N° d'ordre: 2324

THÈSE

Présentée devant

devant l'Université de Rennes 1

pour obtenir

le grade de : Docteur de l'Université de Rennes 1
Mention Informatique

par

François Coste

Équipe d'accueil : Aïda
École Doctorale : Sciences Pour l'Ingénieur
Composante universitaire : IFSIC/IRISA

Titre de la thèse :

Apprentissage d'automates classifieurs en inférence grammaticale

soutenue le 27 Janvier 2000 devant la commission d'examen

M. :	Daniel	Herman	Président
MM. :	Christian	Choffrut	Rapporteurs
	Colin	De la Higuera	
MM. :	Philippe	Besnard	Examinateurs
	Laurent	Miclet	
	Jacques	Nicolas	

ba, ba, ba,...

Ma fille Maëlle, *Premiers mots*

Remerciements

En premier lieu, je souhaite remercier Jacques Nicolas qui m'a permis de réaliser cette thèse dans les meilleures conditions. Je le remercie pour le choix de ce sujet passionnant, mais aussi pour son encadrement, son aide, sa disponibilité permanente (malgré un emploi du temps chargé), ses gâteaux aux chocolat et la confiance qu'il a toujours su me témoigner.

Je remercie également les membres de mon jury, Daniel Herman, qui m'a fait l'honneur de présider le jury, Christian Choffrut et Colin De la Higuera, qui ont bien voulu accepter la charge de rapporteur et se sont acquités de leur tâche dans le court délai que je leur ai laissé, Philippe Besnard et Laurent Miclet, pour leur intérêt accordé à mon travail.

De nombreuses personnes ont contribué au travail présenté. Je tiens à remercier plus particulièrement Colin De la Higuera, Laurent Miclet et Pierre Dupont pour les longues conversations que nous avons pu avoir. Ce mémoire doit beaucoup à leurs idées, conseils et suggestions. Je les remercie également de leur contribution à mon voyage à Albuquerque pour la réalisation de Gowachin. Cela m'a donné la chance de collaborer avec Barak Pearlmutter, que je remercie pour son accueil (ainsi que Stephanie Forrest et la famille Stewart), et Kevin Lang, grand "*hacker*" de l'inférence grammaticale, à la rapidité de réaction très stimulante.

Je remercie aussi Bernard Canet d'avoir toujours été disponible pour m'aider dans ma quête d'une programmation efficace mais propre en C++, je lui dois quelques solutions élégantes exploitant toute la puissance du langage. Je suis également redevable en programmation aux longues discussions et e-mails avec Jacques et Franck, mes collègues du "contrat CNET", avec une pensée émue à l'idée qu'ils sont en train de rédiger.

Enfin, je n'oublie pas l'apport essentiel des échanges réguliers avec Yannick, Daniel, Emmanuel et Catherine (qui n'a pas des goûts cinématographiques aussi douteux que certains le prétendent). Plus généralement, merci à toute l'équipe Aïda pour son accueil, son ambiance de travail chaleureuse, ses séminaires Tahiti et ses pauses café...

Merci à Anne...

Introduction

L'apprentissage automatique occupe aujourd'hui une place majeure au sein de l'Intelligence Artificielle. Un grand intérêt est notamment accordé à l'apprentissage par induction qui consiste à inférer des concepts à partir d'exemples. L'enjeu typique de ce type d'apprentissage est l'acquisition et la construction d'un modèle concis permettant d'expliquer les exemples passés et, dans une certaine mesure, les exemples à venir. L'étude présentée se situe plus particulièrement dans l'apprentissage par induction pour la *reconnaissance de formes*. L'objectif est de concevoir un système apprenant, de manière supervisée, à classer les exemples dans l'une des catégories identifiées. Formellement, les exemples considérés sont sous la forme de couples $(x,\gamma(x))$, tel que x est la description d'un objet à classer et $\gamma(x)$ est la classe de l'objet. L'objet de l'apprentissage inductif est alors l'identification, ou au moins l'approximation, de la fonction de classification inconnue γ, à partir d'un ensemble d'exemples $\mathcal{S} = \{(x_1,\gamma(x_1)),\ldots,x_n,\gamma(x_n)\}$, appelé échantillon d'apprentissage.

Pour un processus d'apprentissage inductif, le choix des représentations utilisées est crucial. Pour la description des exemples, la représentation choisie doit être pertinente et permettre la discrimination des objets entre les différentes classes, i.e. deux objets de classes différentes ne doivent pas être décrits de façon identique. La plupart des méthodes d'apprentissage en reconnaissance des formes travaillent sur une description des objets sous la forme d'un vecteur de valeurs d'attributs. Dans cette étude, nous considérons l'apprentissage inductif de classifications sur des suites de symboles, ce qui nous permet d'envisager l'apprentissage de concepts intégrant la notion de *séquentialité* et d'utiliser les résultats de la théorie des langages. Parmi le grand nombre d'applications concernées par cet apprentissage, on peut par exemple citer des tâches aussi différentes que : reconnaître, étiqueter ou traduire une suite de mots d'un langage, analyser une séquence d'ADN, exécuter ou planifier un enchaînement d'actions, interpréter une séquence d'alarmes, concevoir un circuit logique séquentiel, surveiller un électroencéphalogramme, reconnaître une écriture manuscrite ou détecter une intrusion dans un système informatique. Pour la description des concepts, le choix de la représentation est également très important puisqu'il définit l'ensemble des concepts exprimables et, par conséquent, apprenables par le processus d'apprentissage. Cet ensemble, appelé espace d'hypothèses, doit être soigneusement dimensionné pour permettre d'exprimer l'ensemble des concepts "utiles" tout en restant explorable en un temps "raisonnable".

Pour représenter les fonctions de classification, nous introduisons une extension des automates, que nous nommons *automates classifieurs*. La définition des automates classifieurs permet d'étendre proprement le champ d'application des automates à la classification. Considérer la classification de séquences, plutôt que leur acceptation, permet notamment d'introduire formellement la notion d'univocité : un automate classifieur sera dit *univoque* si il associe à chaque séquence au plus une classe. Cette propriété est requise dans toutes les applications où l'on souhaite répartir les séquences dans des classes distinctes. La notion d'univocité est ainsi sous-jacente dans le problème, classique en inférence grammaticale régulière, de l'apprentissage d'un automate compatible avec un échantillon positif et négatif. Les travaux effectués pour ce problème peuvent alors être adaptés et étendus à l'inférence d'automates classifieurs univoques. Nous reprenons en particulier l'approche par fusions d'états, qui consiste à construire un automate reconnaissant exactement l'échantillon d'apprentissage, puis à le généraliser en essayant de fusionner ses états sous la contrainte qu'aucun exemple négatif ne soit accepté. Cette attitude optimiste consistant à ne considérer que les fusions possibles peut être considérée suffisante pour des approches gloutonnes. Cependant, pour une exploration plus exhaustive de l'espace des hypothèses, nous proposons de considérer aussi les fusions impossibles, la représentation choisie nous permettant de détecter efficacement l'ensemble des états incompatibles, c'est à dire l'ensemble des paires d'états fusionnés permettant de classer une séquence dans plus d'une classe.

Le plan de ce mémoire de thèse est linéaire et découpé en trois parties. Dans la première partie, nous situons le problème de l'inférence d'automates classifieurs au sein de l'inférence grammaticale et de l'identification de machines à états finies. La deuxième partie est consacrée à la définition et l'étude des propriétés de l'espace de recherche. Cette étude permet de définir un espace de recherche implicite et d'exprimer le problème de l'inférence d'automates classifieurs déterministes univoques comme un problème de satisfaction de contraintes. Ces résultats sont utilisés dans la troisième partie pour identifier deux nouveaux schémas génériques d'algorithmes efficaces pour la recherche de solutions minimales, applicables également à l'inférence d'automates compatibles. Différentes variantes de ces schémas ont été implémentées, formant les premiers blocs d'une plate-forme d'inférence de machines à états finies. Une première comparaison expérimentale de différentes stratégies pour le problème, peu traité parce que difficile, de la recherche exacte de solutions minimales conclut cette étude.

Chapitre 1

Inférence grammaticale : vers l'inférence C-régulière

Au sein de l'intelligence artificielle, l'inférence grammaticale est un problème d'inférence inductive de séquences [AS83] consistant à apprendre une grammaire à partir de sous ensembles de mots d'un langage cible (inconnu).

L'inférence grammaticale a été étudiée depuis le développement de la théorie des grammaires formelles dans les années 60 pour la construction d'un modèle formel de l'acquisition du langage naturel. Depuis l'article fondateur de Gold [Gol67] introduisant la notion *d'identification à la limite*, de nombreux travaux ont été effectué au sein de diverses communautés (apprentissage, reconnaissance de formes syntaxiques et structurelles, traitement de la langue, réseaux neuronaux, théorie des langages formels, théorie de l'information, conception de circuits électroniques) pour établir le cadre théorique de l'inférence grammaticale et imaginer des algorithmes d'inférence efficaces. Plus récemment, des applications réelles de ces méthodes à des problèmes pratiques ont vu le jour, notamment pour le traitement de la langue naturelle [CVO93, RS97] et la recherche génomique [SBH+94, BJVU98].

Il existe déjà plusieurs excellents articles de présentation de l'inférence grammaticale auxquels se référer : notamment récemment [Sak97],[Alq97] et [DM98]. Dans ce chapitre, l'objectif est de spécifier et situer le problème de l'inférence d'automates classifieurs au sein de l'inférence grammaticale. Après une présentation des principaux résultats d'apprenabilité de l'inférence grammaticale en section 1.2, et des approches les plus fructueuses pour l'inférence d'automates en section 1.3, l'extension du point de vue de l'inférence régulière à l'inférence d'automates classifieurs est introduite et formulée en termes d'inférence de machines à états finies à partir d'exemples d'entrées/sorties.

Mais, en premier lieu, pour introduire les notations et concepts utilisés, la première section est consacrée à quelques définitions et résultats classiques de la théorie des langages.

1.1 Éléments de la théorie des langages

1.1.1 Langages

Un *alphabet* Σ est un ensemble fini non vide de *symboles* distincts. Un *mot* est une suite finie de symboles. L'ensemble de tous les mots sur l'alphabet Σ, y compris le mot vide noté ϵ, est le monoïde libre noté Σ^*. La concaténation d'un mot u avec un mot v sera notée $u \cdot v$ ou uv. L'élément neutre de la concaténation est le mot vide. Pour tout mot $w \in \Sigma^*$, on a :

$$w \cdot \epsilon = \epsilon \cdot w = w.$$

Un *langage* L est une partie de Σ^*. L'ensemble de tous les langages est $\mathcal{P}(\Sigma^*)$. Puisque Σ^* est un monoïde, l'ensemble des parties $\mathcal{P}(\Sigma^*)$ devient naturellement un demi anneau : la somme de deux langages L et L' est simplement leur union $L \cup L'$ et leur produit LL' est défini par :

$$LL' = \{w \in \Sigma^* / \exists u \in L, \exists v \in L', w = uv\}$$

L'élément neutre de l'union d'ensemble est l'ensemble vide noté $\emptyset$ et est l'élément absorbant pour le produit. On a, pour tout langage L :

$$L \cup \emptyset = \emptyset \cup L = L$$

$$L\emptyset = \emptyset L = \emptyset.$$

Le quotient gauche (ou préfixe) d'un langage L par un langage D et son quotient droit (ou suffixe) par un langage G sont définis par :

$$LD^{-1} = \{u \in \Sigma^* / \exists w \in L, \exists v \in D, w = uv\}$$

$$G^{-1}A = \{v \in \Sigma^* / \exists w \in L, \exists u \in G, w = uv\}.$$

On peut alors définir l'ensemble des préfixes $Pr(L)$ et des suffixes $Suf(L)$ du langage L :

$$Pr(L) = L(\Sigma^*)^{-1}$$

$$Suf(L) = (\Sigma^*)^{-1}L.$$

Si il n'y a pas d'ambiguité, les langages constitués d'un seul mot seront désignés par ce mot, de même les mots constitués d'un seul symbole seront désignés par le symbole et les délimiteurs seront omis. La notation u pourra alors distinguer suivant le contexte aussi bien une lettre u, un mot u ou le singleton $\{u\}$.

1.1.2 Grammaires

La représentation d'un langage sous la forme de l'énumération du sous ensemble de mots de Σ^* qui le compose n'est pas forcément pratique surtout si le langage n'est pas fini. On peut faire appel aux notations ensemblistes comme dans la section précédente. Cependant, les langages sont plus généralement représentés à l'aide des grammaires formelles introduites par Chomsky. Une grammaire formelle est un ensemble de règles de production permettant d'engendrer tous les mots du langage par un système de réécriture.

Définition 1 *Une grammaire est définie par un quadruplet* (Σ,V,S,P) *où :*
- Σ *est un alphabet dit* terminal
- V *est un autre alphabet dit* non-terminal
- S *est une lettre distinguée de* V *appelé* axiome *ou symbole d'entrée*
- P *est un sous-ensemble de fini de* $(\Sigma \cup V)^+ \times (\Sigma \cup V)^*$ *dont un élément* (α,β) *est appelé* règle de production *et noté* $\alpha \to \beta$.

□

La relation de réécriture (ou de dérivation) $\to$ suivant les règles de production de P est définie par :

$$\forall u,v \in (\Sigma \cup V)^*, u \to v \text{ } ssi \text{ } \exists(\alpha,\beta) \in P, \exists u_1,u_2 \in (\Sigma \cup V)^* u = u_1 \alpha u_2 \text{ } et \text{ } v = u_1 \beta u_2$$

Le langage engendré par une grammaire est alors l'ensemble des mots de Σ^* obtenus par la fermeture réflexive et transitive de $\to$ notée $\to^*$:

Définition 2 *Le langage* $L(G)$ *engendré par une grammaire* $G = (\Sigma,V,S,P)$ *est l'ensemble* $\{w \in \Sigma^* / S \to^* w\}$. □

La représentation sous la forme de grammaires permet d'effectuer la classification de Chomsky des langages en fonction de la forme des règles de production de la grammaire les engendrant.

