INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Inférence grammaticale régulière : fondements théoriques et principaux algorithmes

Pierre Dupont et Laurent Miclet

No 3449

Juillet 1998

THÈME 3

ISSN 0249-6399

INRIA
RENNES

Inférence grammaticale régulière : fondements théoriques et principaux algorithmes

Pierre Dupont * et Laurent Miclet †

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projet REPCO

Rapport de recherche n 3449 — Juillet 1998 — 80 pages

Résumé : L'objet de cette étude est l'apprentissage automatique d'une grammaire formelle à partir d'exemples. En particulier, nous nous concentrons sur l'inférence de grammaires régulières. Nous proposons une étude détaillée des bases théoriques et de l'espace de recherche de ce problème. Nous y démontrons des propriétés originales définissant un cadre formel et des contraintes pour la conception de nouveaux algorithmes. Ensuite, nous présentons un éventail d'algorithmes d'inférence en comparant la nature des données qu'ils utilisent, leur critère de généralisation et leur complexité calculatoire. Finalement, nous étudions l'inférence régulière sur base d'une présentation séquentielle des données d'apprentissage. Une extension incrémentale d'un algorithme existant est développée. Nous en démontrons la convergence et la complexité théorique.

Mots-clé : Inférence grammaticale, algorithmes d'apprentissage symbolique, apprentissage incrémental.

(Abstract: pto)

Une version préliminaire de ce rapport a servi de support écrit au cours de la 5ème école d'été sur le Traitement des Langues Naturelles, organisée à Trégastel (France), du 25 au 29 septembre 1995.

* EURISE – Département de Mathématiques, Université Jean Monnet. 23, rue du Docteur Paul Michelon 42023 Saint-Etienne Cedex 2 – France. Email : pdupont@univ-st-etienne.fr

† IRISA-ENSSAT – Technopola Anticipa, BP 447 – 6, rue de Kerampont 22305 Lannion Cedex – France. Email : miclet@enssat.fr

Unité de recherche INRIA Rennes

Regular grammar induction: theoretical foundations and algorithms

Abstract: In this work we study the automatic learning of a formal grammar from examples. In particular we focus on regular grammatical inference. We cover in detail the theoretical foundations and the definition of the search space of this problem. We prove new properties that define a framework and constraints which together serve to help the design of new algorithms. Next we present a set of existing inference algorithms. We compare them on the basis of the type of data they use, their generalization criteria and computational complexity. Finally we study grammatical inference from sequential presentation of learning data. An incremental extension of an existing algorithm is proposed. We prove its convergence and complexity.

Key-words: Grammar induction, symbolic learning algorithms, incremental learning.

Table des matières

Table des figures

Chapitre 1

Bases théoriques de l'inférence régulière

1.1 Introduction

Nous nous intéressons ici à l'apprentissage de modèles structurels pouvant être décrits par le biais d'une grammaire formelle. Ce problème relève de l'*inférence grammaticale* dans le sens classique du terme. Elle consiste en l'apprentissage d'une grammaire représentant un langage à partir d'un ensemble d'exemples. Cet ensemble est formé au moins d'un *échantillon positif*, c'est-à-dire un sous-ensemble fini d'un langage. Nous pouvons également disposer d'un *échantillon négatif*, c'est-à-dire un ensemble fini de chaînes n'appartenant pas au langage.

L'inférence grammaticale a été étudiée depuis le développement de la théorie des grammaires formelles. Outre son intérêt théorique, elle offre un ensemble d'applications potentielles, en particulier dans les domaines de la *Reconnaissance des Formes Syntaxiques et Structurelles* , le *Traitement de la Langue Naturelle* et le *Traitement de la Parole*. Nous nous limiterons ici à l'*inférence régulière*, c'est-à-dire l'apprentissage d'une grammaire représentant un langage supposé régulier. En effet, c'est dans ce cadre qu'ont été menés la plupart des travaux jusqu'à ce jour.

Nous rappelons au paragraphe 1.2 les notions algébriques de base qui nous permettent de définir l'espace des solutions possibles de l'inférence régulière: langages réguliers, automates finis, partitions d'un ensemble fini, dérivation d'un automate relativement à une partition de son ensemble d'états, treillis d'automates. En particulier, nous étendons la notion classique de *complétude structurelle* d'un échantillon relatif à un automate.

La caractérisation et l'évaluation d'une méthode d'inférence requiert un cadre formel définissant l'objet de l'apprentissage. Nous présentons au paragraphe 1.3 quelques critères d'inférence correcte. En particulier, nous détaillons l'identification exacte d'un langage telle qu'elle a été définie par Gold [Gol67, Gol78].

Nous rappelons au paragraphe 1.4 les principaux résultats de la complexité théorique d'identification d'un langage régulier [Pit89]. Bien que l'identification exacte d'un langage régulier à partir d'échantillons positif et négatif constitue un problème NP-difficile, nous insistons sur le fait qu'une identification quasi-exacte puisse être obtenue en temps polynomial [Lan92, OG92].

Nous consacrons le paragraphe 1.5 à une description formelle de l'inférence régulière. Elle est vue comme un problème de *recherche explicite* d'une généralisation de l'échantillon positif. Nous revoyons ici les théorèmes fondamentaux conformément à notre définition de complétude structurelle. Il en résulte que l'espace de recherche est un treillis d'automates, construit sur la base de l'échantillon positif. Nous étudions certaines propriétés de ce treillis. En particulier, nous montrons que le treillis construit à partir de l' *automate canonique maximal* est plus général que celui construit à partir de l'*accepteur des préfixes*.

Nous définissons ensuite la notion d'*ensemble frontière* (paragraphe 1.6), c'est-à-dire l'ensemble des solutions compatibles les plus générales, et nous étudions quelques propriétés de cet ensemble. Notons que nous avons présenté les principaux résultats des paragraphes 1.5 et 1.6 dans [DMV94].

Finalement, nous démontrons au paragraphe 1.7 un autre résultat original. Il s'agit de l'existence d'un demi-treillis constitué de tous les automates déterministes du treillis associé à l'accepteur des préfixes.

1.2 Définitions et notations

Le lecteur familier avec les définitions classiques de la théorie des automates peut passer le paragraphe 1.2.1. Nous présentons ces notions par souci de complétude et afin de spécifier les notations utilisées par la suite.

1.2.1 Langages, automates et partitions

Notions de base

Soit Σ un *alphabet fini* et les *lettres* a, b, c des éléments de Σ. Nous désignons par u, v, w, x des éléments de Σ^*, c'est-à-dire des chaînes de longueur finie sur Σ, et par λ la *chaîne vide* . $|u|$ désigne la longueur de la chaîne u.

Définition 1.1 Nous disons que u est un *préfixe* de v s'il existe w tel que $uw = v$.

Définition 1.2 Nous disons que u est un *suffixe* de v s'il existe w tel que $wu = v$.

Définition 1.3 Un langage L est un sous-ensemble quelconque de Σ^*. Les éléments de L sont des chaînes qui, dans ce contexte, sont généralement appelées *mots*[1].

Définition 1.4 Soit $Pr(L) = \{u | \exists v, uv \in L\}$ l'ensemble des préfixes de L. Nous désignons par $L/u = \{v | uv \in L\}$ le *quotient-droit* de L par u, encore appelé *ensemble des finales* de u. Nous avons donc $L/u \neq \phi$ si et seulement si $u \in Pr(L)$.

Automates finis

Définition 1.5 Un *automate fini* est un quintuple $(Q, \Sigma, \delta, q_0, F)$ où Q est un ensemble fini d'*états*, Σ est un *alphabet fini*, δ est une *fonction de transition*, c'est-à-dire une application de $Q \times \Sigma \rightarrow 2^Q$, q_0 est l'*état initial* et F est un sous-ensemble de Q identifiant les *états finaux* ou *d'acceptation*.

Définition 1.6 Si, pour tout q de Q et tout a de Σ, $\delta(q, a)$ contient au plus un élément (respectivement exactement un élément), l'automate A est dit *déterministe* (respectivement *complet*). Par la suite, nous utiliserons l'abréviation **AFD** pour un "automate fini déterministe" et **AFN** pour un "automate fini non-déterministe".

1. Ces désignations sont classiques en théorie des langages. Elles peuvent néanmoins induire une confusion. Si nous nous référons aux appelations classiques en traitement de la langue naturelle, les mots ou entrées lexicales correspondent ici aux lettres et les phrases correspondent ici aux mots.

Considérons, par exemple, l'automate A_1, représenté à la figure 1.1. Il comporte 5 états, $Q = \{0, 1, 2, 3, 4\}$. Il est défini sur un alphabet de 2 lettres, $\Sigma = \{a, b\}$. L'état initial[2] q_0 est l'état 0 et les états 3 et 4 sont finaux[3], $F = \{3, 4\}$. La fonction de transition est représentée par les arcs reliant les états. Il s'agit ici d'un automate non déterministe car il a deux arcs étiquetés a à partir de l'état 0. En d'autres termes, $\delta(0, a) = \{1, 2\}$.

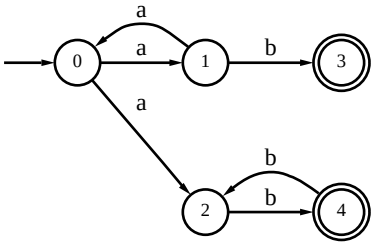


FIG. 1.1: *Une représentation de l'automate A_1.*

Langage accepté par un automate fini

Définition 1.7 Une *acceptation* d'une chaîne $u = a_1 \dots a_l$ par un automate A, éventuellement non-déterministe, définit une séquence, éventuellement non unique, de $l + 1$ états $(q^0, \dots, q^l)$ telle que $q^0 = q_0$, $q^l \in F$ et $q^{i+1} \in \delta(q^i, a_{i+1})$, pour $0 \leq i \leq l \Leftrightarrow 1$. Les $l + 1$ états sont dits *atteints* pour cette acceptation et l'état q^l est *utilisé comme état d'acceptation*. De façon similaire, les l transitions, c'est-à-dire des éléments de δ, sont dites *exercées* par cette acceptation.

Par exemple, la séquence des états $(0, 1, 0, 2, 4, 2, 4)$ correspond à une acceptation de la chaîne $aaabbb$ dans l'automate A_1.

Définition 1.8 Un automate A est *ambigu* s'il existe au moins une chaîne u pour laquelle il y a plusieurs acceptations. Dans ce cas, l'automate A est nécessairement non-déterministe. Un ensemble de transitions exercées par un ensemble de chaînes S_s est l'union des ensembles de transitions *exercées* par une acceptation de chaque chaîne de S_s. Nous dirons que les états associés sont *atteints* par une acceptation de S_s.

L'automate A_1 est ambigu car la chaîne ab peut être acceptée par les séquences d'états $(0, 1, 3)$ et $(0, 2, 4)$.

Définition 1.9 Le langage $L(A)$ accepté par un automate A est l'ensemble des chaînes acceptées par A. Le langage L est accepté par un automate A si et seulement s'il constitue un *ensemble régulier*, c'est-à-dire qu'il peut être défini par une *expression régulière*[4] [AU72].

Dans la suite de ce document, un automate signifiera, par défaut, un automate *fini* et un langage signifiera, par défaut, un langage *régulier*.

Définition 1.10 La différence symétrique $L1 \oplus L2$ des langages $L1$ et $L2$ est définie comme suit :

$$L1 \oplus L2 = (L1 \cup L2) \setminus (L1 \cap L2).$$

2. L'état initial est représenté graphiquement par $\rightarrow$(0).

3. Les états finaux sont représentés graphiquement par ◎.

4. Nous supposons le lecteur familier avec les notations utilisées pour spécifier les expressions régulières [AU72].

Définition 1.11 Un état q d'un automate A est *utile* s'il existe une chaîne u de $L(A)$ pour laquelle l'état q peut être atteint pour une acceptation de u. Sinon, l'état q est *inutile*. Un automate *élagué* ne contient aucun état inutile.

Définition 1.12 L'automate $A(L) = (Q, \Sigma, \delta, q_0, F)$ désigne l'AFD qui possède le nombre minimal d'états pour un langage L donné. $A(L)$ est généralement appelé *automate déterministe minimal* ou *automate canonique* de L, et peut être défini comme suit :

$Q = \{L/u \mid u \in Pr(L)\}$,
$q_0 = L/\lambda$,
$F = \{L/u \mid u \in L\}$,
$\delta(L/u, a) = L/ua$ où $u, ua \in Pr(L)$.

On démontre que $A(L)$ est unique à une permutation près des indices de ses états [AU72]. Tout automate déterministe acceptant exactement L et possédant le nombre minimal d'états pour ce langage, est donc *isomorphe* à $A(L)$. Par la suite, nous parlerons indifféremment de l'automate canonique ou de l'automate déterministe minimal de L.

Par exemple, l'automate canonique représenté à la figure 1.2 accepte le langage défini par l'expression régulière $L = a(aa)^*b(bb)^*$. Il s'agit du langage accepté également par l'automate A_1 de la figure 1.1. Il n'existe pas d'automates déterministes comportant moins d'états et acceptant ce langage.

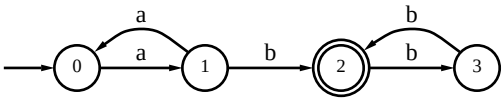


FIG. 1.2: *Automate canonique du langage* $L = a(aa)^*b(bb)^*$.

Automates dérivés

Définition 1.13 Pour tout ensemble S, une partition π est un ensemble de sous-ensembles de S, non vides et disjoints deux à deux, dont l'union est S. Si s désigne un élément de S, $B(s, \pi)$ désigne l'unique élément, ou *bloc*, de π comprenant s. Une partition π_i *raffine*, ou *est plus fine que*, une partition π_j si et seulement si tout bloc de π_j est un bloc de π_i ou est l'union de plusieurs blocs de π_i.

Définition 1.14 Si $A = (Q, \Sigma, \delta, q_0, F)$ est un automate, l'automate $A/\pi = (Q', \Sigma, \delta', B(q_0, \pi), F')$ *dérivé de A relativement à la partition π* de Q, aussi appelé l'*automate quotient* A/π, est défini comme suit :

$Q' = Q/\pi = \{B(q, \pi) | q \in Q\}$,
$F' = \{B \in Q' | B \cap F \neq \phi\}$,
$\delta' : Q' \times \Sigma \to 2^{Q'} : \forall B, B' \in Q', \forall a \in \Sigma, B' \in \delta'(B, a)$ si et seulement si
$\exists q, q' \in Q, q \in B, q' \in B'$ et $q' \in \delta(q, a)$.

Nous dirons que les états de Q appartenant au même bloc B de la partition π sont *fusionnés*.

Si nous reprenons l'automate A_1, représenté à la figure 1.1, et que nous définissons la partition de son ensemble d'états, $\pi = \{\{0, 1\}, \{2\}, \{3, 4\}\}$. L'automate quotient A_1/π, obtenu en fusionnant tous les états appartenant à un même bloc de π, est représenté à la figure 1.3[5].

5. Remarquons qu'il ne s'agit pas d'un automate quotient de l'automate canonique, acceptant le même langage, représenté à la figure 1.2.

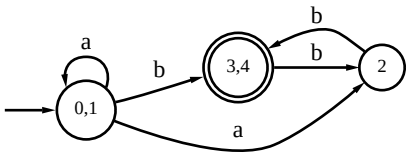


FIG. 1.3: *L'automate quotient* A_1/π.

1.2.2 Echantillons d'un langage et automates associés

Définition 1.15 Nous désignons par I_+ un sous-ensemble fini, appelé *échantillon positif*, d'un langage L quelconque. Nous désignons par I_- un sous-ensemble fini, appelé *échantillon négatif*, du langage complémentaire $\Sigma^* \Leftrightarrow L$. Par conséquent, I_+ et I_- sont des sous-ensembles finis et disjoints de Σ^*.

Complétude structurelle

Définition 1.16 Un échantillon I_+ est *structurellement complet* relativement à un automate A acceptant L, s'il existe une acceptation $\mathcal{AC}(I_+, A)$ de I_+ telle que :

(1) toute transition de A soit exercée.
(2) tout élément de F (l'ensemble des états finaux de A) soit utilisé comme état d'acceptation.

Les définitions classiques de la complétude structurelle [FB75a, Mic79, Mic80, Ang82] n'incluent pas la seconde condition. Nous verrons au paragraphe 1.5.1 qu'elle est cependant nécessaire.

A titre d'exemple, l'échantillon $I_+ = \{aaab, ab, abbbbb\}$ est structurellement complet relativement à l'automate A_1.

Automate Canonique Maximal, Arbre Accepteur des Préfixes et Automate Universel

Soit $I_+ = \{u_1, \ldots, u_M\}$ un échantillon positif, où $u_i = a_{i,1} \ldots a_{i,|u_i|}, 1 \leq i \leq M = |I_+|$.

Définition 1.17 Nous désignons par $MCA(I_+) = (Q, \Sigma, \delta, q_0, F)$ l'*automate maximal canonique*[6] *relatif à* I_+ [Mic79]. Il est construit comme suit :

Σ est l'alphabet sur lequel I_+ est défini.
$Q = \{v_{i,j} | 1 \leq i \leq M, 1 \leq j \leq |u_i|, v_{i,j} = a_{i,1} \ldots a_{i,j}\} \cup \{\lambda\}$,
$q_0 = \lambda$,
$F = I_+$,
$\delta(\lambda, a) = \{a_{i,1} | a_{i,1} = a, 1 \leq i \leq M\}$,
$\delta(v_{i,j}, a) = \{v_{i,j+1} | v_{i,j+1} = v_{i,j}a\}, 1 \leq i \leq M, 1 \leq j \leq |u_i| \Leftrightarrow 1$.

Par conséquent, $L(MCA(I_+)) = I_+$ et $MCA(I_+)$ est le plus grand automate, c'est-à-dire l'automate ayant le plus grand nombre d'états, pour lequel I_+ est structurellement complet. Remarquons que $MCA(I_+)$ est généralement non-déterministe.

6. L'abréviation **MCA** pour "Maximal Canonical Automaton" provient de la terminologie anglaise. Le qualificatif *canonique* se rapporte ici à un échantillon. Le MCA ne doit pas être confondu avec l'automate canonique d'un langage (voir déf. 1.12).

A titre d'exemple, l'automate représenté à la figure 1.4 est l'automate maximal canonique relatif à l'échantillon $I_+ = \{a, ab, bab\}$.

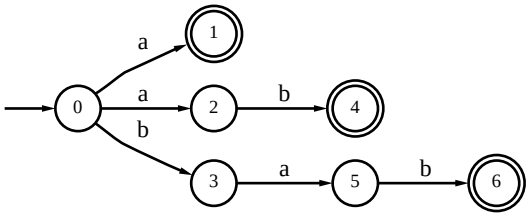


FIG. 1.4: *L'automate maximal canonique $MCA(I_+)$, avec $I_+ = \{a, ab, bab\}$.*

Définition 1.18 Nous désignons par $PTA(I_+)$ l'*arbre accepteur des préfixes*[7] de I_+ [Ang82]. Il s'agit de l'automate quotient $MCA(I_+)/\pi_{I_+}$ où la partition π_{I_+} est définie comme suit :

$$B(q, \pi_{I_+}) = B(q', \pi_{I_+}) \text{ si et seulement si } Pr(q) = Pr(q').$$

En d'autres termes, le $PTA(I_+)$ peut être obtenu à partir du $MCA(I_+)$ en fusionnant les états partageant les mêmes préfixes. Remarquons que le $PTA(I_+)$ est déterministe[8].

A titre d'exemple, l'automate représenté à la figure 1.5 est l'arbre accepteur des préfixes relatif à l'échantillon $I_+ = \{a, ab, bab\}$.

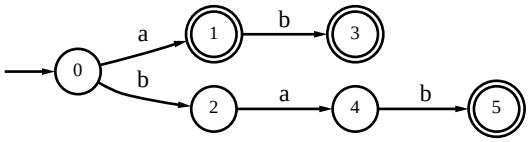


FIG. 1.5: *L'arbre accepteur des préfixes $PTA(I_+)$, avec $I_+ = \{a, ab, bab\}$.*

Définition 1.19 Nous désignons par UA *l'automate universel*[9]. Il accepte toutes les chaînes définies sur l'alphabet Σ, c'est-à-dire $L(UA) = \Sigma^*$. Il s'agit donc du plus petit automate pour lequel des échantillons de Σ^* peuvent être structurellement complets.

L'automate universel défini sur l'alphabet $\Sigma = \{a, b\}$ est représenté à la figure 1.6.

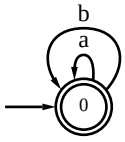


FIG. 1.6: *L'automate universel sur l'alphabet $\Sigma = \{a, b\}$.*

7. L'abréviation **PTA** pour "Prefix Tree Acceptor" provient de la terminologie anglaise.

8. Puisque le $PTA(I_+)$ est l'automate déterministe acceptant I_+ et comprenant le plus grand nombre d'états, il pourrait être appelé *automate maximal déterministe*.

9. L'abréviation **UA** pour "Universal Automaton" provient de la terminologie anglaise.

1.2.3 Treillis d'automates

Définition 1.20 Nous désignons par $P(A)$ l'ensemble des partitions de l'ensemble des états d'un automate A. Soit $r(\pi_i)$, ou simplement r_i, le nombre de blocs de la partition π_i. Supposons que $\pi_1 = \{B_{11}, \ldots, B_{1r_1}\}$ et π_2 soient deux éléments de $P(A)$. La partition π_2 *dérive directement de* π_1 s'il existe j et k, compris entre 1 et r_1 ($j \neq k$), tels que

$$\pi_2 = \{B_{1j} \cup B_{1k}\} \cup (\pi_1 \setminus \{B_{1j}, B_{1k}\}),$$

Par conséquent, $r_2 = r_1 \Leftrightarrow 1$.

Cette opération de dérivation définit une relation d'ordre partiel sur $P(A)$. Nous désignons cette relation par $\preceq$. En particulier, nous pouvons écrire $\pi_1 \preceq \pi_2$. Nous désignons également la fermeture transitive de cette relation par $\ll$. En d'autres termes, nous avons $\pi_i \ll \pi_j$ si et seulement si π_i est plus fine que π_j. Par extension, nous disons que A/π_i est *plus fin que* A/π_j ou, réciproquement, que A/π_j *dérive de* A/π_i.

Par construction d'un automate quotient, nous pouvons à présent énoncer la propriété d'inclusion des langages réguliers [FB75a] :

Propriété 1.1 *Si un automate A/π_j dérive d'un automate A/π_i, alors le langage accepté par A/π_i est inclus dans celui accepté par A/π_j. Formellement, nous avons :*

$$\textit{Si } A/\pi_i \ll A/\pi_j \textit{ alors } L(A/\pi_i) \subseteq L(A/\pi_j).$$

Définition 1.21 L'ensemble des automates partiellement ordonné par la relation $\preceq$ est un *treillis complet complémenté*[10], que nous désignons par $Lat(A)$[11][HS66]. Les automates A et UA, l'automate universel, en sont respectivement les éléments nul et universel.

Définition 1.22 La *profondeur* d'un automate A/π dans $Lat(A)$ est donnée par la formule $N \Leftrightarrow r(\pi)$, où N désigne le nombre d'états de A. Dès lors, la profondeur de l'automate A dans $Lat(A)$ est nulle tandis que la profondeur de l'automate universel UA est égale à $N \Leftrightarrow 1$.

1.3 Objet de l'inférence

Une définition informelle de l'*apprentissage* est, par exemple, "l'acquisition d'une capacité à réaliser une tâche". L' *induction,* ou *inférence inductive,* constitue un type particulier d'apprentissage où un "raisonnement" sur quelques cas observés est appliqué à un ensemble "plus général" de cas. Une définition précise de l'induction requiert une caractérisation des généralisations recherchées. Celles-ci répondent à des critères formellement établis qui définissent ce qu'est un apprentissage correct ou réussi.

L'inférence grammaticale est une instance de l'inférence inductive. Il s'agit d'apprendre une grammaire qui constitue une représentation possible d'un langage, à partir d'exemples de ce langage. En d'autres termes, les cas observés sont ici des exemples, ou chaînes, appartenant à un langage inconnu et la généralisation est le langage complet représenté par la grammaire.

Dans la suite de ce paragraphe, nous nous inspirons de l'article de Angluin et Smith sur les méthodes d'inférence inductive [AS83].

10. Un *treillis* est un ensemble ordonné E dans lequel tout couple d'éléments admet un plus petit majorant et un plus grand minorant.
Un treillis E est *complet* si tout sous-ensemble E' de E constitue également un treillis.
Un treillis E est *complémenté* s'il possède un élément nul et un élément universel et si tout élément x de E admet au moins un complément.

11. L'abréviation *Lat* pour "Lattice" provient de la terminologie anglaise.

1.3.1 La spécification d'un problème d'inférence grammaticale

Afin de définir un problème d'inférence grammaticale, cinq points doivent être spécifiés :

- La *classe de grammaires* à inférer. Dans cette partie de notre travail, il s'agit essentiellement de la classe des grammaires *régulières*[12] ou d'une de ses sous-classes.
- L'*espace d'hypothèses*, c'est-à-dire l'ensemble des descriptions tel que chaque grammaire de la classe considérée possède au moins une description dans l'espace. Pour les grammaires régulières, il peut s'agir de l'espace des *automates*, des *expressions régulières* ou des *ensembles réguliers*.
- Un *ensemble d'exemples*, pour chaque grammaire à inférer et un protocole de présentation de ces exemples. En particulier, nous définissons au paragraphe 1.3.2 les présentations dites *admissibles*.
- La *classe des méthodes d'inférence* en considération. Il s'agit en particulier des méthodes constructives ou heuristiques (voir paragraphe 1.1).
- Les critères d'une inférence réussie, définis aux paragraphes 1.3.3 et 1.3.4.

Une *méthode d'inférence* est alors un processus calculable qui lit des exemples conformément à une présentation admissible et propose des solutions dans l'espace des hypothèses.

1.3.2 Présentations admissibles

Nous allons à présent définir trois protocoles de présentation d'exemples d'un langage. Remarquons que ces définitions supposent que les langages considérés soient de cardinalité infinie.

Définition 1.23 Une *présentation positive* d'un langage L est une séquence infinie d'exemples, comportant au moins une occurrence de chaque élément de L. On parle également d'*exemples positifs* du langage L.

Définition 1.24 Une *présentation négative* d'un langage L est une séquence infinie d'exemples comportant au moins une occurrence de chaque élément de $\Sigma^* \Leftrightarrow L$. On parle indifféremment d'*exemples négatifs* ou de *contre-exemples* du langage L.

Définition 1.25 Une *présentation complète* d'un langage L est une séquence infinie ordonnée de paires (x, d) de $\Sigma^* \times \{0, 1\}$. Tout élément x de Σ^* apparaît comme premier argument d'une paire de la séquence et $d = 1$ si et seulement si $x \in L$, $d = 0$ sinon.

Un échantillon positif étant un ensemble fini d'éléments d'un langage, nous pouvons considérer une présentation positive comme une séquence infinie d'échantillons positifs inclus et de taille croissante. Une méthode d'inférence séquentielle lit les exemples présentés un par un et propose une nouvelle solution dans l'espace d'hypothèses après chaque nouvel exemple lu. Notons que telles qu'elles ont été définies les présentations ne supposent pas d'ordre particulier sur les exemples. Ces présentations sont dénommées *admissibles*. Une présentation *non admissible* pourrait alors être définie récursivement de la manière suivante : toute séquence d'exemples telle que l'exemple suivant

12. Les grammaires régulières sont celles qui, par définition, engendrent ou acceptent un langage régulier (voir paragraphe 1.2).

présenté contredise nécessairement la solution proposée par la méthode d'inférence, sur base des exemples déjà lus.

Une autre présentation admissible d'exemples d'un langage est la *présentation par oracle*. Un oracle est, par définition, capable de répondre à des questions concernant le langage. Par exemple, la méthode d'inférence propose un exemple et l'oracle répond à une *question d'appartenance*, c'est-à-dire s'il s'agit ou non d'un élément du langage. En théorie, cette présentation par oracle est équivalente à une présentation complète mais ce type de présentation modifie profondément la conception des algorithmes d'inférence. En effet, nous ne disposons en pratique que d'un sous-ensemble *fini* d'une présentation positive ou complète. L'existence d'un oracle permet alors de classifier, à la demande, des exemples n'appartenant pas aux échantillons observés.

1.3.3 L'identification à la limite

L'*identification à la limite* a été proposée par Gold [Gol67]. Il s'agit d'un critère d'inférence correcte défini par le biais d'un protocole théorique d'inférence. De nombreux résultats découlent de ce critère, tant sur la possibilité d'identifier ou non un langage par induction, que de la complexité calculatoire de ce processus d'identification [Gol78, Pit89]. En outre, ces travaux constituent une justification de l'utilisation d'échantillons négatifs.

