pour tout mot l et r de Σ^* , $lur \in L \iff lvr \in L$.

Cela définit une notion de substituabilité forte dans le sens où si $u \equiv_L v$, alors dans un mot de Σ^* substituer un facteur u par v (ou l'inverse) n'affecte pas l'appartenance (ou la non appartenance) du mot au langage L . Cette relation d'équivalence \equiv_L est une congruence sur le monoïde Σ^* car il est trivial que :

$$u \equiv_L v \text{ implique que pour tout } l, r \in \Sigma^* \text{ } lur \equiv_L lvr.$$

Le monoïde syntaxique d'un langage L est défini comme le quotient de Σ^* par les classes de congruence définies par \equiv_L . Le résultat suivant est bien connu :

Propriété 5.1 *Un langage L est régulier si et seulement si son nombre de classes de congruence syntaxique est fini.*

Une autre façon de définir cette propriété est de s'intéresser à l'ensemble des contextes d'un mot dans un langage. On rappelle que cet ensemble est défini comme suit : $C_L(u) = \{(l, r) \in \Sigma^* : lur \in L\}$. On a alors

$$\text{Pour tout } u, v \text{ dans } \Sigma^*, u \equiv_L v \text{ si et seulement si } C_L(u) = C_L(v).$$

Comme cette notion de substituabilité n'est pas vérifiable sur un échantillon d'apprentissage, nous introduisons une notion plus faible :

Définition 5.2 (Substituabilité faible)

Étant donné un langage L , on dit que deux mots u et v sont faiblement substituables par rapport à L , noté $u \doteq_L v$, s'il existe l et r dans Σ^* tels que $lur \in L$ et $lvr \in L$.

Une autre façon de voir cette propriété est de dire que deux mots u et v sont faiblement substituables si l'intersection de leur ensemble de contextes est non vide, c'est-à-dire $C_L(u) \cap C_L(v) \neq \emptyset$. Cette relation ne définit pas en général une congruence, ni même une relation transitive.

Concernant l'apprentissage, notre connaissance d'un langage cible L est contenue dans un échantillon d'exemples positifs $S \subseteq L$. Il est clair que $u \doteq_S v$ implique $u \doteq_L v$ (car $S \subseteq L$). C'est pourquoi nous confondons désormais \doteq_S et \doteq_L .

Nous pouvons maintenant définir la classe de langages qui nous intéresse.

Définition 5.3 (Langage substituable)

Un langage L sur un alphabet Σ est substituable si pour chaque paire de mots $u, v \in \Sigma^*$, $u \doteq_L v$ implique $u \equiv_L v$.

En terme de contextes, un langage est substituable si et seulement si dès que les ensembles de contextes de deux mots ne sont pas disjoints, ils sont identiques. La classe, qui nous intéresse, des *langages hors-contextes substituables* est simplement la classe contenant les langages à la fois hors-contextes et substituables. Des exemples de tels langages sont développés dans la section 5.4.1.

Un premier problème se pose que nous choisissons de contourner de suite : le mot vide λ apparaît dans tous les contextes possibles, ce qui réduit considérablement la classe. Nous décidons de ne pas considérer le mot vide, ce qui implique que les mots u et v des définitions précédentes ne font pas partie de Σ^* mais de Σ^+ . Une autre conséquence est que les langages ainsi définis ne peuvent contenir le mot vide.

Exemple 5.1 *Savoir si un langage donné est substituable ne semble pas être décidable mais ce peut être vérifié sur des langages particuliers. Par exemple, Σ^* est un langage substituable, ainsi que $\{wcv^R \mid w \in \Sigma^*\}$ (le langage des palindromes avec un centre marqué) et $\{a^n cb^n \mid n \geq 0\}$. Par contre, $L = \{a^n b^n \mid n \geq 0\}$ n'est pas substituable (par exemple, $a \doteq_L aab$, car $\underline{ab} \in L$ et $\underline{aabb} \in L$, mais $a \not\equiv_L aab$, car $\underline{aabb} \in L$ mais $\underline{aababb} \notin L$).*

D'autres exemples de langages substituables (et non substituables) seront donnés dans la suite (voir section 5.4.1).

5.3 Un algorithme d'apprentissage des langages hors-contextes substituables

Nous nous intéressons maintenant à l'apprentissage de la classe des langages substituables. Pour ce faire, nous explicitons deux algorithmes (voir les algorithmes 5.1 et 5.2), nommé SGL-G et SGL-RS, qui, à partir d'un échantillon d'exemples positifs d'un langage cible, retournent respectivement une grammaire hors-contexte pour le premier, et un système de réécriture de mots pour le second. Dans les deux cas, la représentation retournée est cohérente avec les données.

Les deux algorithmes reposent sur le même outil : les graphes de substitution [CE05].

Définition 5.4 (Graphe de substitution)

Étant donné un échantillon S d'exemples positifs, le graphe de substitution associé à S , $GS(S) = \langle N, A \rangle$ est défini comme suit :

$$N = \{u \in \Sigma^+ : \exists l, r \in \Sigma^*, lur \in S\}$$

$$A = \{(u, v) \in N \times N : u \neq v, \exists l, r \in \Sigma^*, lur \in S \wedge lvr \in S\}$$

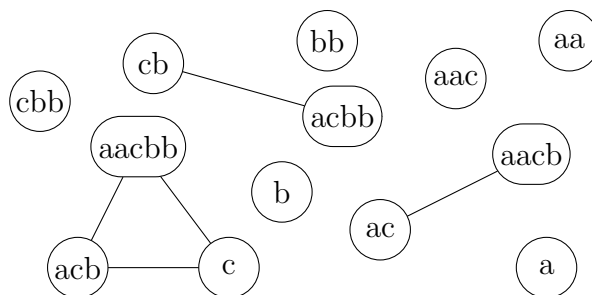


FIG. 5.1 – Le graphe de substitution correspondant à l'ensemble $S = \{c, acb, aacbb\}$.

Moins formellement, les noeuds N du graphe sont les facteurs (non vides) des mots de S et il existe une arête ($\in A$) entre deux noeuds ssi ces deux facteurs apparaissent au moins une fois dans le même contexte dans S . Un exemple de graphe de substitution est donné figure 5.1. Notons que pour un langage substituable L , chaque composante connexe du graphe (*i.e.* chaque ensemble de sommets reliés directement ou indirectement entre eux) correspond à une classe de congruence syntaxique de L . La composante dite *principale* est celle contenant tous les mots de l'échantillon (qui existe puisqu'ils apparaissent tous dans le même contexte (λ, λ)).

Comme les deux algorithmes précédemment cités sont très proches l'un de l'autre, nous détaillerons moins le fonctionnement et la preuve de convergence du second.

5.3.1 Apprendre des grammaires

SGL-G (voir l'algorithme 5.1) commence par construire le graphe de substitution à partir de l'échantillon d'apprentissage S (*construire_graphe_de_substitution*) puis les règles produisant les lettres : $\forall a \in \Sigma, [a] \rightarrow a \in \hat{P}$. Ensuite, il crée un non-terminal pour chaque composante connexe (s'il n'existe déjà) en repérant la principale (*est_composante_principale*), c'est-à-dire celle qui contient tous les mots de l'échantillon d'apprentissage. On ajoute ensuite à l'ensemble \hat{P} toutes les règles de production de la forme $\{[u] \rightarrow [v][w] \mid u = vw, u \in N, |v| > 0, |w| > 0\}$. Moins formellement, pour chaque mot $u = vw$ dont v et w sont des facteurs non vides, le non terminal $[u]$ correspondant à la composante du mot u dérive la forme sententielle $[v][w]$, où $[v]$ est le non terminal correspondant à la composante de v et $[w]$ celui de la composante de w . La grammaire obtenue est donc sous forme

Algorithme 5.1 : SGL-G (Substitution Graph Learner for Grammars)**Données** : un échantillon d'exemples positifs S **Résultat** : une grammaire hors-contexte $G = \langle \Sigma, \hat{V}, \hat{P}, \hat{S} \rangle$ **début**

```

 $\Sigma \leftarrow \text{lettres\_présentes}(S); \hat{V} \leftarrow \emptyset; \hat{P} \leftarrow \emptyset;$ 
 $GS \leftarrow \text{construire\_graphe\_de\_substitution}(S);$ 
pour chaque  $a \in \Sigma$  faire
   $[a] \leftarrow \text{créer\_nouveau\_non\_terminal}(a);$ 
   $\hat{V} \leftarrow \hat{V} \cup \{[a]\}; \hat{P} \leftarrow \hat{P} \cup \{[a] \rightarrow a\};$ 
pour chaque Composante connexe  $C_i$  de  $GS$  faire
   $u_i \leftarrow \min_{\triangleleft} \{w \in C_i\};$ 
  si  $u_i \notin \Sigma$  alors
     $[u_i] \leftarrow \text{créer\_nouveau\_non\_terminal}(u_i);$ 
     $\hat{V} \leftarrow \hat{V} \cup \{[u_i]\};$ 
  si est_composante_principale $(C_i, S)$  alors
     $\hat{S} \leftarrow [u_i];$ 
pour chaque Noeud  $u$  de  $GS$  faire
   $\hat{P} \leftarrow \hat{P} \cup \{[u_i] \rightarrow [u_j][u_k] \mid u = vw, u \in C_i, v \in C_j \text{ et } w \in C_k\};$ 
retourner  $\langle \Sigma, \hat{V}, \hat{P}, \hat{S} \rangle$ 

```

fin

normale de Chomsky. Une conséquence est que, pour les composantes contenant plus d'un élément, un unique non-terminal dérive l'ensemble des éléments de cette composante. Enfin, certaines règles pouvant apparaître plusieurs fois, l'algorithme supprime les doublons.