Définition 3 *Soit une grammaire* $G = (\Sigma,V,S,P)$.
- G *est de* type 0 *ou* à structure de phrase *si* P *est inclus dans* $(\Sigma \cup V)^+ \times (\Sigma \cup V)^*$.
- G *est de* type 1 *ou* contextuelle *si* P *est inclus dans* $(\Sigma \cup V)^* V (\Sigma \cup V)^* \times (\Sigma \cup V)^*$.
- G *est de* type 2 *ou* algébrique *si* P *est inclus dans* $V \times (\Sigma \cup V)^*$.
- G *est de* type 3 *ou* régulière *ou* linéaire à droite *(resp.* linéaire à gauche*) si* P *est inclus dans* $V \times (\Sigma^* V) \cup \Sigma^*$ *(resp.* $V \times (V \Sigma^*) \cup \Sigma^*$*).*

□

Cette classification introduit une hiérarchie stricte entre les classes de langages. Nous étudierons plus particulièrement l'inférence des langages engendrés par des grammaires de type 3. Nous appelerons ces langages, les *langages réguliers*. Ces langages sont également appelés *rationnels* car ils forment la plus petite famille de langages sur Σ^* clos par les fonctions rationnelles que sont l'union, le produit de concaténation et l'opération d'itération "étoile" de Kleene. Ces langages sont également appelés *reconnaissables* car ils peuvent être représentés par un automate fini qui reconnaît (accepte) l'ensemble des mots du langage.

1.1.3 Automates

Les automates à états finis constituent la représentation des langages réguliers la plus couramment utilisée en inférence grammaticale.

Définition 4 *Un automate (à états fini) A est un quintuplet* (Σ,Q,q_0,F,E) *où :*

- Σ *est un alphabet,*
- Q *est un ensemble fini d'états,*
- q_0 *est un état initial,*
- $F \subseteq Q$ *est un ensemble d'états finals,*
- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ *est l'ensemble des transitions.*

□

Les automates peuvent être vus comme une représentation sous forme de graphe des grammaires régulières, les états correspondant aux symboles non-terminaux et les transitions aux dérivations. Le même type de représentation visuelle que pour les graphes peut être utilisé. Nous utiliserons la convention d'utiliser un double cercle pour les états finals, une flèche rentrante pour l'état initial.

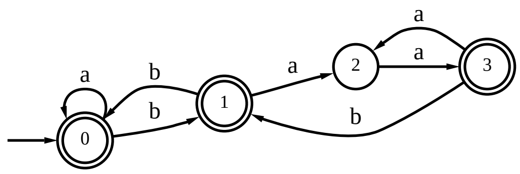


FIG. 1.1 – *Automate*

Un automate fini permet de représenter un ensemble de mots sur l'alphabet Σ. Les mots acceptés par l'automate sont ceux pour lesquels il existe un chemin de l'état initial à un état final suivant les symboles du mot. Pour une définition plus formelle, on introduit la fermeture transitive de l'ensemble des transitions $\hat{E} \subseteq Q \times \Sigma^* \times Q$ qui est le plus petit ensemble tel que :

- Pour tout q de Q, $(q,\epsilon,q) \in \hat{E}$
- Pour tout w de Σ^*, si $(q_1,w,q_2) \in \hat{E}$ et $(q_2,a,q_3) \in E$ alors $(q_1,wa,q_3) \in \hat{E}$

L'ensemble des mots reconnus (ou acceptés) par un automate $A = (\Sigma,Q,q_0,F,E)$ est alors le langage $L(A)$ défini par :

$$L(A) = \{w \in \Sigma^* / \exists q_f \in F, (q_0,w,q_f) \in \hat{E}\}$$

Si, pour tout état q de Q et tout symbole a de Σ, il existe au plus un état q' de Q (respectivement exactement un état q' de Q) tel que $(q,a,q') \in E$, l'automate est dit *déterministe* (respectivement *complet*). Nous utiliserons par la suite les abréviations classiques DFA pour "automate fini déterministe" et NFA pour "automate fini non-déterministe". L'existence d'un algorithme de déterminisation d'un NFA permet d'assurer que les NFA et DFA permettent de représenter la même classe des langages rationnels.

Plusieurs automates pouvant représenter le même langage, on appelle *automate canonique* d'un langage L, noté $A(L)$, le plus petit automate (en nombre d'états) déterministe représentant L. On démontre que cet automate est unique à une permutation de ses états près et qu'il peut être obtenu à partir de tout automate représentant L par les algorithmes de déterminisation et de minimisation des automates [HU80].

1.1.4 Transducteurs

La fonction réalisée par un automate est l'acceptation de mots. Si l'on ajoute au symbole d'entrée de chaque transition un symbole de sortie, la machine à états finie correspondante est appelée *transducteur rationnel* et permet de réaliser une relation de transductions $t : \Sigma_i^* \to \Sigma_o^*$, des mots acceptés sur l'alphabet d'entrée Σ_i vers des mots sur l'alphabet de sortie Σ_o. Nous introduisons dans cette section les différents types de transducteurs de la littérature [Ber79, RS97] afin d'illustrer les principaux concepts associés aux machines à états finies suivant le point de vue fonctionnel présenté par Booth [Boo67].

Définition 5 *Un transducteur rationnel (à états fini) T est défini par un sextuplet $(\Sigma_i,\Sigma_o,Q,q_0,F,E)$ où :*
- *Σ_i et Σ_o sont respectivement les alphabets d'entrée et de sortie*
- *Q est un* ensemble fini d'états,
- *q_0 est* l'état initial,
- *$F \subseteq$ est l'ensemble* des états finals,
- *$E \subseteq Q \times \Sigma_i^* \times \Sigma_o^* \times Q$ est* l'ensemble des transitions.

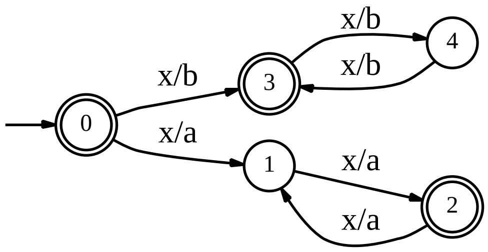
□

Il existe plusieurs autres définitions équivalentes. Une définition alternative consiste à remplacer l'ensemble des transitions par une *fonction de transition* δ de $Q \times \Sigma_i^*$ vers $\mathcal{P}(Q)$ et une *fonction d'émission* λ de $Q\times\Sigma_i^*$ vers $\mathcal{P}(\Sigma_o^*)$ définies sur le même domaine[1]. Les équations suivantes expriment les relations entre ces deux fonctions et l'ensemble des transitions :

$$\delta(q,a) = \{q' \in Q / \exists (q,a,b,q') \in E\}$$

$$\lambda(q,a) = \{b \in \Sigma_o^* / \exists (q,a,b,q') \in E\}$$

1. Pour une fonction partielle f de A vers B, le domaine de f, noté $dom(f)$, est l'ensemble des éléments a de A pour lesquels la fonction $f(a)$ est définie. Si l'on considère la fonction partielle comme une fonction totale de A dans $\mathcal{P}(B)$ ou $B\cup\emptyset$, le domaine de f est défini par $dom(f) = \{a \in A / f(a) \neq \emptyset\}$.



$$\tau : x^* \rightarrow \{a,b\}^*$$

$$\tau(x^n) = \begin{cases} a^n \text{ si } n \text{ pair} \\ b^n \text{ si } n \text{ impair} \end{cases}$$

FIG. 1.2 – *Transducteur rationnel et transduction rationnelle correspondante*

Pour définir formellement la relation de transduction effectuée, on introduit la fermeture transitive des transitions $\hat{E}$, définie comme étant le plus petit sous-ensemble de $Q \times \Sigma_i^* \times \Sigma_o^* \times Q$ tel que :

- Pour tout q de Q, $(q,\epsilon,\epsilon,q) \in \hat{E}$
- Pour tout w_i de Σ_i^*, tout w_o de Σ_o^*, si $(q_1,w_i,w_o,q_2) \in \hat{E}$ et $(q_2,a,b,q_3) \in E$ alors $(q_1,w_ia,w_ob,q_3) \in \hat{E}$

La relation de transduction $L(T)$ sur $\Sigma_i^* \times \Sigma_o^*$ associée à un transducteur rationnel $T = (\Sigma_i,\Sigma_o,Q,q_0,F,E)$ et la transduction τ_{T} de Σ_i^* dans $\mathcal{P}(\Sigma_o^*)$ effectuée par T peuvent alors être définies par :

$$L(T) = \{(w_i,w_o) \in \Sigma_i^* \times \Sigma_o^* / \exists q_f \in F,(q_0,w_i,w_o,q_f) \in \hat{E}\}$$

$$\tau_{\mathrm{T}}(w_i) = \{w_o \in \Sigma_o^* / (w_i,w_o) \in L(T)\}$$

L'ensemble des transductions pouvant être réalisées par des transducteurs rationnels forme la classe la plus générale de transductions pouvant être effectuées par une machine à états finie. Ces transductions sont appelées *rationnelles.*

Définition 6 *Une transduction* $\tau : \Sigma_i^* \rightarrow \mathcal{P}(\Sigma_o^*)$ *est une* transduction rationnelle *si il existe un transducteur rationnel* T *tel que* $\tau = \tau_T$*. Si pour chaque mot* w *en entrée de* Σ_i^*, $\tau(w)$ *est soit un singleton, soit l'ensemble vide,* τ *définit une* fonction rationnelle. □

En général, on préfère manipuler des machines déterministes pour l'efficacité calculatoire qu'elles procurent. Plusieurs définitions de transducteurs "déterministes" coexistent dans la littérature. Nous en considérons ici deux : la définition des *transducteurs séquentiels*, qui permet de définir les machines séquentielles de Mealy et de Moore, et la définition des *transducteurs sous-séquentiels*, au pouvoir d'expression plus élevé.

Définition 7 *Un transducteur déterministe par rapport à l'entrée (i.e.* $((q,a,b_1,q_1) \in E \wedge (q,a,b_2,q_2) \in E) \Rightarrow (b_1 = b_2 \wedge q_1 = q_2)$*) dont tous les états sont finals est appelé* transducteur séquentiel *et la transduction définie est alors nommée* fonction séquentielle. □

Les transducteurs séquentiels sont également appelés *machines de Mealy*. Lorsque la production dépend uniquement de l'état destination au lieu de la transition, la fonction de production sur les transitions λ définie de $Q \times \Sigma_i$ vers Σ_o peut économiquement être remplacée par une fonction de production sur les états ρ définie seulement de Q vers Σ_o. La machine est alors appelée *machine séquentielle de Moore*. Notons que la différence entre ces deux types de machines est essentiellement au niveau de la représentation interne, l'ensemble des transductions réalisables étant pour chacune l'ensemble des fonctions séquentielles.

Les fonctions séquentielles forment un sous ensemble des fonctions rationnelles caractérisé par la propriété de conservation des préfixes, c'est à dire que pour une fonction séquentielle τ : $\tau(\epsilon) = \epsilon$ et si $\tau(uv)$ existe, alors $\tau(uv) \in \tau(u)\Sigma_o^*$. Pour outrepasser cette restriction et augmenter la classe des transductions possibles tout en gardant les propriétés de parcours déterministes, une variante des transducteurs rationnels appelée transducteur sous-séquentiel a été introduite [Sch77]. Un transducteur sous-séquentiel est un transducteur rationnel déterministe par rapport à l'entrée auquel a été ajouté une fonction ρ d'émission finale de F dans Σ_o^*. Dans la définition suivante, l'ensemble d'états finals F est omis, remplacé par une fonction *d'émission finale* ρ définie de Q dans $\Sigma_o^* \cup \emptyset$ telle que ρ vaut l'élément absorbant $\emptyset$ pour tout état non final.

Définition 8 *Un transducteur sous-séquentiel SST est défini par un septuplet $(\Sigma_i,\Sigma_o,Q,q_0,\delta,\lambda,\rho)$ où :*
- *Σ_i,Σ_o,Q,q_0 sont définis comme pour les transducteurs ;*
- *δ est une fonction de transition déterministe de $Q \times \Sigma_i$ vers $Q \cup \emptyset$;*
- *λ est une fonction d'émission déterministe de $Q \times \Sigma_i$ vers $\Sigma_o^* \cup \emptyset$;*
- *ρ est une fonction d'émission finale déterministe de Q dans $\Sigma_o^* \cup \emptyset$;*

le domaine de définition de δ et λ devant être identique. □

Pour simplifier les expressions, le domaine des fonctions peut être étendu pour ajouter que, par convention, une fonction n'est pas définie si l'un de ses paramètre n'est pas défini. Nous utiliserons notamment le fait que $\delta(\emptyset,a) = \lambda(\emptyset,a) = \rho(\emptyset) = \emptyset$, quelque soit a de Σ_i.

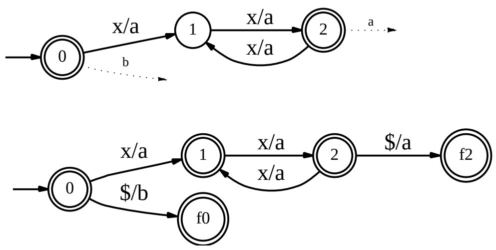
L'extension des fonctions δ et λ déterministes aux mots peut être effectuée classiquement pour tout w de Σ_i^*, tout q de Q et tout a de Σ_i par :

$$\delta(q,aw) = \delta(\delta(q,a),w)$$

$$\lambda(q,aw) = \lambda(q,a).\lambda(\delta(q,w),a))$$

La transduction effectuée par un transducteur sous-séquentiel SST peut alors être définie par :

$$\tau_{\mathrm{SST}}(w) = \lambda(q_0,w)\rho(\delta(q_0,w))$$



$$\tau_{\text{SST}} : x^* \rightarrow \{a,b\}^*$$

$$\tau_{\text{SST}}(x^n) = \begin{cases} b \text{ si } n = 0 \\ a^{n+1} \text{ si } n \text{ pair, } n \neq 0 \\ \emptyset \text{ si } n \text{ impair} \end{cases}$$

FIG. 1.3 – *SST et machine de Mealy avec symbole de fin (ici $) correspondante.*

On peut ainsi voir les transducteurs sous-séquentiels comme la fusion d'une machine de Mealy et d'un machine de Moore, la production de la machine de Moore n'étant considérée que pour le dernier état atteint. Un transducteur sous-séquentiel peut être simulé par une machine de Mealy seule avec l'ajout d'un marqueur de fin de séquence (voir figure 1.3).