Le protocole d'identification à la limite est défini comme suit. Soit M une méthode d'inférence et $S = x_1, x_2, \ldots$ une présentation admissible d'exemples d'un langage L. Nous notons S_i le sous-ensemble fini des i premiers éléments de S. La méthode d'inférence doit proposer une solution dans l'espace d'hypothèses après chaque nouvel exemple lu. Soit $H(S_i)$, l'hypothèse proposée par la méthode M après avoir lu les i premiers exemples présentés.

Définition 1.26 La méthode d'inférence M *identifie à la limite* le langage L, s'il existe un indice fini j, tel que

- $H(S_i) = H(S_j)$, pour tout $i \geq j$.
- $L(H(S_j)) = L$.

La première condition impose que la méthode converge à partir d'un indice fini, la seconde qu'elle converge vers **une** représentation de la solution correcte. Nous appelons *point de convergence* pour la présentation S, l'indice fini j à partir duquel la convergence est atteinte.

Définition 1.27 Une méthode d'inférence M identifie à la limite une classe $\mathcal{C}$ de langages, si M identifie à la limite n'importe quel langage L de la classe $\mathcal{C}$ pour n'importe quelle présentation admissible d'exemples de L.

Ces définitions correspondent à ce que nous désignerons par la suite comme l'*identification exacte* d'un langage ou d'une classe de langages. Nous verrons au paragraphe 1.3.5 que le type de présentation des exemples influencent les possibilités d'identification.

L'abréviation EX désigne la classe de tous les ensembles U de fonctions récursives totales[13] telle qu'il existe une méthode d'inférence qui identifie exactement U à la limite. Remarquons que l'ensemble de toutes les fonctions récursives totales ne peut pas être identifié à la limite [Gol67].

13. Les fonctions récursives totales sont les fonctions entières calculables par des Machines de Turing qui s'arrêtent pour toute entrée.

1.3.4 Autres critères d'identification

Malgré la popularité du critère d'identification à la limite, de nombreux autres critères d'identification ont été proposés. Ils constituent généralement une relaxation de l'identification exacte. Nous en présentons quelques-uns ci-après.

BC-identification

L'appellation *BC-identification* provient de la terminologie anglaise pour "Behaviorally Correct Identification". Ce critère n'impose pas que la méthode d'inférence converge vers une seule hypothèse dans l'espace des hypothèses mais, qu'à partir du point de convergence, toutes les hypothèses proposées constituent une description possible de la solution correcte. La classe BC est une sous-classe stricte de la classe EX [Bar74, CS83].

ε-identification

Ce critère d'identification est dû à Wharton [Wha74]. Il définit une fonction de pondération des chaînes $w : \Sigma^* \rightarrow \mathbb{R}$. Chaque chaîne x de Σ^* est affectée d'un *poids* de telle sorte que la somme des poids vaut l'unité. On définit alors une distance $d(L_1, L_2)$ entre deux langages comme la somme des poids des chaînes dans leur différence symétrique $L_1 \oplus L_2$. Cette distance définit une métrique sur l'espace des langages.

Définition 1.28 Pour tout $\varepsilon > 0$, une méthode d'inférence M *ε-identifie* un langage L, si et seulement si elle converge vers une grammaire G telle que $d(L(G), L) < \varepsilon$.

Un résultat positif de cette théorie est que pour tout ε, la classe de tous les langages définis sur un alphabet fixé est ε-identifiable à la limite à partir de présentations positives par une méthode énumérative qui produit des grammaires engendrant des langages finis [Wha77].

EX^k-identification

Cette relaxation de l'identification exacte concerne le nombre maximal d' *anomalies* entre le langage proposé après convergence et le langage correct.

Définition 1.29 Pour tout entier naturel k, une méthode d'inférence M identifie à la limite un langage L avec au plus k anomalies, si et seulement si elle converge vers une grammaire G, telle que la différence symétrique $L(G) \oplus L$ contienne au plus k éléments.

L'abréviation EX^k désigne la classe de tous les ensembles U de fonctions récursives telle qu'il existe une méthode d'inférence qui identifie U à la limite avec au plus k anomalies. Par conséquent, $EX^0 = EX$. De façon similaire, EX^* désigne la classe de tous les ensembles U de fonctions récursives telle qu'il existe une méthode d'inférence qui identifie U à la limite avec un nombre fini d'anomalies. Case et Smith ont montré que $EX^0, EX^1, EX^2, \ldots$ est une hiérarchie stricte dont l'union est strictement inclue dans EX^* [CS83].

1.3.5 Influence du type de présentations

Nous étudions dans ce paragraphe l'effet des différents types de présentation sur les possibilités d'identification exacte d'un langage [Gol67].

Définition 1.30 Une classe *superfinie* de langages est une classe qui contient tous les langages de cardinalité finie et au moins un langage de cardinalité infinie.

Propriété 1.2 *Aucune classe superfinie de langages n'est identifiable à la limite à partir de présentations positives [Gol67].*

Cette propriété implique en particulier que la classe des langages réguliers n'est pas identifiable à la limite à partir de présentations positives.

Propriété 1.3 *Toute classe énumérable de langages récursifs est identifiable à la limite à partir de présentations complètes [Gol67].*

Cette classe inclut la classe des langages sensibles au contexte et donc également celle des langages réguliers. Ce résultat justifie l'utilisation d'échantillons négatifs car il assure que, pour peu que les échantillons soient suffisamment grands, une identification exacte de n'importe quel langage régulier est possible. Remarquons cependant que l'identification exacte a une complexité calculatoire non-polynomiale. Cette question sera étudiée plus en détail au paragraphe 1.4.

1.3.6 Identification par utilisation de questions d'équivalence

Angluin a proposé un autre protocole théorique d'identification d'un langage L sur base d'une présentation par oracle. A chaque étape, la méthode d'inférence propose une hypothèse H qui est une description possible d'un langage, désigné par $L(H)$. Une *question d'équivalence* est alors posée à l'oracle : si $L(H) = L$, l'oracle répond "oui" et l'identification est atteinte, sinon l'oracle propose un contre-exemple, c'est-à-dire un élément de $L(H) \oplus L$ [Ang87].

Les résultats sur la complexité de l'identification par oracle avec questions d'équivalence ou d'appartenance sont présentés dans [Pit89]. Nous ne développerons pas plus ces résultats, étant donné leur limitation qui consiste à disposer, en pratique, d'un oracle.

1.3.7 Identification stochastique

Un langage *stochastique* est un langage L_s muni d'une distribution de probabilité D sur ses éléments [FB75b, GT78]. L'espace d'hypothèses d'un problème d'inférence stochastique est généralement la classe des *grammaires stochastiques*. Une *présentation stochastique* relative à un langage stochastique L_s est une séquence infinie d'exemples qui sont supposés avoir été générés conformément à la probabilité attribuée à ses éléments. Ce type de présentation ne comporte donc pas de contre-exemples explicites. Dans cette partie de notre exposé, nous ne détaillerons pas l'inférence d'une grammaire stochastique. Signalons simplement qu'elle répond à des critères d'identification qui lui sont propres. Angluin a démontré que, pour une distribution de probabilité D quelconque, si l'on désire garantir l'identification avec une probabilité 1, le problème se réduit à un problème d'identification classique [Ang88]. Par contre, elle démontre que si l'on se restreint à des distributions rationnelles [14] l'identification exacte peut être obtenue, avec probabilité 1, à partir d'une présentation

14. Plus précisément, Angluin démontre que l'on peut identifier à la limite des distributions supposées provenir d'une séquence de distributions approximables par des fonctions rationnelles [Ang88].

stochastique. En particulier, Horning montre que les langages engendrables par les grammaires hors-contexte stochastiques, dont les règles sont munies de probabilités rationnelles, peuvent être identifiés à la limite, avec une probabilité 1, à partir d'une présentation stochastique [Hor69].

1.3.8 Identification-PAC

Valiant a introduit la notion d'apprentissage PAC, pour *probablement approximativement correct* [Val84]. Dans le cas particulier de l'apprentissage d'automates déterministes[15], l'*identification-PAC* peut être définie comme suit[16] :

Définition 1.31 Les AFDs sont PAC-identifiables si et seulement s'il existe un algorithme A (éventuellement aléatoire) tel que, pour tout $\varepsilon > 0$, $\delta > 0$, pour tout AFD M de taille n, pour tout entier naturel m et pour toute distribution de probabilité D définie sur les chaînes de Σ^* de longueur au plus égale à m, si A reçoit en entrée des exemples classifiés relativement à M et générés aléatoirement conformément à la distribution D, alors A produit un AFD M' tel que, avec une probabilité au moins égale à $1 \Leftrightarrow\delta$, la probabilité de l'ensemble des chaînes de la différence symétrique $L(M) \oplus L(M')$ est au plus égale à ε. La complexité temporelle de A doit être polynomiale en fonction de $n, m, \frac{1}{\varepsilon}, \frac{1}{\delta}$, et $|\Sigma|$ [Pit89].

Il s'agit donc d'une relaxation de l'identification exacte, dans un double sens. D'une part, l'identification ne doit être garantie qu'avec une probabilité bornée par $1 \Leftrightarrow\delta$ et, d'autre part, un ensemble de chaînes peuvent être mal classifiées pour peu que la probabilité de l'ensemble de ces chaînes, conformément à la distribution D, soit bornée par ε. Par contre, le résultat doit être valable pour toute distribution D.

Ce cadre théorique est encore en plein développement. Remarquons que les résultats actuels sont plutôt négatifs [PW89]. Par ailleurs, les AFDs sont PAC-identifiables sur base d'une présentation par oracle et questions d'appartenance [Nat91].

1.4 Complexité calculatoire de l'identification d'un automate

Pitt a proposé un excellent article de synthèse sur la complexité calculatoire de l'inférence inductive des automates [Pit89]. Nous ne détaillerons pas ces aspects ici mais nous présentons quelques résultats qui devraient clarifier la complexité théorique et pratique de l'inférence d'un AFD à partir d'échantillons positif I_+ et négatif I_-.

Le problème du *plus petit AFD compatible* est défini comme suit. Il s'agit de trouver un AFD A tel que :

- $L(A) \supseteq I_+$,
- $L(A) \cap I_- = \emptyset$,
- A est un AFD comportant le nombre minimal d'états.

Nous pouvons énoncer les principaux résultats théoriques négatifs :

- Le problème du plus petit AFD compatible est NP-difficile [Ang78, Gol78].

15. La taille d'un automate déterministe est généralement définie comme son nombre d'états.

16. La définition présentée dans [Nat91] ne requiert pas la complexité polynomiale évoquée dans la définition de Pitt. Néanmoins, on s'intéresse, en pratique, aux algorithmes de complexité polynomiale.

– Si $P \neq NP$, il n'existe pas d'algorithme polynomial d'approximation qui garantirait de produire un AFD compatible dont la taille serait bornée par un quelconque polynôme en fonction de la taille de la solution optimale [PW89].

– L'inférence approchée d'automates à partir d'un sous-ensemble fini quelconque d'une présentation complète est NP-difficile si un adversaire choisit la machine à identifier et les données [KV89].

Par ailleurs, Trakhtenbrot et Barzdin proposent un algorithme par fusion[17] d'états de complexité $\mathcal{O}(mn^2)$ [TB73] pour construire le plus petit AFD compatible avec un échantillon *complet*, où m est la taille du $PTA(I_+)$ et n la taille de l'AFD *cible*, c'est-a-dire l'automate à identifier. Cet échantillon complet est constitué de toutes les chaînes de Σ^* de longueur au plus égal à l, chacune étant étiquetée comme exemple positif ou négatif. La longueur l dépend de certaines caractéristiques[18] de l'automate cible. Elle est au plus égale à $2n \Leftrightarrow 1$ [TB73]. Dans le pire des cas, la taille d'un échantillon complet croît donc exponentiellement avec la taille de l'automate cible. De plus, Angluin a démontré que, dans le pire des cas, l'identification exacte est impossible dès qu'une fraction arbitrairement petite de l'échantillon complet est manquante [Ang78].

Malgré les résultats négatifs énoncés, la complexité calculatoire semble être meilleure *en moyenne*. Pour des AFDs générés aléatoirement conformément à une distribution de probabilité prédéfinie, la valeur estimée de la taille d'un échantillon complet est [TB73] :

$$|S|_{complet} = (|\Sigma|^2 n^C \log_2(n \Leftrightarrow 1))/(|\Sigma| \Leftrightarrow 1)$$

où C ne dépend que de $|\Sigma|$.

La valeur estimée de la taille d'un échantillon complet pourrait suggérer que l'identification exacte d'AFDs est en moyenne envisageable, du moins pour des alphabets restreints. Cependant, il n'y a aucune garantie de disposer, en pratique, d'un échantillon complet, quelqu'en soit la taille. Seule la disponibilité de données *éparses* peut être supposée. Dans ce cadre, Lang a montré expérimentalement [Lan92] qu'une identification quasi-exacte peut être obtenue en n'utilisant qu'une faible fraction (aléatoirement générée) d'un échantillon complet. De plus, les résultats de Lang montrent que cette fraction décroît, en proportion, avec la croissance de la taille de l'automate cible.

L'algorithme polynomial proposé par Lang [Lan92], et indépendamment par Oncina *et al.* [OG92], est similaire à celui proposé par Trakhtenbrot *et al.* [TB73]. Cet algorithme sera présenté plus en détail au paragraphe 2.2.2 et au chapitre 3.

1.5 Espace de recherche

Nous pouvons définir l'inférence régulière comme la recherche d'un automate A inconnu à partir duquel un échantillon positif I_+ est supposé avoir été généré. Moyennant l'hypothèse supplémentaire de complétude structurelle de I_+ relativement à l'automate A, nous pouvons considérer ce problème comme la recherche à travers un treillis construit sur la base de I_+. Cette propriété résulte de deux théorèmes partiellement démontrés, respectivement dans [Mic79] et [Ang82], que nous avons revus dans [DMV94]. En particulier, nous montrons la nécessité de la seconde condition de la complétude structurelle de l'échantillon I_+ (voir définition 1.16).

17. Ce type d'algorithme sera exposé aux paragraphes 2.1 et 2.2

18. Ces caractéristiques sont le *degré de distingabilité* de l'automate et sa *profondeur* dans le sens défini dans [TB73].

1.5.1 Théorèmes fondamentaux

Théorème 1.1 *Soit I_+ un échantillon positif d'un quelconque langage régulier L et soit A n'importe quel automate acceptant exactement L . Si I_+ est structurellement complet relativement à A alors A appartient à $Lat(MCA(I_+))$.*

Preuve :
Nous allons construire le $MCA(I_+)$ à partir d'une acceptation de I_+ par A. Cette construction définit une partition π telle que A soit isomorphe à
$MCA(I_+)/\pi$.
Soit $I_+ = \{u_1, \ldots, u_M\}$, où $u_i = a_{i,1} \ldots a_{i,|u_i|}$, $1 \leq i \leq M$, un échantillon positif structurellement complet relativement à A. L'acceptation $\mathcal{AC}(I_+, A)$ définit pour chaque chaîne u_i une séquence $(q_A^{i,0}, \ldots, q_A^{i,|u_i|})$ de $|u_i| + 1$ états, où $q_A^{i,0} = q_{0_A}$, $q_A^{i,|u_i|} \in F_A$ et $q_A^{i,j+1} \in \delta_A(q_A^{i,j}, a_{i,j+1})$, $1 \leq i \leq M, 0 \leq j \leq |u_i| \Leftrightarrow 1$.
Nous commençons la construction du MCA en considérant son état initial, $q_{0_{MCA}}$ avec la définition : $q_{MCA}^{i,0} = q_{0_{MCA}},\ 1 \leq i \leq M$. Chaque fois qu'une transition de A est exercée, nous ajoutons un nouvel état dans Q_{MCA} et nous adaptons la fonction de transition δ_{MCA} comme suit:

$$q_A^{i,j+1} \in \delta_A(q_A^{i,j}, a_{i,j+1}) \Leftrightarrow q_{MCA}^{i,j+1} \in \delta_{MCA}(q_{MCA}^{i,j}, a_{i,j+1}),$$

pour $1 \leq i \leq M, 0 \leq j \leq |u_i| \Leftrightarrow 1$.
De plus, nous construisons l'ensemble des états d'acceptation F_{MCA} comme suit :

$$F_{MCA} = \{q_{MCA}^{i,|u_i|}, 1 \leq i \leq M\}.$$

Nous définissons également une fonction φ qui fait correspondre les états du MCA à ceux de A :

$$\varphi : Q_{MCA} \to Q_A : \varphi(q_{MCA}^{i,j}) = q_A, \text{ lorsque } q_A = q_A^{i,j},\ 1 \leq i \leq M, 0 \leq j \leq |u_i|.$$

Finalement, nous définissons la partition π comme suit :

$$B(q_{MCA}^l, \pi) = B(q_{MCA}^k, \pi) \Leftrightarrow \varphi(q_{MCA}^l) = \varphi(q_{MCA}^k).$$

La définition même de la partition π implique que A soit isomorphe à MCA/π, puisque la complétude structurelle de I_+ implique que $\delta_{MCA/\pi}$ corresponde exactement à δ_A. La seconde condition requise par la complétude structurelle de I_+ impose que

$$\forall q_A \in F_A, \exists i, 1 \leq i \leq M \text{ tel que } q_A^{i,|u_i|} = q_A.$$

Par conséquent, $F_{MCA} = \{q_{MCA} | \varphi(q_{MCA}) \cap F_A \neq \phi\}$ et $F_{MCA/\pi}$ correspond exactement à F_A. □

FIG. 1.7: *L'automate A et le $MCA(I_+)$, avec $I_+ = \{ab\}$.*

En guise d'illustration de la dernière partie de cette démonstration, nous considérons l'exemple suivant. Conformément à la définition 1.16, l'échantillon $I_+ = \{ab\}$ n'est pas structurellement complet relativement à l'automate A de la figure 1.7, où q_1 et q_2 sont des états d'acceptation. Pour cette

raison, l'automate A ne peut pas être dérivé du $MCA(I_+)$, et évidemment pas non plus du $PTA(I_+)$. En effet, il n'est pas possible de définir une partition π de l'ensemble des états Q_{MCA} telle que A corresponde à $MCA(I_+)/\pi$ avec $q_1 \in F_{MCA/\pi}$. Au contraire, l'échantillon $\{ab, a\}$ est structurellement complet relativement à l'automate A qui peut donc être dérivé du MCA, ou du PTA, associés à cet échantillon.

Théorème 1.2 *Soit I_+ un échantillon positif d'un quelconque langage régulier L et soit $A(L)$ l'automate* ***canonique*** *acceptant L. Si I_+ est structurellement complet relativement à $A(L)$ alors $A(L)$ appartient à $Lat(PTA(I_+))$.*

Preuve :
Nous pouvons faire appel au raisonnement utilisé dans la preuve du théorème 1.1, à l'exception de l'acceptation de I_+ par $A(L)$ qui, à présent, est unique puisque $A(L)$ est déterministe. De façon similaire, chaque entrée de la fonction de transition du $PTA(I_+)$ contient au plus un élément, puisque le $PTA(I_+)$ est déterministe. □

Théorème 1.3 *Soit $\mathcal{A}$ l'ensemble des automates relativement auxquels un échantillon positif I_+ est structurellement complet. L'ensemble $\mathcal{A}$ est $Lat(MCA(I_+))$.*

Preuve :
(1) *Si I_+ est structurellement complet relativement à un automate A, alors $A \in Lat(MCA(I_+))$.*
Il s'agit exactement du théorème 1.1.
(2) *Si un automate A appartient à $Lat(MCA(I_+))$ alors I_+ est structurellement complet relativement à A* [FB75a].
Par construction, I_+ est structurellement complet relativement au $MCA(I_+)$. Puisque n'importe quel automate appartenant à $Lat(MCA(I_+))$ peut être dérivé du $MCA(I_+)$ pour une certaine partition π, le résultat est une conséquence directe de la définition 1.14 d'un automate quotient. □

1.5.2 Propriétés de l'espace de recherche

Grâce aux résultats présentés au paragraphe 1.5.1, nous savons que si nous disposons d'un échantillon positif I_+ d'un langage inconnu L et si nous supposons que I_+ est structurellement complet relativement à un automate inconnu A acceptant exactement L, alors nous pouvons dériver A pour une certaine partition π de l'ensemble des états du $MCA(I_+)$. Nous pouvons dès lors considérer l'inférence régulière comme un problème de recherche, dans un treillis, de la partition π.

Signalons, tout d'abord, que le théorème 1.1 est plus général que le théorème 1.2. En effet, le premier suppose seulement la complétude structurelle de I_+ relativement à un quelconque automate acceptant un langage L, et non nécessairement relativement à l'automate canonique[19] de L. De plus, nous pouvons énoncer la propriété d'inclusion des treillis :

Propriété 1.4 $Lat(PTA(I_+)) \subseteq Lat(MCA(I_+))$.

Cette propriété découle directement de la définition 1.18 du $PTA(I_+)$ qui est un automate quotient du $MCA(I_+)$. Comme le treillis $Lat(PTA(I_+))$ est généralement *strictement* inclus dans le treillis $Lat(MCA(I_+))$, rechercher une solution dans $Lat(PTA(I_+))$ au lieu de $Lat(MCA(I_+))$ permet de considérer un espace de recherche plus restreint.

19. Un échantillon structurellement complet relativement à $A(L)$ est parfois appelé *représentatif pour L* [Mug84].

La taille de l'espace de recherche est le nombre de partitions $|\mathcal{P}(N)|$ d'un ensemble à N éléments où N est ici le nombre d'états de l'élément nul du treillis, c'est-à-dire $PTA(I_+)$ ou $MCA(I_+)$. Ce nombre peut être calculé par la formule suivante :

$$|\mathcal{P}(N)| = \sum_{i=1}^{N} S(N,i)$$

où $S(N,i)$ désigne le nombre de partitions comportant i blocs d'un ensemble à N éléments. $S(N,i)$ est le nombre de Stirling de deuxième espèce qui est donné par l'application de la formule de récurrence [DB62] :

$S(N,1) = 1$, $N \geq 1$,
$S(N,i) = 0$, si $i > N$,
$S(N,i) = i \cdot S(N \Leftrightarrow 1, i) + S(N \Leftrightarrow 1, i \Leftrightarrow 1)$, pour $2 \leq i \leq N$.

A titre d'exemple, $|\mathcal{P}(30)| = 8.5 \times 10^{23}$.

Nous détaillons dans la suite de ce paragraphe d'autres propriétés des treillis, que nous avons établies [DMV94]. La première caractéristique importante est que chaque élément des treillis correspond à un automate particulier mais que beaucoup d'automates différents peuvent représenter, c'est-à-dire accepter, le même langage. Donc, nous pourrions envisager de restreindre la recherche à des automates déterministes. Cependant, nous avons la propriété suivante :

Propriété 1.5 *Il existe des échantillons positifs I_+ pour lesquels certains langages ne sont représentés que par des AFNs dans $Lat(MCA(I_+))$.*

Notons que cette propriété s'applique également à $Lat(PTA(I_+))$ en conséquence de l'inclusion des treillis (propriété 1.4).

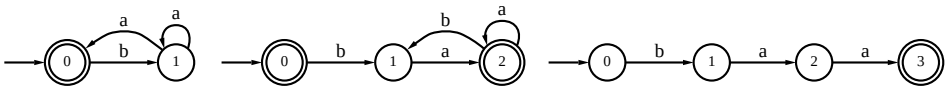


FIG. 1.8: $AFN(L_1)$, $A(L_1)$ *et* $MCA(I_+)$, *avec* $L_1 = (ba^*a)^*$ *et* $I_+ = \{baa\}$.

Considérons l'automate non-déterministe de la figure 1.8. Il accepte le langage L_1 défini par l'expression régulière $(ba^*a)^*$. L'échantillon $I_+ = \{baa\}$ est structurellement complet relativement à cet automate. Cependant, cet échantillon n'est pas structurellement complet relativement à l'automate déterministe minimal (définition 1.12) du même langage et donc relativement à aucun automate déterministe acceptant le langage L_1. Par conséquent, nous ne pouvons pas identifier le langage L_1 si nous restreignons la recherche à des automates déterministes.

Propriété 1.6 *Il existe des échantillons positifs I_+ pour lesquels l'ensemble des langages qui peuvent être identifiés dans $Lat(PTA(I_+))$ est strictement inclus dans l'ensemble des langages qui peuvent être identifiés dans*
$Lat(MCA(I_+))$.

Par la propriété 1.4 d'inclusion des treillis, nous savons que l'ensemble des automates qui peuvent être dérivés de $MCA(I_+)$ inclut l'ensemble de ceux qui peuvent être dérivés de $PTA(I_+)$. La propriété 1.6 résulte du fait que, d'une part, cette inclusion peut être stricte et que, d'autre part, les automates qui appartiennent à $Lat(MCA(I_+))$ sans appartenir à $Lat(PTA(I_+))$ n'ont pas d'automate équivalent, c'est-à-dire n'importe quel automate acceptant le même langage, dans $Lat(PTA(I_+))$.

Considérons le langage $L_2 = (ba + b(aa)^*)$, qui peut être représenté par l'automate non-déterministe $AFN(L_2)$ de la figure 1.10 et par son automate canonique $A(L_2)$. Supposons que l'échantillon positif I_+ soit $\{ba, baa\}$. Le $MCA(I_+)$ et le $PTA(I_+)$ sont représentés à la figure 1.9. Nous pouvons clairement dériver $AFN(L_2)$ à partir du $MCA(I_+)$ et non du $PTA(I_+)$. Cela provient du fait que les états q_1 et q_2 du MCA, respectivement q_3 et q_4, sont fusionnés dans le PTA. De plus, puisque I_+ est structurellement complet relativement à $AFN(L_2)$ mais pas relativement à $A(L_2)$, ce dernier automate ne peut être dérivé du PTA ni aucun automate déterministe acceptant le même langage.

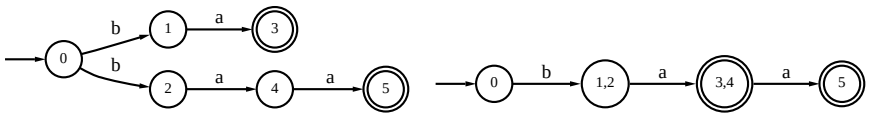


FIG. 1.9: $MCA(I_+)$ *et* $PTA(I_+)$, *avec* $I_+ = \{ba, baa\}$.

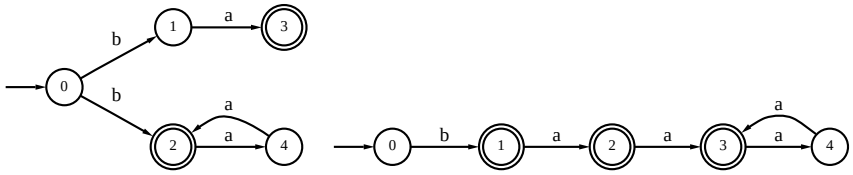


FIG. 1.10: $AFN(L_2)$ *et* $A(L_2)$, *avec* $L_2 = (ba + b(aa)^*)$.

1.6 Ensemble frontière

Supposons que nous disposions d'un échantillon négatif du langage inconnu L. Dans ce cas, nous savons que nous pouvons, en principe, identifier au moins n'importe quel langage régulier (voir paragraphe 1.3.5). L'inférence peut être alors considérée comme la découverte d'un automate compatible avec les échantillons positif et négatif (voir paragraphe 1.4). Rappelons que cela consiste à chercher un automate A tel que $I_+ \subseteq L(A)$ et $I_- \cap L(A) = \phi$. Il existe un grand nombre d'automates compatibles appartenant à $Lat(MCA(I_+))$ et, en particulier, le $MCA(I_+)$ satisfait ces conditions mais ne généralise pas l'échantillon positif. Par conséquent, nous pouvons chercher une solution plus générale sous le contrôle de l'échantillon négatif.

Si nous choisissons la simplicité de l'automate inféré comme critère de généralité et que nous restreignons la recherche à des automates déterministes, la solution cherchée est alors l'AFD compatible comportant le nombre minimal d'états. Il s'agit du problème du plus petit AFD compatible, déjà évoqué au paragraphe 1.4. Nous avons vu qu'il n'existe pas d'algorithme polynomial résolvant ce problème dans tous les cas mais qu'il existe un algorithme polynomial qui permet, en général, de trouver une solution quasi-exacte. Nous donnerons un aperçu de cet algorithme au paragraphe 2.2.2. Deux autres algorithmes seront abordés aux paragraphes 2.2.1 et 2.2.3. Afin de comparer ces trois algorithmes et d'étudier le problème du plus petit AFD compatible, nous poursuivons la caractérisation de l'espace de recherche introduite au paragraphe 1.5.

La notion d'ensemble frontière [MdG94] a été proposée, dans le domaine de l'inférence grammaticale, par Miclet. Cet ensemble constitue la limite de la généralisation possible à partir de l'échantillon positif sous contrôle de l'échantillon négatif. En ce sens, l'ensemble frontière est l'ensemble des *généralisations les plus générales* conformément à la terminologie propre à l'étude de l'*espace des versions* [Mit78, Nic93]. Cet espace est, dans notre cas, l'ensemble de tous les automates quotient du

$MCA(I_+)$ et compatibles avec I_-. Notons que l'ensemble des *généralisations les plus spécifiques* se réduit, dans $Lat(MCA(I_+))$, au $MCA(I_+)$.