Nous obtenons immédiatement le résultat suivant :

Théorème 5.1 *Le temps d'exécution de SGL est polynomial dans la taille des données en entrée.*

Preuve : Supposons que l'échantillon d'apprentissage est $S = \{w_1, w_2, \dots, w_n\}$. On définit $E = \sum |w_i|$ et $L = \max |w_i|$. Le nombre de facteurs des mots de S (i.e. le nombre de noeuds du graphe de substitution) est inférieur à E^2 . Pour deux mots w_1, w_2 , trouver l'ensemble de leurs facteurs u, v tels que $u \doteq v$ prend au plus un temps L^2 . Par conséquent, la construction des arêtes du graphe de substitution est réalisable en un temps inférieur à $L^2 n^2$. Identifier les composantes du graphe prend un temps polynomial dans le nombre de noeuds et d'arêtes du graphe. Le nombre de règles de production est égal au nombre de façons de diviser les noeuds en deux facteurs non-nuls : au maximum la dernière boucle est effectuée LE^2 fois. \square

Pour clarifier le fonctionnement de SGL, nous détaillons maintenant deux exemples. Le premier est une exécution pas à pas sur un petit échantillon, le deuxième permet une discussion sur la grammaire inférée par SGL pour un langage particulier.

Exemple 5.2 *Supposons que l'échantillon d'apprentissage est $S = \{a, aa\}$. Il est clair que $aa \doteq a$. L'algorithme commence par créer le non terminal $[a]$ et la règle $[a] \rightarrow a$. Il n'y a qu'une seule composante dans le graphe de substitution et le plus petit facteur de cette composante C est a : aucun nouveau non-terminal n'est créé. Le seul élément de C qui peut se découper en deux facteurs non vides est aa , ce qui donne la règle $[a] \rightarrow [a][a]$. En sortie, SGL retourne donc la grammaire $\langle \{a\}, \{[a]\}, \{[a] \rightarrow a, [a] \rightarrow [a][a]\}, [a] \rangle$ qui génère le langage régulier a^+ .*

Exemple 5.3 *Considérons maintenant le langage $L = \{a^n cb^n \mid n \geq 0\}$ et supposons que nous avons un grand échantillon S de mots de ce langage. Le graphe de substitution sera formé de composantes $C_i \subset \{a^m cb^{m+i} \mid m \geq 0, i \in \mathbb{Z}\}$, de composantes $A_i = \{a^i\}$ et de composantes $B_i = \{b^i\}$. La composante principale est C_0 , par conséquent l'axiome est C_0 (par souci de simplicité, nous nommons ici les non-terminaux par l'identifiant de leur composante plutôt que par le représentant de la classe de congruence associée). La grammaire générée à partir de S aura des règles de la forme suivante ($i \geq 0$) :*

$C_i \rightarrow C_j B_{i-j}, C_{-i} \rightarrow A_{i-j} C_j, A_{i+j} \rightarrow A_i A_j, A_1 \rightarrow a, B_{i+j} \rightarrow B_i B_j, B_1 \rightarrow b.$

Il est à noter que la recherche des classes de congruence a pour conséquence de créer nombre de non-terminaux substantiellement plus grand que celui de la grammaire habituellement utilisée pour représenter ce langage.

5.3.2 SGL-G identifie à la limite les langages substituables

Théorème 5.2 *SGL identifie à la limite en temps et données polynomiaux la classe des langages substituables.*

Pour prouver ce théorème, nous commençons par définir un ensemble caractéristique.

Ensemble caractéristique.

Soit $L = \mathcal{L}(G)$ (où $G = \langle \Sigma, V, P, S \rangle$ est une grammaire hors-contexte) le langage substituable cible. Pour décrire l'ensemble caractéristique, nous avons besoin de définir les deux notations suivantes :

- $w(\alpha)$ est le plus petit mot de Σ^* (par rapport à l'ordre \triangleleft) pouvant être produit à partir de $\alpha \in (V \cup \Sigma)^+$ ($w(\alpha) = \alpha$ si $\alpha \in \Sigma^*$), et

- $C(N)$, pour $N \in V$, est le plus petit contexte (par rapport à \triangleleft) $(l,r) \in \Sigma^*$ tel que $S \xrightarrow{*} lNr$.

Nous pouvons maintenant définir notre ensemble caractéristique : $CS = \{lwr \mid \exists(N \rightarrow \alpha) \in P, (l,r) = C(N), w = w(\alpha)\}$. Comme $|CS| = |P|$, la taille de cet ensemble est polynomiale dans la taille de la cible.

Exemple 5.4 Soit $G = \langle \{a,b,c\}, \{S\}, \{S \rightarrow aSb, S \rightarrow c\}, S \rangle$ une grammaire hors-contexte engendrant le langage $\{a^n cb^n, n \geq 0\}$. Pour la règle $S \rightarrow c$, la construction donne $C(S) = (\lambda, \lambda)$ et $w(c) = c$ donc le mot c fait partie de l'ensemble caractéristique. En ce qui concerne la règle $S \rightarrow aSb$, on a $C(S) = (\lambda, \lambda)$ et $w(aSb) = acb$. L'ensemble caractéristique est donc $CS = \{c, acb\}$.

Convergence.

Il nous faut maintenant prouver qu'à partir d'un échantillon d'apprentissage $S \supseteq CS$, SGL-G retourne une grammaire $\hat{G} = \langle \Sigma, \hat{V}, \hat{P}, \hat{S} \rangle$ telle que $\mathcal{L}(\hat{G}) = L$.

Lemme 5.1 $L \subseteq \mathcal{L}(\hat{G})$.

Preuve : Par définition de l'ensemble caractéristique, pour chaque production $N \rightarrow \alpha$ il existe un facteur $w(\alpha)$ et un facteur $w(N)$ appartenant à la même composante du graphe de substitution. Si $\alpha = a_1 a_2 \dots a_n$ ($\forall i, a_i \in (V \cup \Sigma)$) alors $w(\alpha) = w(a_1)w(a_2) \dots w(a_n)$ (car le plus petit mot dérivable à partir de α est égal à la concaténation des plus petits mots dérivables à partir des éléments constituant α). De par la construction des règles, il existe une production $[w(\alpha)] \rightarrow [w(a_1)][w(a_2) \dots w(a_n)]$ et $(n-2)$ productions $[w(a_i)w(a_{i+1}) \dots w(a_n)] \rightarrow [w(a_i)][w(a_{i+1}) \dots w(a_n)]$. Ce qui implique $[w(N)] \xrightarrow{*}_{\hat{G}} [w(a_1)][w(a_2)] \dots [w(a_n)]$. Nous allons montrer par induction sur le nombre de pas de dérivation que pour tout $u \in \Sigma^*$, si $S \xrightarrow{*}_G u$ alors $\hat{S} \xrightarrow{*}_{\hat{G}} u$. Initialement, si $S \Rightarrow u$ alors $(S \rightarrow u) \in P$. Supposons que $u = a_1 \dots a_n, (\forall i, a_i \in \Sigma)$. Alors $[u] \xrightarrow{*} [a_1] \dots [a_n]$ et, par la construction des règles pour les lettres, $\forall i, ([a_i] \rightarrow a_i) \in \hat{P}$. Comme $u \in L$, $[u] = \hat{S}$, d'où $\hat{S} \xrightarrow{*}_{\hat{G}} u$. Supposons la propriété vraie pour toute dérivation de taille inférieure à k et prenons u un mot se dérivant en $k+1$ pas dans G . Alors $S \Rightarrow_G \alpha_1 \dots \alpha_n \xrightarrow{k}_G u$. Il existe $u_1, \dots, u_n \in \Sigma^*$ tels que $\forall i, \alpha_i \xrightarrow{m_i}_G u_i$ et $u = u_1 \dots u_n$. Comme $\forall i, m_i < k+1$, l'hypothèse d'induction implique $\hat{S} \xrightarrow{*}_{\hat{G}} [\alpha_1] \dots [\alpha_n] \xrightarrow{*}_{\hat{G}} u_1 \dots u_n = u$. □

Il nous reste à prouver que $\mathcal{L}(\hat{G}) \subseteq L$, c'est-à-dire que la grammaire inférée n'engendre par un langage trop large. Pour cela nous montrons le lemme suivant, qui stipule que les classes de congruence syntaxique sont stables par dérivation dans \hat{G} .

Lemme 5.2 Pour tout $v \in \Sigma^*$, pour tout facteur u de l'échantillon S , $[u] \xrightarrow{*}_{\hat{G}} v$ implique $u \equiv_L v$.

Preuve : Pour commencer, notons que la construction implique $[u] \xrightarrow{*}_{\hat{G}} u$. La preuve se fait par induction sur le nombre de pas de dérivation. Si $[u] \Rightarrow_{\hat{G}} v$, alors $v \in \Sigma$ et v est dans la même composante que u , donc $u \equiv_L v$. Supposons maintenant que c'est vrai pour toute dérivation de taille inférieure strictement à k et prenons une dérivation de taille k : $[u] \Rightarrow [v][w] \xrightarrow{k-1} x$. Il existe x_1, x_2 tels que $x = x_1x_2$, $[v] \xrightarrow{*} x_1$ et $[w] \xrightarrow{*} x_2$. Par hypothèse d'induction, on a $v \equiv_L x_1$ et $w \equiv_L x_2$. Comme $([u] \rightarrow [v][w]) \in \hat{P}$, il existe v' et w' tel que $v'w'$ est dans la même composante que u , et $v' \equiv_L v$, $w' \equiv_L w$ et $u \equiv_L v'w'$. Comme \equiv_L est une congruence, $u \equiv_L v'w' \equiv_L vw' \equiv_L vw \equiv_L x_1w \equiv_L x_1x_2 = x$, ce qui finit la preuve. \square

Ce résultat suffit pour démontrer que $\mathcal{L}(\hat{G}) \subseteq L$ car pour un v dans S , si $\hat{S}(= [v]) \xrightarrow{*}_{\hat{G}} u$ alors $u \equiv_L v$ ce qui implique $u \in L$.