Notons enfin qu'une extension des transducteurs sous-séquentiels appelé transducteurs p sous-séquentiels a été introduite par Mohri qui consiste à considérer une fonction d'émission finale ρ indéterministe, mais avec un nombre d'émission pour chaque état borné par un entier p ($\rho : Q \rightarrow \mathcal{P}(\Sigma_o^*)$ et $\forall q \in Q, |\rho(q)| \leq p$). Le parcours est alors toujours déterministe alors que la sortie ne l'est pas.

La famille des transductions sous-séquentielles (i.e. pouvant être effectuées par les transducteurs sous-séquentiels) inclut la famille des fonctions séquentielles (il suffit de poser $\forall q \in Q,\ \rho(q) = \epsilon$) mais l'ajout de la fonction d'émission finale permet de ne pas forcément conserver les préfixes par transduction. La classe des transductions sous-séquentielles reste cependant strictement incluse dans la classe des transductions rationnelles à cause du déterminisme. Ainsi, la transduction rationnelle τ associée au transducteur rationnel de la figure 1.2 et définie par :

$$\forall w \in \{x\}^*, \tau(w) = \begin{cases} a^{|w|} \text{ si } |w| \text{ est pair} \\ b^{|w|} \text{ sinon} \end{cases}$$

ne peut pas être réalisée par un transducteur séquentiel ou sous-séquentiel. En effet, pour écrire la production, il faudrait d'abord lire toute la séquence d'entrée qui peut être arbitrairement longue avant de produire la sortie. Par soucis de complétude, on peut cependant mentionner qu' une transduction rationnelle peut être, en revanche, décrite par la composition d'une fonction séquentielle droite et d'une fonction séquentielle gauche (appelée bimachine [Sch61]). Les fonctions séquentielles gauches sont les fonctions séquentielles déjà présentées, les fonctions séquentielles droites sont des fonctions séquentielles lisant les mots de droite à gauche.

1.2 Modèles d'apprentissage

Pour obtenir des résultats d'apprenabilité sur des classes de langage à partir d'exemples, il convient de définir un critère de succès de l'inférence, le type des exemples ainsi que leur disponibilité. En inférence grammaticale, trois modèles d'apprentissage sont principalement utilisés : l'*identification à la limite* de Gold [Gol67], *l'apprentissage avec Oracle* de Angluin [Ang88] et *l'apprentissage PAC* de Valiant [Val84]. Leur présentation, jumelée avec les résultats qu'il permettent d'établir, fait l'objet des trois prochaines sections.

1.2.1 Identification à la limite

L'identification à la limite proposée par Gold [Gol67] considère l'apprentissage comme un processus infini. Dans ce modèle, à chaque pas du processus d'inférence, la méthode d'apprentissage reçoit un exemple et doit proposer une solution dans l'espace d'hypothèses. La méthode d'apprentissage *identifie à la limite* un concept $\mathcal{C}$, si il existe un pas du processus d'inférence à partir duquel chaque solution proposée par la méthode d'apprentissage est équivalente au concept $\mathcal{C}$.

Ce modèle d'apprentissage permet d'obtenir des résultats sur la possibilité d'identifier ou non un langage par induction, notamment en fonction du type de présentation des exemples.

Définition 9 *On appelle* présentation positive *(resp.* négative*) d'un langage L une séquence infinie d'exemples, comportant au moins une occurrence de chaque élément de L (resp. $\Sigma^* - L$).* □

Définition 10 *On appelle* présentation complète *d'un langage L une séquence infinie ordonnée de paires (x,d) de $\Sigma^* \times \{0,1\}$ telle que tous les éléments x de Σ^* apparaissent comme premier argument d'une paire de la séquence et pour chaque paire (x,d), $d = 1$ si et seulement si $x \in L$ ($d = 0$ sinon).* □

On a alors les résultats théoriques suivants :

Théorème 1 *Aucune classe de langage contenant tous les langages de cardinalité finie et au moins un langage de cardinalité infinie n'est identifiable à la limite à partir de présentations positives [Gol67].*

Ce théorème implique que même la classe des langages rationnels n'est pas identifiable à la limite à partir d'exemples positifs. Intuitivement, lorsque l'un des langages de l'espace d'hypothèse est de cardinalité infinie, la seule donnée d'exemples positifs ne permet pas d'éviter une surgénéralisation. Par contre, si l'on dispose de contre exemples, en particulier une présentation complète, la classe de langages identifiable à la limite est beaucoup plus vaste :

Théorème 2 *Toute classe de langages récursivement énumérable est identifiable à la limite à partir de présentations complètes [Gol67].*

Ce résultat théorique 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 est possible.

Cependant la taille de l'échantillon requis et/ou la complexité calculatoire de l'algorithme d'apprentissage peuvent être rédhibitoires pour l'identification de langages algébriques et même rationnels. En pratique, les algorithmes NP-Complet étant inutilisables sauf pour des problèmes de très petite taille, on peut considérer que l'apprenabilité d'une classe de concepts est liée à l'existence de méthodes d'apprentissage de complexité polynomiale. Pitt [Pit89] a discuté et défini la complexité polynomiale pour l'identification à la limite. Il propose la définition suivante : un algorithme d'inférence est polynomial si le temps de mise à jour de son hypothèse pour une nouvelle donnée est polynomial en fonction de la taille des exemples déjà traités et si l'algorithme fait au plus un nombre polynomial d'erreurs implicites (erreur de prédiction d'un nouvel exemple). Mais suivant cette définition, même les automates déterministes ne sont pas identifiables polynomialement alors que l'intuition et des résultats expérimentaux suggèrent le contraire.

Le paradigme du nombre d'exemples infini étant peu adapté aux situations pratiques où le nombre d'exemples disponibles et traitables est rarement infini, Gold [Gol78] propose *un modèle d'identification à partir de données fixées*, où la méthode d'apprentissage doit retourner un concept compatible avec une paire d'ensembles finis d'exemples donnée (S_+,S_-), avec S_+ ensemble d'exemples positifs et S_- ensemble d'exemples négatifs. En posant que pour tout concept, il existe un *ensemble d'exemples caractéristique* tel que l'inclusion de cet ensemble dans l'ensemble d'exemples implique que la méthode d'apprentissage retourne la solution correcte, une classe est dite *polynomialement identifiable à la limite à partir de données fixées* si, comme dans [Pit89], le temps de mise à jour de l'hypothèse pour une nouvelle donnée est polynomial en fonction de la taille des exemples déjà traités mais sous la condition supplémentaire que l'algorithme infère le concept cible dès qu'un ensemble caractéristique, devant être de taille polynomiale par rapport au concept cible, est présent dans l'ensemble d'exemples. Dans le même article, Gold a montré que les automates déterministes sont polynomialement identifiables à la limite à partir de données fixées. C'est également le cas des grammaires linéaires équilibrées [Tak88, SG94] ainsi que des grammaires linéaires universelles [Tak94]. De plus, les grammaires équilibrées déterministes et les fonctions subséquentielles totales ont été démontrées polynomialement identifiables à la limite à partir de données fixées seulement positives dans, respectivement, [KMT95] et [OGV93].

Une généralisation du modèle de Gold [Gol78] et une restriction naturelle de la définition d'identification avec remise à jour polynomiale de Pitt [Pit89]. Proposée par De la Higuera,elle permet de prendre en compte la complexité spatiale et temporelle de l'apprentissage relative à la taille du concept à inférer :

Définition 11 *[Hig96] Une classe de représentation $\mathcal{R}$ est polynomialement identifiable à partir de données fixées si et seulement si il existe deux polynômes p et q et un algorithme A tel que :*

1. *Étant donné un échantillon (S_+,S_-) de taille m, A renvoie une représentation R dans $\mathcal{R}$ compatible avec (S_+,S_-) en temps inférieur à $p(m)$.*
2. *Pour toute représentation R de taille n, il existe un échantillon caractéristique (CS_+,CS_-) de taille au plus $q(n)$, pour lequel, sur les données (S_+,S_-) avec $S_+ \supseteq CS_+$ et $S_- \supseteq CS_-$, A retourne une représentation R' équivalente à R.*

□

Ce cadre n'est pas trivial, il existe plusieurs classes qui ne sont pas polynomialement identifiables à partir de données fixées : les langages définis par les grammaires algébriques, les grammaires linéaires, les grammaires déterministes simples et les automates non déterministes [Hig96]. Par contre, pour les classes apprenables dans ce modèle, ce cadre permet de rendre compte des résultats positifs observés expérimentalement avec des algorithmes heuristiques mais cependant déterministes. Une suggestion de l'auteur est que ce modèle fournit également un moyen de comparer différents algorithmes d'inférence en fonction de la taille de l'échantillon caractéristique nécessaire à chacun.

1.2.2 Apprentissage avec Oracle

Angluin [Ang88] a considéré la situation d'apprentissage où l'on dispose d'un Oracle capable de répondre à certains types de requêtes sur la grammaire inconnue G. Deux types de requêtes pouvant être soumises à l'Oracle par la méthode d'apprentissage ont particulièrement été mises en évidence ("Minimally Adequate Teacher") :

- *Requête d'appartenance d'une chaîne $x \in \Sigma^*$* : en réponse, l'Oracle indique si x est engendré par G ou non.
- *Requête d'équivalence d'une grammaire G'* : en réponse, l'Oracle indique si la grammaire G' est équivalente à G (i.e. elle génère le même langage que G). Dans le cas où elle n'est pas équivalente, l'Oracle fournit également un contre-exemple x appartenant à la différence symétrique entre le langage $L(G)$ généré par G et le langage $L(G')$ généré par G'.

En théorie, la requête d'appartenance correspond à la disponibilité d'une présentation complète et permet d'identifier à la limite les mêmes classes de règle. La complexité calculatoire n'est cependant pas comparable, la requête d'appartenance permettant de choisir l'ordre de présentation des exemples classifiés par l'Oracle. Ainsi, par exemple, la classe des automates déterministes peut être identifiée en temps polynomial à partir de requêtes d'équivalence et d'appartenance [Ang87]. Un résultat similaire avec uniquement des requêtes d'appartenance peut être énoncé moyennant une hypothèse supplémentaire sur l'échantillon d'apprentissage [Ang81] (voir section 2.2.1). Par contre, si l'on ne dispose que de requêtes d'équivalence, Angluin a montré qu'il n'existait aucun algorithme polynomial identifiant la classe des automates déterministes ou non déterministes, des grammaires algébriques, ou des formules booléennes sous forme normale disjonctive ou conjonctive.

1.2.3 Apprentissage PAC

Valiant [Val84] a introduit le modèle d'apprentissage *approximativement probablement correct* (apprentissage PAC) dans lequel une approximation du concept avec un taux d'erreur faible doit être apprise en temps polynomial avec une grande probabilité quelque soit la distribution des exemples. Malgré le relâchement du critère d'identification, l'apprentissage PAC était jusqu'à présent relativement peu utilisé en inférence grammaticale, la classe des automates déterministes n'étant pas apprenable dans ce modèle [KV89].

Cependant, des résultats plus intéressants peuvent être obtenus à condition de relâcher le critère portant sur la distribution des exemples, notamment en privilégiant en situation d'apprentissage les exemples simples. Li et Vitányi [LV91] proposent le modèle d'*apprentissage PAC par exemples simples* dans lequel la condition d'apprentissage pour toutes les distributions est remplacée par un apprentissage pour toutes les distributions simples. Les distributions simples forment une classe étendue qui inclut toutes les distributions énumérables. Plus précisément, ces distributions sont celles dominées multiplicativement par la distribution universelle énumérable $\mathbf{m}$, qui assigne de grandes probabilités aux exemples dont la complexité de Kolmogorov $K(x)$ est faible. Ce modèle a été étendu par Denis, D'Halluin et Gilleron [DDG96] pour prendre en compte la complexité de représentation du concept c à apprendre. Les exemples sont alors supposés être tirés suivant la distribution conditionnelle de Solomonoff-Levin $\mathbf{m}_c$. C'est à dire que les exemples les plus probables sont ceux qui ont la complexité de Kolmogorov conditionnelle $K(x/c)$ la plus faible. Une classe de concepts est dite *PACS apprenable* si elle est PAC-apprenable pour des exemples tirés suivant la loi de probabilité $\mathbf{m}_c$.

Dans ce modèle d'apprentissage, la classe des automates déterministes a été montrée PAC apprenable par exemples simples [PH97]. Une application intéressante de ce modèle est de considérer que pour une présentation positive conformément à la distribution $\mathbf{m}_c$, l'absence d'un exemple simple peut être interprétée comme étant équivalente à un exemple négatif. Denis [Den98] propose ainsi une définition d'apprentissage PAC par exemples positifs et montre que la classe des automates déterministes est apprenable à partir d'exemples positifs simples.

L'introduction de ces nouveaux modèles permet également de faire le lien avec l'apprentissage par requêtes présenté en section 1.2.2 et d'en utiliser les résultats. Castro et Guijarro ont montré [CGL97] que les classes de concepts identifiables par requêtes sont également PAC-apprenables par exemples simples et que, par conséquent, l'apprenabilité des automates déterministes est une conséquence de son apprenabilité dans le modèle d'apprentissage par requêtes.