Nous avons étudié dans [DMV94] quelques-unes des propriétés de l'ensemble frontière que nous rappelons ci-après.

Définition 1.32 Une *antichaîne* $\overline{AS}$ dans un treillis d'automates est un ensemble d'automates tel que chaque élément de $\overline{AS}$ n'est relié par la relation $\preceq$ avec aucun autre élément de $\overline{AS}$.

Définition 1.33 Nous disons qu'un automate A est *à une profondeur maximale* dans un treillis d'automates s'il n'existe pas, dans ce treillis, d'automate A', différent de A, tel que A' puisse être dérivé de A et tel que $L(A') \cap I_- = \phi$.

Définition 1.34 L'*ensemble frontière*[20] $BS_{MCA}(I_+, I_-)$ est l'antichaîne dont chaque élément est à une profondeur maximale dans $Lat(MCA(I_+))$.

Définition 1.35 L'*ensemble frontière* $BS_{PTA}(I_+, I_-)$ est l'antichaîne dont chaque élément est à une profondeur maximale dans $Lat(PTA(I_+))$.

Par conséquent, l'ensemble frontière d'un treillis est l'antichaîne des automates les plus généraux sous le contrôle de l'échantillon négatif. Le problème du plus petit AFD compatible consiste à découvrir le plus petit AFD appartenant à l'ensemble frontière[21].

Propriété 1.7 $BS_{PTA}(I_+, I_-) \subseteq BS_{MCA}(I_+, I_-)$.

Il s'agit d'une conséquence directe de l'inclusion des treillis (propriété 1.4).

Propriété 1.8 *Plusieurs langages distincts peuvent être représentés dans* $BS_{MCA}(I_+, I_-)$.

Cette propriété résulte du fait que la relation d'ordre partiel introduite au paragraphe 1.2.3 s'applique aux automates et non aux langages.

Propriété 1.9 *Il peut exister des AFNs appartenant à* $BS_{MCA}(I_+, I_-)$ *contenant moins d'états que le plus petit AFD compatible.*

Par exemple, ce cas se présente lorsque le plus petit AFD compatible correspond à un langage qui peut être représenté, dans le treillis considéré, par un AFN comportant moins d'états.

Propriété 1.10 *Tous les AFDs appartenant à* $BS_{MCA}(I_+, I_-)$ *sont nécessairement minimaux pour le langage qu'ils acceptent.*

20. L'abréviation **BS** pour "Border Set" provient de la terminologie anglaise.

21. On pourrait également introduire une notion d'*ensemble frontière déterministe*, c'est-à-dire constitué uniquement d'AFDs. Au sens strict, un ensemble frontière déterministe BS_1 peut être distinct d'une restriction à des automates déterministes d'un ensemble frontière quelconque BS_2. En effet, BS_1 peut inclure des AFDs dont on pourrait dériver des AFNs compatibles. Ces AFDs n'appartiendraient donc pas à la restriction déterministe de BS_2.

Remarquons que la même propriété s'applique à $BS_{PTA}(I_+, I_-)$ comme conséquence de la propriété 1.7.
Preuve :
Soit A un AFD acceptant le langage L et compatible avec I_+ et I_-. Supposons que A ne soit pas isomorphe à l'automate canonique $A(L)$. Nous pouvons construire la partition π de l'ensemble des états de A de telle sorte que A/π soit obtenu à partir de A en fusionnant les états partageant les mêmes ensembles de finales. Par construction, $A/\pi = A(L)$. Donc, $A \ll A(L)$ et A ne se trouve pas à une profondeur maximale. Finalement, quoi qu'il en soit de l'appartenance de $A(L)$ à $BS_{MCA}(I_+, I_-)$, A ne peut en être un élément. □
Cette propriété établit qu'un automate déterministe non minimal ne peut pas appartenir à l'ensemble frontière $BS_{MCA}(I_+, I_-)$. Par ailleurs, il existe des automates canoniques qui n'appartiennent pas à $BS_{MCA}(I_+, I_-)$. En effet, tous les automates canoniques du treillis ne sont pas nécessairement compatibles avec I_- et, par ailleurs, il est parfois possible de dériver à partir d'un automate canonique un autre automate compatible.

1.7 Inférence régulière par recherche déterministe

Le théorème 1.2 stipule que nous pouvons identifier n'importe quel langage régulier L en découvrant l'automate canonique $A(L)$ comme automate quotient du $PTA(I_+)$. Rappelons que cela nécessite la complétude structurelle de I_+ relativement à l'automate $A(L)$ recherché. Sous cette hypothèse, nous pouvons explorer le treillis $Lat(PTA(I_+))$. Le $PTA(I_+)$ et l'automate cible sont déterministes. Néanmoins, certains automates construits au cours de la recherche de $A(L)$ peuvent être non déterministes. La possibilité de restreindre la recherche à des automates déterministes est basée sur le théorème 1.4.

Théorème 1.4 *Soit I_+ un échantillon positif d'un quelconque langage régulier L et soit $A(L)$ l'automate canonique acceptant L. Si I_+ est structurellement complet relativement à $A(L)$ alors il existe une séquence d'AFDs telle que*

$$PTA(I_+) = PTA(I_+)/\pi_0 \preceq PTA(I_+)/\pi_1 \preceq \ldots \preceq PTA(I_+)/\pi_n = A(L)$$

Preuve :
Le théorème 1.2 nous assure que la partition π_n existe, avec $0 < n \leq |Q_{PTA(I_+)}| \Leftrightarrow 1$. Nous allons démontrer qu'il existe une séquence de partitions $\pi_0 \preceq ... \preceq \pi_n$ telle que chaque automate quotient associé soit déterministe. Par hypothèse, $PTA(I_+)/\pi_n$ est un AFD. Nous allons démontrer par récurrence que si $PTA(I_+)/\pi_j$ est un AFD, pour $0 < j \leq n$, alors il existe une partition π_{j-1}, telle que $\pi_{j-1} \preceq \pi_j$ et $PTA(I_+)/\pi_{j-1}$ est un AFD.
Nous considérons les $r(j)$ blocs de la partition $\pi_j = \{B_1, ..., B_{r(j)}\}$. Chaque bloc reçoit l'indice de son état de rang minimal conformément à l'ordre standard des états du $PTA(I_+)$. Par conséquent, $B_1 < ... < B_{r(j)}$. Nous désignons par k l'indice, compris entre 1 et $r(j)$, du premier bloc comportant au moins 2 états, avec $B_k = \{q_k^0, ..., q_k^p\}$. Les états de chaque bloc, en particulier ceux du bloc B_k, sont également triés par ordre standard : $q_k^0 < ... < q_k^p$. La partition π_{j-1} est alors obtenue à partir de la partition π_j par la formule suivante :

$$\pi_{j-1} = \pi_j \backslash B_k \cup \{q_k^0\} \cup B_k \backslash q_k^0$$

Par construction : $\pi_{j-1} \preceq \pi_j$.

Supposons que $PTA(I_+)/\pi_{j-1}$ soit non déterministe alors que $PTA(I_+)/\pi_j$ est déterministe (par hypothèse de récurrence). Tous les blocs de rang inférieur à k correspondent nécessairement à un seul état du $PTA(I_+)$. Cela implique qu'il existe un état q du $PTA(I_+)$, avec $q < q_k^0$, et une lettre $a \in \Sigma$, tels que $\exists q' \in B_k \backslash q_k^0$ avec q' et q_k^0 tous les deux a -successeurs de q. Dans ce cas $PTA(I_+)$ serait non déterministe, ce qui est faux. Donc $PTA(I_+)/\pi_{j-1}$ est un automate déterministe. □

Corolaire 1.1 *Tout AFD $A \in Lat(PTA(I_+))$ peut être obtenu par fusions successives en suivant une certaine séquence d'AFDs quotient du $PTA(I_+)$.*

Nous pouvons nous demander s'il existe un théorème dual du théorème 1.4 qui consisterait à trouver une séquence d'automates déterministes entre un automate déterministe quelconque dans $Lat(PTA(I_+))$ et l'automate universel. Il n'en est rien. En guise de contre-exemple, à partir de l'échantillon $I_+ = \{aaa\}$ nous pouvons obtenir le $PTA(I_+)$, les automates A_1, A_2, A_3, A_4, A_5 et l'automate universel UA représentés à la figure 1.11. Ces 7 automates sont déterministes mais toute fusion d'états de l'automate A_1 introduit un non déterminisme. Il n'existe donc pas de séquence d'automates déterministes et quotient de A_1 qui mènerait à UA. Cela nous amène à conclure que la restriction du treillis $Lat(PTA(I_+))$ aux automates déterministes est un *sup demi-treillis* dont l'élément nul est le $PTA(I_+)$. Nous avons représenté à la figure 1.11 le sup demi-treillis de l'exemple évoqué.

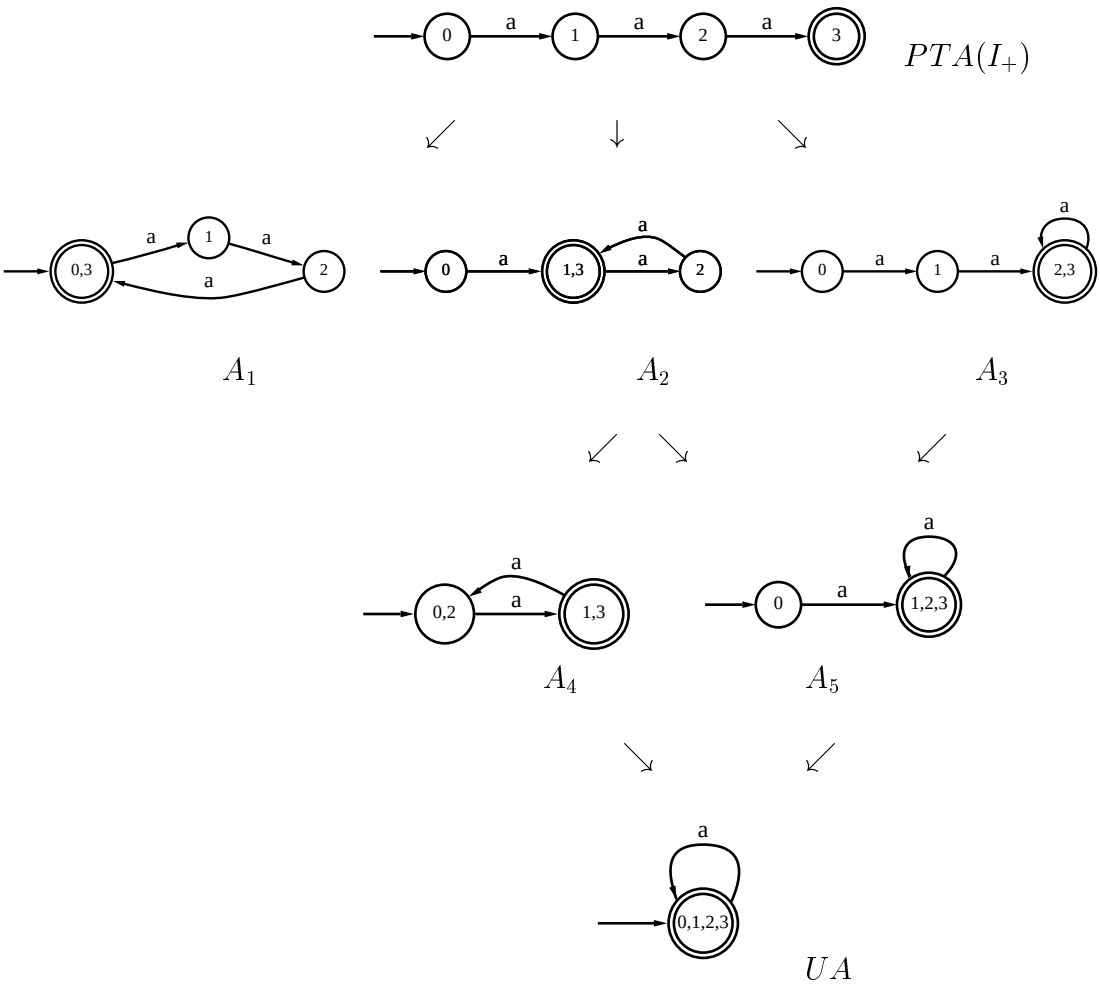


FIG. 1.11: *Sup demi-treillis des AFDs dans $Lat(PTA(I_+))$, pour $I_+ = \{aaa\}$.*

Nous pouvons contraindre les algorithmes de recherche heuristiques, en particulier BRIG, à n'explorer que le sup demi-treillis déterministe. Notons que cette contrainte peut être appliquée par une inspection efficace de la table de transition en utilisant des structures d'ensembles disjoints [CLR90].

Chapitre 2

Algorithmes d'inférence régulière

De nombreux algorithmes d'inférence ont été développés. Certains sont *constructifs* ou *caractérisables* dans le sens qu'ils identifient *avec certitude* n'importe quel langage appartenant à une *classe définie*. Nous qualifions les autres d'algorithmes *heuristiques*, soit parce qu'ils n'identifient pas avec certitude n'importe quel langage d'une classe définie, soit parce qu'il n'existe pas de caractérisation de la classe de langages qu'ils peuvent identifier. Ces algorithmes utilisent un *biais d'apprentissage* ou heuristique pour guider la généralisation effectuée à partir des données.

La majorité des algorithmes d'inférence régulière n'utilisent que des échantillons positifs. La plupart de ces algorithmes sont décrits dans les articles de synthèse [BF72, AS83, Mic90, Gre94]. Nous présentons au paragraphe 2.1 quelques algorithmes constructifs ou heuristiques, qui utilisent ce type de données.

Si nous disposons également d'un échantillon négatif, l'inférence peut consister à construire *le plus petit automate compatible* avec les échantillons positif et négatif. Nous avons étudié le cadre formel de ce problème au paragraphe 1.6. Nous comparons au paragraphe 2.2 trois méthodes d'inférence avec échantillon négatif.

Nous terminons ce chapitre par l'exposé d'une technique originale d'inférence semi-incrémentale. Il s'agit d'une technique heuristique ayant pour objet de réduire la complexité pratique d'une inférence sous contrôle d'un échantillon négatif. Elle est détaillée au paragraphe 2.3.

2.1 Inférence sans échantillon négatif

Etant donné un échantillon positif I_+ d'un langage régulier L, nous avons à définir le critère qui guidera la recherche d'un automate A acceptant I_+. En particulier, le $MCA(I_+)$ et le $PTA(I_+)$ acceptent tous deux l'échantillon positif. Cependant, ils n'acceptent aucune autre phrase du langage L et, donc, ne généralisent pas l'information contenue dans les données d'apprentissage.

Nous savons, par la propriété 1.1, que n'importe quel automate dérivé du $MCA(I_+)$ accepte un langage incluant I_+. En ce sens, n'importe quel automate appartenant à $Lat(MCA(I_+))$ constitue une généralisation possible de l'échantillon positif. Conformément au résultat de Gold présenté au paragraphe 1.3.5, nous savons qu'il n'est pas possible d'identifier la classe entière des langages réguliers à partir d'un échantillon positif seulement. En particulier, aucun exemple positif ne peut éviter une sur-généralisation éventuelle de l'inférence. En d'autres termes, si une solution proposée par un algorithme d'inférence correspond à un langage incluant strictement le langage à identifier, aucun exemple positif ne peut contredire cette solution.

Pour éviter le risque de sur-généralisation, deux approches sont classiquement utilisées. La première possibilité consiste à rechercher la solution dans une sous-classe particulière des langages réguliers[1]; il s'agit alors d'une méthode caractérisable. La seconde possibilité consiste à utiliser une information a priori pour guider la généralisation recherchée, le propre des méthodes heuristiques.

L'algorithme k-TSSI[2], détaillé au paragraphe 2.1.1, permet d'inférer les langages *k-testables au sens strict* [GV90]. Bien que la "méthode du successeur" [RV84] soit généralement présentée comme une méthode heuristique, García et Vidal ont démontré qu'elle identifie les *langages locaux*, encore appelés *langages 2-testables au sens strict* [GVC87]. Elle peut donc être considérée comme un cas particulier de l'algorithme k-TSSI. Un autre exemple, est l'algorithme d'inférence d'un langage appartenant à la classe des langages *k-réversibles*, présenté au paragraphe 2.1.2.

Les deux algorithmes précédents identifient à la limite, pour *k* fixé, une sous-classe définie des langages réguliers. Ils sont donc caractérisables. Nous présentons ensuite deux techniques d'inférence heuristiques. L'algorithme MGGI[3], décrit au paragraphe 2.1.3, constitue une autre extension de la méthode du successeur. L'identification de la classe entière des langages réguliers est possible grâce à l'introduction d'une heuristique d'apprentissage, en l'occurrence le choix d'un *morphisme*. Enfin, l'algorithme ECGI[4], décrit au paragraphe 2.1.4, est une technique d'inférence d'automates sans cycles, c'est-à-dire acceptant la sous-classe des langages finis, qui repose sur une analyse corrective.

2.1.1 L'algorithme k-TSSI

De façon informelle, un *langage k-testable au sens strict* est défini par un ensemble fini de sous-chaînes de longueur k qui ne peuvent apparaître dans les chaînes du langage[5]. La classe k-TSS désigne l'ensemble des langages k-testables au sens strict, pour k fixé.

Définition 2.1 Soit $Z_k = (\Sigma, I_k, F_k, T_k)$ un quadruplet, où Σ est un alphabet fini, $I_k, F_k \subseteq \bigcup_{i=1}^{k-1} \Sigma^i$ deux ensembles, constitués de chaînes de longueur strictement inférieure à k, appelés respectivement, *segments initiaux* et *finaux*, et $T_k \subseteq \Sigma^k$ un ensemble de *segments interdits*, constitués de chaînes de longueur k. Un langage k-testable au sens strict est défini par l'expression régulière suivante :

$$l(Z_k) = (I_k\Sigma^*) \cap (\Sigma^* F_k) \Leftrightarrow (\Sigma^* T_k \Sigma^*)$$

Les chaînes de $l(Z_k)$ sont donc définies comme commençant par un segment initial, se terminant par un segment final et ne comportant aucun segment interdit de longueur k. L'appellation k-testable provient du fait qu'il suffit d'analyser les chaînes avec une fenêtre de longueur k pour définir l'appartenance ou non au langage $l(Z_k)$. En particulier, la classe 2-TSS est aussi appelée la classe des *langages locaux*. Par ailleurs, la classe LTSS des *langages localement testables au sens strict* désigne la classe de tous les langages k-TSS, pour k quelconque et la classe k-T des langages *k-testables* désigne les langages obtenus par fermeture booléenne de langages k-testables au sens strict. Il n'existe pas, à l'heure actuelle, d'algorithme d'inférence à partir d'échantillon positif ni pour la classe LTSS, ni pour la classe k-T [VCG93].

1. Cette sous-classe sera nécessairement non superfinie.

2. L'abréviation **k-TSSI** provient de la terminologie anglaise pour "k-Testable (languages) in the Strict Sense Inference".

3. L'abréviation **MGGI** provient de la terminologie anglaise pour "Morphic Generator Grammatical Inference".

4. L'abréviation **ECGI** provient de la terminologie anglaise pour "Error Correcting Grammatical Inference".

5. Nous désignons par Σ^k l'ensemble des chaînes de longueur k définies sur l'alphabet Σ.

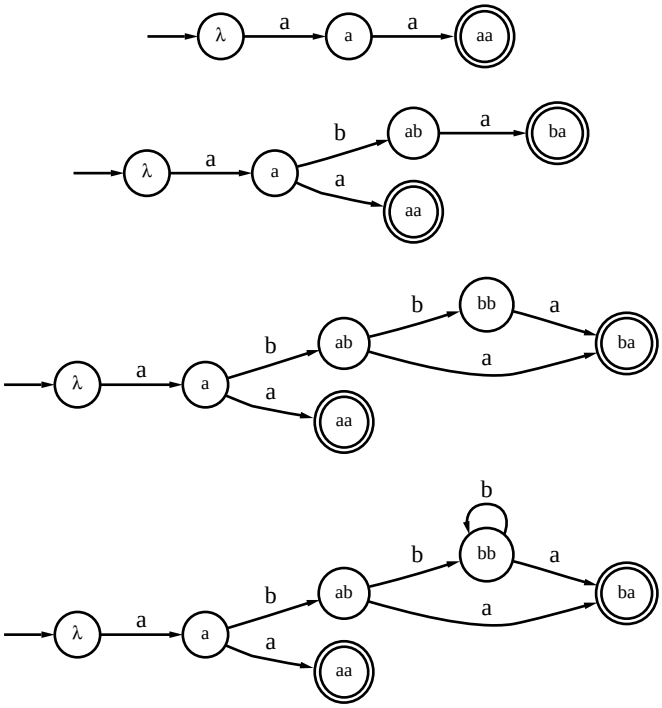


FIG. 2.1: *Exemple d'exécution de l'algorithme k-TSSI.*

Nous présentons ci-dessous l'algorithme k-TSSI qui permet, pour k fixé a priori, d'identifier le plus petit langage appartenant à k-TSS et contenant un échantillon positif I_+ [GV90]. La figure 2.1 représente les automates obtenus par l'application de l'algorithme k-TSSI avec $k = 3$ et $I_+ = \{aa, aba, abba, abbba\}$. Chaque état est identifié par une séquence, comportant au plus $k \Leftrightarrow 1$ symboles, qui correspondent aux derniers symboles lus, jusqu'à cet état, lors de l'acceptation d'une chaîne de I_+. En d'autres termes, lors de l'analyse d'une chaîne les derniers symboles lus sont mémorisés dans une fenêtre glissante de $k \Leftrightarrow 1$ éléments. Chaque contenu de la fenêtre d'analyse identifie un état à créer ou déjà existant. Dans notre exemple, la fenêtre est de longueur 2. Afin d'accepter la chaîne aa, l'on crée les trois premiers états λ, a et aa ; le dernier état étant nécessairement final puisque aa est un élément du langage. L'acceptation de la chaîne aba passe par les états λ et a, déjà existants, et crée les états ab et ba. Ce processus est répété pour toutes les chaînes de I_+. Si nous désignons par $||I_+||$ la somme des longueurs des chaînes de I_+, nous déduisons aisément que cet algorithme est de complexité[6] linéaire, $\mathcal{O}(||I_+||)$.

6. Par défaut, nous appelons complexité d'un algorithme d'inférence sa complexité temporelle asymptotique.

Algorithme k-TSSI

entrée k //L'ordre du modèle
$I_+ = \{x_1, \ldots, x_{|I_+|}\}$ //L'échantillon positif
sortie $A_k = (Q, \Sigma, \delta, q_0, F)$ //Un AFD acceptant le plus petit langage k-TSS incluant I_+

début

$Q \leftarrow q_0$
$\Sigma \leftarrow \phi$
$\delta \leftarrow \phi$
$q_0 \leftarrow \lambda$ // Les états sont représentés par des éléments de Σ^*
$F \leftarrow \phi$
pour $i = 1$ **jusqu'à** $|I_+|$ // Boucle sur les chaînes de I_+
  $q' \leftarrow q_0$
  pour $j = 1$ **jusqu'à** $|x_i|$ // Boucle sur les lettres de x_i
    $\Sigma \leftarrow \Sigma \cup x_{ij}$
    $y \leftarrow q' x_{ij}$
    si $|y| > k - 1$ **alors**
      $y \leftarrow y_{2,\ldots,|y|}$
    fin si
    $q \leftarrow y$
    $Q \leftarrow Q \cup \{q\}$
    $\delta \leftarrow \delta \cup \{q', x_{ij}, q\}$
    si $j = |x_i|$ **alors**
      $F \leftarrow F \cup \{q\}$
    fin si
    $q' \leftarrow q$
  fin pour
fin pour
retourner A_k
fin k-TSSI

Un automate inféré par l'algorithme k-TSSI est, par construction, non ambigu. Une estimation selon le critère du maximum de vraisemblance des probabilités des transitions de l'automate peut être obtenue par le calcul des fréquences relatives d'utilisation des transitions lors de l'analyse d'un ensemble donné de chaînes. Cette estimation peut être faite incrémentalement et simultanément avec la construction de l'automate. Ce modèle stochastique est alors exactement équivalent à un N-gram, avec $N = k$ [Seg93].

2.1.2 Inférence de langages k-réversibles

Angluin a proposé une méthode d'inférence qui identifie à la limite la classe des langages *k-réversibles*, pour k quelconque mais fixé a priori [Ang82]. Nous détaillons ci-après les principales notions nécessaires à la définition de cette méthode d'inférence.

Définition 2.2 Soit un automate $A = (Q, \Sigma, \delta, I, F)$, l'automate inverse $A^r = (Q, \Sigma, \delta^r, F, I)$ est défini comme suit :

$$\forall q \in Q, \quad \forall a \in \Sigma, \quad \delta^r(q, a) = \{q' \in Q | q \in \delta(q', a)\}.$$

Par conséquent, l'automate inverse A^r est obtenu en inversant les états[7] d'entrée et de sortie de l'automate original A, et en inversant également le sens des transitions de A. Nous présentons à la figure 2.2 un exemple d'automate et son inverse.

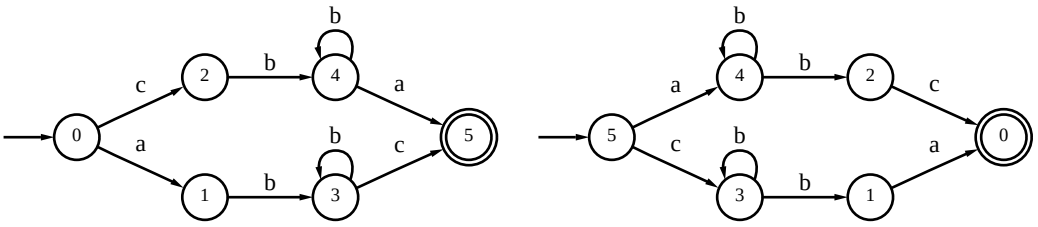


FIG. 2.2: *L'automate A et son inverse A^r.*

Définition 2.3 Un automate $A = (Q, \Sigma, \delta, q_0, F)$ est *déterministe avec anticipation k* si[8]

$$\forall q \in Q, \forall x \in \Sigma^*, |x| > k, \quad |\delta^*(q, x)| \leq 1.$$

Par conséquent, un automate est déterministe avec anticipation k si, lors de l'acceptation d'une chaîne quelconque, il existe au plus un état auquel cette acceptation peut mener en anticipant de k lettres par rapport à la position courante dans la chaîne. Un AFD est déterministe avec anticipation 0. L'automate A^r de la figure 2.2 est déterministe avec anticipation 1.

Définition 2.4 Un automate A est *k-réversible* s'il est déterministe et si son inverse A^r est déterministe avec anticipation k.

Définition 2.5 Un langage L est *k-réversible* s'il existe un automate k-réversible qui l'accepte.

Une définition équivalente est la suivante.

Définition 2.6 Un langage L est *k-réversible* si et seulement si lorsque deux préfixes de L, (w_1 et $w_2 \in Pr(L)$), contenant un même suffixe v de longueur k, ($w_1 = u_1 v$ et $w_2 = u_2 v$, avec $|v| = k$), ont un même suffixe w dans L, ($u_1 vw$ et $u_2 vw \in L$), alors ils ont même ensemble de finales ($L/w_1 = L/w_2$).

Partant de l'accepteur des préfixes $PTA(I_+)$, l'algorithme k-RI[9] consiste à fusionner des états, c'est-à-dire à définir un nouvel automate quotient $PTA(I_+)/\pi$, tant que cet automate n'est pas k-réversible. Plus précisément, la partition π des états du $PTA(I_+)$ est initialement celle qui associe un bloc à chacun de ses états. Ensuite, on détermine récursivement dans l'automate quotient $PTA(I_+)/\pi$, deux états distincts représentés par les blocs B_1 et B_2 tels qu'au moins une des trois conditions suivantes (C1, C2a ou C2b) soit vérifiée.

1. $\exists a \in \Sigma, \exists B \in \pi$ avec $B_1 \in \delta(B, a)$ et $B_2 \in \delta(B, a)$, (C1 : cas d'indéterminisme sur la lettre a).

7. Nous avons défini au paragraphe 1.2, les automates finis comme comprenant un seul état d'entrée et un ensemble d'états de sortie. Ces automates permettent de représenter un langage régulier quelconque. La généralisation aux automates comportant un ensemble d'états d'entrée est triviale.

8. δ^* désigne l'extension classique à $Q \times \Sigma^* \to 2^Q$, de la fonction de transition δ.