5.3.3 Apprendre des systèmes de réécriture

L'algorithme SGL-G a l'inconvénient majeur d'inférer un grand nombre de règles de production qui ne sont pas utiles pour représenter le langage. Il est possible d'imaginer plusieurs de types processus polynomiaux permettant de diminuer ce nombre de productions. On peut par exemple effacer récursivement tous les non terminaux qui ne possèdent qu'une seule règle de production et les remplacer par la partie droite de la règle en question. On peut aussi imaginer qu'on enlève une par une les règles de production en testant à chaque fois si l'échantillon d'apprentissage est toujours reconnu. Si l'échantillon caractéristique est contenu dans les données, les règles ainsi effacées n'étaient pas utiles.

Dans cette section, nous allons présenter une variante de SGL-G dont l'intérêt principal est d'être plus efficace et donc plus utilisable sur des échantillons d'apprentissage de grande taille. L'idée est d'utiliser un système de réécriture de mots (SRM) au lieu d'une grammaire, ce qui est rendu possible car la classe des langages hors-contextes substituables est définie indépendamment d'une représentation choisie.

Le gain en efficacité repose sur un "nettoyage" du graphe de substitution afin d'éliminer les redondances éventuelles. Ainsi, si une composante du graphe contient les mots u et v , tel que $u \triangleleft v$, et qu'une autre composante contient les mots lur et lvr , on peut enlever le noeud lvr . En effet, si une règle de réécriture permet de remplacer les facteurs v par u , il est inutile d'avoir une règle permettant de substituer lvr par lur . Cela revient à éliminer des règles au SRM correspondant au graphe.

Quelques définitions.

Nous rappelons ici quelques définitions qui ont déjà été introduites dans le chapitre précédent, mais dans un contexte différent (celui des systèmes *délimités*

de réécriture de mots).

Un système de réécriture de mots est un ensemble fini de règles de réécriture, notées $u \vdash v$. Ce système est appelé *système de réduction* si et seulement si il existe un ordre total $<$ tel que pour toute règle $u \vdash v$, $v < u$. Dans ce qui suit, nous utilisons l'ordre hiérarchique \triangleleft , précédemment défini, pour orienter les règles. \vdash^* est la fermeture réflexive et transitive de \vdash .

Définition 5.5 (Système de réduction confluent, localement confluent)

Soit \mathcal{R} un système de réduction.

- \mathcal{R} est confluent si pour tout mot $w \in \Sigma^*$ s'il existe w_1 et w_2 , $w_1 \neq w_2$, $w \vdash^* w_1$ et $w \vdash^* w_2$, alors il existe un mot $w' \in \Sigma^*$ tel que $w_1 \vdash^* w'$ et $w_2 \vdash^* w'$.
- \mathcal{R} est localement confluent sur un ensemble S si pour tout mot $w \in S$ s'il existe w_1 et w_2 dans S , $w_1 \neq w_2$, $w \vdash^* w_1$ et $w \vdash^* w_2$, alors il existe un mot $w' \in S$ tel que $w_1 \vdash^* w'$ et $w_2 \vdash^* w'$.

Enfin, nous rappelons qu'un système de réécriture est noethérien s'il n'admet pas de dérivation infinie. Un système de réduction est noethérien.

Un système de réduction confluent définit une congruence car chaque mot se réécrit alors en un unique mot irréductible : deux mots sont congruents s'ils se réduisent dans le même mot. Si les règles de réécriture réduisent la taille ($\forall u \vdash v, |v| < |u|$) alors la taille maximale d'une dérivation est égale à la taille du mot en entrée. Comme nous utilisons un ordre plus faible pour orienter nos règles, l'ordre hiérarchique, la polynomialité de la taille d'une dérivation n'est pas assurée.

Étant donné un système de réduction, on peut définir un langage comme l'union de plusieurs classes de congruence. Ainsi, étant donné un ensemble E de mots irréductibles dans un système de réduction \mathcal{R} , on définit le langage $\mathcal{L}(\mathcal{R}, E) = \{w : \exists e \in E, w \vdash_{\mathcal{R}} e\}$. Ces langages sont appelés *langages congruentiels*. Dans certains cas, c'est une façon plus naturelle de représenter les langages que les grammaires formelles de la hiérarchie de Chomsky.

Réduction d'un graphe de substitution.

Étant donné un graphe de substitution $GS = \langle N, A \rangle$, on dit que GS se réduit en $GS' = \langle N', A' \rangle$ si et seulement si il existe $(u, v) \in N$ et $(l, r) \in \Sigma^*$, $u \triangleleft v$ et $|l| + |r| > 0$ tels que $lvr \in N$, $N' = (N \setminus \{lvr\}) \cup \{lur\}$ et $A' = (A \setminus \{(lvr, y) \in A\}) \cup \{(lur, y) : (lvr, y) \in A\}$.

On dit qu'un graphe de substitution GS est *irréductible* s'il n'existe pas d'autre graphe de substitution GS' tel que GS se réduise en GS' .

Cette définition nous permet d'expliciter l'algorithme SGL-RS (voir algorithme 5.2).

Algorithme 5.2 : SGL-RS (Substitution Graph Learner for Reduction Systems)

Données : un échantillon d'exemples positifs S

Résultat : Un système de réduction \mathcal{R} et un mot irréductible e

début

```

 $\mathcal{R} \leftarrow \emptyset;$ 
 $GS \leftarrow \text{construire\_graphe\_de\_substitution}(S);$ 
tant que  $\text{est\_réductible}(GS)$  faire
   $\lfloor GS \leftarrow \text{réduire\_graphe\_de\_substitution}(GS)$ 
pour chaque Composante connexe  $C_i$  de  $GS$  faire
   $u_{min} \leftarrow \min_{\triangleleft} \{w \in C_i\};$ 
  si  $\text{est\_composante\_principale}(C_i, S)$  alors
     $\lfloor e \leftarrow u_{min};$ 
  pour chaque noeud  $u_i$  de  $C_i$ ,  $u_i \neq u_{min}$  faire
     $\lfloor \mathcal{R} \leftarrow \mathcal{R} \cup \{u_i \vdash u_{min}\};$ 
retourner  $\langle \mathcal{R}, e \rangle$ 

```

fin

Résultat d'indentification.

Nous allons montrer que SGL-RS identifie à la limite en temps et donnée polynomiaux la classe des langages hors-contextes substituables.

Le raisonnement est le même que celui pour SGL-G. En supposant que nous disposons d'un échantillon d'apprentissage d'un langage L généré par une grammaire cible $G = \langle \Sigma, V, P, S \rangle$, qui contient un ensemble caractéristique défini comme pour l'algorithme précédent, il est facile de montrer les deux lemmes suivants.

Lemme 5.3 *Si $N \in V$ et $N \xRightarrow{*} u$ (pour $u \in \Sigma^*$), alors $u \vdash^* w(N)$.*

Preuve : Supposons que $N \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n = u$ est une dérivation de u dans G . On associe à cette dérivation la séquence $(w(N), w(\alpha_1), \dots, w(\alpha_{n-1}), u)$. Considérons un pas de dérivation $\alpha_i \Rightarrow \alpha_{i+1}$: il existe $l, r, \beta \in (\Sigma \cup V)^*$ et $M \in V$ tels que $\alpha_i = lMr$ et $\alpha_{i+1} = l\beta r$, où $M \rightarrow \beta \in P$. On a $w(\alpha_i) = w(l)w(M)w(r)$ et $w(\alpha_{i+1}) = w(l)w(\beta)w(r)$ ce qui implique $w(\alpha_{i+1}) \vdash_{\mathcal{R}}^* w(\alpha_i)$, par construction de \mathcal{R} . D'où $u \vdash^* w(N)$. \square

Lemme 5.4 *Si $v \vdash u$ alors $(v \in L \iff u \in L)$.*

Preuve : $v \vdash u$ implique qu'il existe $x \vdash y \in \mathcal{R}$ et $l, r \in \Sigma^*$ tels que $v = lxr$ et $u = lyr$. Par construction de \mathcal{R} , on a $x \doteq_S y$ qui implique $x \doteq_L y$. Comme L est substituable, on a $x \equiv_L y$, ce qui implique que $v \equiv_L u$ car \equiv_L est une congruence. \square

Le système de réduction inféré est localement confluent sur le langage L : tous les mots du langage se réécrivent dans le plus petit. Malheureusement, dans le cas général, le système, bien que noethérien, n'assure pas la polynomialité des dérivations. Alors que le système inféré est bien souvent largement plus petit que la grammaire retournée par SGL-G, dans certains cas tester si un mot appartient au langage est plus coûteux.

5.4 Sur les langages substituables

Dans cette section nous allons détailler la composition de la classe des langages hors-contextes substituables. Dans un premier temps nous donnerons des exemples (et des contre-exemples) de langages substituables. Puis nous essaierons de caractériser cette classe et les relations qui la lient à la hiérarchie de Chomsky.

Comme nous nous sommes placé dans le cadre de l'identification à la limite à partir d'exemples positifs seulement, nous ne pouvons espérer apprendre tous les langages finis. En effet, dans ce paradigme, plus la classe contient des langages complexes, plus son nombre de langages finis est réduit.

5.4.1 Exemples de langages substituables

- Σ^* est un langage substituable.
- Tous les langages composés d'un unique mot sont substituables.
- Le langage fini $\{a,aa\}$ n'est pas substituable car a apparaît dans les contextes (λ,λ) et, par exemple, (λ,a) , or aa apparaît aussi dans le contexte (λ,λ) mais pas dans (λ,a) . À partir de $S = \{a,aa\}$, nos algorithmes retournent une représentation du langage $\{a^n : n > 0\}$.
- $\{a^n : n > 0\}$ est un langage substituable.
- $\{a^n b^n : n > 0\}$ n'est pas substituable. Par exemple, $a \doteq aab$ mais il est clair que $a \not\equiv_L aab$.
- $\{a^n c b^n : n > 0\}$ est un langage substituable. L'ajout d'un marqueur au centre du mot résout le problème.
- $\{w c w^R : w \in (a,b)^*\}$ (le langage des palindromes avec un centre marqué) est substituable.