1.3 Inférence d'automates

D'après les résultats énoncés dans la section précédente, la classe des langages réguliers est la plus haute classe, dans la hiérarchie de Chomsky, identifiable polynomialement à la limite. Pratiquement, cela signifie que cette classe de langages est suffisamment expressive pour décrire des phénomènes non triviaux de la vie réelle tout en restant apprenable à partir d'exemples. D'autre part, des résultats positifs obtenus pour l'inférence de (sous-classes de) grammaires algébriques l'ont été par la réduction du problème à un problème d'inférence régulière [Tak88, Tak95] ou par utilisation de méthodes d'inférence régulière [Sak90]. L'inférence régulière (ou de façon équivalente l'inférence d'automates) est donc l'une des bases de l'inférence grammaticale et a fait l'objet de nombreuses études. Nous présentons ici les principales approches adoptées, classées suivant que la présentation des exemples est positive (section 1.3.1) ou complète (section 1.3.2). Le problème peut également être abordé sous l'angle de l'inférence de classifieurs de séquences, où l'ensemble des séquences peut être réparti en deux classes selon qu'elles appartiennent ou non au langage. Cette approche peut être généralisée à l'inférence de classifieurs de séquences parmi plusieurs langages (section 1.3.3). La représentation choisie dans cette étude est un type de machine à états finie appelé automates classifieurs. Cette représentation permet de faire le lien avec un domaine à la marge de l'inférence grammaticale qui est l'identification de machines à états finies à partir de couples d'entrées-sorties (section 1.4.1).

1.3.1 Apprentissage à partir d'exemples positifs

L'apprentissage de la classe des langages rationnels à partir d'exemples positifs seuls étant impossible d'après le résultat négatif de Gold [Gol67], une propriété supplémentaire doit être ajoutée à l'énoncé du problème pour éviter la surgénéralisation.

La propriété supplémentaire considérée dans la section suivante est la compatibilité avec des exemples négatifs. On peut se ramener à ce schéma si l'on suppose que les exemples sont donnés suivant une distribution simple et que les exemples simples non présents sont des exemples négatifs "par omission" [Den98]. Les algorithmes par fusion de la section suivante peuvent également être utilisés si l'on remplace la condition de compatibilité pour la fusion de deux états par un critère statistique de similitude entre les langages acceptés à partir de ces deux états pour autoriser ou non une fusion [CO94]. Mais dans ce dernier cas, on ne peut plus caractériser l'ensemble de langages identifiable par l'algorithme d'inférence. Un autre exemple de méthode d'inférence heuristique est ECGI [RV88, Rul92] qui permet d'inférer des automates déterministes sans boucles de façon incrémentale, par minimisation d'une mesure de la distance entre l'automate courant et le nouvel exemple traité. On peut citer également la famille des méthodes d'inférence heuristiques, les méthodes *k-tails* [BF72], *tail-clustering* [Mic80], *successor* [RV84], *predecessor-successor* [VR84] et, plus généralement, l'étude sur les méthodes par similarités partielles [KS88]. Une bonne présentation de ce domaine a été faite par Gregor [Gre94].

Pour établir des résultats d'identifiabilité à partir d'exemples positifs, il est classique de se restreindre à des "sous classes" des langages réguliers. Si l'on a des raisons de penser que pour le domaine d'application considéré, la sous-classe de langage est suffisante, ces algorithmes doivent être utilisés. Ainsi, Angluin [Ang82] a introduit une série de sous-classes de langages appelés *k-reversibles*, avec $k = 0,1,2,\ldots$, identifiable à la limite à partir d'une présentation positive. L'algorithme d'identification profite des contraintes sur la structure des automates reconnaissant ces types de langage pour limiter la généralisation. L'existence d'ensembles d'exemples caractéristiques assure l'identification du langage cible. De même, parmi les sous-classes identifiables à la limite, on peut citer la classe des langages *k-testables* [GV90] et la classe des automates strictement déterministes [Yok95].

1.3.2 Apprentissage à partir d'exemples positifs et négatifs

1.3.2.1 Apprentissage avec oracle

Angluin a montré que la classe des automates déterministes peut être identifié en temps polynomial à partir de requêtes d'équivalence et d'appartenance[Ang87]. L^*, l'algorithme proposé pour cette identification, utilise une structure intéressante de données appelée *"observation table"* et qui est proche de celle utilisée par Gold pour la caractérisation des états. Elle a la forme d'un tableau à deux dimensions indicé par des chaînes de caractère. Le contenu d'une case de la rangée e et de la colonne s est égal à 1 si et seulement si la chaîne $s.e$ est acceptée par l'automate inconnu. L'ensemble des indices des rangées est composé de deux ensembles : un ensemble S de chaînes de caractères clos par préfixes et l'ensemble $S.\Sigma$ des chaînes obtenues en concaténant chaque lettre de l'alphabet aux chaînes de S. L'ensemble des indices des colonnes est un ensemble E clos par suffixes dont le rôle est de distinguer les états potentiels représentés par les éléments de S alors que les chaînes de $S.\Sigma$ sont utilisées pour construire la table de transitions. Initialement $S = E = \{\epsilon\}$, l'algorithme fait alors croître ces ensembles en remplissant les cases par requêtes jusqu'à que le tableau soit "consistant et clos". Une procédure polynomiale permet alors de construire l'automate déterministe minimal, consistant avec les données du tableau. L'algorithme pose au plus $O(mn^2)$ de requêtes d'appartenance et $n-1$ requêtes d'équivalences, où n est le nombre d'états de l'automate (minimal) cible et m la longueur maximale des contre-exemples retournés par l'oracle.

Ce résultat est séduisant mais nécessite en pratique la présence d'un oracle connaissant la solution pour répondre aux requêtes (notamment d'équivalence). Angluin a montré que pourvu que l'ensemble d'exemples soit "représentatif" (*representative sample*, toutes les transitions de l'automate cible sont exercées par l'échantillon d'apprentissage), on pouvait se passer des requêtes d'équivalence [Ang81].

1.3.2.2 Apprentissage par données fixées

Malgré le résultat positif de Gold [Gol67], l'inférence d'automate à partir d'exemples et de contre exemples est difficile en général. Ainsi, Gold [Gol78] a montré que le problème de trouver un automate fini déterministe de taille minimale compatible avec un ensemble fini d'exemples positifs et négatifs du langage est NP-Complet. Pitt et Warmuth [PW89] ont prouvé que le problème de trouver un automate fini déterministe compatible avec un nombre d'états n' proche du nombre n d'états de la solution optimale est lui même NP-Complet (pour $n' = n^{(1-\epsilon) loglogn}$ avec $\epsilon > 0$).

Le problème n'est donc traitable que si l'échantillon d'apprentissage n'est pas quelconque. Trakhenbrot et Barzdin proposent un *algorithme par fusion d'états* de complexité polynomiale en la taille de l'échantillon [TB73] pour construire le plus petit automate fini déterministe compatible avec un échantillon *complet*.

Un échantillon est dit complet si il est constitué de toutes les chaînes de Σ^*, étiquetées comme exemple positif ou négatif, de longueur inférieure ou égale à $d+1+\rho$, où d et ρ sont respectivement la profondeur (depth)[2] et le *degré de distingabilité*[3] de l'automate cible . Dans le pire des cas, cette longueur est égale à $2n-1$ [TB73] et la taille d'un échantillon complet croît alors exponentiellement avec la taille de l'automate cible. Cependant, Trakhenbrot et Barzdin ont montré qu'en moyenne $\rho = log_|\Sigma| log_2 n$ et $d = Clog_|\Sigma| n$ où C est une constante dépendant de la taille de l'alphabet $|\Sigma|$. La taille moyenne d'un échantillon complet est donc $|\Sigma|^2 n^C log_2 n - 1$. Ce résultat doit être mis en rapport avec l'étude de Angluin qui a montré que, dans le pire des cas, l'identification exacte est impossible dès qu'une fraction arbitrairement petite de l'échantillon complet est manquante [Ang78]. L'identification exacte est alors subordonnée à la présence dans l'échantillon d'apprentissage d'un ensemble d'exemples caractéristique pour la méthode d'inférence.

La famille des algorithmes d'inférence par fusion est étudiée plus en détail dans la suite de l'étude. Le principe de ce type d'algorithme consiste à construire l'automate sous forme d'arbre qui accepte exactement les exemples positifs de l'échantillon d'apprentissage et à en fusionner les états tant que l'automate résultant est compatible (i.e. les exemples négatifs ne sont pas acceptés).

Cette famille comprend notamment RPNI proposé par Oncina et Garcia [OG92] et indépendamment par Lang [Lan92] dans lequel, par rapport à l'algorithme de Trakhenbrot et Barzdin, l'ordre de fusion des états est fixé (largeur d'abord). Il permet l'identification à la limite de la classe des langages réguliers [OG92, Dup96] et a été montré expérimentalement efficace pour l'identification approximative des automates déterministes [Lan92]. Cet algorithme de complexité globale en $O((|S_+| + |S_-|).|S_+|^2)$ et dont la taille de l'échantillon caractéristique est en $O(n^2)$, où n est le nombre d'états de l'automate cible, est particulièrement efficace.

2. La profondeur d'un automate est la taille du plus long élément de l'ensemble des plus courts préfixes permettant d'atteindre chaque état

3. Le degré de distingabilité d'un automate est la longueur du plus petit suffixe assurant la non équivalence de deux états.

Les caractéristiques de RPNI permettent donc d'affirmer l'apprenabilité de la classe des automates, dans le modèle d'apprentissage polynomial à la limite à partir de données fixée [Hig96] (ce résultat a été démontré vrai pour un algorithme général par fusion, quelque soit l'ordre des fusions effectuées pourvu que le choix des états à fusionner soit également polynomial [HOV96]). Il a également été utilisé pour montrer l'apprenabilité PAC par exemples simples des automates déterministes [PH97] et l'apprenabilité à partir d'exemples positifs simples. L'efficacité des algorithmes par fusion peut être améliorée en pratique par le choix d'heuristiques "Data-Dependant" [HOV96] qui consistent à prendre en compte la nature des données pour ordonner dynamiquement les fusions d'états à effectuer. Ce type d'heuristique s'est révélé très performant lors de la compétition Abbadingo One [LPP98].

Cependant, la qualité du résultat fourni par ces algorithmes gloutons dépend de la présence d'un échantillon caractéristique dans l'échantillon d'apprentissage en l'absence duquel le résultat peut être très éloigné de la cible sans qu'il y ait moyen de le savoir. Le risque de ne pas trouver l'automate cible peut être diminué en explorant une plus grande partie de l'espace de recherche. Ceci peut être effectué par la réitération de la recherche gloutonne (par exemple, à partir d'une exploration en largeur d'abord bornée) mais d'autres types d'exploration ont également été étudiés. L'algorithme RIG permet une recherche complète [MdG94] et peut être décliné sous une version heuristique explorant l'espace en faisceau appelée BRIG [MdG94]. L'exploration de l'espace peut également être faite par algorithmes génétique [Dup96] ou méta-heuristiques [Gio95].

1.3.3 Inférence k-régulière

L'inférence régulière consiste à apprendre un langage à partir d'exemples de ce langage. Les langages réguliers étant clos par complémentation, cela revient donc également à apprendre le langage complémentaire. Les exemples négatifs ajoutés pour assurer l'identification à la limite suivant le théorème de Gold [Gol78] peuvent ainsi être vus comme un moyen de limiter la surgénéralisation mais aussi comme des exemples d'apprentissage pour le langage complémentaire. Le problème de l'inférence à partir d'échantillons positifs et négatifs peut alors être considéré comme l'apprentissage d'un langage positif et d'un langage négatif sous la contrainte qu'ils soient disjoints. Nous formalisons et généralisons cette approche pour plusieurs langages réguliers :

Définition 12 *[Cos99] Nous appelons* inférence k-régulière *le problème d'inférer (simultanément) k langages réguliers à partir d'exemples de chacun des langages.* □

Ce cadre permet d'étendre l'application de techniques de l'inférence régulière à des problèmes de classification de séquences multi-classes comme ceux rencontrés notamment en reconnaissance des formes. Le problème peut alors être vu comme l'inférence d'un classifieur de séquences parmi un nombre fini de classes ou langages, une séquence pouvant ne pas être classée. Par rapport à un apprentissage successifs de chacun des langages par des méthodes classiques d'inférence grammaticale, l'inférence simultanée des langages permet de contrôler et guider directement la généralisation des différents échantillons, en fonction de l'interaction entre les langages obtenus.

L'interaction considérée dans ce mémoire est une intersection vide entre les langages, ce qui correspond à l'extension naturelle du problème de l'inférence d'automates compatibles avec une présentation complète. Dans ce cas, une séquence pourra être classée dans au plus un langage, on parlera alors de *classification univoque*. Mais d'autres types d'interactions peuvent être envisagées.

L'approche classification, plutôt que acceptation, peut également s'avérer intéressante pour le problème de l'inférence à partir d'exemples positifs et négatifs ; grâce à la possibilité d'exprimer l'indétermination du langage auquel appartiennent certaines séquences, notamment dans le cadre de l'apprentissage à partir de données fixées. En effet, dans ce cadre, l'identification polynomiale n'est possible que si l'échantillon d'apprentissage comprend un échantillon caractéristique pour l'algorithme d'apprentissage. Un échantillon caractéristique doit permettre de reconstituer les états et transitions de l'automate cible mais également de guider l'algorithme pour qu'il "évite les mauvais choix". Si la première condition paraît nécessaire pour pouvoir apprendre quelque chose, la deuxième est proche d'une situation de collusion entre un enseignant codant l'automate sous la forme d'exemples et l'élève la décodant à l'aide de l'algorithme d'apprentissage. Hors de cette situation de collusion, il peut exister des indications statistiques que l'échantillon caractéristique soit contenu dans l'échantillon mais celà peut impliquer un volume d'exemples rarement disponible en pratique. Ainsi, si l'on considère qu'il y a peu de chances que l'échantillon soit caractéristique, envisager le problème de l'inférence à partir d'une présentation complète sous l'angle de la classification permet d'intégrer l'incertitude due à l'insuffisance potentielle des échantillons d'apprentissage. La "quantité d'incertitude" peut alors même constituer un des critères d'évaluation des solutions dans l'espace de recherche ou un critère d'arrêt à la limite de l'inférence.