9. L'abréviation **k-RI** provient de la terminologie anglaise pour "k-Reversible Inference".

2. $\exists v, u_1, u_2 \in \Sigma^*, |v| = k$ avec $B_1 \in \delta^*(q_0, u_1 v)$ et $B_2 \in \delta^*(q_0, u_2 v)$, ($v$ est le suffixe commun des préfixes menant respectivement à B_1 et B_2), et

   (a) B_1 et B_2 sont des états d'acceptation de l'automate quotient (C2a : un suffixe de longueur k menant à deux états d'acceptation).

   (b) $\exists a \in \Sigma$, $\exists B \in \pi$ avec $B \in \delta(B_1, a)$ et $B \in \delta(B_2, a)$ (C2b : un suffixe de longueur k menant à deux états d'où sont issues des transitions pointant vers le même état B sur la lettre a).

Lorsque deux blocs B_1 et B_2 sont identifiés, un nouvel automate quotient est obtenu par fusion des blocs B_1 et B_2. La partition des états du $PTA(I_+)$ est progressivement mise à jour jusqu'à ce qu'il n'existe plus aucune paire de blocs satisfaisant au moins une des trois conditions décrites. L'automate quotient est alors nécessairement k-réversible. Notons que l'ordre exact des paires de blocs à fusionner est arbitraire.

Nous présentons à la figure 2.3 un exemple d'exécution de l'algorithme k-RI avec $k = 1$ et $I_+ = \{ab, bb, aab, abb\}$. La partition initiale de l'arbre des préfixes est

$$\{\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}\}.$$

La condition C2a est vérifiée, par exemple, pour les blocs $\{5\}$ et $\{7\}$, avec $v = b$ un suffixe de longueur 1. De même , la condition C2a est vérifiée pour les blocs $\{4\}$ et $\{6\}$, avec également $v = b$. Après deux étapes de l'algorithme k-RI, on obtient le second automate représenté à la figure 2.3. Il correspond à la partition

$$\{\{0\}, \{1\}, \{2\}, \{3\}, \{4, 6\}, \{5, 7\}\}.$$

La condition C2a est vérifiée pour les blocs $\{4, 6\}$ et $\{5, 7\}$, avec $v = b$. Après fusion de ces deux blocs, la partition est $\{\{0\}, \{1\}, \{2\}, \{3\}, \{4, 5, 6, 7\}\}$. Finalement, la condition C2b est vérifiée pour les blocs $\{1\}$ et $\{3\}$, d'une part, et les blocs $\{2\}$ et $\{4, 5, 6, 7\}$ d'autre part. La partition résultante est $\{\{0\}, \{1, 3\}, \{2, 4, 5, 6, 7\}\}$; elle correspond à un automate quotient qui est 1- réversible.

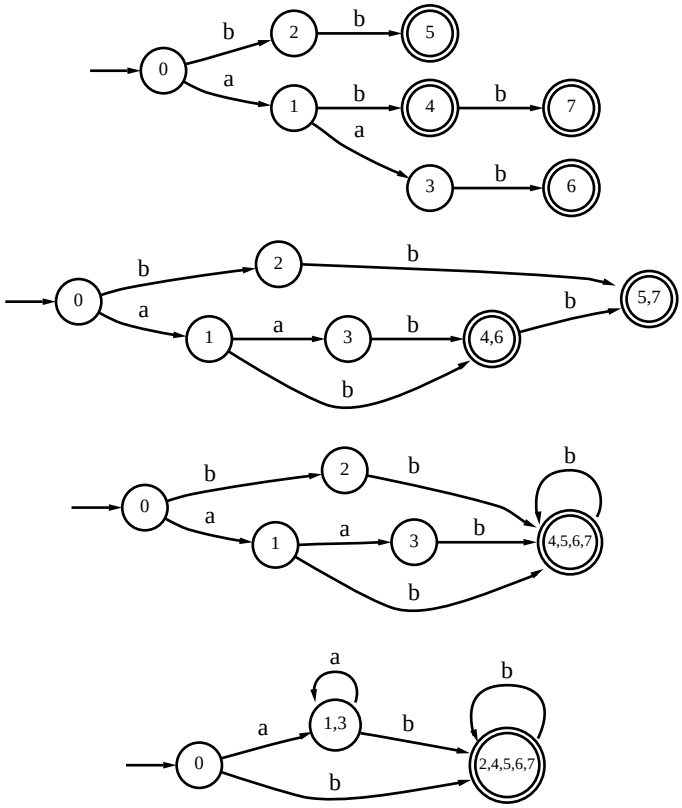


FIG. 2.3: *Exemple d'exécution de l'algorithme k-RI.*

Algorithme k-RI

entrée k //L'ordre du modèle
I_+ //L'échantillon positif
sortie A_k //Un automate canonique acceptant le plus petit langage k-réversible incluant I_+

début

// N désigne le nombre d'états du $PTA(I_+)$

$\pi \leftarrow \{\{0\},\{1\},\ldots,\{N-1\}\}$ // Un bloc par états du $PTA(I_+)$
$A \leftarrow PTA(I_+)$

tant que $((B_1,B_2) \leftarrow$*non-réversible* $(A/\pi,\pi,k)) \neq (\phi,\phi)$ **faire**
$\pi' \leftarrow \pi \backslash \{B_1,B_2\} U \{B_1 U B_2\}$ // Fusion du bloc B_1 et du bloc B_2
$A/\pi' \leftarrow$ *dériver* (A,π')
$\pi \leftarrow \pi'$
fait
retourner A/π
fin k-RI

La fonction *non-réversible* $(A/\pi,\pi,k)$ renvoie une paire de blocs distincts, de la partition π, qui vérifient au moins une des trois conditions citées. Si aucune paire de bloc ne satisfait au moins l'une de ces conditions, elle renvoie (ϕ,ϕ). La fonction *dériver* (A,π') renvoie l'automate quotient de A conformément à la partition π'.

Si l'on pose $n = ||I_+|| + 1$, remarquons que le $PTA(I_+)$ comporte au plus n états et $n \Leftrightarrow 1$ transitions. L'opération qui domine le coût calculatoire de cet algorithme est la fonction *non-réversible.* Moyennant l'introduction de $k+1$ matrices auxiliaires (voir [Ang82]), cette fonction est calculable avec une complexité de $\mathcal{O}(k.n^2)$ pour $\mathcal{O}(n^2)$ paires de blocs à tester. Puisqu'il y a au plus $n \Leftrightarrow 1$ fusions à effectuer, la complexité globale de l'algorithme k-RI est $\mathcal{O}(k.n^3)$.

Angluin a démontré que l'algorithme k-RI permet d'identifier la classe des langages k-réversibles et qu'il produisait, pour I_+ et k fixés, l'automate canonique du plus petit langage k-réversible incluant I_+ [Ang82]. Notons finalement que la classe des langages k-réversibles inclut strictement la classe des langages $k \Leftrightarrow 1$ testables au sens strict [VCG93].

2.1.3 L'algorithme MGGI

L'algorithme MGGI constitue une extension de l'inférence de langages locaux, qui permet d'introduire, par le biais d'un morphisme, une connaissance a priori sur le langage à inférer [GVC87, GSVG90]. Il exploite le théorème énoncé ci-après.

Définition 2.7 Si E et H désignent deux alphabets finis et E^*, H^* les monoïdes libres correspondants, un *morphisme* $h : E^* \rightarrow H^*$ est une fonction qui satisfait les deux conditions suivantes :

$$\forall x, y \in E^* : h(xy) = h(x)h(y), h(\lambda) = \lambda.$$

De plus, la fonction h est *lettre-à-lettre* si $h(a) \in H$, $\forall a \in E$.

Théorème 2.1 *Soit E, un alphabet fini et soit $L \subset E^+$, un langage régulier. Il existe un alphabet fini E', un morphisme lettre-à-lettre $h : E'^* \rightarrow E^*$ et un langage local l de E'^* tel que $L = h(l)$ [GL70].*

Ce théorème stipule donc que tout langage régulier peut être obtenu à partir d'un langage local et l'application d'un morphisme lettre-à-lettre.

Etant donné un échantillon positif $I_+ \subseteq E^*$, l'algorithme MGGI requiert la définition d'une fonction $g : I_+ \rightarrow E'^*$, telle que $g(I_+) = I'_+$ et d'un morphisme lettre-à-lettre $h : E'^* \rightarrow E^*$. Si nous désignons par $l_{g(I_+)}$, le langage 2-TSS inféré par l'algorithme présenté au paragraphe 2.1.1, le langage inféré par l'algorithme MGGI est défini par l'expression $L = h(l_{g(I_+)})$. En d'autres termes, les lettres des chaînes de I_+ sont renommées conformément à la fonction g, donnant lieu à un échantillon positif I'_+ défini sur un nouvel alphabet. A partir de l'échantillon I'_+, un langage local est inféré et ensuite une procédure de renommage (généralement inverse) est appliquée par l'intermédiaire du morphisme g lettre-à-lettre.

Exemple :

$$\begin{aligned}
&E = \{a, b\},\\
&I_+ = \{abba, aaabba, bbaaa, bba\},\\
&E' = \{a^1, a^2, a^3, b^1, b^2\},\\
&I'_+ = g(I_+) = \{a^1b^1b^2a^1, a^1a^2a^3b^1b^2a^1, b^1b^2a^1a^2a^3, b^1b^2a^1\},\\
&h(\alpha^i) = \alpha, \forall \alpha^i \in E'.
\end{aligned}$$

Les chaînes de l'échantillon I_+ sont faites de séquences de a, de longueur 1 ou 3, et de séquences de b, de longueur 2. Dans la mesure où l'on considère qu'il s'agit d'une information pertinente sur le langage à inférer, il est possible de définir la fonction g qui attribue des indices différents à la même lettre, a ou b, dans une sous-chaîne donnée. Les lettres des chaînes de l'échantillon positif I_+ sont renommées par la fonction g. Sur base du nouvel échantillon positif I'_+, un langage local, ou 2-TSS,

est inféré. Il correspond à l'automate représenté à la figure 2.4. Après application du morphisme h qui, dans ce cas, a pour effet de supprimer les indices attachés aux lettres, nous obtenons le langage représenté par l'automate de la figure 2.5. Remarquons qu'il correspond bien au langage acceptant des séquences de $a,$ de longueur 1 ou 3, et des séquences de b, de longueur 2.

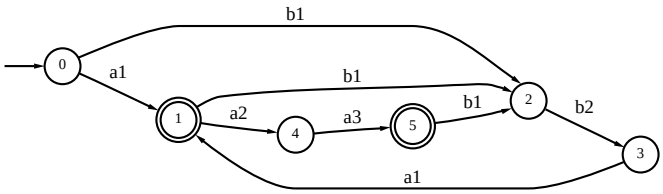


FIG. 2.4: *Automate acceptant le langage local sur l'alphabet E'.*

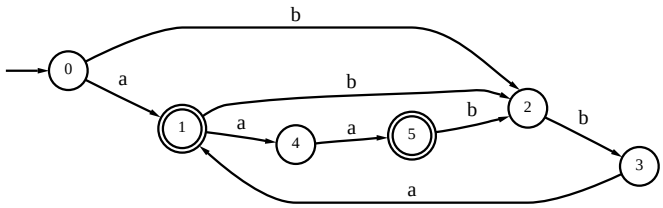


FIG. 2.5: *Automate inféré par l'algorithme MGGI.*

L'algorithme MGGI offre la possibilité d'introduire un biais d'apprentissage par l'intermédiaire des fonctions g et h. Par conséquent, il est intéressant d'appliquer cet algorithme si nous disposons d'une autre information que celle simplement contenue dans l'échantillon I_+. Notons que lorsque les fonctions g et h consistent à renommer chaque lettre par elle-même, l'algorithme MGGI correspond exactement à l'inférence d'un langage 2-testable au sens strict.

La complexité de l'algorithme MGGI dépend de celle de l'inférence du langage 2-testable qui est $\mathcal{O}(||I'_+||)$ (voir paragraphe 2.1.1). L'application du morphisme h lettre-à-lettre est également linéaire en fonction de $||I'_+||$. Par contre, le lien entre $||I'_+||$ et $||I_+||$, dépend de la fonction g. Généralement, $||I'_+|| = ||I_+||$ et la complexité globale est $\mathcal{O}(||I_+||)$. Ce résultat n'est applicable que si la fonction g ne donne pas lieu à un allongement des chaînes de l'échantillon positif.

2.1.4 L'algorithme ECGI

L'algorithme ECGI est une méthode d'inférence heuristique conçue pour extraire l'information sur la longueur de sous-chaînes des éléments de I_+ et de la concaténation de ces sous-chaînes. Cet algorithme est incrémental, c'est-à-dire qu'il mesure une "distance" entre un nouvel exemple et le modèle existant et adapte le modèle conformément. Le langage accepté par la grammaire inférée inclut I_+ ainsi que d'autres chaînes obtenues par concaténation de sous-chaînes des éléments de I_+.

Nous reprenons la description de l'algorithme ECGI tel qu'il a été présenté dans [RV88, Rul92]. En particulier, la représentation utilisée est directement celle des grammaires régulières. Nous illustrons également son fonctionnement à l'aide de la représentation équivalente en automates.

Propriété 2.1 *[RV88] L'algorithme ECGI produit une grammaire $G = (V, \Sigma, P, S)$ régulière, non-déterministe, sans cycle ($\nexists (A \stackrel{+}{\Rightarrow} \alpha A), A \in V, \alpha \in \Sigma^*$) et qui vérifie la condition suivante :*

$$\forall A, B, C \in V, \quad \forall a, b \in \Sigma \quad \textit{si } (B \to aA) \in P \textit{ et } (C \to bA) \in P \textit{ alors } a = b.$$

En d'autres termes, le même terminal est associé avec toutes les productions ayant le même non-terminal en partie droite. Par conséquent, toutes les transitions pointant sur un état donné de l'automate équivalent, sont nécessairement étiquetées par le même terminal.

Définition 2.8 A la grammaire $G = (V, \Sigma, P, S)$ sont associés les *règles d'erreurs* suivantes :

$$\begin{array}{rl}
\text{Insertion de } a: & A \to aA, \forall A \in V, \forall a \in \Sigma \\
\text{Substitution de } b \text{ par } a: & A \to aB, \forall (A \to bB) \in P, \forall a \in \Sigma, a \neq b \\
& A \to a, \forall (A \to b) \in P, \forall a \in \Sigma, a \neq b \\
\text{Suppression de } b: & A \to B, \forall (A \to bB) \in P \\
& A \to \lambda, \forall (A \to b) \in P
\end{array}$$

Par conséquent, il y a $\mathcal{O}(|\Sigma|)$ règles d'insertion par non-terminal de la grammaire G (ou par état de l'automate équivalent) et $\mathcal{O}(|\Sigma|)$ règles de substitution ou de suppression par production de la grammaire G (ou par transition de l'automate équivalent).

Définition 2.9 La *grammaire étendue* $G' = (V, \Sigma, P', S)$ de G est la grammaire obtenue en ajoutant les règles d'erreurs à P.

Définition 2.10 La *dérivation corrective optimale* de la chaîne $\beta \in \Sigma^*$ est la dérivation de β, conformément à la grammaire étendue G' et qui utilise un nombre minimal de règles d'erreurs.

L'algorithme ECGI consiste à construire la grammaire canonique acceptant la première chaîne de I_+. Ensuite, pour chaque nouvel exemple β la dérivation corrective optimale est calculée et la grammaire est étendue conformément à cette dérivation. Cette dérivation est obtenue par une procédure d'analyse corrective qui constitue une application de l'algorithme de Viterbi [For73]. Elle produit la séquence optimale de règles correctes (celles déjà présentes dans la grammaire) et de règles d'erreurs pour l'acceptation de β.

Nous présentons à la figure 2.6 un exemple d'exécution de l'algorithme ECGI prenant en entrée l'échantillon $I_+ = \{aabb, abbb, abbab, bbb\}$ et en représentant chaque grammaire par un automate associé. Les états et transitions en pointillés correspondent à l'extension de la grammaire à chaque étape. Après la construction de l'automate acceptant la première chaîne $aabb$, une analyse corrective construit la dérivation corrective optimale de la chaîne $abbb$. Dans ce cas, la seule règle d'erreur appliquée est la substitution du second a par b. L'automate résultant est représenté à la figure 2.6. Notons que l'état 3 est créé afin de garantir la propriété 2.1 qui stipule que toutes les transitions pointant sur un état doivent être étiquetées par le même symbole. L'analyse corrective de la chaîne $abbab$ conduit à l'insertion d'un symbole a et à la création correspondante de l'état 5. Finalement, l'analyse corrective de la chaîne bbb détermine une erreur de suppression d'un symbole a en début

de chaîne. L'automate final est obtenu en ajoutant une transition étiquetée par b et reliant directement l'état initial à l'état 3.

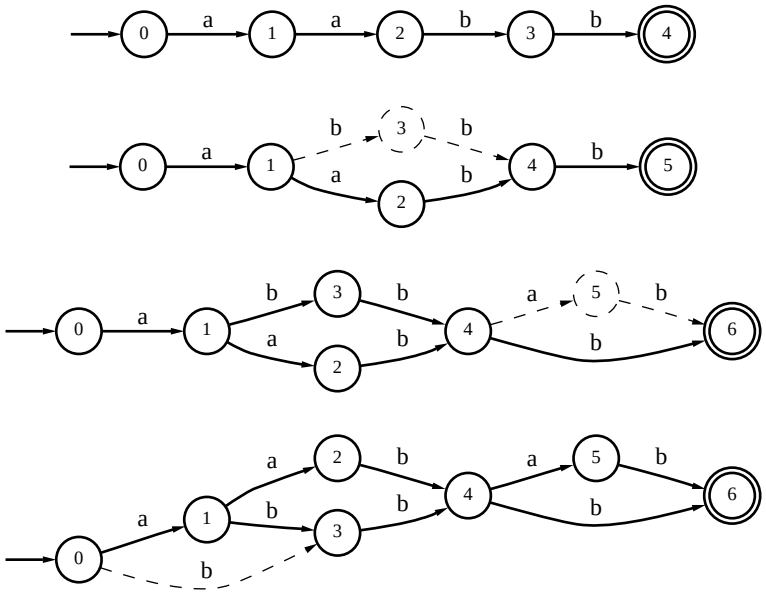


FIG. 2.6: *Exemple d'exécution de l'algorithme ECGI.*

Algorithme ECGI

entrée I_+
sortie Une grammaire G compatible avec I_+ et vérifiant la propriété 2.1

début

$x \leftarrow I_+^1$
$n \leftarrow |x|$
$V \leftarrow \{A_0, \ldots, A_{n-1}\}$
$\Sigma \leftarrow \{a_1, \ldots, a_n\}$
$P \leftarrow \{(A_{i-1} \rightarrow a_i A_i), i = 1, \ldots, n-1\} \cup \{A_{n-1} \rightarrow a_n\}$
$S \leftarrow A_0$
$G_1 \leftarrow (V, \Sigma, P, S)$
pour $i = 2$ **jusqu'à** $|I_+|$ // Boucle sur les éléments de I_+
  $G \leftarrow G_{i-1}$
  $x \leftarrow I_+^i$
  $D \leftarrow$*dériv_optim* (x, G_{i-1})
  pour $j = 1$ **jusqu'à** $|D|$ // Boucle sur les règles de la dérivation optimale D
    $G \leftarrow$*étendre_gram* (G, p_j)
  fin pour
  $G_i \leftarrow G$
fin pour
retourner G
fin ECGI

La fonction *dériv_optim* (x, G_{i-1}) renvoie la dérivation corrective optimale de la chaîne x conformément à la grammaire G_{i-1}. La fonction *étendre_gram* (G, p_j) renvoie la grammaire étendue conformément à la règle p_j.

La complexité de l'algorithme ECGI est dominée par la fonction d'analyse corrective. Pour chaque chaîne x, l'analyse représente $\mathcal{O}(T.|x|)$ opérations où T est le nombre de transitions de l'automate correspondant à la grammaire étendue. Comme il y a au plus $\mathcal{O}(||I_+||)$ transitions et $\mathcal{O}(||I_+||)$ états dans l'automate inféré, l'automate incluant les règles d'erreurs comporte $\mathcal{O}(|\Sigma|.||I_+||)$ transitions. L'analyse corrective d'une chaîne x ayant une complexité de $\mathcal{O}(|\Sigma|.||I_+||.|x|)$, la complexité globale de l'algorithme est $\mathcal{O}(|\Sigma|.||I_+||^2)$.

Il existe une extension stochastique de l'algorithme ECGI qui permet d'inférer une grammaire régulière stochastique par une procédure incrémentale en une seule passe sur l'échantillon positif [Rul92]. En effet, les probabilités associées aux productions de la grammaire peuvent être mise à jour par comptage de la fréquence d'utilisation des règles lors de l'analyse corrective de chaque nouvelle chaîne.

2.2 Inférence avec un échantillon négatif

2.2.1 Les algorithmes RIG et BRIG

Etant donné des échantillons positif I_+ et négatif I_-, un algorithme, nommé RIG[10] [MdG94], de construction complète de l'ensemble frontière $BS_{MCA}(I_+, I_-)$ a été proposé par Miclet. Cet algorithme procède par énumération des automates dérivés du $MCA(I_+)$, c'est-à-dire des partitions

10. L'abréviation **RIG** provient de la terminologie anglaise pour "Regular Inference of Grammars".

dans $Lat(MCA(I_+))$. Il s'agit d'une énumération en largeur à partir de l'élément nul du treillis des partitions, le $MCA(I_+)$. Cette énumération ne conserve que les automates compatibles à chaque profondeur dans le treillis. Par la propriété 1.1 d'inclusion des langages, nous pouvons effectuer un *élagage par héritage* , qui consiste à éliminer tous les automates à la profondeur $i+1$ qui dérivent d'au moins un automate non compatible à la profondeur i. Ensuite, les automates restant à la profondeur $i+1$ subissent un *élagage direct* qui consiste à éliminer les automates non compatibles à cette profondeur. Finalement, l'ensemble frontière est obtenu en mémorisant tous les automates compatibles et qui ne possèdent aucun dérivé compatible. La solution proposée par l'algorithme RIG est, par exemple, le premier automate déterministe rencontré à la profondeur maximale de l'ensemble frontière, mais l'algorithme RIG peut également fournir comme solution tout l'ensemble frontière.

Dans l'algorithme RIG original, des automates identiques peuvent être engendrés à plusieurs reprises. En effet, les partitions à la profondeur $i+1$ sont obtenues en fusionnant, de toutes les façons possibles, deux blocs d'une partition à la profondeur i. Nous présentons ci-dessous une version améliorée de l'algorithme RIG qui évite ce traitement redondant [MDV95]. Elle consiste à introduire un ordre sur les partitions à une profondeur donnée[11] et à définir un ensemble $G(i)$ de partitions, dites *génératrices*. Il s'agit des partitions de rang minimum à partir desquelles peuvent être engendrées toutes les partitions à la profondeur suivante : chaque nouvelle partition à la profondeur $i+1$ est engendrée à partir de l'union de deux blocs d'une partition génératrice à la profondeur i. L'ensemble $NG(i)$ désigne les autres partitions, dites *non génératrices.*

11. Chaque bloc d'une partition prend pour rang celui de son état de rang minimal dans le MCA. Il en résulte un ordre sur les partitions puisqu'elles ont toutes le même nombre de blocs à une profondeur donnée.

Algorithme RIG

entrée I_+, I_-
sortie Un AFD appartenant à $BS_{MCA}(I_+, I_-)$

début

// N désigne le nombre d'états du $MCA(I_+)$

$G(0) \leftarrow \{\{\{0\}, \{1\}, ..., \{N-1\}\}\}$ // L'ensemble des partitions génératrices
$NG(0) \leftarrow \phi$ // L'ensemble des partitions non génératrices
$BS \leftarrow \phi$ // L'ensemble frontière

pour $i = 1$ **jusqu'à** $N-1$ // Boucle sur la profondeur
  $G(i) \leftarrow \phi$
  $NG(i) \leftarrow \phi$
  $BS \leftarrow BS \cup NG(i-1)$
  pour $\pi_j = G(i-1)_1$ **jusqu'à** $G(i-1)_{|G(i-1)|}$ // Boucle sur les partitions génératrices
    $BS\_booléen \leftarrow$ VRAI
    tant que ($\pi_k \leftarrow$ *partition_suivante* $(\pi_j)) \neq \phi$ **faire**
      $A \leftarrow$ ***dériver*** $(MCA(I_+), \pi_k)$
      si *compatible* (A, I_-) **alors** // Elagage par I_-
        $BS\_booléen \leftarrow$ FAUX
        $BS \leftarrow BS \backslash$***dérivable*** $(\pi_k, NG(i-1))$
        si *rang_minimum* (π_k)
          alors $G(i) \leftarrow G(i) \cup \pi_k$
          sinon $NG(i) \leftarrow NG(i) \cup \pi_k$
        fin si
      fin si
    fait
    si $BS\_booléen$ **alors**
      $BS \leftarrow BS \cup \pi_j$
    fin si
  fin pour
fin pour
retourner *extraire_solution* (BS)
fin RIG

La fonction *partition_suivante* (π_j) renvoie, à chaque appel, la partition suivante qui peut être dérivée de la partition génératrice π_j. Lorsque toutes ces partitions ont été générées, elle renvoie ϕ. La fonction *dériver* $(MCA(I_+), \pi_k)$ renvoie l'automate quotient du $MCA(I_+)$ conformément à la partition π_k. La fonction *compatible* (A, I_-) renvoie VRAI si l'automate A n'accepte aucune chaîne de l'échantillon négatif I_-, sinon elle renvoie FAUX. La fonction *dérivable* $(\pi_k, NG(i \Leftrightarrow 1))$ renvoie le sous-ensemble des partitions non génératrices $NG(i \Leftrightarrow 1)$ dont peut dériver la partition π_k. La fonction *rang_minimum* (π_k) renvoie VRAI si la partition π_k est de rang minimum, sinon elle renvoie FAUX. La fonction *extraire_solution* (BS) renvoie, par exemple, le premier automate déterministe à la profondeur maximale appartenant au BS.

L'algorithme RIG construit par énumération et sans redondance tous les éléments de l'ensemble frontière BS. Il n'existe pas, à l'heure actuelle, de preuve de l'identification à la limite de la classe des langages réguliers par l'algorithme RIG. Cela nécessiterait de caractériser la solution extraite du BS pour un ensemble croissant de données et de démontrer la convergence (voir paragraphe 1.3.3).

La complexité de RIG dépend du nombre de partitions distinctes construites et implique une analyse, en général non-déterministe, pour vérifier la compatibilité de chaque automate quotient par rapport à I_-. La complexité globale étant non-polynomiale[12], un algorithme approximatif, nommé BRIG[13], a été proposé [MdG94]. Il consiste à ne considérer qu'une proportion finie, générée aléatoirement, des partitions pouvant être dérivées des génératrices. Cette sélection aléatoire permet de garantir, en pratique, une complexité polynomiale et de construire un sous-ensemble du BS. L'algorithme BRIG est clairement de nature heuristique car il n'existe pas de caractérisation de la classe de langages qu'il identifie. Cette caractérisation n'est pas envisageable car son caractère aléatoire implique que différentes exécutions partant des mêmes données ne conduisent pas nécessairement au même résultat.

Remarquons que la construction complète du BS, par l'algorithme RIG, ou d'un sous-ensemble du BS, par l'algorithme BRIG, peut sembler coûteuse dans la mesure où nous ne retenons qu'une solution à la profondeur maximale. Dans ce cas, une simplification évidente de l'algorithme consiste à ne mémoriser le BS qu'à la dernière profondeur où des automates compatibles sont rencontrés. Par ailleurs, l'étude des différents langages acceptés par les automates du BS permettrait probablement de ne retenir qu'un sous-ensemble "pertinent" des automates du BS.

2.2.2 L'algorithme RPNI

L'algorithme RPNI [OG92], développé indépendamment par
Lang [Lan92], effectue une recherche en profondeur dans $Lat(PTA(I_+))$ et trouve une solution "localement optimale" au problème du plus petit AFD compatible. Nous savons, par la propriété 1.6 que l'identification d'un langage régulier L par exploration du treillis des partitions du $PTA(I_+)$ n'est envisageable que si l'on suppose la complétude structurelle de I_+ relativement à l'automate canonique $A(L)$. Sous cette hypothèse, l'algorithme RPNI est particulièrement efficace.

Par construction du $PTA(I_+)$ (voir définition 1.18), chacun de ses états correspond à un préfixe unique et les préfixes peuvent être triés conformément à l'ordre standard[14] "$<$". Cet ordre s'applique donc également aux états du $PTA(I_+)$. Une partition de l'ensemble des états du $PTA(I_+)$ est faite d'un ensemble de blocs, chaque bloc recevant l'indice de l'état de rang minimal qu'il contient. L'algorithme RPNI procède en $N \Leftrightarrow 1$ étapes où N est le nombre d'états du $PTA(I_+)$. La partition à l'étape i est obtenue en fusionnant les deux premiers blocs, selon l'ordre standard, de la partition à l'étape $i \Leftrightarrow 1$ et qui, de plus, donne lieu à un automate quotient compatible.