5.4.2 Relations avec les autres classes

Les langages hors-contextes substituables sont strictement inclus dans la classe des langages congruents [BO93].

Les langages réguliers *strictement déterministes* [Yok95] sont substituables. Comme l'automate correspondant est déterministe dans les deux sens (de l'état

initial ou de l'état final en utilisant les transitions à l'envers) et que chaque mot ne peut être généré que par une unique séquence d'états, il est facile de montrer que pour deux facteurs u et v tels que $u \doteq v$, la séquence d'états qui génère u doit commencer et se finir dans exactement les mêmes états que la séquence qui génère v .

Les langages très simples [Yok03] sont représentés par les grammaires sous forme normale de Greibach telles qu'aucune lettre de l'alphabet n'apparaît plus d'une fois dans les règles de production. Comme nous l'avons déjà dit dans la première partie de ce document, ces langages sont identifiables polynomialement à la limite. L'algorithme utilisé pour montrer ce résultat est une extension aux langages hors-contextes de celui apprenant les langages réguliers strictement déterministes. Pourtant, certains langages très simples ne sont pas substituables. Par exemple, considérons la grammaire $S \rightarrow bN, S \rightarrow aNP, N \rightarrow xM, N \rightarrow n, P \rightarrow rMP, P \rightarrow p, M \rightarrow m$. Le langage correspondant L est $bn, bxm, anp, axmp, anrmp, \dots$. On a $x \doteq_L nr$ mais pas $x \equiv_L nr$, car, par exemple, bxm est dans le langage mais pas $bnrm$. Néanmoins, remarquons que tous les exemples de langages très simples présents dans l'article les introduisant ([Yok03]) sont substituables.

Comme les exemples précédents en attestent, les langages hors-contextes substituables sont incomparables avec les langages réguliers et les langages finis. Nous conjecturons qu'ils sont strictement inclus dans la classe des langages NTS (voir définition 1.4).

5.4.3 Propriétés des langages substituables

La substituabilité peut être comparée à la réversibilité [Ang82]. On rappelle qu'un langage régulier est *réversible* si chaque fois que deux mots uw et vw sont dans le langage alors ux fait partie du langage si et seulement si vx en fait aussi partie (pour $x \in \Sigma^*$). Ainsi la substituabilité est l'analogue hors-contexte de la réversibilité pour les langages réguliers. Le grand nombre de travaux (voir par exemple [Mäk00, LSG04]) sur ces langages réversibles laissent espérer que nos résultats seront le point de départ d'autres recherches en inférence grammaticale.

On peut aussi faire un rapprochement avec les méthodes et travaux en inférence grammaticale de langages réguliers dont l'approche repose sur les langages *résiduels*. Le langage résiduel d'un langage L par rapport à un mot u (ou, plus simplement, le résiduel de L par u), noté $u^{-1}L$, est l'ensemble $\{w \in \Sigma^* : uw \in L\}$. Si l'on s'intéresse aux langages réguliers, le résiduel $u^{-1}L$ d'un langage L , représenté par l'automate fini déterministe minimal A , est le langage reconnu à partir de l'état q tel que $\delta^*(q_0, u) = q$. La substituabilité peut alors être comparée avec la distinguibilité utilisée pour l'inférence de langages réguliers [RST98, CT04]. L'idée est d'utiliser une mesure de similarité sur les résiduels. Ces travaux montrent

ensuite que la classe des langages réguliers est apprenable dans le cadre PAC si l'échantillon d'apprentissage permet une séparation des résiduels distincts du langage. C'est en reprenant cette idée mais en utilisant les contextes à la place des résiduels qu'Alexander Clark a obtenu un résultat d'apprentissage pour les langages NTS [Cla06a] : il a montré que les langages NTS sont PAC-apprenables si la distribution de l'échantillon d'apprentissage permet de distinguer les différentes classes de congruence syntaxique. Bien entendu, si une analogie peut-être faite entre les deux approches, les idées algorithmiques et leurs justifications sont radicalement différentes.

D'autre part, dans des travaux récents [DLT01], l'importance des techniques d'apprentissage utilisant les résiduels a été montrée. Ainsi, chaque langage régulier peut être caractérisé par un ensemble fini de résiduels et des opérations combinant cette "base". Le résultat de Clark sur les langages NTS tend à prouver qu'un résultat similaire peut, peut-être, être obtenu pour les langages hors-contextes en utilisant non pas une base formée de résiduels mais de classes de congruence. C'est un peu l'intuition qui se cache derrière notre travail : la propriété 5.1 implique qu'un langage purement hors-contexte correspond à un nombre infini de classes de congruence syntaxique mais nos résultats montrent que, au moins pour les langages substituables, un nombre restreint de classes de congruence est suffisant pour représenter ces langages, les autres classes étant obtenues par concaténation.

5.5 Application à un problème de langue naturelle

Depuis que des chercheurs se sont intéressés, dans les années cinquante, à la modélisation de l'acquisition de sa langue maternelle par l'enfant, un débat fait rage au sein de cette communauté. Il oppose les tenants d'un apprentissage reposant principalement sur une connaissance pré-natale de la langue (les linguistes nativistes) et ceux considérant que le langage s'acquiert en grande partie par l'expérience dont l'enfant jouit pendant son développement (les linguistes comportementalistes). Ainsi, les nativistes soutiennent que la rapidité de cet apprentissage ne peut être expliqué que par une connaissance innée de la langue. Leurs opposants soutiennent que les capacités cognitives universelles sont suffisantes pour expliquer ce phénomène.

5.5.1 Problématique

Un des (derniers) arguments des nativistes repose sur la fréquence d'apparition de certaines formes de questions. En anglais, les interrogations polaires (dont la réponse est oui ou non) sont formées à l'aide d'un auxiliaire en tête de phrase

(remplacé par “do” si le verbe principal n’est pas un auxiliaire). Par exemple, l’interrogation correspondant à “The man is hungry” est “Is the man hungry?”. Quand le sujet de la phrase est lié à une phrase relative comportant aussi un auxiliaire, c’est l’auxiliaire de la phrase principale qui est mis en tête, celui de la relative restant à sa place : “The man who is eating is hungry” donne “Is the man who is eating hungry?”. Mais on aurait pu imaginer, par exemple, un inversement complet, donnant la phrase (fausse grammaticalement) “Is the man who eating is hungry?”.

Il n’y a aucune raison pour que l’enfant favorise une des deux formes de questions, car elles sont de difficulté similaire. Pourtant, il a été montré [CN87] que les enfants anglo-saxons infèrent majoritairement (pour ne pas dire tout le temps) la phrase correcte. Noam Chomsky a été le premier à dire que les exemples de ce type de questions sont extrêmement rares dans l’environnement linguistique des enfants. Ce fut donc utilisé par les nativistes comme une preuve empirique de leur théorie. En 2002, Pullum et Scholz [PS02] ont réalisé une étude tendant à montrer qu’en fait ce type de structures n’est pas si rare que ça. Mais les tenants d’une connaissance innée du langage n’ont pas été convaincus par cette étude, remettant en cause la méthodologie utilisée et les conséquences de ce travail.

Reali et Christiansen [RC04] ont présenté une possible solution à ce problème. Ils prétendent que l’utilisation de statistiques locales, en l’occurrence des n -grams, peut être suffisante pour permettre à un apprenant d’inférer la structure correcte et de rejeter celle incorrecte. Toutefois cet argument a été mis à mal par [KST⁺05], qui ont montré que cette étude repose uniquement sur des coïncidences phonétiques. Ce résultat n’est pas surprenant car il était très peu plausible qu’un formalisme tel que les n -grams, qui ne peut modéliser que des relations concomitantes (donc régulières), puissent modéliser l’apprentissage de telles structures (purements hors-contextes).

La thèse que nous développons ici consiste à réfuter un des présupposés du débat, à savoir que la discussion sur la rareté des phrases n’a pas lieu d’être. En effet, nous allons développer une expérience qui montre qu’à partir d’un échantillon d’apprentissage ne contenant pas de phrases du type que nous cherchons à apprendre, SGL est capable de déterminer quelles structures sont correctes et lesquelles ne le sont pas.

5.5.2 Expérimentations

Dans un but de clarté, nous avons utilisé un très petit échantillon d’apprentissage, construit à la main par nos soins, composé de phrases (on devrait dire de mots) que l’on peut raisonnablement trouver présentes dans l’expérience linguistique d’un enfant. La table 5.1 résume l’expérience. Notre but était de montrer que même en absence de question avec une forme relative, l’algorithme était capable

the man who is hungry died .
 the man ordered dinner .
 the man died .
 the man is hungry .
 is the man hungry?
 the man is ordering dinner .

 is the man who is hungry ordering dinner?
 *is the man who hungry is ordering dinner?

TAB. 5.1 – *Place de l’auxiliaire dans des questions polaires. Les phrases au dessus de la ligne forment l’échantillon d’apprentissage à partir duquel SGL a travaillé. L’apprentissage a été testé à l’aide de celles en dessous. L’étoile correspond à une phrase que la sortie de l’algorithme a rejetée.*

de correctement étiqueter des questions de ce type. C’est ce que montre bien nos résultats : la question grammaticalement correcte est acceptée par la grammaire inférée alors que celle comportant une erreur grammaticale est rejetée.

Comme le fonctionnement de l’algorithme est somme toute assez simple, on peut facilement expliquer comment cela se produit. L’échantillon d’apprentissage place “the man” et “the man who is hungry” dans la même classe de congruence, puisqu’il existe deux phrases qui ne diffèrent que pour ces deux blocs de mots. De la même façon, “hungry” et “ordering dinner” sont congruents. Ainsi, la phrase “is the man hungry?” est congruente avec la phrase correcte.