Enfin, cette approche permet de formaliser une pratique, de plus en plus utilisée en inférence régulière pour les algorithmes par fusion, qui consiste à construire l'arbre accepteur des préfixes non seulement des exemples positifs mais aussi des exemples négatifs. Cette technique a été utilisée dans des implémentations de RPNI à Valence, pour éviter d'avoir à *parser* les exemples négatifs pour vérifier la compatibilité de l'automate obtenu par une fusion. La première formalisation de l'utilisation symétrique des exemples positifs et négatifs est faite par Alquezar et Sanfeliu par l'introduction des unbiaised finite state automata [AS95]. L'utilisation simultanée des exemples positifs et négatifs est également employée dans les algorithmes avec les heuristiques de fusion d'état "data driven" de [HOV96] et [LPP98].

Fort de toutes ces motivations, nous proposons une étude de l'inférence k-régulière dans les chapitres suivants. Cette étude sera présentée avec le choix de représenter les ensembles de langages par des automates classifieurs (introduits en section 2.1). Ce choix de représentation par machines à états finies fait ressortir la parenté de l'approche avec le problème de l'identification de machine à partir des entrées-sorties. dont nous présentons les principales caractéristiques et définitions dans la section suivante.

1.4 Inférence de machines à états finies

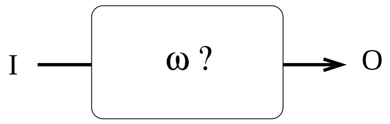


FIG. 1.4 – *Identification de boîte noire à partir des entrées-sorties : trouver une fonction ω telle que pour tout couple d'entrée-sortie (i,o) : $o = \omega(i)$.*

L'identification d'une machine, vue comme une boîte noire, à partir d'exemples de fonctionnement est une situation d'apprentissage couramment traitée en Intelligence Artificielle. Si les entrées-sorties sont sous une forme symbolique, une des hypothèses possibles est que le fonctionnement de la boîte noire peut être décrit par une machine à états finie.

Une machine à états finie est un système dynamique discret qui traduit une séquences de symboles d'entrée en une séquence de symboles de sortie. La traduction est assurée par un ensemble d'états et une fonction de transition qui permet d'assurer le changement d'état, en fonction du symbole d'entrée.

Les états peuvent être considérés comme une mémorisation du passé, de sorte que l'état courant dépend du symbole d'entrée courant mais aussi des symboles précédents dans la séquence d'entrée. Si le passé n'est pas pris en compte, il n'y a qu'un seul état et la machine calcule une *fonction combinatoire.* La mémoire peut être éventuellement effacée si un bouton de réinitialisation permettant de remettre la machine dans un état initial est disponible.

Les machines à états finies avec bouton de réinitialisation, ou de façon équivalente avec l'utilisation d'un symbole spécial de fin de séquence, peuvent ainsi traiter des séquences de séquences par prise en compte de la sortie et réinitialisation après chaque séquence. Trois catégories fonctionnelles de traitement des séquence peuvent alors être discernées : la reconnaissance , la classification et la traduction de séquences.

Les machines à états finies reconnaissant ou acceptant une séquence sont les automates à états finis. Les méthodes d'identification sont alors celles décrites dans la section précédente présentant l'inférence régulière. L'identification de classifieurs de séquences a été assez peu étudiée en tant que telle mais fait l'objet du prochain chapitre. Nous proposons d'étudier, dans les prochaines sections, les travaux effectués pour la traduction de séquences.

1.4.1 Inférence de machines séquentielles

L'identification de machines à partir des entrées-sorties est particulièrement étudiée pour la synthèse et la conception de circuits logiques. L'enjeu de cette recherche est la synthèse de circuits séquentiels ou de programmes à partir d'exemples de fonctionnement. Les machines usuellement considérées sont les machines séquentielles de Mealy et de Moore (définies en section 1.1.4), l'alphabet de sortie considéré étant généralement binaire. L'identification de machines de Mealy peut être faite à partir de séquences de paires d'entrées-sorties à partir de l'état initial. Chaque transition associant un symbole de sortie au symbole d'entrée. L'identification de la machine est alors relativement facile et correspond à une minimisation exacte.

Le problème est plus difficile si des sorties ou des transitions manquent ou sont incomplètement spécifiées comme pour la synthèse de circuits imprimés [KVBSV94]. Dans ce dernier cas, on parle de "don't care output", ce qui signifie que la production de l'état est indifférente, généralement parce qu'elle n'est pas prise en compte dans le circuit. Le problème consiste alors à trouver la machine minimale incluant la spécification d'entrées-sorties donnée en exemple ou, sous une autre formulation, à minimiser la machine à états finie incomplètement spécifiée.

Les premiers travaux sur la synthèse de machines de Moore ou de Mealy à partir d'exemples d'entrées-sorties appartenant à (Σ_i^*,Y) où Y peut être soit Σ_o, soit Σ_o^* ont été effectués par Biermann et Feldman [BF72]. Le choix de Y correspond à une utilisation dynamique de la structure d'une machine de Mealy ou de Moore, vue comme un accepteur (ou classifieur) ou comme un traducteur. La machine est construite en utilisant une variation de la relation d'équivalence de Nerode, bornée par un paramètre k. L'augmentation du paramètre k permet d'obtenir la machine cible à partir d'un certain seuil.

Une évolution de cet algorithme [BBP75] a commencé à prendre en compte les fusions incompatibles, par l'ajout d'une technique de mémorisation ("backmarck"), pour la synthèse de programmes informatiques à partir d'exemples de traces. Cet algorithme a été encore amélioré par Oliveira et Silva, grâce à l'ajout d'une technique de diagnostic des conflits pour permettre un retour arrière plus performant [OS98]. Ce type d'algorithme a été appliqué à la minimisation de machines incomplètement spécifiées et semble meilleur en moyenne [OE96, OS98, PO98] que les approches traditionnellement utilisées dans ce domaine.

Il serait néanmoins intéressant d'étudier si les méthodes utilisées par ces dernières, basées essentiellement sur le calcul "d'ensembles premiers compatibles" (*prime compatible set*), ne pourraient pas être utilisées en inférence grammaticale pour certains types de problèmes (à caractériser). Inversement, la minimisation de machines incomplètement spécifiées est un champ potentiel d'application des algorithmes d'inférence régulière pour lequel la recherche exacte de solution est un enjeu industriel important.

1.4.2 Inférence de transducteurs

Le principal algorithme d'inférence de transducteurs sous-séquentiels est OSTIA proposé par Oncina, Garcìa et Vidal [OGV93], qui constitue une extension de RPNI aux transducteurs sous-séquentiels ; la difficulté consistant à "faire glisser" de façon adéquate les étiquettes des productions. Dans cette optique, les auteurs définissent une forme canonique de répartition des productions émises par les transitions et les états, définissant la classe des "Onward Subsequential Transducer" (OST) telle que les productions sont effectuées le plus tôt possible, c'est à dire le plus "près" possible de l'état initial. L'algorithme d'inférence consiste alors à construire le transducteur arborescent sous forme OST, reconnaissant exactement l'échantillon d'apprentissage. Puis des fusions d'état, conservant l'unicité de production pour chaque traduction possible, sont successivement effectuées, les productions étant éventuellement repoussées plus loin de l'état initial. Les auteurs ont montré que l'algorithme OSTIA permet l'identification à la limite des transductions sous-séquentielles totales à partir d'exemples de transductions. Le terme total indique que tous les mots de Σ_i^* doivent pouvoir être traduits, la notion d'état d'acceptation n'étant alors pas exploitée.

L'extension d'OSTIA à l'inférence de fonctions sous-séquentielles partielles a été étudiée par Oncina et Varò [OV96] qui considèrent deux situations : l'ajout d'information externe sur le domaine sous la forme d'automates ou la mise à disposition d'exemples "négatifs". D'autres travaux ont été effectués pour l'inférence de transducteurs sous-séquentiels sur la base d'OSTIA. Dans [Onc98], une amélioration de l'ordre des fusions est proposée. L'application à la traduction de langues naturelles a également été testée [PV98].

On peut également mentionner les récents travaux de Mäkinen [Mäk99] établissant le lien entre inférence de transducteurs et inférence de grammaires algébriques grâce au théorème suivant :

Théorème 3 *[Ros67] τ est une transduction rationnelle si et seulement si il existe un langage algébrique L tel que $\tau = \{(x,y)/x\#\tilde{y} \in L\}$, où $\#$ est un nouveau symbole et $\tilde{y}$ est le mot miroir de y.*

Sans perte de généralité, on peut considérer les transducteurs sur les symboles (i.e. $E \subseteq Q \times (\Sigma_i \cup \{\epsilon\}) \times (\Sigma_o \cup \{\epsilon\}) \times Q$) et les productions des grammaires de la forme $A \to aBb$ avec $a,b \in \Sigma \cup \{\epsilon\}$ et les productions finales de la forme $A \to \#$. La correspondance entre les deux représentations revient alors à associer à une production $A \to aBb$, une transition (q_A,a,b,q_B) et à une production finale $A \to \#$, une transition $(q_A,\epsilon,\epsilon,q_f)$. On peut alors utiliser les résultats de l'inférence de sous classes de grammaires algébriques [Tak88, Sak92, SN98, KMT97, Mäk94] par transposition des restrictions adoptées aux transducteurs. Par exemple, l'algorithme d'inférence de grammaires linéaires équilibrées sous le contrôle d'un ensemble régulier, proposé par Takada [Tak88], permet d'inférer à la limite, à partir d'exemples positifs et négatifs, les transducteurs sur les symboles préservant la longueur (i.e. tels que $E \subseteq Q \times \Sigma_i \times \Sigma_o \times Q$) [Mäk99].

Avec ce dernier résultat, on voit se dessiner un contour possible de la famille des concepts apprenables en inférence grammaticale, qui serait l'ensemble des fonctions réalisables par machines à états finies et que l'on pourrait appeler les problèmes d'inférence rationelle. Au sein de cette famille, nous proposons d'étudier plus particulièrement, dans la prochaine partie, l'inférence d'une classe de machines à états finies, que nous appelons automates classifieurs. Cette classe peut être vue comme une extension des automates, ou comme une restriction des transducteurs telle que seule l'émission finale pour chaque séquence est considérée. Le travail présenté peut ainsi être également considéré comme l'étude particulière de l'apprentissage de la fonction d'émission finale pour l'inférence de machines à états finies.

Chapitre 2

Inférence d'automates classifieurs

Nous considérons à présent le problème d'inférence k-régulière introduit dans la section 1.3.3. Nous nous plaçons dans le cadre d'un apprentissage de classifications univoques à partir de données fixées. Le problème consiste alors à inférer simultanément un ensemble de k langages réguliers deux à deux disjoints à partir d'exemples de chacun des langages.

Dans ce chapitre, nous proposons d'étudier l'espace de recherche pour ce problème quand la représentation choisie pour un ensemble de langages est une extension des automates finis appelée automate classifieurs introduite en section 2.1. L'espace de recherche associé peut alors être défini par simple extension de l'espace de recherche considéré pour l'inférence des automates [DME94]. Cet espace est présenté en section 2.2. Moyennant une hypothèse sur l'échantillon d'apprentissage, l'espace de recherche est un treillis fini dont l'élément nul est un automate classifieur particulier appelé MCA.

L'opération élémentaire de parcours de l'espace de recherche, employé par les principaux algorithmes d'inférence régulière est la fusion d'états. Nous présentons en sections 2.3 et 2.4 une étude plus particulière des conséquences de chaque fusion effectuée grâce à l'introduction d'un nouvel espace de recherche implicite basé sur les fusions de paires d'états de MCA. Cette étude nous permet notamment de mettre en évidence l'ensemble des paires d'états incompatibles. Nous proposons un algorithme efficace de calcul préalable de cet ensemble de paires d'états ne devant pas être fusionnés et caractérisons la dynamique de propagation de l'ensemble incompatibilités en fonction des choix de fusions effectués lors de l'inférence. Dans le cas déterministe considéré en section 2.4.3, cette étude permet d'exprimer l'ensemble des automates classifieurs univoques déterministes de l'espace de recherche par un système de contraintes sur l'automate classifieur MCA.

Ce système de contraintes peut être considéré comme le nouvel espace de recherche à parcourir pour l'inférence d'automates classifieurs déterministes univoques, mais aussi pour l'inférence classique d'automates déterministes compatibles, comme présenté dans la dernière section de ce chapitre.

2.1 Automates classifieurs

Alquezar et Sanfeliu [AS95] introduisent un nouveau type d'automates appelé “Unbiaised Finite State Automata” (UFSA) pour permettre une inférence d'automate considérant symétriquement les données positives et négatives.

Définition 13 *[AS95] Un $UFSA$ U est un sextuplet (Σ,Q,q_0,F_+,F_-,E) où :*
- *Σ,Q,q_0 et E sont définis comme pour un automate,*
- *$F_+ \subseteq Q$ et $F_- \subseteq Q$ sont les ensembles d'états finals respectivement positifs et négatifs et tels que $F_+ \cap F_- = \emptyset$.*

□

Le langage accepté par un $UFSA$ U est alors :

$$L_A(U) = \{w \in \Sigma^* / \exists q_f \in F_+,(q_0,w,q_f) \in \hat{E}\}$$

Un UFSA permet également de définir le langage rejeté :

$$L_R(U) = \{w \in \Sigma^* / \exists q_f \in F_-,(q_0,w,q_f) \in \hat{E}\}$$

et l'ensemble de mots “ignorés” par U :

$$L_I(U) = \Sigma^* - (L_A(U) \cup L_R(U))$$

ce qui permet d'introduire l'incertitude de classification d'un mot dans un langage si les données ne sont pas suffisantes.

L'introduction de cette structure permet de considérer symétriquement et simultanément les données positives et négatives et permet d'étendre les algorithmes par fusion d'états ou plus généralement d'inférence d'automates pour y inclure un biais d'apprentissage favorisant la généralisation du langage positif, négatif ou des deux [AS95]. Le problème traité est donc un problème d'inférence 2-régulière.