12. Dans le pire des cas, aucun élagage par I_- n'est effectué. Dans ce cas, l'algorithme RIG explore tout le treillis des partitions d'où sa complexité exponentielle. Malgré un élagage effectif par l'échantillon négatif, il a été observé expérimentalement que la complexité de RIG reste non polynomiale en fonction de N, le nombre d'états du $MCA(I_+)$ [MdG94, MDV95].

13. L'abréviation **BRIG** provient de la terminologie anglaise pour "Boosted beam-search Regular Inference of Grammars".

14. L'ordre standard, parfois appelé *ordre hiérarchique* , sur les chaînes de Σ^*, correspond à un ordre par longueur des chaînes et, pour une longueur donnée, à l'ordre alphabétique. Par exemple, pour l'alphabet $\{a, b\}$, les premières chaînes dans l'ordre standard sont : $\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots$ Notons que les auteurs de l'algorithme RPNI désignent, improprement, cet ordre comme étant l'ordre lexicographique.

Algorithme RPNI

entrée I_+, I_-
sortie Un AFD compatible avec I_+, I_-

début

// N désigne le nombre d'états du $PTA(I_+)$

$\pi \leftarrow \{\{0\}, \{1\}, \ldots, \{N-1\}\}$ // Un bloc par préfixe dans l'ordre $<$
$A \leftarrow PTA(I_+)$

pour $i = 1$ **jusqu'à** $|\pi| - 1$ // Boucle sur les blocs de la partition π
  pour $j = 0$ **jusqu'à** $i - 1$ // Boucle sur les blocs de rang inférieur
    $\pi' \leftarrow \pi \backslash \{B_j, B_i\} U \{B_i U B_j\}$ // Fusion du bloc B_i et du bloc B_j
    $A/\pi' \leftarrow$ *dériver* (A, π')
    $\pi'' \leftarrow$ *fusion_déterm* (A/π')
    si *compatible* $(A/\pi''$,$I_-)$ **alors** // Analyse déterministe de I_-
      $A \leftarrow A/\pi''$
      $\pi \leftarrow \pi''$
      sortir boucle // Quitter la boucle sur j
    fin si
  fin pour // Fin boucle sur j
fin pour // Fin boucle sur i
retourner A
fin RPNI

La fonction *dériver* (A, π') renvoie l'automate quotient de A conformément à la partition π'. L'automate quotient A/π' peut être non déterministe. La fonction *fusion_déterm* (A/π') renvoie la partition π'' obtenue en fusionnant récursivement tous les blocs de π' qui créent le non-déterminisme[15]. Si A/π' est déterministe, la partition π'' est égale à la partition π'.

Sachant que le $PTA(I_+)$ comporte $\mathcal{O}(\|I_+\|)$ états, la complexité de la fonction *fusion_ déterm* est $\mathcal{O}(\|I_+\|)$. De plus, comme l'automate quotient A/π'' est nécessairement déterministe, le test de compatibilité avec I_- s'effectue en $\mathcal{O}(\|I_-\|)$. La complexité globale de l'algorithme RPNI est donc $\mathcal{O}((\|I_+\| + \|I_-\|) \cdot \|I_+\|^2)$.

La solution proposée par l'algorithme RPNI est un automate déterministe appartenant à l'ensemble frontière $BS_{PTA}(I_+, I_-)$. Par la propriété 1.10, nous savons qu'il s'agit de l'automate canonique pour le langage qu'il accepte. Cependant, il ne s'agit du plus petit AFD compatible que si les données d'apprentissage satisfont une condition supplémentaire, c'est-à-dire qu'elles contiennent un *échantillon* dit *caractéristique*, formellement défini dans [OG92]. En d'autres termes, lorsque les données d'apprentissage sont suffisamment représentatives, la découverte de l'automate canonique du langage à identifier, est garantie. De plus, cet automate est également la solution du problème du plus petit AFD compatible, dans ce cas particulier. Remarquons qu'Oncina et García ont démontré que la taille d'un échantillon caractéristique propre à cet algorithme est $\mathcal{O}(n^2)$, où n est le nombre d'états de l'automate cible [OG92].

Vu l'intérêt de l'algorithme RPNI et son efficacité calculatoire, nous en avons proposé une version incrémentale qui sera détaillée au chapitre 3. Nous reviendrons alors sur la définition exacte d'un

15. L'opération de fusion pour déterminisation d'un AFN ne doit pas être confondue avec l'algorithme classique de déterminisation d'un AFN, qui a pour objet de produire un AFD acceptant le même langage [AU72]. Conformément à la propriété 1.1 , l'AFD obtenu par fusion des états d'un AFN accepte, en général, un sur-langage de l'AFN dont il provient.

échantillon caractéristique qui garantit l'identification de l'automate cible. Nous étudierons également les résultats de complexité que nous venons d'énoncer.

2.2.3 L'algorithme GIG

Nous avons présenté dans [Dup94] un algorithme heuristique, dénommé GIG[16], qui optimise par un algorithme génétique la recherche d'une solution optimale, c'est-à-dire à la profondeur maximale. Il fait évoluer dans $Lat(PTA(I_+))$ une population d'automates dérivés du $PTA(I_+)$, par le biais d'opérateurs spécifiques de mutation et de croisement.

L'algorithme GIG peut-être présenté de façon synthétique comme suit :

Algorithme GIG

entrée I_+, I_- //Les échantillons positif et négatif
sortie $MCA(I_+)/\pi$ //Un automate quotient du $MCA(I_+)$
var p_c //Le taux de croisement
p_m //Le taux de mutation
p_d //Le taux de descente

Procédure *évaluer* $(P(t))$
début
pour $i = 1$ **jusqu'à** Pop
$\pi \leftarrow P_i(t)$
$A/\pi \leftarrow$*dériver* $(MCA(I_+), \pi)$
évaluer (A/π) //Evaluation de chaque automate quotient
fin pour
fin *évaluer* $(P(t))$

début GIG

// Pop désigne la taille de la population
// N désigne le nombre d'états de $MCA(I_+)$
// $P(t)$ désigne les partitions conservées à la génération t

$t \leftarrow 0$
$\pi_0 \leftarrow \{\{1\}, \ldots, \{N\}\}$
$P(0) \leftarrow$*initialiser* (π_0) //Initialisation de la population
évaluer $(P(0))$
$\pi_{best} \leftarrow$*meilleur* $(P(0))$
tant que $\neg$(*condition d'arrêt*) **faire**
$t \leftarrow t + 1$
$P(t) \leftarrow$*sélectionner* $(P(t-1))$
$P(t) \leftarrow$*croiser* $(P(t), p_c)$
$P(t) \leftarrow$*muter* $(P(t), p_m, p_d)$
évaluer $(P(t))$
fait
$\pi_{best} \leftarrow$*meilleur* $(P(t))$
retourner $MCA(I_+)/\pi_{best}$
fin GIG

Le $MCA(I_+)$ comporte $||I_+|| + 1$ états et $||I_+||$ transitions. L'évaluation de chaque automate quotient distinct requiert une analyse, en général non-déterministe, de I_- avec une complexité de

16. L'abréviation **GIG** provient de la terminologie anglaise pour "Grammatical Inference by Genetic search".

$\mathcal{O}(||I_+||.||I_-||)$. Un critère d'arrêt classique consiste à fixer un nombre maximal e_{max} d'évaluations de solutions distinctes. La complexité globale de l'algorithme GIG est alors $\mathcal{O}(||I_+||.||I_-||.e_{max})$.

2.2.4 Le point de vue connexionniste

Nous terminons ce paragraphe en citant un certain nombre de travaux sur l'utilisation de réseaux connexionnistes en inférence grammaticale, habituellement à partir d'échantillons positif et négatif. Le cadre formel de ces travaux, lorsqu'il existe, ne correspond généralement pas au cadre classique de l'inférence tel que nous l'avons présenté. Cependant une équivalence théorique entre les réseaux de McCullough et Pitts [MP43] et les automates finis déterministes a été démontrée par Minsky [Min54, Min67]. La question générale de déterminer la taille (c'est-à-dire le nombre de neurones) d'un réseau afin que celui-ci reproduise exactement le comportement d'un automate donné, est étudiée dans [ADO91].

Parmi les nombreuses variantes d'architecture, les *réseaux récurrents*, proposés par Elman [Elm88], ont été utilisés pour approcher le comportement d'automates finis [CSSM89]. Des extensions de ces modèles afin d'approximer les *automates à piles* ont été proposées, par exemple, par Pollack [Pol90] ou par Giles [GSC$^+$90].

Des techniques heuristiques ayant pour objet d'approximer la structure d'un langage sont décrites dans [GLGG91, JWT92]. Par ailleurs, Smith et Miller étudient l'apprentissage de grammaires régulières stochastiques selon un critère d'information mutuelle [SM89] et en proposent une implémentation distribuée. Finalement, Nakamura et Shikano ont développé les NETgrams comme approximation des N-grams. L'avantage potentiel des NETgrams est qu'ils ne requièrent pas un nombre de paramètres libres exponentiellement croissant avec N [NS89].

2.3 Inférence régulière semi-incrémentale

Nous avons vu au paragraphe 1.5.2 que la taille de l'espace de recherche était directement liée au nombre d'états de l'élément nul du treillis. Il en résulte une propriété paradoxale de l'inférence régulière. En effet, plus nous disposons d'information positive sur un langage, c'est-à-dire plus $|Q_{MCA(I_+)}|$ augmente, plus la taille de l'espace de recherche croît, rendant l'identification plus difficile. C'est la raison pour laquelle, nous proposons une procédure d'inférence *semi-incrémentale*. Elle suppose que nous disposions d'une certaine information positive et négative fixée mais que nous définissions un algorithme faisant usage de l'information positive de façon incrémentale [Dup94, MDV95].

Nous trions l'échantillon positif I_+ par ordre standard, ce qui nous permet de considérer en premier lieu les chaînes les plus courtes. A partir de la première chaîne I_+^1, nous construisons l'automate $MCA(I_+^1)$ et nous recherchons la partition optimale sous contrôle de l'échantillon négatif complet I_-. Ensuite, nous définissons la procédure semi-incrémentale par la récurrence suivante. Nous désignons par A^i, l'automate quotient relativement à la partition optimale trouvée à l'étape i et par x^{i+1}, la chaîne I_+^{i+1}. Si $x^{i+1} \in L(A^i)$, nous passons à la chaîne suivante. Sinon, nous étendons l'automate A^i afin d'obtenir un nouvel automate $A^i_{étendu}$ tel que $L(A^i_{étendu}) = L(A^i) \cup x^{i+1}$. Il s'agit d'une opération simple de modification d'automate dont nous donnons un exemple à la figure 2.7. Elle peut être effectuée en deux temps : l'ajout d'une nouvelle branche moyennant l'introduction de transitions vides et le calcul de l'automate équivalent sans transitions vides.

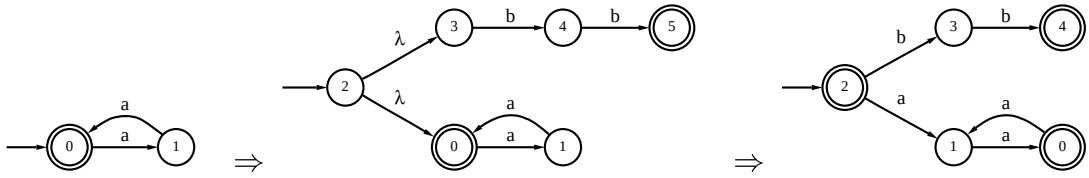


FIG. 2.7: *Construction de l'automate acceptant* $(aa)^* \cup bb$.

Finalement, nous poursuivons la recherche dans $Lat(A^i_{étendu})$ pour trouver la nouvelle partition optimale sous le contrôle de I_-. Cette procédure semi-incrémentale est poursuivie jusqu'à ce que toutes les chaînes de I_+ aient été considérées.

Procédure semi-incrémentale

entrée I_+, I_-
sortie Un automate compatible avec I_+, I_-

début

$A \leftarrow$*inférer* $(MCA(I^1_+), I_-)$

pour $i = 2$ **jusqu'à** $|I_+|$ // Boucle sur les chaînes de I_+
  $x \leftarrow I^i_+$
  si $(x \notin L(A))$ **alors**
    $A' \leftarrow$*étendre* (A, x)
    $A \leftarrow$*inférer* (A', I_-)
  fin pour
retourner A
fin

Remarquons que la procédure semi-incrémentale constitue une technique heuristique pour limiter le coût calculatoire de l'inférence. Il n'y a aucune garantie que cette réduction permette d'obtenir la solution optimale qui résulterait d'une recherche dans le treillis construit sur l'échantillon I_+ complet. Cette procédure n'est pas seulement utile pour la méthode GIG mais peut s'appliquer à tout méthode de recherche explicite dans le treillis des partitions. Elle présente l'avantage supplémentaire de "filtrer" l'échantillon positif puisque les chaînes de I_+ qui sont déjà acceptées, grâce aux généralisations antérieures, ne doivent plus être considérées. Notons que l'on peut aussi envisager une procédure semi-incrémentale par sous-ensembles de plusieurs éléments de I_+. Elle consiste à étendre la solution temporaire pour toutes les chaînes non acceptées dans le sous-ensemble courant et à poursuivre l'inférence, sous contrôle de I_-, après ces extensions.

Chapitre 3

Inférence régulière incrémentale

L'algorithme RPNI, présenté au paragraphe 2.2.2, offre un certain nombre de propriétés intéressantes. Premièrement, il est caractérisable (voir chapitre 2) dans la mesure où il a été démontré que cet algorithme identifie à la limite n'importe quel langage régulier [OG92]. De plus, sa complexité est polynomiale en fonction de la taille des échantillons positif et négatif. Néanmoins, cet algorithme doit recevoir en entrée toute l'information positive et négative disponible, c'est-à-dire qu'il utilise une *présentation à données fixées* (voir paragraphe 3.1.1). Si de nouvelles données d'apprentissage sont présentées, la recherche du nouvel automate solution doit être recommencée sans pouvoir tirer parti de la solution trouvée antérieurement. Nous nous proposons de lever cette limitation et de définir une version incrémentale de l'algorithme RPNI qui soit, elle aussi, caractérisable. Les données d'apprentissage seront supposées être reçues séquentiellement c'est-à-dire une à une, dans un ordre quelconque, et chacune étiquetée comme exemple positif ou négatif du langage à identifier.

Nous allons détailler au paragraphe 3.1 la notion d'échantillon caractéristique d'un langage régulier et étudier un certain nombre de propriétés importantes d'un tel échantillon. Poursuivant l'analyse de l'algorithme RPNI au paragraphe 3.2, nous en proposons une nouvelle variante avec *suppression des finales*. Cette variante garantit également l'identification à la limite mais peut offrir une moins bonne généralisation des données lorsque celles-ci ne contiennent pas d'échantillon caractéristique. Elle permettra néanmoins une analyse de la complexité théorique qui correspond mieux aux observations expérimentales (voir paragraphe 3.2.2). Ensuite, nous poursuivons au paragraphe 3.3 la caractérisation de l'espace de recherche présentée aux paragraphes 1.5 et 1.6. Nous nous intéressons ici à la structure de l'espace de recherche lorsque les données d'apprentissage sont présentées séquentiellement. Enfin, nous exposons au paragraphe 3.4 une version incrémentale de l'algorithme RPNI.

3.1 Caractérisation des données d'apprentissage

3.1.1 Identification à la limite pour une présentation à données fixées

Nous avons vu au paragraphe 1.3.3 que, pour une présentation séquentielle des données, l'identification à la limite est garantie lorsqu'une méthode d'inférence converge avec certitude vers une représentation correcte du langage à identifier. Bien que nous nous intéressions particulièrement aux présentations séquentielles aux paragraphes 3.3 et 3.4, il est utile d'étudier l'identification dans le cadre d'une présentation à *données fixées*. Dans ce cas, l'ensemble d'apprentissage D est fixe et constitué d'une paire d'échantillons positif et négatif, $D = (I_+, I_-)$.

Gold a défini l'identification d'un langage pour une présentation à données fixées [Gol78]. Nous nous inspirons de sa définition pour introduire la notion d'échantillon caractéristique $D^c = (I^c_+, I^c_-)$.

Définition 3.1 Une méthode d'inférence M *identifie à la limite* un langage L *pour une présentation à données fixées* s'il existe un échantillon $D^c = (I^c_+, I^c_-)$ tel que pour tout ensemble de données $D = (I_+, I_-)$, avec $I_+ \supseteq I^c_+$ et $I_- \supseteq I^c_-$, l'hypothèse $H(D)$ fournie par la méthode M est correcte, c'est-à-dire $L(H(D)) = L$. L'échantillon D^c est appelé *caractéristique* pour le langage L et la méthode d'inférence M.

Rappelons que l'hypothèse fournie par une méthode d'inférence est un élément dans l'espace des hypothèses, dans notre cas celui des automates finis.

Cette définition est une caractérisation générale d'une méthode d'inférence. Elle ne précise pas, pour une méthode M et un langage L, comment spécifier D^c. Comme l'automate canonique $A(L)$ d'un langage régulier est unique, la spécification d'un échantillon caractéristique se fait plus naturellement sur base de cet automate. Néanmoins, une méthode d'inférence qui aurait pour objet l'identification d'un langage régulier représenté par un automate non déterministe pourrait requérir une autre spécification.

Nous avons utilisé le terme d'*identification à la limite* dans la définition 3.1 pour insister sur sa similitude avec la définition 1.26 reposant sur une présentation séquentielle.

Propriété 3.1 *Une méthode M identifie un langage L pour une présentation à données fixées si et seulement si elle identifie le langage L pour une présentation séquentielle.*

Preuve :
Si une méthode identifie un langage à partir d'une présentation à données fixées, elle identifiera nécessairement ce langage sur base d'une présentation séquentielle de ces données. Il suffit qu'elle mémorise les données reçues une à une et propose, à chaque étape de la présentation séquentielle, une hypothèse sur base de l'ensemble des données mémorisées. Réciproquement, si une méthode identifie un langage sur base d'une présentation séquentielle, cela veut dire qu'il existe un point de convergence à partir duquel, pour toute nouvelle donnée reçue, l'hypothèse proposée restera identique et correcte. Par la définition 3.1, l'ensemble des données reçues jusqu'au point de convergence inclut donc un échantillon caractéristique. □

3.1.2 Spécification d'un échantillon caractéristique

Partant d'un langage L, représenté par son automate canonique $A(L)$, nous allons définir les conditions que doivent vérifier les données d'apprentissage afin d'être caractéristique relativement à L pour l'algorithme RPNI. A cette fin, Oncina et García ont défini les notions de *préfixes courts* et *noyau* [OG92]. Nous y associons les notions d'*états fixes*, *variables* et *auxiliaires.* Dans la suite, le symbole "$<$ " désigne l'ordre standard sur les chaînes d'un alphabet Σ.

L'ensemble des préfixes courts et le noyau

Définition 3.2 L'ensemble des *préfixes courts* $Sp(L)$ d'un langage L est défini par

$$Sp(L) = \{x \in \mathrm{Pr}(L) \mid \nexists u \in \Sigma^* \text{ avec } L/u = L/x \text{ et } u < x\}$$

Nous savons que dans l'automate canonique $A(L)$ du langage L, il y a autant d'états que d'ensembles de finales L/x des chaînes x appartenant aux préfixes de L (voir définition 1.12). Par conséquent, l'ensemble des préfixes courts est l'ensemble des premières chaînes dans l'ordre standard qui chacune mène à un état de l'automate canonique. Il y a donc également autant de préfixes courts que d'états dans $A(L)$.

A titre d'exemple, l'automate canonique du langage défini sur l'alphabet $\{a, b\}$ et acceptant un nombre pair de b est représenté à la figure 3.1. L'ensemble des préfixes courts de ce langage est $Sp(L) = \{\lambda, b\}$. En effet, λ et b sont les deux premières chaînes dans l'ordre standard permettant de mener respectivement à l'état 0 et à l'état 1.

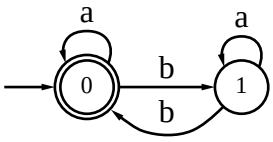


FIG. 3.1: *Automate acceptant un nombre pair de b sur l'alphabet $\{a, b\}$.*

Définition 3.3 Le *noyau* $N(L)$ d'un langage L est défini par

$$N(L) = \{\lambda\} \cup \{xa \mid x \in Sp(L), a \in \Sigma, xa \in \Pr(L)\}$$

Le noyau est constitué, outre de la chaîne λ, des préfixes du langage correspondant à des préfixes courts allongés d'une lettre.

Par construction $Sp(L) \subseteq N(L)$, c'est-à-dire que tous les éléments des préfixes courts appartiennent au noyau. En effet, on peut obtenir chaque préfixe court en allongeant d'une lettre un autre préfixe court du langage.

Si $|Q|$ désigne le nombre d'états de l'automate canonique $A(L)$, le noyau comporte au plus $1 + |Q| \cdot |\Sigma|$ éléments. Les éléments du noyau identifient les transitions de l'automate canonique $A(L)$ puisqu'ils sont obtenus en ajoutant une lettre aux préfixes courts identifiant les états de $A(L)$. De plus, tous les éléments du noyau qui appartiennent également au langage identifient les états d'acceptation dans $A(L)$.

Le noyau du langage de notre exemple est $N(L) = \{\lambda, a, b, ba, bb\}$. Les trois chaînes λ, a et bb appartiennent à L ; elles mènent à l'état d'acceptation de l'automate de la figure 3.1.

Définition 3.4 Un échantillon $D^c = (I^c_+, I^c_-)$ est *caractéristique* relativement à un langage L et pour l'algorithme RPNI s'il vérifie les conditions suivantes

1. $\forall x \in N(L)$,***si*** $x \in L$ ***alors*** $x \in I^c_+$ ***sinon*** $\exists u \in \Sigma^*$ *tel que* $xu \in I^c_+$.
2. $\forall x \in Sp(L), \forall y \in N(L)$ ***si*** $L/x \neq L/y$ ***alors*** $\exists u \in \Sigma^*$ *tel que* $(xu \in I^c_+$ et $yu \in I^c_-)$ *ou* $(xu \in I^c_-$ et $yu \in I^c_+)$.

La condition 1 assure que tous les éléments du noyau soient présents comme chaîne de I^c_+ lorsqu'ils appartiennent au langage ou sinon comme préfixe d'une chaîne de I^c_+. Les éléments du noyau représentant les transitions et les états d'acceptation de l'automate canonique, on vérifie aisément que cette condition assure la complétude structurelle de l'échantillon I^c_+ relativement à $A(L)$ (voir définition 1.16). Dans ce cas, nous savons par le théorème 1.2 que l'automate $A(L)$ appartient au treillis $Lat(PTA(I^c_+))$. Lorsqu'un élément x des préfixes courts et y du noyau n'ont pas même ensemble de

(x, y)	u	I^c_+	I^c_-
(λ, a)	-	-	-
(λ, b)	b	bb	b
(λ, ba)	ab	$baab$	ab
(λ, bb)	-	-	-
(b, λ)	b	bb	b
(b, a)	ba	bba	aba
(b, ba)	-	-	-
(b, bb)	λ	bb	b

TAB. 3.1: *Construction d'un échantillon caractéristique.*

finales ($L/x \neq L/y$), cela signifie qu'il ne correspondent pas au même état dans l'automate canonique. Dans ce cas, la condition 2 assure qu'un suffixe u les distingue[1]. En d'autres termes, la fusion d'un état du $PTA(I^c_+)$ correspondant à un préfixe court x avec un autre état correspondant à un élément y du noyau (incluant les préfixes courts) est rendue incompatible par l'existence de yu dans I^c_- avec xu dans I^c_+ ou l'inverse.

Notons que pour un langage L fixé, il peut exister plusieurs échantillons caractéristiques distincts[2]. En effet, plusieurs suffixes u peuvent satisfaire la première condition ou la seconde. Comme l'ensemble des préfixes courts comporte $|Q|$ éléments et le noyau $\mathcal{O}(|Q| \cdot |\Sigma|)$ éléments, la taille d'un échantillon caractéristique est donnée par

$$|I^c_+| = \mathcal{O}(|Q|^2 \cdot |\Sigma|) \text{ et } |I^c_-| = \mathcal{O}(|Q|^2 \cdot |\Sigma|).$$

Elle ne dépend donc, pour un alphabet Σ fixé, que du nombre d'états de l'automate canonique du langage à identifier.

Un exemple d'échantillon caractéristique

Pour le langage accepté par l'automate $A(L)$ de la figure 3.1, nous avons $Sp(L) = \{\lambda, b\}$ et $N(L) = \{\lambda, a, b, ba, bb\}$. Un échantillon caractéristique peut être construit de la manière suivante. Les chaînes λ, a et bb sont des éléments de $N(L) \cap L$. Elles appartiendront donc nécessairement à I^c_+. Afin de satisfaire complètement la condition 1, il suffit de trouver des suffixes de b et de ba pour obtenir des chaînes de L. Par exemple, la chaîne bb, déjà présente dans I^c_+, contient un suffixe de b et, par exemple, la chaîne $baaaba$ contient un suffixe de ba. La condition 1 est donc satisfaite avec $I^c_+ = \{\lambda, a, bb, baaaba\}$. Cet échantillon est structurellement complet relativement à l'automate $A(L)$.

Afin de compléter l'échantillon caractéristique, nous parcourons les paires constituées d'un élément x de $Sp(L)$ et d'un élément y de $N(L)$. Lorsque $x = y$, les ensembles de finales sont nécessairement égaux. Nous ne considérons donc que les paires d'éléments distincts. Pour chaque paire (x, y), si x et y ont des ensembles de finales différents, c'est-à-dire s'ils mènent à des états distincts dans l'automate canonique, nous trouvons un suffixe u satisfaisant la condition 2. Nous ajoutons les éléments xu et yu dans les échantillons positif ou négatif selon leur appartenance au langage. Ces opérations sont résumées à la table 3.1. Notons que u n'est pas défini lorsque x et y ont un même ensemble de finales.

1. Puisque, par hypothèse $L/x \neq L/y$, l'existence d'au moins un suffixe u satisfaisant la condition 2 est garantie.
2. Il peut même exister une infinité d'échantillons caractéristiques différents si le langage est de cardinalité infinie.

En unifiant les conditions 1 et 2, un échantillon caractéristique pour le langage de notre exemple est donné par

$$I_+^c = \{\lambda, a, bb, bba, baab, baaaba\} \text{ et } I_-^c = \{b, ab, aba\}.$$

Etats fixes, variables et auxiliaires

Nous présentons à la figure 3.2 l'arbre accepteur des préfixes de l'échantillon positif. Nous appelons *états fixes*, les états correspondant aux préfixes courts dans l'arbre des préfixes. Dans notre exemple, les états 0 et 2 sont fixes. Nous appelons *états variables* les états correspondant aux éléments de $N(L) \setminus Sp(L)$ dans l'arbre des préfixes. Dans notre exemple, les états 1, 3 et 4 sont variables. Les autres états de l'arbre des préfixes sont appelés *états auxiliaires*.

Le parcours depuis la racine jusqu'à un état d'acceptation du $PTA(I_+^c)$ est nécessairement composé d'un ou plusieurs états fixes, suivi éventuellement d'un état variable et suivi éventuellement d'un ou plusieurs états auxiliaires. En outre, le long de toute branche du $PTA(I_+^c)$ les préfixes sont ordonnés par l'ordre standard. En d'autres mots, le long de toute branche comportant au moins un état fixe q_{fixe}, un état variable q_{var} et au moins un état auxiliaire q_{aux}, nous avons $q_{fixe} < q_{var} < q_{aux}$. Dans notre exemple, l'acceptation de la chaîne bba passe par les états 0, 2, 4 et 6 ; nous vérifions l'ordre 0 (fixe) $<$ 2 (fixe) $<$ 4 (variable) $<$ 6 (auxiliaire).

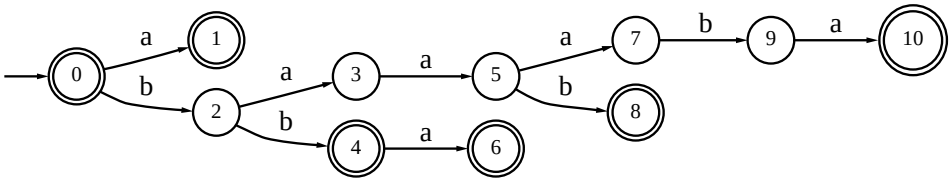


FIG. 3.2: *Arbre accepteur des préfixes de* $\{\lambda, a, bb, bba, baab, baaaba\}$.