Une dérivation de cette phrase est la suivante : [is the man hungry?] \Rightarrow [is the man hungry][?] \Rightarrow [is the man][hungry][?] \Rightarrow [is][the man][hungry][?] \Rightarrow [is][the man who is hungry][hungry][?] \Rightarrow [is][the man who is hungry][ordering dinner][?].

La table 5.2 résume une deuxième expérience que nous avons menée. Nous ne détaillerons pas cette étude, mais ce type de structures, à l’instar de celles étudiées précédemment, a été utilisé pour étayer la thèse des nativistes [PB85]. Cette fois aussi, l’algorithme est capable d’apprendre la structure correcte et de rejeter celles incorrectes, et ce en l’absence d’exemples de ce type dans les données. Notons toutefois que la dernière phrase est grammaticalement correcte en anglais : cela vient de l’ambiguïté syntaxique de la langue, puisqu’alors “rain” n’est pas un verbe mais un nom commun.

5.5.3 Discussion

Chomsky a été le premier à montrer les limites de l’approche de Harris, et il est certain que notre algorithme surgénéralise énormément (des phrases ne faisant pas partie du langage sont certainement acceptées). Sur des échantillons d’apprentissage bien plus grands, il est même possible que certaines structures

it rains
it may rain
it may have rained
it may be raining
it has rained
it has been raining
it is raining
it may have been raining
*it may have been rained
*it may been have rain
*it may have been rain

TAB. 5.2 – *Utilisation correcte des auxiliaires. L'échantillon d'apprentissage se trouve au dessus de la ligne, l'ensemble de test au dessous.*

incorrectes soient acceptées.

À la lumière de la solution que nous proposons, il est intéressant d'examiner pourquoi les nativistes ont pensé que ces problèmes étaient si importants.

Tout d'abord, le débat s'est toujours focalisé sur comment construire l'interrogation à partir de la phrase déclarative. C'est ainsi qu'a été posée la question de savoir quel auxiliaire doit passer en tête de phrase. Cela revient à dire implicitement que la phrase interrogative ne peut être obtenue qu'à *partir* de la phrase déclarative, ce qui a donné un grand nombre de travaux se basant sur des transformations grammaticales. Aujourd'hui, grâce à l'étude de nouveaux formalismes permettant l'inférence directe de telles structures, sans passer par une étape de transformation, il est évident que cette supposition était fautive. S'il existe bien sûr un lien, sémantique, entre ces deux structures, il n'implique pas la nécessité d'une relation syntaxique/grammaticale.

Ensuite, le présupposé sur les algorithmes d'apprentissage est très réducteur. Il consiste à penser que seule la présence de structures grammaticalement identiques dans les données peut permettre l'induction. Nous avons, au minimum, réussi à démontrer que cette supposition est fautive. La structure correcte peut être apprise en l'absence de structure similaire dans les données : tant que les différentes parties peuvent être apprises séparément, il est possible d'inférer la globalité de la structure.

Une question de plus grande importance est de savoir si les bons résultats de notre algorithme ne sont pas dus au fait qu'il soit biaisé pour répondre à ce genre de problèmes et seulement à ceux-là. De tels algorithmes sont appelés "domaine-spécifiques" en opposition aux algorithmes "génériques" qui eux permettent de répondre à plus de problèmes mais dont les solutions ne sont pas forcément optimales. Clairement notre algorithme est biaisé : il surgénéralise massivement. Un

de ses avantages est que sa simplicité permet une explication remarquablement claire de son biais. Mais est-ce que ce biais est spécifique au domaine de la langue anglaise? Il ne se réfère en aucune façon à quelque chose de spécifique au type de problèmes qu'il résout : il n'utilise aucune connaissance *a priori* sur la parole ou les phrases d'apprentissage, ni même à une éventuelle structure hiérarchique de ces phrases. Il n'est même pas spécifiques aux langues naturelles! Il est de nos jours largement entendu que ce type d'approches forme des algorithmes génériques [JP05].

5.6 Conclusion et perspectives

Les algorithmes d'apprentissage de langages réguliers se focalisent sur l'identification des états de l'automate fini déterministe minimal. Ces états sont isomorphes aux résiduels permettant de représenter le langage en question. Quand on s'intéresse aux langages hors-contextes dans cette optique, l'idée naturelle est d'identifier, par exemple, les configurations d'un automate à pile, c'est-à-dire les paires formées d'un état et d'une composition potentielle de la pile. Un des problèmes de cette approche est que la structure dépend de la représentation (ici les automates à pile). Une façon de voir le travail développé dans ce chapitre, est de dire qu'une meilleure approche consiste à identifier les éléments du monoïde syntaxique. Ce monoïde représente la structure du langage dans une forme dépouillée, sans les "fioritures" inhérentes au choix d'une représentation particulière. Du point de vue de l'apprentissage, cette approche est intéressante et plus naturelle car elle est purement syntaxique : elle est liée au langage en lui-même et pas à une de ses représentations.

Un autre point important est que nos algorithmes ne reposent pas sur l'identification de constituants, c'est-à-dire des facteurs ayant été générés par le même non terminal d'une *grammaire* cible. De nombreux algorithmes d'apprentissage de langages hors-contextes se sont concentrés sur ce but, alors que c'est une approche qui, d'un certain point de vue, n'est pas pertinente : il existe un grand nombre de grammaires engendrant le même langage, et les constituants de l'une ne sont pas forcément les constituants des autres.

Une des faiblesses de ce travail est que nous n'avons pas obtenu de caractérisation grammaticale de la classe apprise, ni un algorithme permettant de déterminer si un langage hors-contexte est substituable. Il semble clair que cette propriété est indécidable dans le cas général, mais il est peut-être possible de décider si un langage NTS est substituable.

D'un point de vue plus général, notre approche est basée sur l'identification des facteurs syntaxiquement congruents. Les langages substituables possèdent une propriété qui permet une procédure simple pour déterminer si deux facteurs sont

congruents. Mais cette approche peut sembler un peu naïve en ce sens qu'elle correspond à un fonctionnement dit du "tout ou rien". Cependant, il est facile d'imaginer des propriétés plus complexes, par exemple sur la distribution des contextes de facteurs, et ainsi d'étudier (et d'apprendre) des classes plus importantes que celle des langages hors-contextes substituables. On peut aussi imaginer la définition de langages k -substituables, à l'instar des langages k -réversibles : un langage serait k -substituable si dès que deux facteurs admettent le même contexte de taille k ils partagent alors tous les contextes de cette taille.

Pour conclure, nous avons montré qu'une formalisation simple de l'idée de substitution introduite par Harris permet un apprentissage polynomial d'une sous-classe des langages hors-contextes, sous-classe suffisamment intéressante pour répondre à des problèmes d'autres disciplines que l'inférence grammaticale.

Conclusions

Le premier apport de ce document réside dans le second chapitre. C'est à notre connaissance la première tentative de synthèse visant à répertorier et à analyser les problèmes propres à l'inférence grammaticale des langages hors-contextes. Nous extrayons ensuite trois grandes classes de solutions regroupant les différentes approches suivies par les chercheurs de la communauté.

Nos contributions explorent trois pistes permettant d'apprendre des langages hors-contextes, chacune contournant quelques-unes des difficultés inhérentes à l'apprentissage des langages hors-contextes.

La première consistait à structurer des exemples d'apprentissage dans le but d'utiliser, par la suite, un algorithme existant capable d'identifier à la limite l'ensemble de la classe des langages hors-contextes. Bien que nos résultats n'aient pas été aussi positifs qu'espérés, ce travail met en évidence les problèmes posés par une telle approche.

Dans un deuxième temps, nous avons proposé une approche reposant sur un changement de représentation. Un grand nombre de problèmes ayant été mis en évidence lors de l'utilisation de grammaires hors-contextes, il nous a semblé judicieux de chercher un autre formalisme pour représenter les langages algébriques. C'est ainsi que nous avons introduit la classe des systèmes délimités de réécriture de mots hybrides et presque non-chevauchants et obtenu un résultat d'identification à la limite sur une sous-classe de ces systèmes. De plus, une étude expérimentale a permis de montrer l'efficacité de notre approche, relativement aux meilleurs algorithmes de l'état de l'art [EdlHJ04, EdlHJ06].

Enfin, nous avons présenté un troisième travail, reposant sur une restriction à une sous-classe des langages hors-contextes. Notre but était d'apprendre des langages à partir d'exemples de mots du langage seulement. Or des résultats théoriques démontrent que la classe entière n'est, dans ce cas, pas identifiable à la limite. Contrairement aux restrictions précédemment proposées, nous définissons notre sous-classe à l'aide de contraintes sur les langages et non sur les règles des grammaires. Cela permet une indépendance bien venue vis-à-vis de la classe de représentations. Nous obtenons ainsi un résultat d'identification de la classe des langages substituables, que ce soit à l'aide de grammaires ou de systèmes de réduction [CE05, CE06].

Le travail reposant sur un changement de représentation des langages offre de multiples perspectives. De nombreuses propriétés ont été introduites en théorie des langages pour les systèmes de réécriture, comme par exemple la “basicité” [Sén98], et certaines peuvent sans doute permettre des résultats d’apprentissage. D’autre part, nous avons choisi de gérer la confluence à l’aide d’une contrainte simple qui peut certainement être raffinée. D’autant plus que nous nous sommes rendu compte des problèmes qu’elle posait lors de l’apprentissage.

Le contenu du dernier chapitre de cette thèse pourrait être le point de départ de bien d’autres travaux. La substituabilité, que nous avons formalisée, étant l’analogie hors-contexte de la réversibilité des langages réguliers, le grand nombre de résultats obtenus sur cette dernière classe et ses classes dérivées laisse présager du fort potentiel de cette approche.