L'extension à plus de deux langages est immédiate. Dans sa thèse, Alquezar [Alq97] présente l'extension des UFSA de deux ensembles d'états finals à un nombre k fixé d'ensembles d'états finals $(F_1,\ldots,F_C)$ qu'il appelle *k-Classes Unbiaised Finite State Automaton* (k-UFSA) et mentionne leur utilité potentielle pour des problèmes de reconnaissance multi-classes. Ce type de machine pourrait aussi être défini à l'aide du formalisme des semi-automates présentés par Berstel [Ber79] . Cependant, en rapport avec la fonctionnalité de ces machines, nous préférons utiliser le formalisme des machines à états finies et parler alors plus simplement d'*automate classifieurs*.

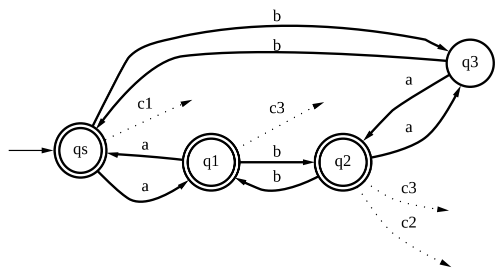


FIG. 2.1 – 3-*DFA classant les mots dans les classes* c1*,* c2 *et* c3 *définies respectivement par : le mot a un nombre pair de a et un nombre pair de b (*c1*), un nombre impair de a et un nombre impair de b (*c2*) un nombre impair de a (*c3*). L'ensemble des mots ignorés par cet automate classifieur est constitué des mots comportant un nombre pair de a et un nombre impair de b.*

Définition 14 *Un automate classifieur A à k classes d'acceptation (également appelé automate k-classifieur et noté k-FSA), est défini par un sextuplet $(\Sigma,\Gamma,Q,q_0,\delta,\rho)$ où :*

- Σ_i *et* Γ *sont respectivement les alphabets d'entrée et de classification,* Γ *comportant* k *symboles appelés* classes.
- Q *est un* ensemble fini d'états,
- q_0 *est* l'état initial,
- δ *est une fonction de* $Q \times \Sigma$ *dans* $\mathcal{P}(Q)$ *appelée* fonction de transition.
- ρ *est une fonction de* Q *dans* $\mathcal{P}(\Gamma)$ *appelée* fonction d'émission finale.

L'extension aux mots de la fonction de transition δ étant effectuée classiquement, la fonction de classification γ_A *de A est définie de $\Sigma^* \times Q$ dans $\mathcal{P}(\Gamma)$ en fonction de ρ par :*

$$\gamma_A(q,w) = \bigcup_{q' \in \delta(q,w)} \rho(q')$$

□

La classification γ_{A} de Σ^* dans $\mathcal{P}(\Gamma)$ effectuée par un automate classifieur $A = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ peut alors être définie pour tout mot w de Σ^* par :

$$\gamma_{\mathrm{A}}(w) = \gamma_{\mathrm{A}}(q_0,w)$$

La relation associée étant :

$$L(A) = \{(w,k) \in \Sigma^* \times \Gamma / k \in \gamma_{\mathrm{A}}(w)\}$$

On appelle *domaine de classification* d'un k-FSA A, noté $dom(A)$, l'ensemble des mots de Σ^* dont la classification est définie :

$$dom(A) = \{w \in \Sigma^* / \gamma_{\mathrm{A}}(w) \neq \emptyset\}$$

La classification sera dite *univoque* si tout mot du domaine peut être classé dans au plus une classe, elle sera dite *ambigüe* sinon. Elle sera dite *complète* si tous les mots de Σ^* sont classés dans au moins une classe.

Pour rester compatible avec la terminologie des automates, nous définissons le déterminisme d'un automate classifieur par rapport au déterminisme de sa fonction de transition seulement. Le déterminisme d'un automate classifieur assure donc essentiellement l'unicité et la séquentialité de l'analyse d'un mot par l'automate.

Définition 15 *un* automate classifieur déterministe *(noté k-DFA) est un automate classifieur dont la fonction de transition est déterministe. Nous noterons k-NFA les automate classifieur non déterministe.* □

Dans cette définition, le déterminisme d'un automate classifieur n'est donc pas dépendant du déterminisme de la fonction d'émission finale. Concernant le déterminisme de la fonction de classification, on préferera se référer à la notion d'univocité. On peut remarquer que tout automate classifieur déterministe tel que la fonction d'émission finale est également déterministe est univoque. Dans le sens inverse, l'univocité de la fonction de classification implique le déterminisme de la fonction d'émission finale mais pas forcément celle de la fonction de transition. Enfin, notons que pour tout automate classifieur, un automate classifieur déterministe peut être construit suivant l'algorithme classique de déterminisation des machines à états finies [Boo67].

L'algorithme de minimisation par classes d'équivalence de Nerode [AU72] peut également être adapté aux automates classifieurs en considérant les classes d'équivalences entre états induites par la fonction $\gamma(q,w)$. La déterminisation et la minimisation d'automates classifieurs assurant l'unicité de l'automate obtenu, la notion d'automate classifieur canonique peut être définie comme pour les automates classiques.

Définition 16 *Étant donné une relation de classification $\gamma \subseteq \Sigma^* \times \Gamma$, l'automate classifieur canonique de γ, noté $A(\gamma)$, est l'automate classifieur déterministe minimal réalisant γ.* □

Un k-FSA peut également être vu comme une superposition de k automates finis et permet la représentation d'un ensemble de k langages réguliers :

Définition 17 *Un k-FSA A accepte un ensemble de k langages réguliers $\mathcal{L}(A) = \{L_k(A)\}_{k\in\Gamma}$, défini par :*

$$\forall k \in \Gamma : L_k(A) = \{w \in \Sigma^* / k \in \gamma_A(w)\}.$$

□

On peut alors reformuler certaines des définitions précédentes en termes de langages. L'union des langages acceptés par A est un langage régulier qui représente le domaine de classification de A, on a donc

$$dom(A) = \bigcup_{k\in\Gamma} L_k(A).$$

Les mots du langage complémentaire $L_I(A) = \Sigma^* - L_{dom}(A)$ sont les mots pour lesquels la classification est indéfinie.

Les classifications univoques et totales peuvent alors être caractérisées par :

$$\forall i,j \in \Gamma, i \neq j, L_i(A) \cap L_j(A) = \emptyset \text{ (univocité)}$$

$$L_I(A) = \emptyset \text{ (complétude)}$$

De même, la définition de l'automate canonique peut être relative à un ensemble de langages $\mathcal{L}$. On notera $A(\mathcal{L})$, l'automate classifieur déterministe minimal acceptant exactement $\mathcal{L}$.

Un automate classifieur simule un ensemble de k automates accepteurs, chacun pouvant être obtenu par restriction de l'automate classifieur. La restriction à un langage L_k d'un automate classifieur consiste à supprimer les branches inutiles pour l'acceptation des mots dans le langage L_k et à transformer tous les états dont l'émission finale contient k en état final. La transformation d'un automate en un automate classifieur peut être faite de la façon inverse. Un automate classifieur équivalent à un ensemble de k automates peut être obtenu en effectuant la transformation de chacun des automates en automate classifieur avec un symbole de classe différent puis en réalisant l'union des automates classifieurs obtenus.

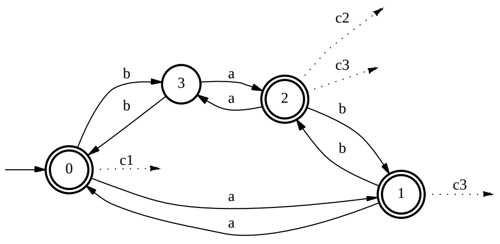
Un automate classifieur déterministe A peut être représenté par un transducteur sous-séquentiel SST avec une fonction d'émission λ définie seulement de $Q \times \Sigma$ dans $\{\epsilon\}$ et une fonction d'émission finale définie de Q dans Γ (voir figure 2.2). Dans ce cas :

$$\tau_{\text{SST}}(w) = \lambda(q_0,w)\rho(\delta(q_0,w) = \rho(\delta(q_0,w)) = \gamma_{\text{A}}(w).$$

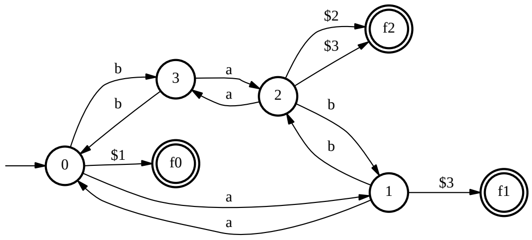
Pour les automates classifieurs non déterministes, la même transformation peut être considérée si l'on ajoute une fonction d'émission finale non déterministe à la définition d'un transducteur rationnel, ce qui ne change pas son pouvoir d'expression. Sinon, dans le cas non déterministe, l'automate peut être vu comme un transducteur en ajoutant au graphe un nouvel état, qui sera l'unique état final, et des ϵ-transitions (q,ϵ,k,q_f) pour tous les états q tels que $k \in \gamma(q)$.

L'inférence d'automates classifieurs se situe donc, au sein de l'inférence de machines à partir des entrée-sorties, entre l'inférence d'automates classique qu'elle inclut ("inférence 1-régulière") et l'inférence de transducteurs dont elle peut être considérée comme le pendant limité à la partie acceptation/émission finale.

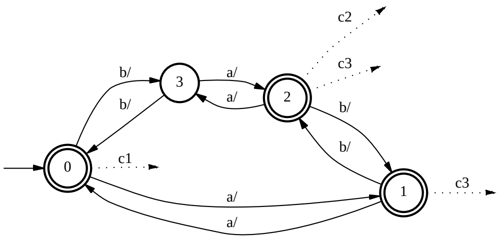
Nous nous intéresserons plus particulièrement au cas où k est supérieur ou égal à deux, ce qui permet de considérer l'inférence de classifieur de séquences pour des problèmes multiclasses. Les méthodes développées pour cette inférence peuvent également être utiles pour l'inférence de transducteurs : soit par application des techniques employées avec l'ajout de la prise en compte de la fonction d'émission des transitions,



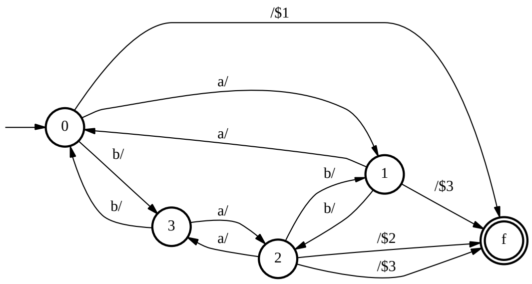
Automate classifieur



Automate avec symbole de fin



Transducteur sous séquentiel



Transducteur

FIG. 2.2 – *Représentations équivalentes de classifieurs*

soit par la décomposition du problème en la minimisation d'une machine de traduction incomplètement spécifiée d'une part et l'identification du domaine de traduction et de la fonction d'émission finale par automate classifieur d'autre part. Enfin, on pourrait éventuellement considérer l'inférence de "transducteurs classifieurs" si une application requérant une classification de la traduction effectuée était identifiée.

L'étude du cas particulier $C = 2$, où les échantillons sont supposés être ceux d'un langage et de son complémentaire, permet de plus une approche alternative intéressante au problème de l'inférence d'automates classique à partir d'exemples positifs et négatifs et constitue la motivation originelle de notre approche. Dans la section suivante, nous proposons de commencer l'étude de l'inférence d'automates classifieurs par la définition d'un espace de recherche.

2.2 Espace de recherche

Cette section est dédiée à la définition de l'espace de recherche pour l'inférence d'automates classifieurs dans le cadre de l'apprentissage à partir de données fixées. Nous supposons qu'un échantillon d'apprentissage $\mathcal{S} = (S_1, \ldots ,S_C)$ est donné, tel que pour k compris entre 1 et k, S_k est un ensemble fini de mots du langage cible L_k, appelé échantillon d'apprentissage de L_k. Nous noterons $\{w_{k,1}, \ldots ,w_{k,|S_k|}\}$ les mots d'un échantillon S_k et $(a_{k,i,1} \ldots a_{k,i,|w_{k,i}|})$ les symboles d'un mot $w_{k,i}$.

Pour définir l'espace de recherche, une hypothèse classique en inférence grammaticale consiste à supposer que l'échantillon d'apprentissage est structurellement complet relativement à l'automate cible. En définissant la complétude structurelle relative aux automates classifieurs, nous présentons dans cette section une extension simple aux automates classifieurs des définitions et propriétés de l'espace de recherche sous cette hypothèse, présentées dans [DME94] [DM98] et [Alq97]. Le plan de cette présentation est le suivant : nous commençons par introduire la complétude structurelle et l'espace de recherche sous cette hypothèse en section 2.2.1 avant de considérer la restriction de l'espace de recherche aux éléments déterministes en section 2.2.2 ou univoques en section 2.2.3.

2.2.1 Complétude structurelle

L'identification exacte d'un langage à partir d'un échantillon d'apprentissage ne peut être assurée que si l'échantillon est suffisamment "représentatif" du langage à apprendre. Pour l'inférence de grammaires régulières, l'échantillon positif S_+ est généralement supposé structurellement complet relativement au langage cible.

Définition 18 *[FB75] Un échantillon S_+ est* structurellement complet *relativement à un langage $L(G)$ si chaque règle de production de la grammaire G est utilisée pour la génération d'au moins un mot de S_+.* □

De façon induite, un échantillon S_+ est structurellement complet relativement à un automate A [DME94] s'il existe une acceptation de S_+ par A telle que :

1. toute transition de A est exercée ;
2. tout état final de A est utilisé comme état d'acceptation.

L'hypothèse de la complétude structurelle de l'échantillon relativement au langage cible semble raisonnable pour l'inférence d'automates. En effet comment deviner une transition (ou un état d'acceptation) inutile pour l'acceptation des mots de l'échantillon d'apprentissage ?