Nous avons vu qu'un échantillon caractéristique pour un langage L n'est pas nécessairement unique. Cependant, tout échantillon caractéristique $D^c = (I_+^c, I_-^c)$ est tel que, par la condition 1, les **mêmes** états fixes et variables soient présents dans le $PTA(I_+^c)$ puisque ces états constituent les éléments de $N(L)$. De plus, puisque les états fixes correspondent aux préfixes courts, c'est-à-dire aux états de $A(L)$, ils ont nécessairement des ensembles de finales distincts. Tout échantillon caractéristique est, par la condition 2, tel que la fusion de deux états fixes est rendue incompatible par une chaîne de I_-^c. Finalement, comme il y a autant d'ensembles de finales dans le langage L que d'états fixes, cela implique que chaque état variable partage nécessairement l'ensemble de finales d'un état fixe. La condition 2 assure également que tout fusion d'un état variable avec un état fixe ayant un ensemble de finales différent est rendue incompatible par une chaîne de I_-^c. En d'autres termes, la fusion d'un état variable avec tout état fixe est niée par un contre-exemple sauf pour le "bon" état fixe, c'est-à-dire celui ayant même ensemble de finales.

Toutes les propriétés évoquées au paragraphe précédent restent valables si les données d'apprentissage $D = (I_+, I_-)$ incluent strictement un échantillon caractéristique, c'est-à-dire $I_+ \supset I_+^c$ et $I_- \supset I_-^c$ puisque l'ajout de nouveaux exemples correspond à l'ajout d'états auxiliaires, sans modifier les contraintes sur les fusions possibles entre états fixes et variables. De même, de nouveaux contre-exemples peuvent définir comme incompatibles des fusions entre états fixes et variables mais ces fusions étaient déjà nécessairement niées par des éléments de I_-^c.

Etats auxiliaires, les "responsables" du non-déterminisme

Les états d'un automate canonique sont représentés par les états fixes de l'arbre des préfixes et ses transitions et états d'acceptation par les éléments du noyau incluant états fixes et variables. L'automate canonique étant déterministe, le non-déterminisme d'un automate quotient du $PTA(I_+^c)$ est nécessairement introduit par des transitions pointant vers un état auxiliaire. Nous allons illustrer cette propriété par un exemple.

Reprenant l'arbre des préfixes de la figure 3.2, nous avons défini un automate quotient par la partition

$$\pi = \{\{0,1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\},\{9\},\{10\}\}.$$

Cet automate est représenté[3] à la figure 3.3 (a). Comme nous le verrons, il peut être obtenu au cours de l'application de l'algorithme RPNI avec l'échantillon caractéristique défini précédemment. Rappelons que les états 0 et 2 sont fixes, les états 1, 3 et 4 variables et les autres états auxiliaires. L'automate de la figure 3.3 (a) est déterministe. Si l'on fusionne l'état variable 3 et l'état fixe 2, on obtient l'automate représenté à la figure 3.3 (b). Si l'on n'y considère que les états fixes 0 et 2 et les transitions pointant vers ces états, le sous-automate obtenu est déterministe. Par contre, dans l'automate complet, la transition pointant vers l'état auxiliaire 5 introduit un non-déterminisme sur la lettre a. Puisque l'on cherche à construire un automate déterministe, il faut "replier" cette transition sur celle qui est déjà issue de l'état 2. Cela consiste à fusionner les états 2 et 5. Cette opération donne lieu à la *fusion pour déterminisation*, une des procédures de l'algorithme RPNI. Notons que la fusion de l'état 5 avec un autre état que l'état 2 ne permettrait pas de réduire le non-déterminisme. Par contre, cette fusion introduit un nouveau non-déterminisme que l'on peut supprimer en appliquant de nouveau la même opération.

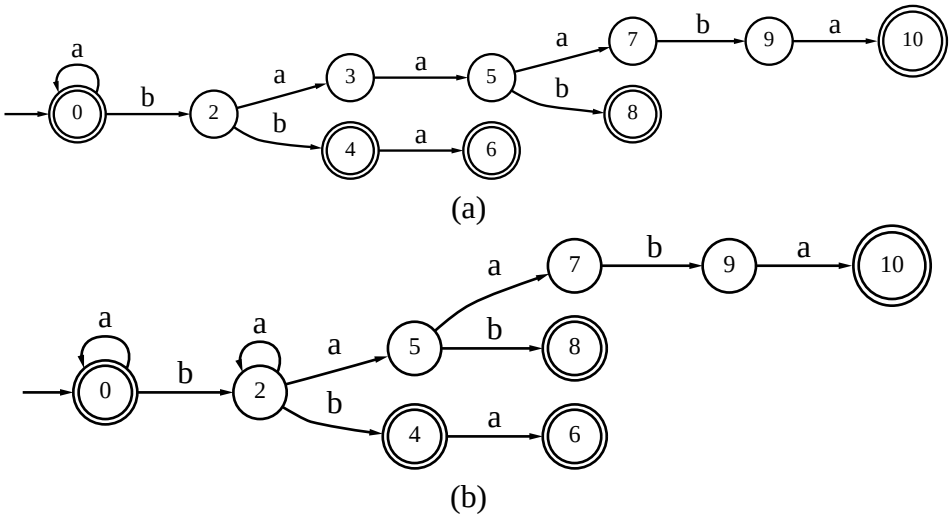


FIG. 3.3: *Automates quotient déterministe (a) et non-déterministe (b).*

Nous définissons formellement la fusion pour déterminisation comme suit. Chaque bloc B_i de la partition π a pour rang celui de son état de rang minimal dans l'ordre standard.

> **Si** B_i et B_j sont deux blocs de la partition π, avec $B_i < B_j$, tels qu'il existe un bloc $B < B_j$ et $a \in \Sigma$ avec $B_i \in \delta(B,a)$ et $B_j \in \delta(B,a)$ **alors** la nouvelle partition π' est

3. Chaque bloc est représenté par son état de rang minimal.

définie par

$$\pi' = \{B_i \cup B_j\} \cup (\pi \setminus \{B_i, B_j\})$$

En d'autres termes, l'automate quotient relativement à la partition π' est obtenu en fusionnant les blocs B_i et B_j . La fusion pour déterminisation peut entraîner un nouveau non-déterminisme introduit par un autre état. Ce nouvel état est également auxiliaire puisque nous avons, le long d'une branche du $PTA(I_+^c)$, la relation $q_{fixe} < q_{var} < q_{aux}$. Par conséquent, la fusion par déterminisation est appliquée à nouveau jusqu'à l'obtention d'un automate déterministe.

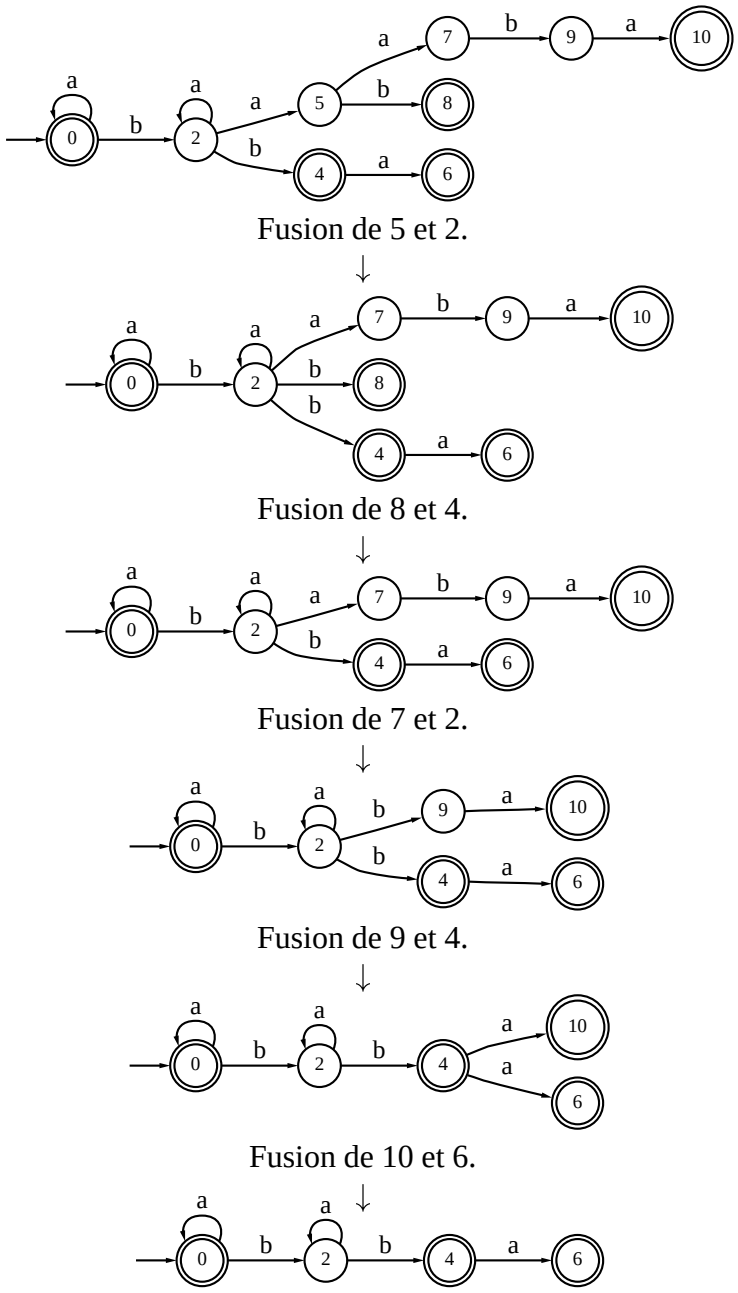


FIG. 3.4: *Fusion pour déterminisation.*

La figure 3.4 illustre la séquence d'automates obtenus lors de la fusion pour déterminisation de l'automate de la figure 3.3 (b).

3.1.3 Identification à la limite de l'algorithme RPNI

Nous reprenons la présentation de l'algorithme RPNI du paragraphe 2.2.2 pour en donner une preuve de l'identification à la limite[4].

Algorithme RPNI

entrée I_+, I_-
sortie Un AFD compatible avec I_+, I_-

début

// N désigne le nombre d'états du $PTA(I_+)$

$\pi \leftarrow \{\{0\},\{1\},\ldots,\{N-1\}\}$ // Un bloc par préfixe dans l'ordre $<$
$A \leftarrow PTA(I_+)$

pour $i = 1$ **jusqu'à** $|\pi| - 1$ // Boucle sur les blocs de la partition π
  pour $j = 0$ **jusqu'à** $i - 1$ // Boucle sur les blocs de rang inférieur
    $\pi' \leftarrow \pi \backslash \{B_j, B_i\} U \{B_i U B_j\}$ // Fusion du bloc B_i et du bloc B_j
    $A/\pi' \leftarrow$ *dériver* (A, π')
    $\pi'' \leftarrow$ *fusion_déterm* (A/π')
    si *compatible* (A/π''),I_-) **alors** // Analyse déterministe de I_-
      $A \leftarrow A/\pi''$
      $\pi \leftarrow \pi''$
      sortir boucle // Quitter la boucle sur j
    fin si
  fin pour // Fin boucle sur j
fin pour // Fin boucle sur i
retourner A
fin RPNI

Théorème 3.1 *L'algorithme RPNI identifie à la limite la classe des langages réguliers.*

Preuve :
Les états de l'arbre du $PTA(I_+)$ sont parcourus dans l'ordre standard (boucle sur l'indice i). Pour chaque état, on teste la fusion avec les états de rang inférieur (boucle sur l'indice j). Remarquons que si l'automate A/π' est incompatible avec I_-, c'est-à-dire $L(A/\pi') \cap I_- \neq \phi$, alors l'automate A/π'' est également incompatible avec I_- puisque la fusion pour déterminisation implique $L(A/\pi') \subseteq L(A/\pi'')$. La fusion pour déterminisation permet d'effectuer une analyse déterministe afin de vérifier cette compatibilité.

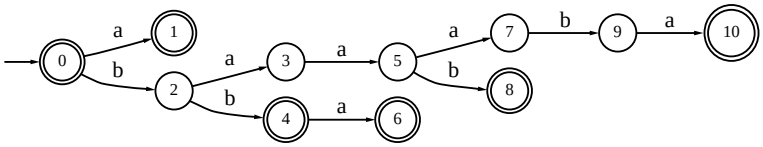
Nous supposons que les données d'apprentissage contiennent un échantillon caractéristique pour le langage régulier L. Par conséquent, si les états i et j sont des états fixes, leur fusion sera nécessairement niée par un contre-exemple et l'automate A/π'' sera incompatible avec I_-. Cela conduit à incrémenter j et à tester une autre fusion. La première fusion, dans l'ordre d'exploration, qui donnera lieu à un automate A/π' compatible correspond nécessairement à un état j fixe et un état i variable, avec $j < i$. Tous les états du sous-arbre dont l'état i est la racine sont alors des états auxiliaires. Ils introduisent un non-déterminisme qui sera éliminé par la fusion pour déterminisation. Nous avons vu que cette opération consiste à replier les transitions pointant vers les états auxiliaires sur des transitions

4. L'identification à la limite de l'algorithme RPNI a été démontrée dans [OG92]. Nous en donnons une démonstration reformulée qui, en particulier, inclut explicitement la fusion pour déterminisation.

pointant vers des états fixes. Dans ce cas, si l'automate A/π' est compatible avec I_-, l'automate A/π'' l'est également.

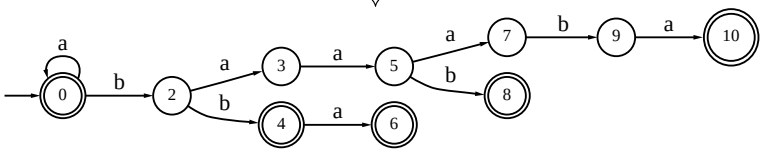
En résumé, si les données d'apprentissage contiennent un échantillon caractéristique, aucune fusion entre états fixes ne donne lieu à un automate compatible avec I_-. De plus, seule la fusion de chaque état variable avec l'état fixe partageant le même ensemble de finales, donne lieu à un automate compatible. Enfin, les états auxiliaires sont fusionnés nécessairement après la fusion d'un état variable sur un état fixe. Comme la seule manière de réduire le non-déterminisme est de replier les transitions pointant vers les états auxiliaires sur des transitions pointant déjà vers des états fixes, cette opération ne modifie pas, dans ce cas particulier, la compatibilité avec I_-. Finalement, comme tous les états fixes et variables sont nécessairement présents, les ensembles d'états (éventuellement d'acceptation) et transitions de $A(L)$ sont donc obtenus et l'identification du langage L est garantie. □

La figure 3.5 présente un exemple d'exécution complète de l'algorithme RPNI avec un échantillon caractéristique ; $I_+^c = \{\lambda, a, bb, bba, baab, baaaba\}$ et $I_-^c = \{b, ab, aba\}$. Chaque état y est identifié par l'état de rang minimal du bloc qu'il représente. Le symbole "↓" relie les automates obtenus par fusion pour déterminisation. Le symbole "⇓" relie les automates dérivés par une autre fusion. Lorsque l'automate obtenu est incompatible, une autre fusion est effectuée dans l'ordre défini.

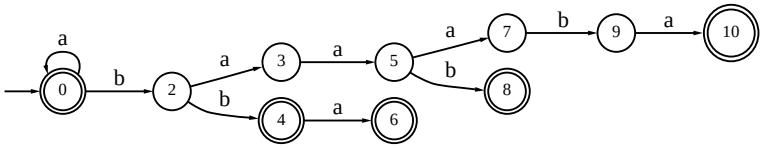


Fusion de 1 et 0.

⇓

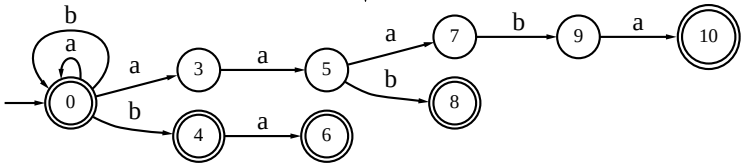


Compatibilité avec I_-



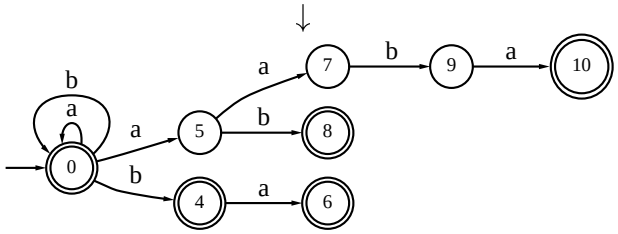
Fusion de 2 et 0.

⇓

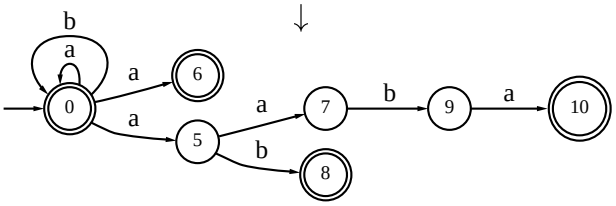


Fusion de 3 et 0.

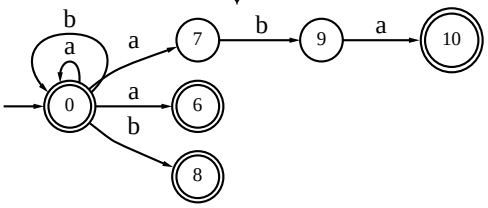
↓



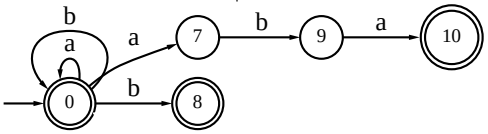
Fusion de 4 et 0.



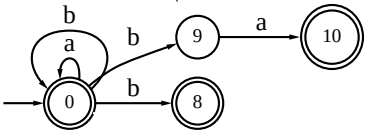
Fusion de 5 et 0.



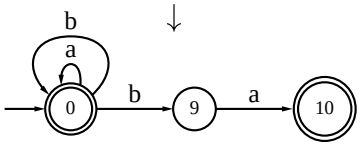
Fusion de 6 et 0.



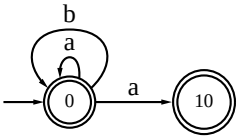
Fusion de 7 et 0.



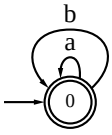
Fusion de 8 et 0.



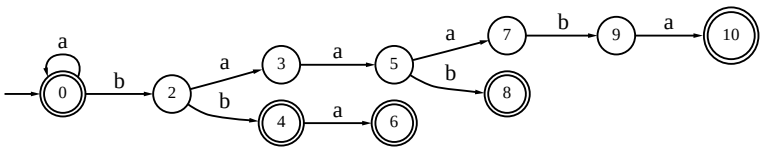
Fusion de 9 et 0.



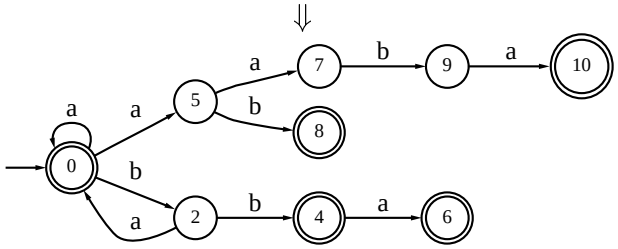
Fusion de 10 et 0.



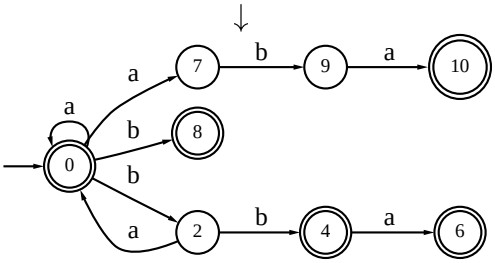
Incompatibilité avec I_-



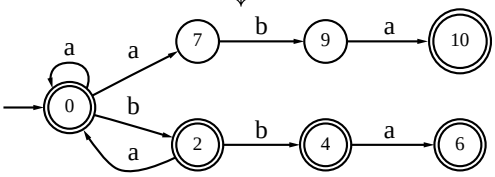
Fusion de 3 et 0.



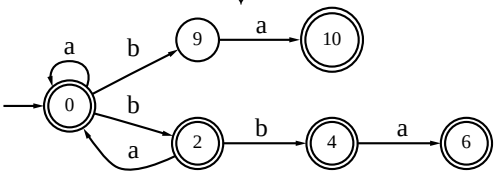
Fusion de 5 et 0.



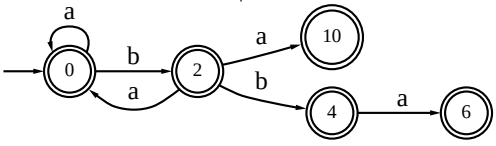
Fusion de 8 et 2.



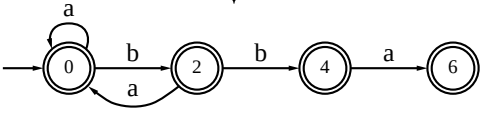
Fusion de 7 et 0.



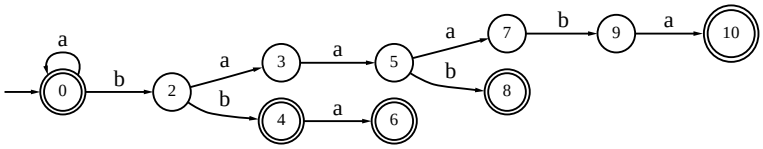
Fusion de 9 et 2.



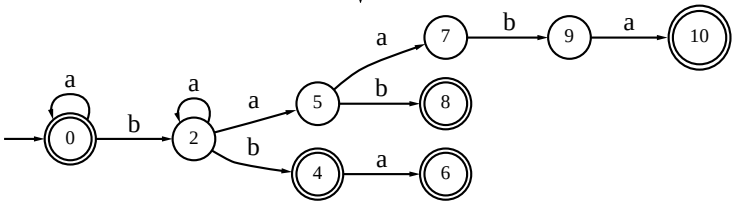
Fusion de 10 et 0.



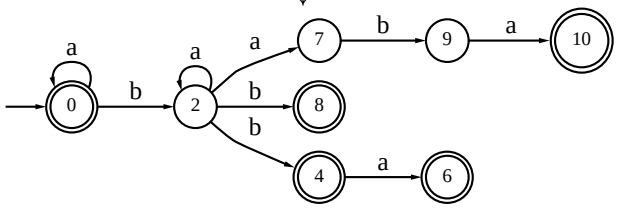
Incompatibilité avec I_-



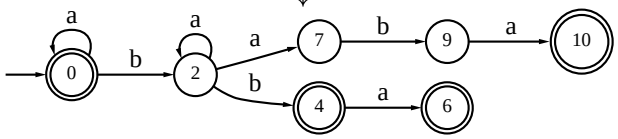
Fusion de 3 et 2.



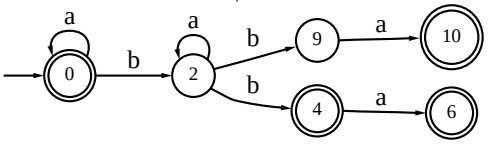
Fusion de 5 et 2.



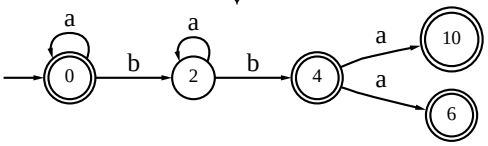
Fusion de 8 et 4.



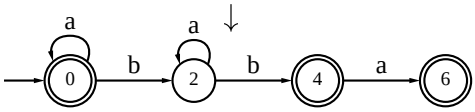
Fusion de 7 et 2.



Fusion de 9 et 4.



Fusion de 10 et 6.



Compatibilité avec I_{-}

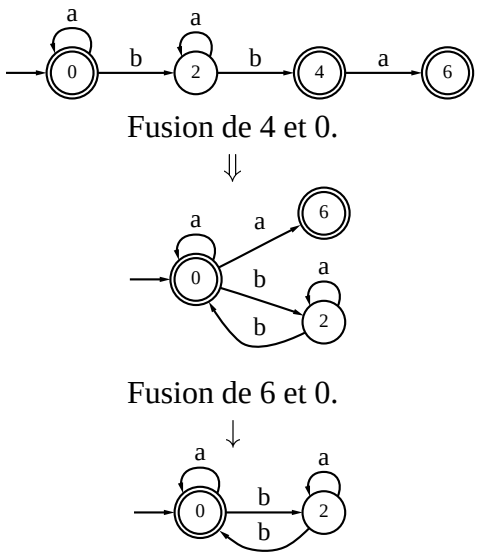


FIG. 3.5: *Exemple d'exécution de l'algorithme RPNI.*

3.1.4 Un élément remarquable de l'ensemble frontière

Théorème 3.2 *Lorsque les données d'apprentissage contiennent un échantillon caractéristique pour le langage L, l'automate $A(L)$ est l'automate déterministe comportant le moins d'états et appartenant à $BS_{PTA}(I_+, I_-)$. Par conséquent, $A(L)$ est la solution au problème du plus petit AFD compatible.*

Preuve
Comme les données contiennent un échantillon caractéristique, tous les états fixes et variables sont présents dans le $PTA(I_+)$. Aucun automate comportant moins d'états que le nombre d'états fixes, c'est-à-dire le nombre d'états de $A(L)$, ne peut être compatible avec I_- car cela nécessiterait la fusion d'au moins deux états fixes. De plus, tout automate A déterministe, comportant le même nombre d'états que $A(L)$ et compatible, est nécessairement isomorphe à $A(L)$. En effet, pour assurer la compatibilité de A, il faut que chaque état variable soit fusionné avec l'état fixe partageant le même ensemble de finales. Si tous les états variables sont fusionnés avec leur état fixe respectif, la seule manière d'obtenir un automate déterministe est de replier les états auxiliaires conformément à la fusion pour déterminisation.□

Le lecteur aura sans doute remarqué la similitude entre les preuves des théorèmes 3.1 et 3.2. Nous avons distingué ces deux résultats pour insister sur le fait que le second complète la caractérisation de l'espace de recherche, indépendamment de l'algorithme d'inférence utilisé. En ce sens, la définition d'un échantillon caractéristique devient plus générale. En résumé, le théorème 3.2 garantit l'existence d'un élément remarquable de l'ensemble frontière alors que le théorème 3.1 garantit que l'on trouve cet élément par application de l'algorithme RPNI.

Bien que l'algorithme RPNI soit de complexité polynomiale, le théorème 3.2 ne contredit pas le caractère NP-difficile du problème du plus petit AFD compatible. En effet, ce résultat négatif suppose des données quelconques alors que les données qui assurent l'identification contiennent un échantillon caractéristique.

Le théorème 3.2 est également cohérent avec le résultat de Gold qui, dans le cas des langages réguliers, consiste à obtenir l'identification en énumérant des automates déterministes de taille croissante à partir d'un automate à un seul état [Gol78]. En appliquant une stratégie conservatrice, qui consiste

à ne poursuivre l'énumération que si le dernier automate construit n'est pas compatible, l'énumération s'arrêtera nécessairement sur le plus petit AFD compatible. Cet algorithme d'énumération a évidemment une complexité non-polynomiale.

Finalement, le théorème 3.2 est une justification théorique des algorithmes heuristiques BRIG et GIG puisqu'il démontre l'utilité de rechercher un élément compatible à une profondeur maximale dans le treillis $Lat(PTA(I_+))$. Néanmoins, ceci illustre également la limite des algorithmes BRIG et GIG car si l'on suppose que les données contiennent un échantillon caractéristique, l'algorithme qui s'impose est RPNI.

3.2 L'algorithme RPNI avec suppression des finales

3.2.1 Présentation de l'algorithme

Nous savons que les états auxiliaires sont ceux qui introduisent le non-déterminisme lors des fusions dans l'arbre des préfixes. La fusion pour déterminisation a été introduite dans l'algorithme RPNI afin de supprimer le non-déterminisme. Elle a pour effet de replier les transitions pointant sur des états auxiliaires sur celles pointant vers des états fixes. Dans la mesure où l'on replie des transitions sur des transitions déjà existantes, les états auxiliaires et les transitions associées sont superflus. Nous proposons une variante de l'algorithme RPNI avec *suppression des finales*. Elle consiste à éliminer tous les états qui sont considérés comme états auxiliaires (voir paragraphe 3.1.2).

Le problème consiste à déterminer quels sont les états fixes, variables et auxiliaires dans le $PTA(I_+)$ sans pour autant connaître l'automate canonique du langage cible. Nous savons que, si les données contiennent un échantillon caractéristique, les fusions entre états fixes sont interdites car elles donnent lieu à des automates incompatibles avec I_-. Par contre, si une fusion entre un état j et un état i, avec $j < i$, est permise et si les fusions sont effectuées dans l'ordre propre à l'algorithme RPNI alors l'état i est nécessairement variable et l'état j est fixe. Le sous-arbre dont l'état i est la racine est constitué d'états auxiliaires à l'exception de l'état i. Dès lors, avant fusion avec l'état j, toutes les finales de l'état i correspondent aux finales de l'état i dans le $PTA(I_+)$ et peuvent être supprimés du nouvel automate quotient.

L'algorithme RPNI avec suppression des finales peut être résumé de la manière suivante.