Nos deux derniers travaux présentés, bien que l’approche soit différente, partagent plusieurs caractéristiques. Tout d’abord, la plus évidente est l’utilisation de systèmes de réécriture de mots. Dans un cas il s’agit du fondement même de l’approche alors que dans le second travail c’est simplement une possibilité. Ce formalisme de représentation de langages n’a été que très peu utilisé en inférence grammaticale mais ces premiers résultats en font un très bon candidat pour suppléer les grammaires, et l’attention de membres de la communauté commence à se concentrer dessus [MOP06].

Une autre caractéristique commune à ces deux travaux est que le processus d’apprentissage se base sur une étude des contextes des facteurs. L’algorithme LARS accepte une règle si, quel que soit le contexte, la partie gauche de la règle dans ce contexte appartient au langage si et seulement si la partie droite dans ce même contexte en fait partie. Quant aux langages hors-contextes substituables, leur définition même repose sur une contrainte similaire au niveau des contextes. Ainsi, pour apprendre des langages hors-contextes nos deux approches se focalisent sur l’étude... des contextes !

Si les travaux utilisant des systèmes de réécriture de mots constituent une perspective intéressante pour l’apprentissage de langages algébriques, d’autres voies sont en train de s’ouvrir. Par exemple, l’utilisation de machines à noyaux [MMR⁺01], dotées de noyaux explicitement construits pour travailler sur des mots, est une idée séduisante [CFWS06].

Une autre approche, intéressant fortement les chercheurs de la communauté du traitement de la langue naturelle, consiste à se restreindre à une classe de langages transversale à la hiérarchie de Chomsky : les langages *midly context-sensitive*. Si, dans la définition informelle initiale [Jos85], cette classe contenait strictement l’ensemble des langages hors-contextes, les travaux récents ont tendance à la considérer comme transversale à la hiérarchie de Chomsky. Par exemple, quel est l’intérêt, quand on cherche à modéliser les langues naturelles,

à ce que la classe contienne l'ensemble des langages finis ? Quoi qu'il en soit, la plupart des structures hors-contextes y sont représentables. Cette classe peut être représentée entièrement ou en partie par plusieurs formalismes, la plupart étant des systèmes de réécriture particuliers. L'inférence grammaticale de cette classe est un challenge nouveau et prometteur [BBY04, OABBA06].

Ces résultats tendent à montrer que la hiérarchie de Chomsky, définie par des contraintes syntaxiques sur les règles des grammaires, n'est pas pertinente du point de vue de l'apprentissage. La classe des langages hors-contextes souffre ainsi d'une hétérogénéité importante au niveau des propriétés des langages qu'elle regroupe. À l'exception de celui de Sakakibara, tous les résultats positifs obtenus dans le cadre de l'identification polynomiale à la limite l'ont été sur des classes transversales à la hiérarchie. Et le fait que l'on soit capable d'apprendre une classe contenant des langages qui ne sont pas hors-contextes abonde dans ce sens. Ce qui précède contribue à remettre en question la pertinence de l'utilisation de grammaires en inférence grammaticale.

Saint-Etienne,
le 10 novembre 2006.

Bibliographie

- [Abe88] N. Abe, *Feasible learnability of formal grammars and the theory of natural language acquisition.*, COLING, 1988, p. 1–6.
- [Abe95] ———, *Characterizing pac-learnability of semilinear sets*, Inform. and Comput. **116** (1995), 81–102.
- [AFvZ02] P. Adriaans, H. Fernau, et M. van Zaannen (eds.), *Grammatical inference: Algorithms and applications, proceedings of icgi '02*, LNAI, vol. 2484, Berlin, Heidelberg, Springer-Verlag, 2002.
- [AGH00] K. Arnold, J. Gosling, et D. Holmes, *The java programming language*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
- [AKZ01] N. Abe, R. Khardon, et T. Zeugmann (eds.), *Proceedings of alt 2001*, LNCS, no. 2225, Berlin, Heidelberg, Springer-Verlag, 2001.
- [AM97] N. Abe et H. Mamitsuka, *Predicting protein secondary structure using stochastic tree grammars*, Machine Learning Journal **29** (1997), 275–301.
- [A.N58] A. Nerode, *Linear automaton transformations*, Proceedings of American Mathematics Society, 1958, p. 541–544.
- [Ang82] D. Angluin, *Inference of reversible languages*, Journal of the Association for Computing Machinery **29** (1982), no. 3, 741–765.
- [Ang87] ———, *Queries and concept learning*, Machine Learning Journal **2** (1987), 319–342.
- [ASA01] H. Arimura, H. Sakamoto, et S. Arikawa, *Efficient learning of semi-structured data from queries*, in Abe et al. [AKZ01], p. 315–331.
- [Aut87] J.M. Autebert, *Langages algébriques*, Études et Recherches en Informatique, Masson, 1987.
- [AV02] P. Adriaans et M. Vervoort, *The EMILE 4.1 grammar induction toolbox*, in Adriaans et al. [AFvZ02], p. 293–295.
- [AW99] A. et L. Wachowski, *The matrix*, 1999, produit par J. Silver.

- [Bak79] J. K. Baker, *Trainable grammars for speech recognition*, Speech Communication Papers for the 97th Meeting of the Acoustical Society of America (D. H. Klatt et J. J. Wolf, eds.), 1979, p. 547–550.
- [BBY04] Leonor Becerra-Bonache et Takashi Yokomori, *Learning mild context-sensitiveness: Toward understanding children’s language learning.*, 7th International Colloquium on Grammatical Inference, Lecture Notes in Computer Science, vol. 3264, 2004, p. 53–64.
- [BJVU98] A. Brazma, I. Jonassen, J. Vilo, et E. Ukkonen, *Pattern discovery in biosequences*, in Honavar and Slutski [HS98], p. 257–270.
- [BM04] J. Besombes¹ et J.-Y. Marion, *Learning reversible categorial grammars from structures*, *Categorial Grammars*, 2004, p. 121–153.
- [BO93] R. Book et F. Otto, *String-rewriting systems*, Springer-Verlag, 1993.
- [Boa80] L. Boasson, *Grammaire à non-terminaux séparés*, Proc. 7th ICALP, LNCS 85, 1980, p. 105–118.
- [Cam85] J. Cameron, *The terminator*, 1985, produit par Cinema 84, Hemdale Film Corporation et Pacific Western.
- [Cas95] F. Casacuberta, *Statistical estimation of stochastic context-free grammars*, *Pattern Recognition Letters* **16** (1995), 565–573.
- [CDG⁺97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, et M. Tommasi, *Tree automata techniques and applications*, Available on: <http://www.grappa.univ-lille3.fr/tata>, 1997, release October, 1st 2002.
- [CE05] A. Clark et R. Eyraud, *Identification in the limit of substitutable context-free languages*, *Proceedings of ALT’05*, LNAI, no. 3734, Springer, 2005, p. 283–296.
- [CE06] ———, *Learning auxiliary fronting with grammatical inference*, *Proceedings of the 10th Conference of Computational Natural Language Learning*, Association for Computational Linguistics, 2006, p. 125–132.
- [CFWS06] A. Clark, C. Costa Florencio, C. Watkins, et M. Serayet, *Planar languages and learnability*, *Proceedings of ICGI*, september 2006.
- [Chi01] B. Chidlovskii, *Schema extraction from xml: A grammatical inference approach*, *Proceedings of the 8th International Workshop on Knowledge Representation meets Databases* (M. Lenzerini, D. Nardi, W. Nutt, et D. Suciu, eds.), *CEUR Workshop Proceedings*, vol. 45, 2001.
- [Cho56] N. Chomsky, *Three models for the description of language*, *Transactions on Information Theory* **3** (1956), 113–124.

- [Cla01] Alexander Clark, *Unsupervised induction of stochastic context-free grammars using distributional clustering*, Proceedings of CoNLL-2001 (Walter Daelemans et Rémi Zajac, eds.), 2001, p. 105–112.
- [Cla06a] A. Clark, *Learning deterministic context free grammars: the omphalos competition*, Machine Learning Journal (2006), To be published.
- [Cla06b] ———, *Pac-learning unambiguous nts languages*, Proceedings of ICGI, september 2006.
- [CM02] A. Cornuéjols et L. Miclet, *Apprentissage artificiel; concepts et algorithmes*, Eyrolles, 2002.
- [CN87] S. Crain et M. Nakayama, *Structure dependence in grammar formation*, Language **63** (1987), 522–543.
- [CO94] R. C. Carrasco et J. Oncina (eds.), *Grammatical inference and applications, proceedings of icgi '94*, LNAI, no. 862, Berlin, Heidelberg, Springer-Verlag, 1994.
- [COCR01] R. C. Carrasco, J. Oncina, et J. Calera-Rubio, *Stochastic inference of regular tree languages*, Machine Learning Journal **44** (2001), no. 1, 185–197.
- [CR36] A. Church et J.B. Rosser, *Some properties of conversion*, Transactions of the American Mathematical Society **3** (1936), no. 36, 472–482.
- [CT04] A. Clark et F. Thollard, *Pac-learnability of probabilistic deterministic finite state automata.*, Journal of Machine Learning Research **5** (2004), 473–497.
- [Den01] F. Denis, *Learning regular languages from simple positive examples*, Machine Learning Journal **44** (2001), no. 1, 37–66.
- [DJ90] N. Dershowitz et J. Jouannaud, *Rewrite systems*, Handbook of Theoretical Computer Science: Formal Methods and Semantics (J. van Leeuwen, ed.), vol. B, North Holland, Amsterdam, 1990, p. 243–320.
- [dlH97] C. de la Higuera, *Characteristic sets for polynomial grammatical inference*, Machine Learning Journal **27** (1997), 125–138.
- [dlH03] ———, *A bibliographical study of grammatical inference*, Pattern Recognition (2003), To appear.
- [dlHAVZO03] C. de la Higuera, P. Adriaans, M. van Zaanen, et J. Oncina (eds.), *Proceedings of the workshop and tutorial on learning context-free grammars*, ISBN 953-6690-39-X, 2003.
- [dlHO02] C. de la Higuera et J. Oncina, *Learning deterministic linear languages*, in Kivinen and Sloan [KS02], p. 185–200.