D'autres arguments plaident en faveur de l'hypothèse de complétude structurelle. Cette hypothèse est notamment la contre-partie au résultat positif d'identification de l'automate à l'aide de requêtes d'équivalence et d'appartenance à un oracle ([Ang87], voir section 1.2.2). La deuxième condition de la complétude structurelle sur les états d'acceptation joue un rôle semblable aux requêtes d'appartenance. Quant aux requêtes d'équivalence, Angluin a montré qu'on pouvait s'en passer pourvu que l'échantillon d'apprentissage utilise toutes les transitions de l'automate cible [Ang81], ce qui est stipulé dans la première condition. L'hypothèse de complétude structurelle de l'échantillon d'apprentissage peut donc être vue comme un palliatif à l'absence d'oracle en pratique. Enfin, notons que les ensembles d'échantillons caractéristiques assurant l'identification des langages k-réversibles suivant l'algorithme présenté par Angluin ([Ang82], voir section 1.3.1) sont structurellement complets par construction.

Par la suite nous supposerons donc que l'échantillon d'apprentissage est structurellement complet, mais nous considérons une définition étendue pour prendre en compte l'acceptation de plusieurs langages par les automates classifieurs.

Définition 19 *Un échantillon $\mathcal{S} = \{S_k\}_{k\in\Gamma}$ est dit structurellement complet relativement à un k-FSA $A = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ s'il existe une acceptation des mots de $\mathcal{S}$ par A telle que :*

1. toute transition de A est exercée, i.e. :

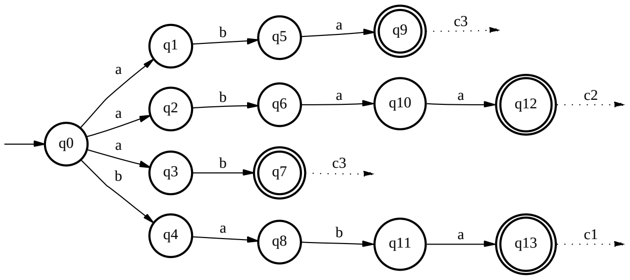
$$\forall q,q' \in Q, \forall a \in \Sigma, q' \in \delta(q,a) \;\Rightarrow\; \exists u,v \in \Sigma^*, \exists k \in \Gamma, uav \in S_k,\ q \in \delta(q_0,u), k \in \gamma_A(q',v)$$

2. toute émission finale de A est exercée, i.e :

$$\forall q \in Q, \forall k \in \rho(q),\ \exists w \in S_k,\ q \in \delta(q_0,w)$$

□

Cette définition illustre l'intérêt de considérer l'inférence simultanée des langages par automate classifieur. En effet, la définition de la complétude structurelle proposée ici porte sur l'ensemble de l'échantillon d'apprentissage $\mathcal{S}$ par rapport à la machine cible. Tous les échantillons d'apprentissage participent à la complétude structurelle, même pour l'acceptation de mots d'une autre classe. Un seul exemple peut ainsi définir une transition utile à la reconnaissance de mots de classes différentes. Le nombre



$$\mathcal{S} = \langle S_{c1}, S_{c2}, S_{c3} \rangle = \langle \{baba\}, \{abaa\}, \{ab, aba\} \rangle.$$

FIG. 2.3 – $MCA(\mathcal{S})$.

d'exemples nécessaire à la complétude structurelle est alors potentiellement beaucoup plus petit que si l'on avait dû considérer la complétude structurelle de chacun des sous-échantilllons S_k par rapport aux automates canoniques $A(L_k)$ correspondants.

Pour un échantillon donné $\mathcal{S}$, on peut construire *l'automate classifieur canonique maximal*, noté $MCA(\mathcal{S})$, pour lequel $\mathcal{S}$ est structurellement complet. Cet automate classifieur en forme d'arbre est composé d'une branche par mot de $\mathcal{S}$. $MCA(\mathcal{S})$ est en général non déterministe, le non déterminisme étant exclusivement localisé sur les transitions sortantes de l'état initial. L'ensemble de langages acceptés par $MCA(\mathcal{S})$ étant exactement $\mathcal{S}$, $MCA(\mathcal{S})$ correspond à un apprentissage par coeur de l'échantillon.

Définition 20 *L'*automate classifieur canonique maximal relatif à $\mathcal{S}$*, noté* $MCA(\mathcal{S})$ [1], *est le k-FSA* $(\Sigma,\Gamma,Q,q_0,\delta,\rho)$ *tel que :*

- Σ *est l'alphabet sur lequel* $\mathcal{S}$ *est défini ;*
- Γ *est l'alphabet des classes correspondant aux ensembles de* $\mathcal{S}$ *;*
- $Q = \{q_{k,i,j} / k \in \Gamma, i \in [1,|S_k|], j \in [1,|w_{k,i}|]\} \cup \{q_0\}$ *;*
- $\forall k \in \Gamma, \forall i \in [1,|S_k|]$, $\rho(q_{k,i,|w_{k,i}|}) = k$ *;*
- $\forall a \in \Sigma$, $\delta(q_0,a) = \{q_{k,i,1} / a_{k,i,1} = a, k \in \Gamma, i \in [1,|S_k|]\}$ *;*
- $\forall k \in \Gamma, \forall i \in [1,|S_k|], \forall j \in [1,|w_{k,i}| - 1]$, $\delta(q_{k,i,j}, a_{k,i,j+1}) = \{q_{k,i,j+1}\}$ *;*

□

L'espace de recherche peut alors être ordonné et défini par une relation d'ordre partielle déduite de la *dérivation* d'automates selon une partition. Rappelons qu'une partition π d'un ensemble S est un ensemble de sous ensembles (appelés blocs) de S, non vides et disjoints deux à deux, dont l'union est S. Si s est un élément de S, nous notons par s_π l'unique bloc de π qui contient s. On peut alors énoncer la définition de la dérivation d'automates :

1. De la terminologie anglaise *Maximal Canonical Automaton*

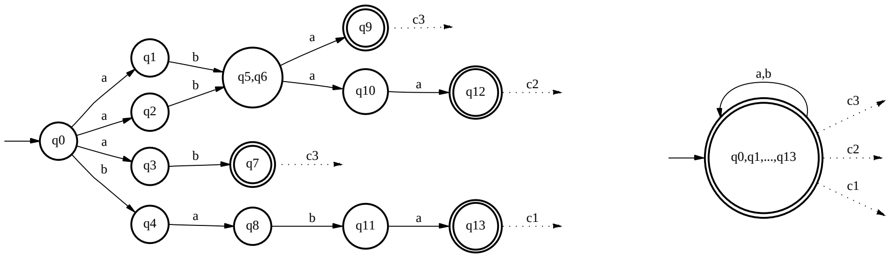


FIG. 2.4 – *Éléments de* $Lat(MCA(\mathcal{S}))$. *L'automate de gauche est un élément du treillis obtenu à partir de* $MCA(\mathcal{S})$ *de la figure 2.3 par la fusion des états* $q5$ *et* $q6$*, alors que celui de droite, est l'élément le plus général du treillis. Il est obtenu par fusion de tous les état de* $MCA(\mathcal{S})$ *et est appelé automate universel (* $UA(\mathcal{S})$*).*

Définition 21 *Étant donné un k-FSA* $A = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ *et une partition* π *des états de* Q*, le* k-FSA dérivé de A par π *est* $A/\pi = (\Sigma,\Gamma,Q',q_{0\pi},\delta',\rho')$ *défini comme suit :*
- $Q' = Q/\pi = \{q_\pi /\ q \in Q\}$
- $\forall B \in Q'$, $\rho'(B) = \bigcup_{q\in B} \rho(q)$
- $\forall B \in Q', \forall a \in \Sigma$, $\delta'(B,a) = \{B' \in Q' /\ \exists q \in B, \exists q' \in B', q' \in \delta(q,a)\}$.

□

Les états de Q appartenant au même bloc B de la partition sont dits *fusionnés*.

L'opération de fusion de paires d'états définit une relation d'ordre partiel sur les automates que nous désignons par $\preceq$. La fermeture transitive de cette relation est notée $\ll$, $A \ll A'$ signifie que A' est dérivé de A. L'ensemble partiellement ordonné des k-FSA dérivés de A forme un treillis que nous désignerons par $Lat(A)$.

La dérivation d'automate conservant la complétude structurelle, le théorème suivant permet de définir l'espace de recherche.

Théorème 1 *Soit* $\mathcal{S}$ *un échantillon d'un quelconque ensemble de langages réguliers* $\mathcal{L}$ *et soit* A *n'importe quel k-FSA acceptant exactement* $\mathcal{L}$*. Si* $\mathcal{S}$ *est structurellement complet relativement à* A*, alors* A *appartient à* $Lat(MCA(\mathcal{S}))$. □

Preuve Nous reformulons ici la preuve de [Alq97] basée sur [DME94] pour prendre en compte la différence de traitement des états finaux. L'idée de la preuve consiste à reconstruire $MCA(\mathcal{S})$ à partir d'une acceptation $\mathcal{AC}(\mathcal{S},A)$ de $\mathcal{S}$ par A, vérifiant les conditions de la complétude structurelle. De cette acceptation, on déduit la partition permettant de dériver un automate isomorphe à A à partir de $MCA(\mathcal{S})$. Nous utilisons les notations suivantes : $MCA(\mathcal{S}) = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ et $A = (\Sigma,\Gamma,Q',q'_0,\delta',\rho')$.

L'acceptation $\mathcal{AC}(\mathcal{S},A)$ définit pour chaque mot $w_{k,i} \in S_k$ une séquence d'états de Q' $(q'_{k,i,0},\dots,q'_{k,i,|w_{k,i}|})$, telle que $q'_{k,i,0} = q'_0$ et $\forall j \in [1,|w_{k,i}|], q'_{k,i,j} \in \delta'(q'_{k,i,j-1},a_{k,i,j-1})$.

En adoptant les notations de la définition de $MCA(\mathcal{S})$ (définition 20), une fonction $\phi : Q \rightarrow Q'$ peut alors être définie par :

1. $\phi(q_0) = q'_0$
2. $\forall k \in \Gamma, \forall i \in [1,|S_k|], \forall j \in [1,|w_{k,i}|]$, $\phi(q_{k,i,j}) = q'_{k,i,j}$.

On peut alors définir une partition π par

$$\forall q_k, q_l \in Q, B_\pi(q_k) = B_\pi(q_l) \Leftrightarrow \phi(q_k) = \phi(q_l)$$

Alors A est isomorphe à $MCA(\mathcal{S})/\pi$, le respect des propriétés de la complétude structurelle par $\mathcal{AC}(\mathcal{S},A)$ impliquant les deux propriétés suivantes :

1. La fonction de transition de $MCA(\mathcal{S})/\pi$ correspond exactement à δ' (car toutes les transitions de δ' sont exercées par $\mathcal{AC}(\mathcal{S},A)$) ;
2. La fonction d'émission de $MCA(\mathcal{S})/\pi$ correspond exactement à ρ' (car $\forall k \in \Gamma, \forall q' \in Q', k \in \rho'(q')\ \exists i \in [1,|S_k|]\ q'_{k,i,|w_{k,i}|} = q'$).

Par conséquent, A appartient au treillis $Lat(MCA(\mathcal{S}))$. □

L'espace de recherche des automates classifieurs (voir figure 2.5) pour lesquels un échantillon donné est structurellement complet est donc l'ensemble des automates pouvant être dérivé de $MCA(\mathcal{S})$ et dont l'élément supérieur est l'automate universel obtenu par fusion de tous les états de $MCA(\mathcal{S})$.

Définition 22 *L'*automate universel *relatif à $\mathcal{S}$, noté $UA(\mathcal{S})$ [2], est le k-DFA $(\Sigma,\Gamma,Q,q_0,\delta,\rho)$ défini comme suit :*

- *Σ est l'alphabet sur lequel $\mathcal{S}$ est défini ;*
- *Γ est un alphabet des classes correspondant aux ensembles de $\mathcal{S}$;*
- *$Q = \{q_0\}$;*
- *$\rho(q_0) = \{k \in \Gamma / S_k \neq \emptyset\}$*
- *$\forall a \in \Sigma$, $\delta(q_0,a) = q_0$*

□

UA($\mathcal{S}$) est donc le k-DFA à un état qui associe à chaque sous-échantillon non vide S_k le langage $L_k = \Sigma^*$, où Σ est l'alphabet sur lequel est défini $\mathcal{S}$, et à chaque échantillon vide $S_{k'}$ le langage vide $\emptyset$ (voir figure 2.4). C'est le plus petit automate pour lequel $\mathcal{S}$ est structurellement complet. On peut noter ici que l'automate universel n'est univoque que dans le cas très particulier où tous les langages, sauf un, sont vides.

2. De la terminologie anglaise *Universal Automaton*

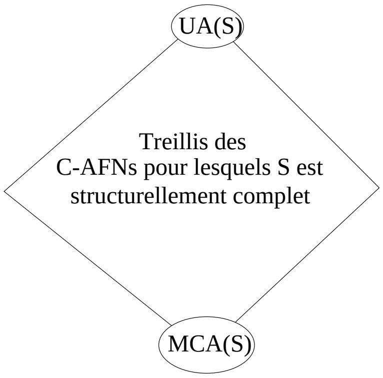


FIG. 2.5 – *Espace de recherche.*

Notons que, bien que l'espace soit de dimension finie, la taille de cet espace en fonction du nombre d'états N du $MCA(\mathcal{S})$ est $\Sigma_{i=1}^{N} S(N,i)$ où $S(N,i)$ est le nombre de Stirling de seconde espèce défini récursivement par :
$S(N,1) = 1, N \geq 1$,
$S(N,i) = 0$, si $i > N$,
$S(N,i) = iS(N-1,i) + S(N-1,i-1)$, pour $2 \leq i \leq N$.
ce qui ne permet pas d'envisager une exploration par énumération en pratique.