Algorithme RPNI avec suppression des finales

entrée I_+, I_-
sortie Un AFD compatible avec I_-

début

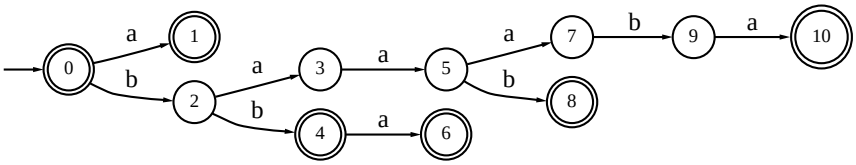
// N désigne le nombre d'états du $PTA(I_+)$

$\pi \leftarrow \{\{0\}, \{1\}, \ldots, \{N-1\}\}$ // Un bloc par préfixe dans l'ordre $<$
$A \leftarrow PTA(I_+)$

pour $i = 1$ **jusqu'à** $|\pi| - 1$ // Boucle sur les blocs de la partition π
  pour $j = 0$ **jusqu'à** $i - 1$ // Boucle sur les blocs de rang inférieur
    $\pi' \leftarrow \pi \backslash \{B_j, B_i\} U \{B_i U B_j\}$ // Fusion du bloc B_i et du bloc B_j
    $A/\pi' \leftarrow$ *dériver* (A, π')
    si *compatible* $(A/\pi', I_-)$ **alors** // Analyse non-déterministe de I_-
      $A'/\pi'' \leftarrow$ *suppression_finales* $(A/\pi', i)$
      $A \leftarrow A'/\pi''$
      $\pi \leftarrow \pi''$
      sortir boucle // Quitter la boucle sur j
    fin si
  fin pour // Fin boucle sur j
fin pour // Fin boucle sur i
retourner A
fin RPNI avec suppression des finales

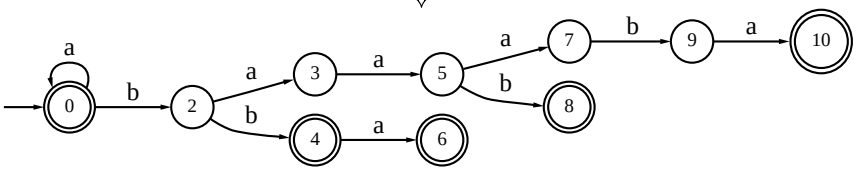
La fonction *compatible* $(A/\pi', I_-)$ renvoie VRAI si l'automate quotient est compatible avec I_-, sinon elle renvoie FAUX. La fonction *suppression_finales* $(A/\pi', i)$ renvoie l'automate obtenu en supprimant de l'automate A/π' les finales de l'état i dans le $PTA(I_+)$. Elle adapte la partition π' et renvoie l'automate A'/π''.

La figure 3.6 présente un exemple d'exécution de l'algorithme RPNI avec suppression des finales. Les données sont celles de l'échantillon caractéristique de l'exemple déjà cité, c'est-à-dire $I_+^c = \{\lambda, a, bb, bba, baab, baaaba\}$ et $I_-^c = \{b, ab, aba\}$. Le symbole "$\Downarrow$" relie les automates dérivés par une fusion. Lorsque l'automate obtenu est incompatible, une autre fusion est effectuée dans l'ordre défini. Le symbole "$|$" relie les automates obtenus par suppression des finales.

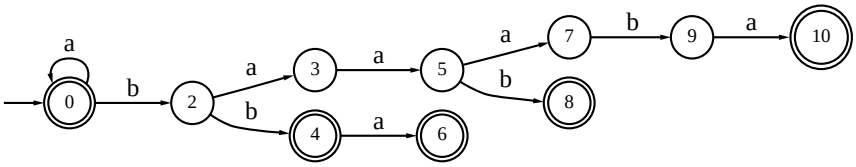


Fusion de 1 et 0.

⇓

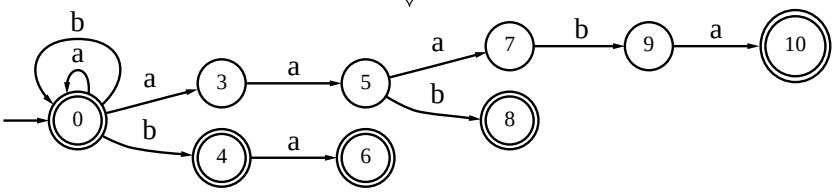


Compatibilité avec I_-

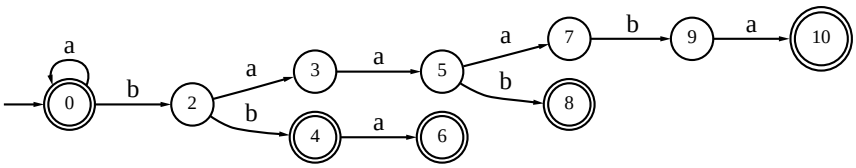


Fusion de 2 et 0.

⇓

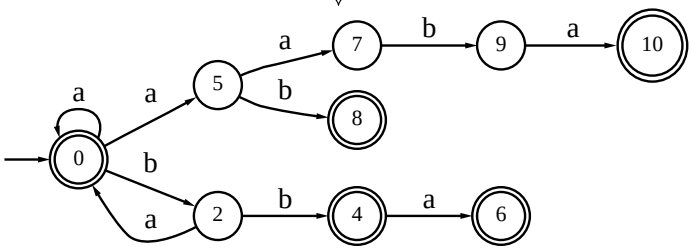


Incompatibilité avec I_-

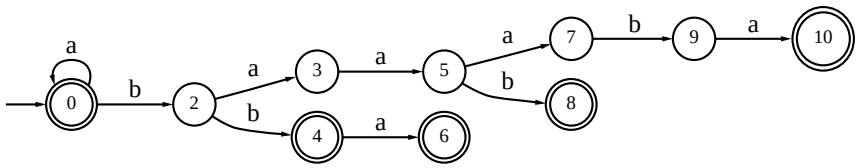


Fusion de 3 et 0.

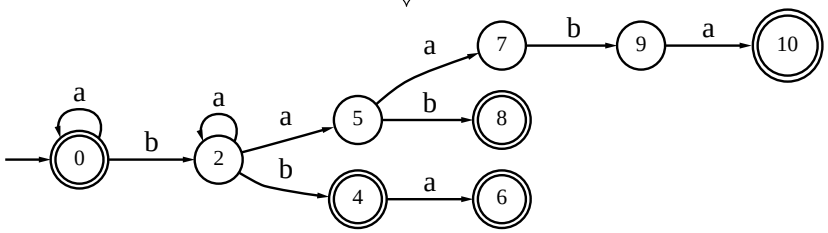
⇓



Incompatibilité avec I_-

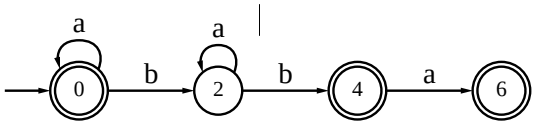


Fusion de 3 et 2.

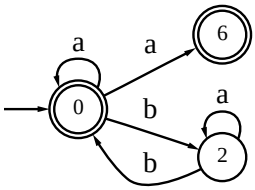


Compatibilité avec I_-

Suppression des finales de 3 avant sa fusion avec 2.



Fusion de 4 et 0.



Compatibilité avec I_-

Suppression des finales de 4 avant sa fusion avec 0.

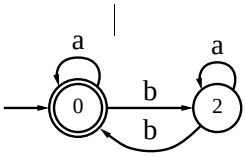


FIG. 3.6: *Exécution de l'algorithme RPNI avec suppression des finales.*

3.2.2 Propriétés de l'algorithme RPNI avec suppression des finales

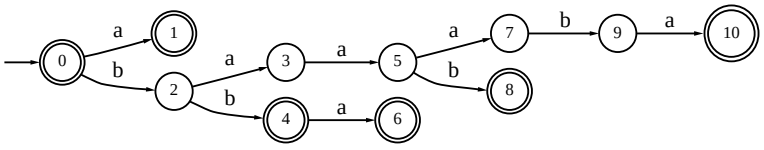
Cette variante de l'algorithme RPNI offre une autre analyse de sa complexité. Nous avions vu au paragraphe 2.2.2 qu'elle est estimée à $\mathcal{O}((\|I_+\| + \|I_-\|) \cdot \|I_+\|^2)$. Or, nous constatons que les fusions importantes sont celles qui concernent les état fixes et les états variables puisque la compatibilité avec I_- ne doit être testée que pour ces fusions. Dans l'algorithme RPNI original, la compatibilité est testée après la fusion pour déterminisation. Elle nécessite donc une analyse déterministe de I_- dont la complexité globale est $\mathcal{O}(||I_-||)$. Dans notre variante, la compatibilité est testée sur un automate qui peut être non-déterministe. Ceci est dû aux états auxiliaires qui ne sont éliminés que lorsque la compatibilité est assurée. Cependant, le non-déterminisme est limité car, par hypothèse de récurrence de l'algorithme, chaque fois que l'on sort de la boucle intérieure (la boucle sur j) l'automate courant

A est déterministe. Seul l'état i, une fois fusionné avec l'état j, peut présenter un non-déterminisme, éventuellement sur plusieurs symboles. Chaque chaîne de I_- présente au plus deux acceptations par l'automate quotient. L'analyse non-déterministe peut donc être calculée par deux analyses déterministes, dont la complexité globale est $\mathcal{O}(||I_-||)$. Finalement, en négligeant la complexité de la suppression des finales, sachant qu'il y a $|Q|$ états fixes et que l'union des états variables et fixes représentent $\mathcal{O}(|Q| \cdot |\Sigma|)$ états, la complexité globale de l'algorithme peut être estimée à $\mathcal{O}(|Q|^2 \cdot |\Sigma| \cdot ||I_-||)$. Cette complexité dépend donc *linéairement* de la taille des données d'apprentissage. Bien que la complexité calculée soit optimiste, puisque nous avons négligé le coût de la suppression des finales, elle correspond mieux à l'évolution, observée expérimentalement, du temps de calcul en fonction du volume de données d'apprentissage.

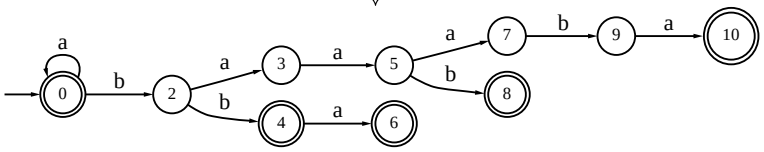
Dans la mesure où le traitement n'est modifié, par rapport à l'algorithme original, que pour les états auxiliaires, l'algorithme RPNI avec suppression des finales identifie à la limite la classe des langages réguliers.

Cet algorithme souffre néanmoins d'une limitation importante car faute d'un échantillon caractéristique, la suppression des finales peut conduire à une solution qui ne soit pas compatible avec l'échantillon positif I_+. Cette propriété est illustrée par l'exemple suivant.

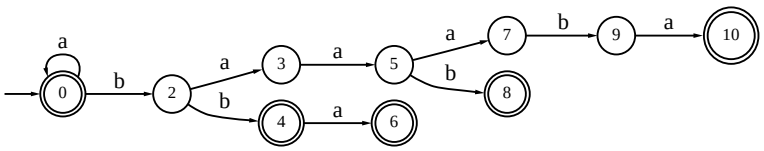
Nous considérons le même échantillon positif mais l'échantillon négatif est réduit à $I_- = \{b\}$. L'exécution de l'algorithme est présenté à la figure 3.7. La fusion de l'état 3 et de l'état 0 engendre un automate qui est compatible avec l'échantillon négatif. Par conséquent l'état 3 est considéré comme un état variable et ses finales, dans le $PTA(I_+)$, sont supprimées. La solution obtenue n'est pas compatible avec I_+, car la chaîne $baab$ n'est pas acceptée par la solution. Au contraire, dans l'algorithme original la solution proposée est toujours compatible avec I_+ et I_-, car les états considérés comme auxiliaires sont fusionnés avec d'autres au lieu d'être supprimés.



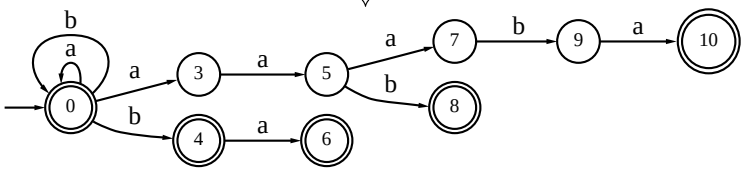
Fusion de 1 et 0.



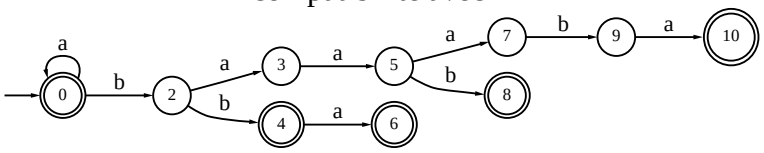
Compatibilité avec I_-



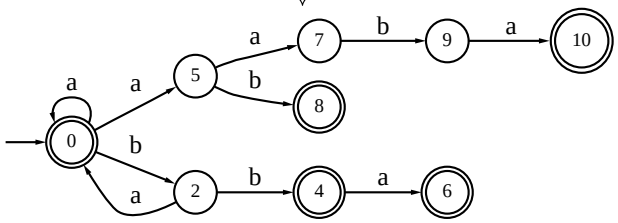
Fusion de 2 et 0.



Incompatibilité avec I_-

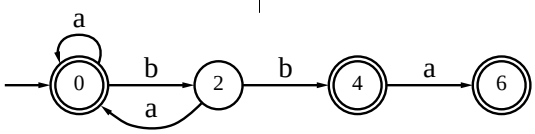


Fusion de 3 et 0.



Compatibilité avec I_-

Suppression des finales de 3 avant sa fusion avec 0.



Incompatibilité avec I_+ : rejet de la chaîne $baab$.

FIG. 3.7: *Incompatibilité vis-à-vis de l'échantillon positif par l'exécution de l'algorithme RPNI avec suppression des finales.*

3.3 Construction incrémentale de l'espace de recherche

Dans ce paragraphe, nous étudions la structure de l'espace de recherche pour une présentation séquentielle des données d'apprentissage. Dans ce cas, les données sont reçues une par une dans un ordre quelconque et elles sont étiquetées comme exemple ou contre-exemple. L'alphabet du langage à identifier est également construit au fur et à mesure en fonction des lettres présentes dans les données reçues.

Hypothèses de récurrence

Nous supposons, par hypothèse de récurrence, que la solution courante à l'étape k est un automate $A(k)$ déterministe et compatible avec l'ensemble des données reçues jusque là. Plus précisément, nous mémorisons l'ensemble des p exemples constituant l'échantillon positif courant $I_+(p)$, sous la forme de l'arbre des préfixes $PTA(I_+(p))$, et nous mémorisons l'ensemble des q contre-exemples constituant l'échantillon négatif courant $I_-(q)$, avec $k = p$ (exemples) $+ q$ (contre-exemples). De plus, la solution $A(k)$ est soit l'automate vide soit supposée appartenir à l'ensemble $BS_{PTA}(I_+(p), I_-(q))$.

Tous les exemples et contre-exemples reçus sont supposés distincts. Cette hypothèse sera justifiée par les propriétés 3.4 et 3.5 qui impliquent notamment qu'une donnée déjà présentée ne peut modifier la solution courante.

Initialisation de la récurrence

Nous pouvons aisément initialiser la récurrence en appliquant l'algorithme RPNI avec la première chaîne. S'il s'agit d'un exemple, la solution sera l'automate universel et s'il s'agit d'un contre-exemple la solution sera l'automate vide. Si la solution $A(k)$ est supposée déterministe et appartenant à l'ensemble frontière, nous savons qu'il s'agit de l'automate canonique pour le langage qu'il accepte (voir propriété 1.10).

Construction des treillis successifs

La propriété 3.2 caractérise la relation entre les treillis successifs lorsqu'un nouvel exemple est reçu.

Propriété 3.2 $Lat(PTA(I_+(p))) \not\subseteq Lat(PTA(I_+(p+1)))$.

Preuve :
Les deux treillis peuvent être complètement distincts, c'est-à-dire

$$Lat(PTA(I_+(p))) \cap Lat(PTA(I_+(p+1))) = \phi.$$

Lorsque l'exemple d'indice $p+1$ introduit un nouveau symbole de l'alphabet, l'automate universel sur le nouvel alphabet n'appartient pas à $Lat(PTA(I_+(p)))$. Comme l'automate universel est quotient de chacun des automates du treillis $Lat(PTA(I_+(p+1)))$, aucun de ces automates ne peut appartenir à $Lat(PTA(I_+(p)))$.

Par ailleurs, lorsque l'exemple d'indice $p+1$ n'introduit pas de nouvel élément de l'alphabet, l'automate universel est commun aux deux treillis, mais l'inclusion des treillis est impossible car $PTA(I_+(p)) \notin Lat(PTA(I_+(p+1)))$. En effet, le $PTA(I_+(p+1))$ est obtenu en étendant le $PTA(I_+(p))$ pour accepter le nouvel exemple. Ces deux automates diffèrent par au-moins un état d'acceptation et éventuellement plusieurs états et transitions. En particulier, $I_+(p+1)$ n'est pas inclus dans le langage accepté par le $PTA(I_+(p))$. Comme, le langage accepté par les automates de $Lat(PTA(I_+(p+1)))$ comprend $I_+(p+1)$, on en déduit que le $PTA(I_+(p))$ n'appartient pas à ce treillis. □

Acceptation d'un nouvel exemple

Partant d'une solution $A(k)$ nous vérifions si le nouvel exemple x appartient à $L(A(k))$. Sinon, nous pouvons étendre $A(k)$ par *le plus court suffixe*. Cette opération consiste à chercher le plus long

préfixe u de x accepté par l'automate $A(k)$. L'acceptation du préfixe u suppose qu'il existe un état q tel que $q \in \delta^*(q_0, u)$[5]. Comme $A(k)$ est déterministe, l'état q est nécessairement unique. Depuis l'état q, l'automate $A(k)$ est étendu par le suffixe v pour accepter $x = uv$.

Nous notons $A^x(k)$ l'automate obtenu par extension du plus court suffixe de x à partir de l'automate $A(k)$. $A^x(k)$ est nécessairement déterministe. Si x est déjà accepté par $A(k)$, le plus court suffixe v est la chaîne vide et $A^x(k) = A(k)$. Un autre cas particulier se présente si l'automate $A(k)$ est l'automate vide. L'extension correspond alors à la construction de l'arbre accepteur des préfixes de la chaîne x.

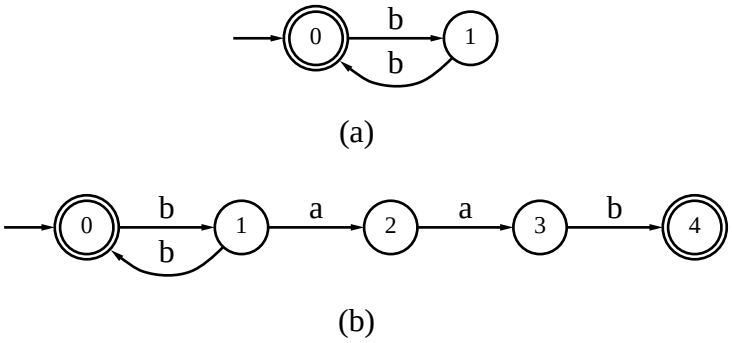


FIG. 3.8: *Extension par le plus court suffixe.*

A titre d'exemple, nous supposons que l'automate $A(k)$ est celui présenté à la figure 3.8 (a) et nous supposons que le nouvel exemple est $bbbaab$. Le plus long préfixe accepté est bbb et mène à l'état 1. Depuis cet état, l'automate est étendu par le suffixe aab. L'automate résultant est présenté à la figure 3.8 (b).

L'extension par le plus court suffixe peut créer un automate incompatible avec l'échantillon négatif courant même si $A(k)$ est compatible. Dans notre exemple, $L(A(k)) = (bb)^*$. L'extension a permis d'accepter la chaîne $bbbaab$ mais également toute chaîne de la forme $b(bb)^*aab$. Par exemple si la chaîne $baab$ appartient à l'échantillon négatif courant, la solution étendue est incompatible. Cet inconvénient sera résolu au paragraphe 3.4.1.

La propriété 3.3 stipule que l'extension par le plus court suffixe garantit que, si la solution courante à l'étape k appartient à l'espace de recherche construit sur les p premiers exemples, l'automate étendu appartient nécessairement à l'espace de recherche construit par l'ajout du nouvel exemple.

Propriété 3.3 $A(k) \in Lat(PTA(I_+(p))) \Rightarrow A^x(k) \in Lat(PTA(I_+(p+1)))$, *avec* x *l'exemple d'indice* $p+1$.

Preuve :
Par hypothèse, $A(k)$ est un automate quotient de $PTA(I_+(p))$. Donc, $I_+(p)$ est structurellement complet relativement à $A(k)$. Nous désignons par v le plus court suffixe correspondant à l'extension de $A(k)$ pour obtenir $A^x(k)$. Nous désignons par w le plus court suffixe correspondant à l'extension du $PTA(I_+(p))$ pour obtenir $PTA(I_+(p+1))$. Par construction, v est un suffixe de w et donc $I_+(p+1)$ est structurellement complet relativement à $A^x(k)$. Par conséquent, $A^x(k) \in Lat(MCA(I_+(p+1)))$. Finalement, puisque $A^x(k)$ est déterministe, il appartient à $Lat(PTA(I_+(p+1)))$. □

5. Au cas où le plus long préfixe accepté est la chaîne vide, l'état q correspond à l'état initial q_0.

La propriété 3.4 stipule que si l'automate $A(k)$, qui, par hypothèse de récurrence, appartient à l'ensemble frontière du treillis à l'étape k, accepte le nouvel exemple alors il appartient également à l'ensemble frontière dans le nouveau treillis.

Propriété 3.4 $A(k) \in BS_{PTA}(I_+(p), I_-(q))$ *et* $A(k) = A^x(k) \Rightarrow A^x(k) \in BS_{PTA}(I_+(p+1), I_-(q))$, *avec* x *l'exemple d'indice* $p+1$.

Preuve :
Nous avons $A(k) = A^x(k)$ si la chaîne x appartient au langage accepté par $A(k)$. Par hypothèse $A(k)$ appartient à $BS_{PTA}(I_+(p), I_-(q))$, c'est-à-dire qu'il est à une profondeur maximale dans le treillis $Lat(PTA(I_+(p)))$. Comme l'échantillon négatif n'est pas modifié $A(k)$ est également à une profondeur maximale dans le nouveau treillis $Lat(PTA(I_+(p+1)))$. □

Cette propriété justifie l'hypothèse que tous les exemples reçus soient distincts car un exemple x déjà rencontré impliquerait nécessairement $A^x(k) = A(k)$. Par ailleurs, l'égalité $A^x(k) = A(k)$ peut être obtenue lorsqu'un exemple x n'a pas encore été rencontré mais que les généralisations déjà effectuées jusqu'à l'étape k permettent l'acceptation de x par $A(k)$.

Réception d'un nouveau contre-exemple

La réception d'un nouveau contre-exemple ne modifie pas l'espace de recherche puisque, en vertu du théorème 1.2, la solution est à rechercher dans le treillis relatif à l'arbre accepteur des préfixes de l'échantillon positif. Par contre, l'ensemble frontière est généralement modifié par l'ajout d'un nouveau contre-exemple. En d'autres termes, la solution courante $A(k)$ appartenant à l'ensemble frontière $BS_{PTA}(I_+(p), I_-(q))$ peut ne pas appartenir à $BS_{PTA}(I_+(p), I_-(q+1))$. La propriété 3.5 stipule que si l'automate $A(k)$, qui, par hypothèse de récurrence appartient à l'ensemble frontière du treillis à l'étape k, n'accepte pas le nouveau contre-exemple alors il appartient également à l'ensemble frontière dans le même treillis avec un échantillon négatif étendu.

Propriété 3.5 $A(k) \in BS_{PTA}(I_+(p), I_-(q))$ *et* $x \notin L(A(k)) \Rightarrow A(k) \in BS_{PTA}(I_+(p), I_-(q+1))$, *avec* x *le contre-exemple d'indice* $q+1$.

Preuve :
Puisque la chaîne x n'appartient pas au langage accepté par $A(k)$ elle ne modifie pas la compatibilité de $A(k)$ relativement à l'échantillon négatif. L'automate $A(k)$ étant par hypothèse à une profondeur maximale conformément aux q premiers contre-exemples, il reste à une profondeur maximale par l'ajout du nouveau contre-exemple. Par conséquent, $A(k)$ appartient à $BS_{PTA}(I_+(p), I_-(q+1))$.□

Cette propriété justifie également l'hypothèse que tous les contre-exemples reçus soient distincts car dans le cas de la réception d'un contre-exemple déjà rencontré l'ensemble frontière ne serait pas modifié et la solution courante resterait forcément compatible.

3.4 L'algorithme RPNI2

3.4.1 Comment garantir la compatibilité avec I_-

Les différentes propriétés démontrées dans le paragraphe 3.3 caractérisent l'espace de recherche et son ensemble frontière lors de l'ajout d'un nouvel exemple ou d'un nouveau contre-exemple. De plus, nous avons vu qu'il y a deux cas possibles d'incompatibilité avec l'échantillon négatif. Le premier cas résulte de l'extension par le plus court suffixe. Elle permet d'étendre la solution courante et

d'obtenir un élément qui appartient nécessairement au nouvel espace de recherche. Cet élément peut néanmoins être incompatible avec l'échantillon négatif. Le second cas concerne l'ajout d'un nouveau contre-exemple qui rend la solution courante incompatible.

Deux cas équivalents d'incompatibilité

Les deux cas d'incompatibilité avec I_- peuvent être traités de façon identique. Nous avons un automate A déterministe qui appartient à $Lat(PTA(I_+))$ et tel que $L(A) \cap I_- \neq \phi$. De plus, nous connaissons la partition π telle que $A = PTA(I_+)/\pi$. En effet, cette partition peut être mémorisée d'une étape à la suivante lorsque un nouveau contre-exemple est reçu ou facilement adaptée lors de l'extension par le plus court suffixe[6].

Un exemple d'incompatibilité avec I_-

A titre d'exemple, considérons l'arbre des préfixes et un automate quotient A déterministe représentés à la figure 3.9. Les données déjà reçues sont supposées être $I_+ = \{\lambda, a, bb, bba, baab, baaaba\}$ et $I_- = \{abbb\}$.

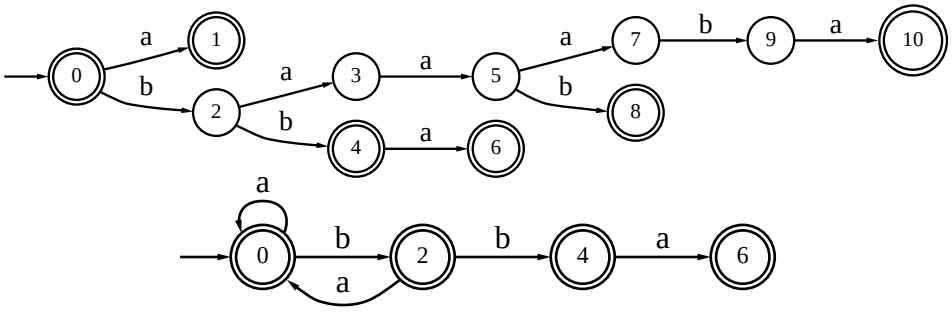


FIG. 3.9: *L'arbre des préfixes et un automate quotient déterministe.*

On vérifie aisément que la partition permettant d'obtenir A à partir du $PTA(I_+)$ est donnée par

$$\pi = \{\{0, 1, 3, 5, 7, 10\}, \{2, 8, 9\}, \{4\}, \{6\}\}.$$

Supposons que nous recevions b comme nouveau contre-exemple. L'automate A accepte la chaîne b et devient donc incompatible avec $I_- = \{b, abbb\}$. L'automate A est incompatible car il y a au moins une fusion d'états qui permet l'acceptation de la chaîne b. Nous cherchons une partition π' plus fine que π et tel que $PTA(I_+)/\pi'$ soit compatible avec I_-.

Traitement d'un contre-exemple qui crée l'incompatibilité avec I_-

Nous savons que le rôle des contre-exemples est de distinguer les éléments du noyau et des préfixes courts qui n'ont pas même ensemble de finales. Plus précisément, nous supposons que le contre-exemple w participe à la définition d'un échantillon caractéristique. Dans ce cas, il existe un préfixe $u \in \mathrm{Pr}(L)$ et une chaîne v tels que $w = uv$ et $uv \in I_-^c$. Connaissant le contre-exemple w et l'arbre des préfixes, il est aisé de déterminer les chaînes u et v. Dans notre cas, le contre-exemple b mène à l'état 2 du $PTA(I_+)$. Nous avons $u = b$ et $v = \lambda$. Puisque le contre-exemple est accepté par l'automate

6. Chaque état créé par l'extension est associé à un nouveau bloc dont il est le seul élement.

quotient, il y a nécessairement un état q_v dont le suffixe est v et qui appartient au bloc du préfixe u. Dans notre exemple, l'état q_v a λ pour suffixe et correspond à l'état 8 du $PTA(I_+)$. Il appartient dans la partition π au bloc de l'état 2. L'opération que nous venons de décrire constitue la *définition du préfixe d'acceptation*. Il détermine en particulier le *rang* de l'état q_v. Nous dirons que le bloc de l'état q_v est *fissionné* ou que l'on sort l'état q_v de son bloc pour créer un nouveau bloc $\{q_v\}$.