- [DLT01] F. Denis, A. Lemay, et A. Terlutte, *Learning regular languages using RFSA*, in Abe et al. [AKZ01], p. 348–363.
- [DNM98] C.L. Blake D.J. Newman, S. Hettich et C.J. Merz, *UCI repository of machine learning databases*, 1998.
- [Dup94] P. Dupont, *Regular grammatical inference from positive and negative samples by genetic search: the GIG method*, in Carrasco and Oncina [CO94], p. 236–245.
- [EdlHJ04] R. Eyraud, C. de la Higuera, et J.C. Janodet, *Representating languages by learnable rewriting systems*, Proceedings of the 7th International Colloquium on Grammatical Inference (NCSR-Demokritos, Athens, Greece), LNAI, vol. 3264, Springer, October 2004, p. 139–150.
- [EdlHJ06] R. Eyraud, C. de la Higuera, et J.-C. Janodet, *Lars: a learning algorithm for rewriting systems*, Machine Learning Journal (2006), À paraître.
- [ET93] B. Efron et R.J. Tibshirani, *An introduction to the bootstrap*, Chapman and Hall, Londre, 1993.
- [Fer01] H. Fernau, *Learning XML grammars*, Machine Learning and Data Mining in Pattern Recognition MLDM'01 (P. Perner, ed.), LNCS, no. 2123, Springer-Verlag, 2001, p. 73–87.
- [Fer02] ———, *Learning tree languages from text*, in Kivinen and Sloan [KS02], p. 153–168.
- [FJ94] M. Frazier et C.D. Page Jr, *Prefix grammars: An alternative characterisation of the regular languages*, Information Processing Letters **51** (1994), no. 2, 67–71.
- [GM96] S. A. Goldman et H. Mathias, *Teaching a smarter learner*, Journal of Computer and System Sciences **52** (1996), no. 2, 255–267.
- [GO93] P. García et J. Oncina, *Inference of recognizable tree sets*, Tech. Report DSIC-II/47/93, Departamento de Lenguajes y Sistemas Informáticos, Universidad Politécnica de Valencia, Spain, 1993.
- [Gol67] E. M. Gold, *Language identification in the limit*, Information and Control **10** (1967), no. 5, 447–474.
- [Gol78] ———, *Complexity of automaton identification from given data*, Information and Control **37** (1978), 302–320.
- [GSVG94] P. García, E. Segarra, E. Vidal, et I. Galiano, *On the use of the morphic generator grammatical inference (mgi) methodology in automatic speech recognition*, International Journal of Pattern Recognition and Artificial Intelligence **4** (1994), 667–685.
- [Har54] Z. Harris, *Distributional structure*, Word **10** (1954), no. 2-3, 146–62.

- [HBJ02] A. Habrard, M. Bernard, et F. Jacquenet, *Generalized stochastic tree automata for multi-relational data mining*, in Adriaans et al. [AFvZ02], p. 120–133.
- [HS98] V. Honavar et G. Slutski (eds.), *Grammatical inference, proceedings of icgi '98*, LNAI, no. 1433, Berlin, Heidelberg, Springer-Verlag, 1998.
- [Ish89] I. Ishizaka, *Learning simple deterministic languages*, Proceedings of COLT 89, 1989.
- [Ish95] H. Ishizaka, *Polynomial time learnability of simple deterministic languages*, Machine Learning Journal **5** (1995), 151–164.
- [JLP01] A. Jagota, R. B. Lyngsø, et C. N. S. Pedersen, *Comparing a hidden Markov model and a stochastic context-free grammar*, Proceedings of WABI '01 (Berlin, Heidelberg), LNCS, no. 2149, Springer-Verlag, 2001, p. 69–74.
- [Jos85] A. K. Joshi, *How much context-sensitivity is required to provide reasonable structural description: Tree adjoining grammars*, Natural Language Parsing: Psychological, Computational and Theoretical Perspectives, Cambridge University Press, 1985, p. 206–250.
- [JP05] R. Jackendoff et S. Pinker, *The nature of the language faculty and its implications for evolution of language (reply to fitch, hauser, and chomsky)*, Cognition **97** (2005), 211–225.
- [KB96] T. Kammeyer et R. K. Belew, *Stochastic context-free grammar induction with a genetic algorithm using local search*, Foundations of Genetic Algorithms IV (University of San Diego, CA, USA) (R. K. Belew et M. Vose, eds.), Morgan Kaufmann, 1996.
- [Kil92] P. Kilpeläinen, *Tree matching problems with applications to structured text databases*, Ph.D. thesis, University of Helsinki, novembre 1992.
- [Klo92] J. W. Klop, *Term rewriting systems*, Handbook of Logic in Computer Science (S. Abramsky, D. Gabbay, et T. Maibaum, eds.), vol. 2, Oxford University Press, 1992, p. 1–112.
- [KMT97] T. Koshiha, E. Mäkinen, et Y. Takada, *Learning deterministic even linear languages from positive examples*, Theoretical Computer Science **185** (1997), no. 1, 63–79.
- [KMT00] ———, *Inferring pure context-free languages from positive data*, Acta Cybernetica **14** (2000), no. 3, 469–477.
- [KS94] T. Knuutila et M. Steinby, *Inference of tree languages from a finite sample: an algebraic approach*, Theoretical Computer Science **129** (1994), 337–367.

- [KS02] J. Kivinen et R. H. Sloan (eds.), *Proceedings of colt 2002*, LNAI, no. 2375, Berlin, Heidelberg, Springer-Verlag, 2002.
- [KST⁺05] X.-N. Kam, I. Stoyneshka, L. Tornyoova, W.G. Sakas, et J.D. Fodor, *Non-robustness of syntax acquisition from n-grams: A cross-linguistic perspective*, The 18th Annual CUNY Sentence Processing Conference, 2005.
- [KV89] M. Kearns et L. Valiant, *Cryptographic limitations on learning boolean formulae and finite automata*, 21st ACM Symposium on Theory of Computing, 1989, p. 433–444.
- [Lee96] L. Lee, *Learning of context-free languages: A survey of the literature*, Technical Report TR-12-96, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, 1996.
- [LG90] P. Laird et E. Gamble, *Ebg and term rewriting systems*, Algorithmic Learning Theory, 1990, p. 425 – 440.
- [LPP98] K. J. Lang, B. A. Pearlmutter, et R. A. Price, *Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm*, in Honavar and Slutski [HS98], p. 1–12.
- [LS00] P. Langley et S. Stromsten, *Learning context-free grammars with a simplicity bias*, Proceedings of ECML 2000, 11th European Conference on Machine Learning,, LNCS, vol. 1810, Springer-Verlag, 2000, p. 220–228.
- [LS02] E. Lehman et A. Shelat, *Approximation algorithms for grammar-based compression*, SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms (Philadelphia, PA, USA), Society for Industrial and Applied Mathematics, 2002, p. 205–212.
- [LSG04] D. López, J. M. Sempere, et P. García, *Inference of reversible tree languages.*, IEEE Transactions on Systems, Man, and Cybernetics, Part B **34** (2004), no. 4, 1658–1665.
- [LV91] M. Li et P. M. B. Vitanyi, *Learning simple concepts under simple distributions*, SIAM J. Comput. **20** (1991), no. 5, 911–935.
- [LY90] K. Lari et S.J. Young, *The estimation of stochastic context-free grammars using the inside-outside algorithm*, Computer Speech and language **4** (1990), no. 1, 35–56.
- [Mäk96] E. Mäkinen, *A note on the grammatical inference problem for even linear languages*, Fundamenta Informaticae **25** (1996), no. 2, 175–182.
- [Mäk00] Erkki Mäkinen, *On inferring zero-reversible languages.*, Acta Cybern. **14** (2000), no. 3, 479–484.

- [MG05] W. Moczydlowski et A. Geser, *Termination of single-threaded one-rule semi-thue systems*, Proceedings of the 16th International Conference on Rewriting Techniques and Applications, LNCS 3467, 2005, p. 338–352.
- [Mit97] T. M. Mitchell, *Machine learning*, McGraw-Hill, 1997.
- [MMR⁺01] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, et B. Schölkopf, *An introduction to kernel-based learning algorithms*, IEEE Neural Networks **12** (2001), no. 2, 181–201.
- [MNO88] R. McNaughton, P. Narendran, et F. Otto, *Church-Rosser Thue systems and formal languages*, Journal of the Association for Computing Machinery **35** (1988), no. 2, 324–344.
- [MOP06] F. Mraz, F. Otto, et M. Platek, *Learning analysis by reduction from positive data*, Proceedings of the 8th International Colloquium on Grammatical Inference, september 2006.
- [MW97] C. Manning et I. Witten, *Inferring lexical and grammatical structure from sequences*, SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences (Washington, DC, USA), IEEE Computer Society, 1997, p. 265.
- [Niv70] M. Nivat, *On some families of languages related to the dyck language*, Proc. 2nd Annual Symposium on Theory of Computing, 1970.
- [NM05] K. Nakamura et M. Matsumoto, *Incremental learning of context-free grammars based on bottom-up parsing and search*, Pattern Recognition **38** (2005), no. 9, 1384–1392.
- [NMW97] C. Nevill-Manning et I. Witten, *Identifying hierarchical structure in sequences: a linear-time algorithm*, Journal of Artificial Intelligence Research **7** (1997), 67–82.
- [NMWO96] Craig G. Nevill-Manning, Ian H. Witten, et Dan R. Olsen, *Compressing semi-structured text using hierarchical phrase identification.*, Data Compression Conference, 1996, p. 63–72.
- [OABBA06] T. Oates, T. Armstrong, L. Becerra-Bonache, et M. Atamas, *Inferring grammars for mildly context sensitive languages in polynomial-time*, Proceedings of the 8th International Colloquium on Grammatical Inference, september 2006.
- [O'D77] M. J. O'Donnell, *Computing in systems described by equations*, LNCS, vol. 58, Springer, 1977.
- [OG92] J. Oncina et P. García, *Identifying regular languages in polynomial time*, Advances in Structural and Syntactic Pattern Recognition (H. Bunke, ed.), Series in Machine Perception and Artificial Intelligence, vol. 5, World Scientific, 1992, p. 99–108.