Respect de l'ordre standard, considéré en sens inverse

Nous cherchons à trouver une partition π' plus fine que la partition π tel que l'automate quotient soit compatible tout en garantissant l'identification à la limite. Or nous avons vu, par la démonstration du théorème 3.1, que l'ordre des fusions est important. Nous cherchons, afin de garantir l'identification, à retrouver l'automate quotient que l'algorithme RPNI construirait à partir du $PTA(I_+)$ sous contrôle du même échantillon négatif I_-. Une des caractéristiques de l'algorithme RPNI est d'essayer les fusions dans l'ordre standard. Par conséquent, si l'on nie la fusion d'un état q_v avec un autre état, il est nécessaire de nier la fusion de tous les états de rang supérieur à q_v. Dans notre exemple, q_v étant l'état 8, nous devons sortir les états 8, 9 et 10 de leurs blocs respectifs. Nous effectuons ces fissions dans l'ordre inverse c'est-à-dire que nous commençons par l'état 10. La figure 3.10 (a) représente l'automate original et la figure 3.10 (b) représente l'automate obtenu après avoir sorti l'état 10.

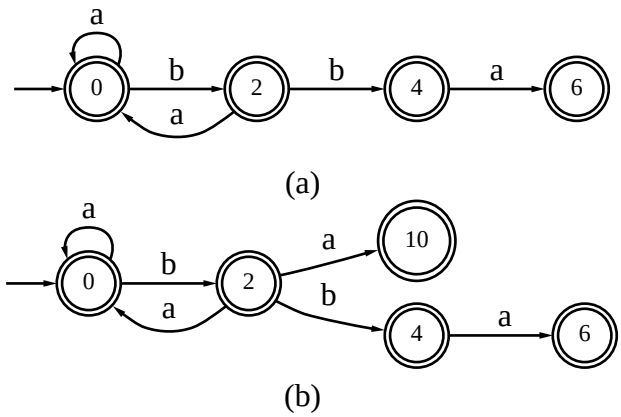


FIG. 3.10: *Fission d'un automate quotient déterministe.*

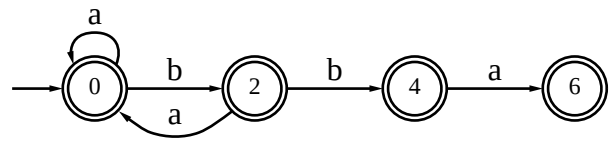
La fission pour déterminisation

Alors que l'automate original est déterministe, la fission du bloc de l'état 10 introduit un non-déterminisme. Cela signifie que l'état 10 de notre exemple est un état auxiliaire. Nous pouvons appliquer une opération de *fission pour déterminisation*, réciproque de la fusion pour déterminisation. Elle est formellement définie comme suit.

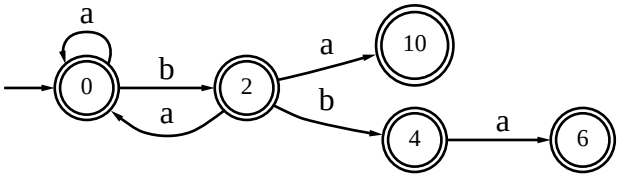
> Si $A = PTA(I_+)/\pi$ et si Bq désigne le bloc de l'état q dans la partition π**,** la *fission* de π *relativement* à q consiste à définir une nouvelle partition
>
> $$\pi' = (\pi \setminus Bq) \cup \{Bq \setminus q\} \cup \{q\}$$
>
> **Si** dans le $PTA(I_+)$ l'état q a un prédécesseur q_p sur la lettre a, c'est-à-dire $q \in \delta_{PTA}(q_p, a)$, et **si** $\exists Bq_p, B' \in \pi'$ avec $B' \neq \{q\}$ et $\delta(Bq_p, a) \supseteq \{\{q\}, B'\}$ dans $PTA(I_+)/\pi'$ **alors** effectuer la fission de π' relativement à q_p.

En d'autres termes, la fission de l'état q introduit un non déterminisme sur la lettre a puisqu'il existe un état q_p appartenant au bloc Bq_p, prédécesseur de q, duquel est issu, dans l'automate quotient $PTA(I_+)/\pi'$, une autre transition sur la lettre a. Dans ce cas, on effectue la fission de π' relativement à q_p.

L'opération de fission pour déterminisation est récursive, c'est-à-dire qu'elle implique une fission qui peut elle même induire un non-déterminisme qui sera réduit par une nouvelle fission. Pour les raisons déjà évoquées, nous désirons respecter un ordre des fissions qui soit l'ordre inverse des fusions dans l'algorithme RPNI. Chaque fois que l'on effectue la fission de la partition π relativement à l'état q, nous sortons également de leurs blocs respectifs les états du sous-arbre dont l'état q est la racine dans le $PTA(I_+)$. Ces états correspondent aux finales de q dans le $PTA(I_+)$. Dans notre exemple, ces opérations sont résumées à la figure 3.11. Le symbole "↓" relient les automates obtenus par fission pour déterminisation.

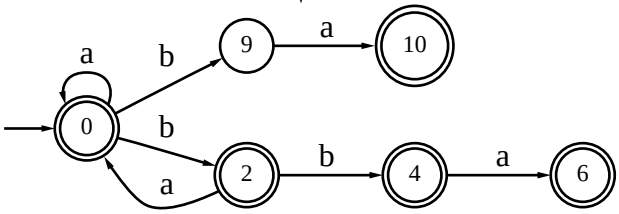


10 est sorti du bloc de 0.



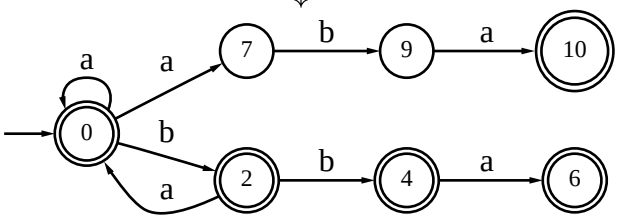
9 est sorti du bloc de 2.

↓



7 est sorti du bloc de 0.

↓



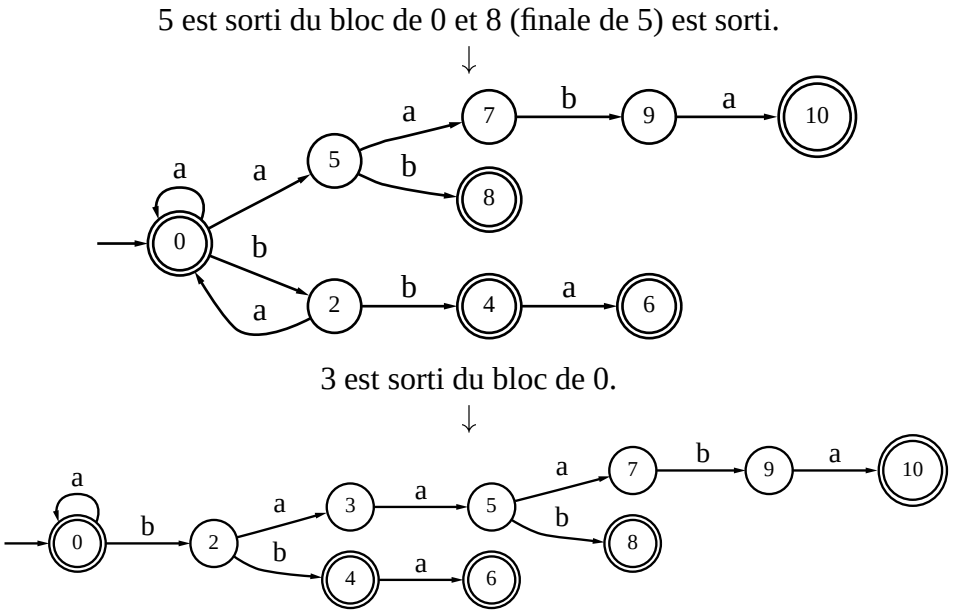


FIG. 3.11: *Fission pour déterminisation.*

Propriété 3.6 *L'opération de fission pour déterminisation s'arrête nécessairement sur un automate déterministe. Si les données contiennent un échantilllon caractéristique, elle s'arrête par la fission relativement à un état variable.*

Preuve :
Par construction, la fission pour déterminisation garantit l'obtention d'un automate déterministe. Lorsque les données incluent un échantillon caractéristique, les états auxiliaires responsables du non-déterminisme ont été sortis de leurs blocs respectifs. De plus, la définition du préfixe d'acceptation détermine le rang minimal de tous les états qui doivent participer à une fission. Lorsque tous ces états ont été traités, l'automate résultant est compatible. Puisque tous les états auxiliaires ont été sortis, le dernier état sorti est nécessairement un état variable. □

3.4.2 Identification à la limite de l'algorithme RPNI2

L'algorithme RPNI2[7] constitue l'extension incrémentale de l'algorithme RPNI. Il peut être résumé de la manière suivante.

A la réception d'un nouvel exemple x, l'arbre des préfixes est étendu par le biais de la fonction *étendre* ($PTA(I_+(p)), x$). La fonction *compatible* (A, I) renvoie VRAI si l'automate A est compatible avec l'échantillon I, sinon elle renvoie FAUX. Lorsque I est un échantillon positif, la compatibilité requiert l'acceptation de toutes les phrases de I. Réciproquement, lorsque I est un échantillon négatif, la compatibilité requiert le rejet de toutes les phrases de I. Dès lors, si l'automate courant reste compatible après la réception d'un nouvel exemple ou d'un nouveau contre-exemple, il constitue la nouvelle solution qui est retournée. Insistons sur le fait que ceci n'implique pas que la nouvelle donnée ait été vue auparavant. Par contre, lorsqu'il y a incompatibilité avec un nouvel exemple x, l'automate courant est étendu par le plus court suffixe de x. La fonction *extension_suffixe* ($A/\pi, x$) réalise cette opération. A ce stade, l'automate courant A appartient nécessairement à l'espace de recherche défini

7. L'abréviation **RPNI2** provient de la terminologie anglaise pour "Regular Positive and Negative Incremental Inference".

par l'ensemble des données reçues jusqu'alors. Cet automate est éventuellement incompatible avec I_-. La fonction *préfixe_acceptation* (A, I_-) renvoie le rang minimal de l'état relativement auquel la partition π sera fissionnée. Si, plusieurs éléments de I_- sont acceptés, c'est le rang minimal défini sur l'ensemble des acceptations qui sera considéré. La fonction *fission_déterminisation* $(A/\pi, j)$ effectue la fission de l'automate A/π relativement à l'état j et recherche, par fissions successives, un automate déterministe. Dans le pire des cas, l'automate obtenu après fission correspond à l'arbre des préfixes. Finalement, l'algorithme RPNI original est appliqué en partant de l'automate obtenu après fission. Dès lors, dans le pire des cas, l'algorithme RPNI est appliqué sans tirer parti de la solution temporaire qui avait été obtenue à l'étape précédente de la présentation séquentielle. Cependant, en général, l'algorithme RPNI sera appliqué sur un automate comportant moins d'états que l'arbre des préfixes. En d'autres termes, la recherche s'effectue dans un sous-treillis de $Lat(PTA(I_+))$.

Algorithme RPNI2

entrée $PTA(I_+(p))$ // L'arbre accepteur des préfixes de p exemples
$I_-(q)$ // L'échantillon négatif comportant q contre-exemples
A/π // L'automate solution courante
x // La nouvelle donnée à traiter
sortie Un AFD compatible avec $(I_+(p), I_-(q))$ et x

début

si x est un exemple **alors**
  $PTA(I_+(p+1)) \leftarrow$*étendre* $(PTA(I_+(p)), x)$ // Extension de l'arbre des préfixes
  si *compatible* $(A/\pi,\{x\})$ **alors** // Vérification de l'acceptation du nouvel exemple
    retourner A/π
  sinon
    $A/\pi' \leftarrow$ *extension_suffixe* $(A/\pi, x)$ // Extension par le plus court suffixe
    $A \leftarrow A/\pi'$
    $\pi \leftarrow \pi'$
  fin si
fin si

si x est un contre-exemple **alors**
  si *compatible* $(A/\pi,\{x\})$ **alors** // Vérification du rejet du nouveau contre-exemple
    retourner A/π
  sinon
    $A \leftarrow A/\pi'$
    $\pi \leftarrow \pi'$
  fin si
fin si

$i \leftarrow$ *préfixe_acceptation* (A, I_-) // Rang minimal des états à fissioner
// N désigne le nombre d'états du PTA
pour $j = N$ **jusqu'à** i // Fission des états dans l'ordre $<$ inverse
    $A/\pi' \leftarrow$ *fission_déterminisation* $(A/\pi, j)$
    $\pi \leftarrow \pi'$
fin pour
$A \leftarrow A/\pi$

// Application de l'algorithme RPNI à partir de l'automate A
pour $i = 1$ **jusqu'à** $|\pi| - 1$ // Boucle sur les blocs de la partition π
    pour $j = 0$ **jusqu'à** $i - 1$ // Boucle sur les blocs de rang inférieur
      $\pi' \leftarrow \pi \backslash \{B_j, B_i\} U \{B_i U B_j\}$ // Fusion du bloc B_i et du bloc B_j
      $A/\pi' \leftarrow$ *dériver* (A, π')
      $\pi'' \leftarrow$ *fusion_déterm* (A/π')
      si *compatible* $(A/\pi'',I_-)$ **alors** // Analyse déterministe de I_-
        $A \leftarrow A/\pi''$
        $\pi \leftarrow \pi''$
        sortir boucle // Quitter la boucle sur j
      fin si
    fin pour // Fin boucle sur j
fin pour // Fin boucle sur i
retourner A
fin RPNI2

Théorème 3.3 *L'algorithme RPNI2 identifie à la limite la classe des langages réguliers.*

Preuve :
Par hypothèse de récurrence et par la propriété 3.3, nous avons la garantie que l'automate obtenu, éventuellement après son extension par le plus court suffixe, est déterministe et appartient au nouvel espace de recherche. En vertu des propriétés 3.4 et 3.5, nous savons que s'il est compatible avec la nouvelle donnée alors il appartient à l'ensemble frontière. Si, par contre, il est incompatible avec I_- l'automate sera fissionné. La propriété 3.6 nous garantit que, lorsque les données contiennent un échantillon caractéristique, le résultat de la fission pour déterminisation est un automate compatible et que le dernier état sorti de son bloc est un état variable. En d'autres termes, un état variable est soit déjà fusionné avec l'état fixe partageant les mêmes finales, soit il est sorti de son bloc ainsi que toutes les finales de cet état dans le $PTA(I_+)$. Par conséquent, l'ordre des fusions ultérieures est cohérent avec celui de l'algorithme RPNI et, en vertu du théorème 3.1, l'identification à la limite est garantie.□

Le théorème 3.3 garantit l'identification de l'automate cible lorsque les données, reçues séquentiellement, contiennent un échantillon caractéristique. L'identification peut également être obtenue avec les mêmes données par l'algorithme RPNI appliqué sur une présentation à données fixées. Cela revient à mémoriser toutes les données reçues séquentiellement et à appliquer l'algorithme RPNI sur cet ensemble de données. Dans ce cas particulier, la solution proposée par les deux algorithmes est identique. Par contre, lorsque les données n'incluent pas d'échantillon caractéristique il n'y a pas d'assurance que les solutions proposées soient identiques. Cependant, des évaluations expérimentales [Dup96a] ont montré que ces deux algorithmes fournissent en pratique des résultats équivalents. L'intérêt de l'algorithme RPNI2 sur l'algorithme RPNI est qu'il permet de réduire la complexité globale de l'inférence. En effet, en cas d'incompatibilité de la solution trouvée à l'étape précédente, il cherche à construire une nouvelle solution dans un espace de recherche qui est généralement un sous-espace de celui de l'algorithme RPNI [Dup96a].

3.4.3 Complexité de l'algorithme RPNI2

Dans ce paragraphe, nous allons comparer la complexité théorique de l'algorithme RPNI2 par rapport à celle de l'algorithme RPNI. Si l'on désire appliquer l'algorithme RPNI lorsque les données sont présentées de façon séquentielle, il suffit de mémoriser les échantillons positif I_+ et négatif I_- reçus jusqu'à l'étape en cours. Dans la mesure où ces informations sont également mémorisées pour l'application de l'algorithme RPNI2, leur complexité spatiale est identique[8] et vaut $\mathcal{O}(||I_+|| + ||I_-||)$.

Nous désirons évaluer le coût du traitement lors de la réception d'un nouvel exemple dans le cas où la solution de l'étape précédente est incompatible avec la nouvelle donnée. Dans ce cas, l'application de l'algorithme RPNI est effectuée dans le nouvel espace avec une complexité globale de $\mathcal{O}((\|I_+\| + \|I_-\|) \cdot \|I_+\|^2)$ (voir paragraphe 2.2.2). La dernière partie de l'algorithme RPNI2 consiste à appliquer l'algorithme RPNI à partir d'un automate quotient de l'arbre des préfixes. Dans le pire des cas, cet automate est l'arbre des préfixes lui-même et la complexité de l'algorithme RPNI2 est donc également $\mathcal{O}((\|I_+\| + \|I_-\|) \cdot \|I_+\|^2)$. Les autres opérations propre à l'algorithme RPNI2 ont une complexité négligeable asymptotiquement. En effet, l'extension par le plus court suffixe de la chaîne x a une complexité linéaire $\mathcal{O}(|x|)$. La définition du préfixe d'acceptation nécessite une analyse de tout l'échantillon négatif, car plusieurs contre-exemples peuvent être acceptés après l'extension par le plus court suffixe. Comme à ce stade l'automate courant est nécessairement déterministe, cette analyse

8. En toute rigueur, la complexité spatiale de l'algorithme RPNI2 est légèrement supérieure puisque l'automate solution à l'étape précédente est également mémorisé. Néanmoins, la complexité spatiale asymptotique est effectivement $\mathcal{O}(||I_+|| + ||I_-||)$.

présente une complexité linéaire $\mathcal{O}(||I_-||)$. Pour chaque contre-exemple accepté, il faut déterminer le préfixe d'acceptation dans le $PTA(I_+)$. Ceci représente une deuxième analyse déterministe des contre-exemples acceptés qui est également de complexité linéaire $\mathcal{O}(||I_-||)$. Finalement, la fission pour déterminisation consiste à sortir au plus autant d'éléments de la partition courante qu'il y a d'états dans le $PTA(I_+)$. Elle a donc une complexité qui dépend de la taille de l'arbre des préfixes, c'est-à-dire $\mathcal{O}(||I_+||)$.

En résumé, les algorithmes RPNI et RPNI2 ont une complexité spatiale et temporelle identique asymptotiquement. L'apport de l'inférence incrémentale, outre son intérêt conceptuel, permet de réduire la complexité pratique. En effet, en général, l'algorithme RPNI2 applique des fusions dans un sous-ensemble de $Lat(PTA(I_+))$ [Dup96b].

Bibliographie

[ADO91] N. Alon, A.K. Dewdey, and T.J. Ott. Efficient simulation of finite automata by neural nets. *Journal of the Association for Computing Machinery*, 38(2):495–514, 1991.

[Ang78] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[Ang82] D. Angluin. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29(3):741–765, 1982.

[Ang87] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1987.

[Ang88] D. Angluin. Identifying languages from stochastic examples. Technical Report YALEU/DCS/RR-614, Yale University, March 1988.

[AS83] D. Angluin and C.H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, 1983.

[AU72] A. Aho and J. Ullman. *The Theory of Parsing, Translation and Compiling, Vol. 1: Parsing*. Prentice-Hall, Englewood Cliffs, 1972.

[Bar74] J. M. Barzdin. Two theorems on the limiting synthesis of functions. Zapiski 210, Latv. Gos. Univ., 1974.

[BF72] A.W. Biermann and J.A. Feldman. A survey of results in grammatical inference. In S. Watanabe, editor, *Frontiers of Pattern Recognition*, pages 31–54. Academic Press, New York, 1972.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[CS83] J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theor. Comput. Science*, 25:193–220, 1983.

[CSSM89] A. Cleeremans, D. Servan-Schreiber, and J. McClelland. Finite state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381, 1989.

[DB62] F.N. David and D.E. Barton. *Combinatorial Chance*. Charles Griffin & Co. Ltd., London, 1962.

[DMV94] P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 25–37. Springer Verlag, 1994.

[Dup94] P. Dupont. Regular grammatical inference from positive and negative samples by genetic search : the GIG method. In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, pages 236–245. Springer Verlag, 1994.

[Dup96a] P. Dupont. Incremental regular inference. In *Grammatical Inference : Learning Syntax from Sentences, ICGI'96*, number 1147 in Lecture Notes in Artificial Intelligence, pages 222–237. Springer Verlag, 1996.

[Dup96b] P. Dupont. *Utilisation et Apprentissage de Modèles de Langage pour la Reconnaissance de la Parole Continue*. Thèse de Doctorat, Ecole Nationale Supérieure des Télécommunications, Paris, France, 1996.

[Elm88] J.L. Elman. Finding structure in time. Technical Report CRL Tech. Report 9901, University of California, San Diego, California, 1988.

[FB75a] K.S. Fu and T.L. Booth. Grammatical inference: Introduction and survey, part 1. *IEEE Transactions on Systems, Man and Cybernetics*, 5:85–111, 1975.

[FB75b] K.S. Fu and T.L. Booth. Grammatical inference: Introduction and survey, part 2. *IEEE Transactions on Systems, Man and Cybernetics*, 5:409–423, 1975.

[For73] G.D. Forney. The Viterbi algorithm. *IEEE Proceedings*, 3:268–278, 1973.

[GL70] M. Gross and A. Lentin. *Notion sur les Grammaires Formelles*. Gauthier-Villars, Paris, 2-ème edition, 1970.

[GLGG91] A.L. Gorin, S.E. Levinson, A.N. Gertner, and E. Goldman. Adaptive acquisition of language. *Computer Speech and Language*, 5:101–132, 1991.

[Gol67] E.M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[Gol78] E.M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[Gre94] J. Gregor. Data-driven inductive inference of finite-state automata. *International Journal of Pattern Recognition and Artificial Intelligence*, 8(1):305–322, 1994.

[GSC+90] C.L. Giles, G.Z. Sun, H.H. Chen, Y.C. Lee, and D. Chen. High order recurrent networks and grammatical inference. In D.S. Touretzky, editor, *Advances in Neural Information Processing System*, pages 380–387. Morgan Kaufman, San Mateo, 1990.

[GSVG90] P. García, E. Segarra, E. Vidal, and I. Galiano. On the use of the Morphic Generator Grammatical Inference (mggi) methodology in automatic speech recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 4(4):667–685, 1990.

[GT78] R.C. Gonzalez and M. G. Thomason. *Syntacic Pattern Recognition, An Introduction*. Addition-Wesley, Reading, Massachusetts, 1978.

[GV90] P. García and E. Vidal. Inference of K-testable languages in the strict sense and applications to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990.

[GVC87] P. García, E. Vidal, and F. Casacuberta. Local languages, the Successor method, and a step towards a general methodology for the inference of regular grammars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6):841–845, 1987.

[Hor69] J.J. Horning. *A Study of Grammatical Inference*. Ph. D. dissertation, Computer Science Department, Stanford University, Stanford, California, 1969.

[HS66] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice-Hall, Englewood Cliffs, New Jersey, 1966.

[JWT92] A.N. Jain, A. Waibel, and D.S. Touredsky. PARSEC: a structured connectionist parsing system for spoken language. In *International Conference on Acoustic, Speech and Signal Processing*, pages 205–208, 1992.

[KV89] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *21st ACM Symposium on Theory of Computing*, pages 433–444, 1989.

[Lan92] K.J. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *5th ACM workshop on Computational Learning Theory*, pages 45–52, 1992.

[MdG94] L. Miclet and C. de Gentile. Inférence grammaticale à partir d'exemples et de contre-exemples : deux algorithmes optimaux (big et rig) et une version heuristique (brig). In *Neuvièmes Journées Francophones sur l'Apprentissage*, pages F1–F13, Strasbourg, France, 1994.

[MDV95] L. Miclet, P. Dupont, and S. Vial. Inférence grammaticale régulière : Méthodes semi-itératives et mesure de performance. In *Journées Acquisition, Validation, Apprentissage*, pages 103–116, Grenoble, France, 1995.

[Mic79] L. Miclet. *Inférence de Grammaires Régulières*. Thèse de Doctorat, Ecole Nationale Supérieure des Télécommunications, Paris, France, 1979.

[Mic80] L. Miclet. Regular inference with a tail-clustering method. *IEEE Transactions on Systems, Man and Cybernetics*, 10:737–743, 1980.

[Mic90] L. Miclet. Grammatical inference. In H. Bunke and A. Sanfeliu, editors, *Syntactic and Structural Pattern Recognition: Theory and Applications*, volume 7 of *Series in Computer Science*, pages 237–290. World Scientific, Singapore, 1990.

[Min54] M.L. Minsky. *Neural Nets and the Brain Model problem*. Ph. D. dissertation, Princeton University, Princeton, New Jersey, 1954.

[Min67] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, New York, 1967.

[Mit78] T. Mitchell. *Version Spaces: an approach to Concept Learning*. Ph. D. dissertation, Standford University, 1978.

[MP43] W.S. McCullough and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophysics*, 5:115–133, 1943.

[Mug84] S. Muggleton. Induction of regular languages from positive examples. Technical Report TIRM-84-009, Turing Institute, 1984.

[Nat91] B.L. Natarajan. *Machine Learning : A Theoretical Approach*. Morgan Kauffman Pub., San Mateo, CA, 1991.

[Nic93] J. Nicolas. L'espace des versions. In *Ecole sur l'Apprentissage automatique*, pages 19–32, Saint-Raphaël, 1993.

[NS89] M. Nakamura and K. Shikano. A study of english word category prediction based on neural networks. In *International Conference on Acoustic, Speech and Signal Processing*, pages 731–734, Glasgow, 1989.

[OG92] J. Oncina and P. García. Inferring regular languages in polynomial update time. In N. Pérez de la Blanca, A. Sanfeliu, and E.Vidal, editors, *Pattern Recognition and Image Analysis*, volume 1 of *Series in Machine Perception and Artificial Intelligence*, pages 49–61. World Scientific, 1992.

[Pit89] L. Pitt. Inductive inference, dfa's, and computational complexity. In K.P. Jantke, editor, *Analogical and Inductive Inference*, number 397 in Lecture Notes in Artificial Intelligence, pages 18–44. Springer-Verlag, Berlin, 1989.

[Pol90] J.B. Pollack. Recursive distribute representation. *Artificial Intelligence*, 46:77–105, 1990.

[PW89] L. Pitt and M. Warmuth. The minimum consistent dfa problem cannot be approximated within any polynomial. In *21st ACM Symposium on Theory of Computing*, pages 421–444, 1989.

[Rul92] H. Rulot. *ECGI : Un Algoritmo de Inferencia Gramatical Mediante Corrección de Errores*. Ph. D. dissertation, Universitát de València, 1992.

[RV84] M. Richetin and F. Vernadat. Efficient regular grammatical inference for pattern recognition. *Pattern Recognition*, 17(2):245–250, 1984.

[RV88] H. Rulot and E. Vidal. An efficient algorithm for the inference of circuit-free automata. In G. Ferratè, T. Pavlidis, A. Sanfeliu, and H. Bunke, editors, *Advances in Structural and Syntactic Pattern Recognition*, pages 173–184. NATO ASI, Springer-Verlag, 1988.

[Seg93] E. Segarra. *Una Aproximacion Inductiva a la Comprension del Discurso Continuo*. Ph. D. dissertation, Universidad Politécnica de Valencia, 1993.

[SM89] K.R. Smith and M.I. Miller. Learning regular grammars on connection architectures. In *International Conference on Acoustic, Speech and Signal Processing*, pages 2501–2504, Glasgow, 1989.

[TB73] B. Trakhtenbrot and Ya. Barzdin. *Finite Automata: Behavior and Synthesis*. North Holland Pub. Comp., Amsterdam, 1973.

[Val84] L.G. Valiant. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11):1134–1142, 1984.

[VCG93] E. Vidal, P. Casacuberta, and P. García. Grammatical inference and applications to automatic speech recognition. Technical Report DSIC II/41/93, Universidad Politécnica de Valencia, 1993.

[Wha74] R.M. Wharton. Approximate language identification. *Information and Control*, 26:236–255, 1974.

[Wha77] R.M. Wharton. Grammar enumeration and inference. *Information and Control*, 33:253–272, 1977.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399