- [PB85] S. F. Pilato et R. C. Berwick, *Reversible automata and induction of the english auxiliary system.*, ACL, 1985, p. 70–75.
- [Pit89] L. Pitt, *Inductive inference, DFA's, and computational complexity*, Analogical and Inductive Inference, LNAI, no. 397, Springer-Verlag, Berlin, Heidelberg, 1989, p. 18–44.
- [PPK⁺04] G. Petasis, G. Paliouras, V. Karkaletsis, C. Halatsis, et C. Spyropoulos, *E-grids: Computationally efficient grammatical inference from positive examples*, Grammars **7** (2004), 69–110.
- [PS02] G. K. Pullum et B. C. Scholz, *Empirical assessment of stimulus poverty arguments*, The Linguistic Review **19** (2002), 9–50.
- [PW88] L. Pitt et M. Warmuth, *Reductions among prediction problems: on the difficulty of predicting automata*, 3rd Conference on Structure in Complexity Theory, 1988, p. 60–69.
- [Rad91] Radzinoki, *Chinese number-name, tree adjoining languages and mildly context-sensitivity*, computational linguistics **17** (1991), 277–299.
- [Rao04] M. R. K. Krishna Rao, *Inductive inference of term rewriting systems from positive data*, Algorithmic Learning Theory, 2004, p. 69–82.
- [RC04] F. Realis et M. Christiansen, *Structure dependence in language acquisition: Uncovering the statistical richness of the stimulus*, Proceedings of the 26th Annual Conference of the Cognitive Science Society, 2004.
- [RST98] D. Ron, Y. Singer, et N. Tishby, *On the learnability and usage of acyclic probabilistic finite automata*, Journal of Computer and System Sciences (JCSS) **56** (1998), no. 2, 133–152.
- [Sak87] Y. Sakakibara, *Inferring parsers of context-free languages from structural examples*, Tech. Report 81, Fujitsu Limited, International Institute for Advanced Study of Social Information Science, Numazu, Japan, 1987.
- [Sak92] ———, *Efficient learning of context-free grammars from positive structural examples*, Information and Computation **97** (1992), 23–60.
- [SB97] J. A. Sánchez et J. M. Benedí, *Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformation*, IEEE Trans. on Pattern Analysis and Machine Intelligence **19** (1997), no. 9, 1052–1055.
- [SB02] I. Salvador et J-M. Benedí, *RNA modeling by combining stochastic context-free grammars and n-gram models*, International Journal

- of Pattern Recognition and Artificial Intelligence **16** (2002), no. 3, 309–316.
- [SBH⁺94] Y. Sakakibara, M. Brown, R. Hughley, I. Mian, K. Sjolander, D.-Haussler, et R. Underwood, *Stochastic context-free grammars for tRNA modeling*, Nuclear Acids Res. **22** (1994), 5112–5120.
- [SCvZ04] B. Starkie, F. Coste, et M. van Zaanen, *Omphalos context-free language learning competition*, 2004.
- [Sén97] G. Sénizergues, *The equivalence problem for deterministic push-down automata is decidable.*, Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7-11 July 1997, Proceedings (Pierpaolo Degano, Roberto Gorrieri, et Alberto Marchetti-Spaccamela, eds.), Lecture Notes in Computer Science, vol. 1256, Springer, 1997, p. 671–681.
- [Sén98] G. Sénizergues, *A polynomial algorithm testing partial confluence of basic semi-thue systems.*, Theor. Comput. Sci. **192** (1998), no. 1, 55–75.
- [SF04] Bradford Starkie et Henning Fernau, *The boisdale algorithm - an induction method for a subclass of unification grammar from positive data.*, ICGI (Georgios Paliouras et Yasubumi Sakakibara, eds.), Lecture Notes in Computer Science, vol. 3264, Springer, 2004, p. 235–247.
- [SG94] J. M. Sempere et P. García, *A characterisation of even linear languages and its application to the learning problem*, in Carrasco and Oncina [CO94], p. 38–44.
- [Shi85] S. M. Shieber, *Evidence against the context-freeness of natural language*, Linguistics and Philosophy **8** (1985), 333–343.
- [SK99] Y. Sakakibara et M. Kondo, *Ga-based learning of context-free grammars using tabular representations*, Proceedings of 16th International Conference on Machine Learning (ICML-99), 1999, p. 354–360.
- [ST00] M. Steinby et W. Thomas, *Trees and term rewriting in 1910: On a paper by axel thue*, Bulletin of the European Association for Theoretical Computer Science **72** (2000), 256+.
- [Sto95] A. Stolcke, *An efficient probabilistic context-free parsing algorithm that computes prefix probabilities*, Comp. Linguistics **21** (1995), no. 2, 165–201.
- [Tak88] Y. Takada, *Grammatical inference for even linear languages based on control sets*, Information Processing Letters **28** (1988), no. 4, 193–199.
- [Tak94] ———, *A hierarchy of language families learnable by regular language learners*, in Carrasco and Oncina [CO94], p. 16–24.

- [TDdlH00] F. Thollard, P. Dupont, et C. de la Higuera, *Probabilistic DFA inference using kullback-leibler divergence and minimality*, Proc. 17th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA, 2000, p. 975–982.
- [Thu14] A. Thue, *probleme über veränderungen von zeichenreihen nach gegebenen regeln*, Kra. Vidensk. Selsk. Skrifter. I. Mat.Nat Kl. (1914).
- [TN90] A. Togashi et S. Noguchi, *Inductive inference of term rewriting systems realizing algebras*, Algorithm Learning Theory, 1990, p. 411–424.
- [Val84] L. G. Valiant, *A theory of the learnable*, Communications of the Association for Computing Machinery **27** (1984), no. 11, 1134–1142.
- [Vap98] V. Vapnik, *Statistical learning theory*, Wiley-Interscience, 1998.
- [vD35] Walther von Dyck, *Jahresberichte der deutschen mathematiker*, Vereinigung **45** (1935), 89–98.
- [WAL⁺90] P. Weis, M.-V. Aponte, A. Laville, M. Mauny, et A. Suárez, *The CAML reference manual*, Tech. Report 121, INRIA, 1990.
- [Yok89] T. Yokomori, *Learning context-free languages efficiently: a report on recent results in Japan*, Analogical and Inductive Inference: Proc. of the International Workshop AII'89 (K. P. Jantke, ed.), Springer-Verlag, Berlin, Heidelberg, 1989, p. 104–123.
- [Yok91] ———, *Polynomial time learning of very simple grammars from positive data*, Proceedings of COLT '91, 1991, p. 213–227.
- [Yok95] ———, *On polynomial-time learnability in the limit of strictly deterministic automata*, Machine Learning Journal **19** (1995), 153–179.
- [Yok03] ———, *Polynomial-time identification of very simple grammars from positive data*, Theor. Comput. Sci. **1** (2003), no. 298, 179–206.
- [You67] D. H. Younger, *Recognition and parsing of context-free languages in time n^3* , Information and Control **2** (1967), no. 10, 189–208.

Liste des Algorithmes

3.1	SEQUITUR (taille des motifs égale à 2)	53
3.2	RT	57
4.1	LARS (Learning Algorithm for Rewriting Systems)	74
5.1	SGL-G (Substitution Graph Learner for Grammars)	94
5.2	SGL-RS (Substitution Graph Learner for Reduction Systems)	99

Résumé

L'inférence grammaticale a pour but l'apprentissage automatique de langages formels. Jusqu'à récemment, l'attention des chercheurs s'est principalement focalisée sur les langages réguliers. Mais il est plus difficile de s'attaquer à l'inférence des langages hors-contextes, la classe de complexité suivante dans la hiérarchie de Chomsky. En effet, des barrières importantes ont été découvertes : plusieurs résultats théoriques montrent l'impossibilité d'apprendre l'intégralité de cette classe. Pourtant dans de nombreux domaines, allant de la génétique au traitement des langues naturelles, en passant par la compression de texte, ce type de langages est nécessaire.

Dans cette thèse, nous commençons par analyser et par classer les difficultés inhérentes à l'inférence des hors-contextes. Nous montrons ensuite comment, dans la littérature, les auteurs ont tenté de contourner ces difficultés, avec plus ou moins de succès. Nous étudions, en particulier, une solution consistant à modifier un algorithme de compression afin de structurer des exemples d'apprentissage (extraction de squelettes) ; ces exemples peuvent ensuite être utilisés par l'algorithme de Sakakibara (1992) qui permet théoriquement d'identifier l'intégralité des hors-contextes à partir de squelettes. Or nous montrons que cette approche ne peut pas être fructueuse, ce qui limite, du même coup, le résultat de Sakakibara aux données intrinsèquement structurées.

Nous présentons ensuite deux contributions originales visant à contourner les écueils de l'apprentissage des hors-contextes. Notre première approche repose sur un changement de représentation : nous utilisons des systèmes spécialisés de réécriture de mots pour représenter les hors-contextes ; nous étudions ensuite les bonnes propriétés d'un nouvel algorithme d'apprentissage, tant d'un point de vue formel (identification polynomiale à partir d'exemples positifs et négatifs) qu'expérimental. Notre seconde approche consiste à restreindre la classe des hors-contextes étudiés : nous définissons les langages substituables, et proposons un algorithme capable de les identifier polynomialement à partir d'exemples positifs seulement. Nous développons enfin une application à un problème de modélisation de l'acquisition par l'enfant de sa langue maternelle.