2.2.2 Déterminisme

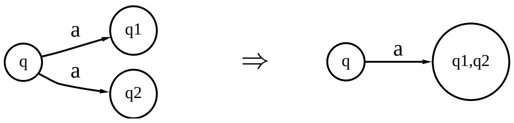


FIG. 2.6 – *Fusion pour déterminisation : pour chaque configuration du type de celle de gauche, les états q_1 et q_2 sont fusionnés (configuration de droite) pour obtenir un automate déterministe*

Pour considérer les automates classifieurs déterministes du treillis défini par $MCA(\mathcal{S})$, nous introduisons l'opération de *fusion pour déterminisation* qui consiste à fusionner les destinations des transitions indéterministes :

Définition 23 *La* fusion pour déterminisation *d'un k-FSA* $(\Sigma,\Gamma,Q,q_0,\delta,\rho)$ *est une procédure de fusion successive des états atteints par le même symbole à partir d'un même état :*

tant que $\exists q \in Q, \exists a \in \Sigma, \exists q_1,q_2 \in \delta(q,a), q_1 \neq q_2$ ***faire***
  fusionner(A,q_1,q_2)
fin tant que

□

La fusion pour déterminisation effectue une déterminisation dans le treillis : l'automate résultant des éventuelles fusions est la plus petite borne supérieure déterministe de l'automate original dans le treillis. Cependant, il faut garder à l'esprit que cet automate est éventuellement plus général que l'automate d'origine et que cette opération ne doit donc pas être confondue avec la déterminisation classique [AU72] qui produit une machine déterministe équivalente mais éventuellement hors du treillis.

En particulier, on peut considérer le résultat de cette opération sur le $MCA(\mathcal{S})$. L'automate résultant est appelé l'arbre accepteur des préfixes de $\mathcal{S}$ et peut être aussi défini de la façon suivante :

Définition 24 *L'arbre accepteur des préfixes de $\mathcal{S}$, noté $PTA(\mathcal{S})$*[3] *est le k-DFA quotient $MCA(\mathcal{S})/_\pi$ où la partition π est définie de sorte que les états acceptant les mêmes préfixes soient fusionnés :*

$$q_\pi = q'_\pi \Leftrightarrow Pref(q) = Pref(q')$$

où $Pref(q)$ est une notation désignant l'ensemble des mots permettant d'atteindre l'état q : $Pref(q) = \{u \in \Sigma^* | \delta(q_0,u) = q\}$ □

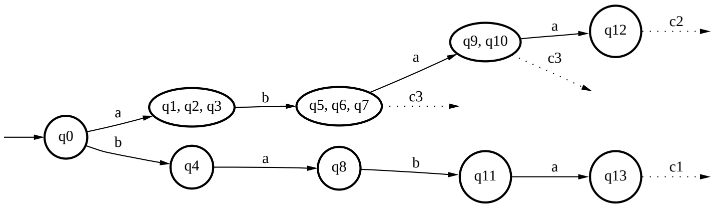


FIG. 2.7 – $PTA(\mathcal{S})$ *(Même échantillon d'apprentissage qu'en figure 2.3 pour $MCA(\mathcal{S})$.)*

L'arbre accepteur des préfixes est par construction déterministe et accepte exactement $\mathcal{S}$. Comme il est obtenu par dérivation de $MCA(\mathcal{S})$, la propriété d'inclusion des treillis peut être énoncée :

Propriété 1 *$Lat(PTA(\mathcal{S})) \subseteq Lat(MCA(\mathcal{S}))$.* □

Une autre propriété intéressante est que l'ensemble des automates classifieurs déterministes du treillis de $MCA(\mathcal{S})$ est inclus dans le treillis de $PTA(\mathcal{S})$. Ainsi, si l'on se limite à la recherche d'automates classifieurs déterministes, l'espace de recherche peut être restreint à $Lat(PTA(\mathcal{S}))$:

Théorème 2 *Soit $\mathcal{S}$ un échantillon d'un quelconque ensemble de langages réguliers $\mathcal{L}$ et soit A un k-DFA acceptant exactement $\mathcal{L}$. Si $\mathcal{S}$ est structurellement complet relativement à A, alors A appartient à $Lat(PTA(\mathcal{S}))$.* □

3. De la terminologie anglaise *Prefix Tree Acceptor*

Preuve Le même type de preuve que pour le théorème 1 peut être utilisé, en considérant la propriété supplémentaire, dûe au déterminisme de A, qu'il n'existe qu'une seule acceptation de $\mathcal{S}$ à partir de laquelle identifier les états du $PTA(\mathcal{S})$. $\square$

Ce théorème permet également de spécifier également l'espace de recherche des automates classifieurs canoniques sous l'hypothèse de la complétude structurelle. L'automate classifieur canonique étant par définition le plus petit automate classifieur déterministe, il est également inclus dans $Lat(PTA(\mathcal{S}))$.

Corollaire 1 *Soit $\mathcal{S}$ un échantillon d'un quelconque ensemble de langages réguliers $\mathcal{L}$ et soit $A(\mathcal{L})$ le k-DFA canonique de $\mathcal{L}$. Si $\mathcal{S}$ est structurellement complet relativement à $A(\mathcal{L})$, alors $A(\mathcal{L})$ appartient à $Lat(PTA(\mathcal{S}))$.* $\square$

Remarquons qu'obtenir l'automate classifieur canonique d'une solution déterministe dans le treillis est facile. L'automate étant déjà déterministe, il suffit de le minimiser. La minimisation d'un automate peut être effectuée classiquement par calcul des classes d'équivalence des états, puis par dérivation selon la partition obtenue [AU72]. L'automate canonique appartient alors aussi au treillis. Plus généralement l'automate canonique peut être obtenu par fusion d'états à partir de tout automate déterministe du treillis acceptant le même ensemble de langages. On peut aussi en déduire que tout automate déterministe, tel que toute fusion d'une paire de ses états augmente le langage reconnu, est canonique.

Bien que l'ensemble des k-DFA soit inclus dans $Lat(PTA(\mathcal{S}))$, des automates classifieurs non déterministes sont encore inclus dans ce treillis. En général, pour les éviter, l'opération de fusion utilisée est l'opération de *fusion déterministe* de RPNI [OG92] qui consiste en une fusion classique suivie d'une fusion pour déterminisation qui peut alors être concentrée sur les successeurs des états fusionnés. Cependant, une approche alternative est suggérée par le théorème suivant et son corollaire :

Théorème 3 *Soit $\mathcal{S}$ un échantillon d'un ensemble de langages réguliers $\mathcal{L}$ et soit $A(\mathcal{L})$ le k-DFA canonique de $\mathcal{L}$. Si $\mathcal{S}$ est structurellement complet relativement à $A(\mathcal{L})$, alors il existe une séquence de k-DFA telle que :*
$PTA(\mathcal{S}) = PTA(\mathcal{S})/\pi_0 \preceq PTA(\mathcal{S})/\pi_1 \preceq \cdots \preceq PTA(\mathcal{S})/\pi_n = A(\mathcal{L})$ *avec* $n \leq |Q_{PTA(\mathcal{S})}| - 1$. $\square$

Preuve La preuve de ce théorème proposée par Dupont [Dup96] consiste à ordonner les fusions à effectuer pour effectuer en premier celles impliquant les états les plus loin de la racine. Cette preuve est indépendante de la notion d'acceptation, excepté l'utilisation du thèorème 2, ces résultats sont toujours valides pour les automates classifieurs. $\square$

Corollaire 2 *Tout k-DFA $A \in Lat(PTA(\mathcal{S}))$ peut être obtenu par fusions successives en suivant une certaine séquence de k-DFA quotient de $PTA(\mathcal{S})$.* $\square$

L'exploration de l'espace de recherche des k-DFA pour lesquels $\mathcal{S}$ est structurellement complet peut donc être effectuée à partir du $PTA(\mathcal{S})$ en ne considérant que les fusions de paires d'états donnant une machine déterministe. Nous caractérisons ces paires d'états dans la section 2.4.1.

2.2.3 Univocité

L'application de la propriété d'inclusion des langages lors de la dérivation d'automates [FB75] à chaque langage d'un k-FSA permet d'obtenir la propriété suivante :

Propriété 2 *Si un k-FSA A' reconnaissant l'ensemble de langages $\{L'_k\}_{k\in\Gamma}$ dérive d'un k-FSA A reconnaissant $\{L_k\}_{k\in\Gamma}$, alors $\forall k \in \Gamma, L_k \subseteq L'_k$.* □

La dérivation d'un automate classifieur induit donc également une relation d'ordre partiel sur les généralisations dont on peut déduire la proposition suivante :

Proposition 1 *Soient deux automates classifieurs A et A' tels que A' dérive de A ($A \ll A'$)*

- *Si A est ambigu, alors A' est ambigu.*
- *Si A' est univoque, alors A est univoque.*

Par conséquent, dans un treillis de k-FSA, on peut délimiter l'ensemble des k-FSA univoques par un ensemble frontière BS[4] [MdG94] tel que tout k-FSA du treillis dérivé de l'un des éléments de la frontière est ambigu. Nous rappelons la notion d'anti-chaîne avant de donner la définition formelle de BS.

Définition 25 *Une* antichaîne $\overline{AC}$ *dans un treillis est un ensemble d'éléments incomparables du treillis. Une antichaîne est dite* maximale *si elle n'est contenue strictement dans aucune autre.* □

Définition 26 *Soit un treillis de k-FSA $Lat(A)$. L'*ensemble frontière BS_A *est l'antichaîne maximale composée de k-FSA univoques du treillis telle que tout k-FSA dérivé d'un élément de l'antichaîne est ambigu.* □

Si l'on considère le treillis basé sur $MCA(\mathcal{S})$, l'ensemble frontière $BS_{MCA(\mathcal{S})}$ constitue la limite de la généralisation possible des k-FSA compatibles pour lesquels $\mathcal{S}$ est structurellement complet. En ce sens, l'ensemble frontière est l'ensemble des *généralisations les plus générales G*, conformément à la terminologie propre à l'*espace des versions* [Mit78], qui est dans notre cas le treillis $Lat(MCA(\mathcal{S}))$. L'*ensemble des généralisations les plus spécifiques S* est réduit à $MCA(\mathcal{S})$. L'ensemble des k-FSA compatibles est donc caractérisé implicitement par la donnée de l'ensemble frontière BS (figure 2.8).

4. De la terminologie anglaise *Border Set*

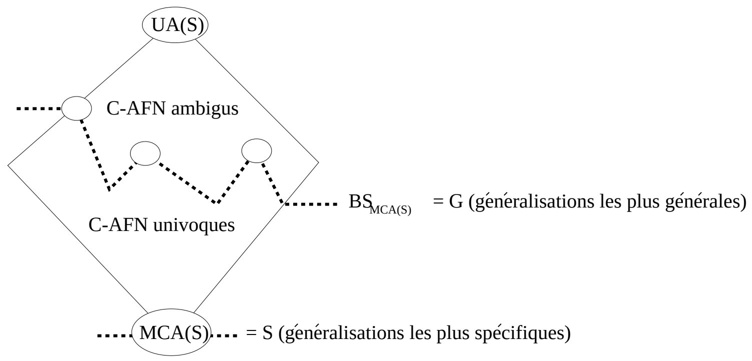


FIG. 2.8 – *Espace de recherche des k-FSA univoques.*

Dans le cadre de la recherche de k-DFA, l'ensemble frontière déterministe $BS_d(\mathcal{S})$ peut également être introduit :

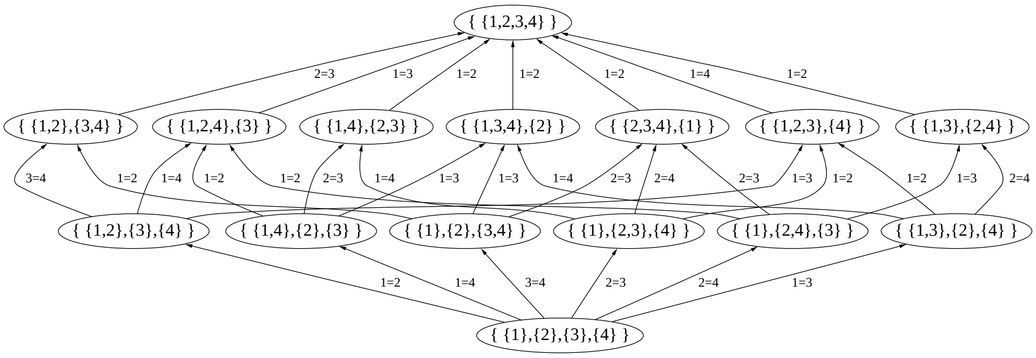
Définition 27 *L'*ensemble frontière déterministe $BS_d(\mathcal{S})$ *est l'antichaîne composée de k-DFA univoques du treillis* $Lat(PTA(\mathcal{S}))$ *telle que tout k-DFA dérivé d'un élément de l'antichaîne est ambigu.* □

Cependant, même pour le cas déterministe, la taille de l'ensemble frontière BS rend pratiquement impossible la gestion de l'ensemble de ses éléments pour des problèmes réalistes. Pour contourner le problème de la taille de l'espace de recherche, plusieurs auteurs proposent de travailler sur des représentations implicites de l'espace des solutions [Hir92, Nic93, Seb94]. Dans cet esprit, bien que les représentations attributs/valeurs soient pour cet espace très différentes de celles traitées, nous introduisons dans la section suivante une représentation alternative implicite des k-FSA univoques pour l'inférence k-régulière, sous l'hypothèse de la complétude structurelle. Cette approche basée, sur les fusions de paires d'états, nous permet d'étudier la dynamique des choix possibles lors de l'exploration de l'espace et permet la caractérisation de l'ensemble des solutions par un ensemble de contraintes sur les paires d'états.

2.3 Espaces de recherches implicites

2.3.1 Espace des partitions

D'après les théorèmes 1 et 2, l'hypothèse de la complétude structurelle de l'échantillon d'apprentissage $\mathcal{S}$ permet de limiter l'espace de recherche au treillis des k-FSA dérivés d'un k-FSA A_0 tel que $A_0 = MCA(\mathcal{S})$ pour la recherche de k-FSA et $A_0 = PTA(\mathcal{S})$ pour la recherche de k-DFA. L'espace de recherche induit par un échantillon d'apprentissage $\mathcal{S}$ peut donc être représenté implicitement par l'élément nul du treillis correspondant à l'échantillon d'apprentissage A_0 et l'ensemble des partitions des états de A_0. Cette représentation est particulièrement pratique et économique pour une exploration intensive du treillis. Nous appellerons *espace des partitions* cette représentation implicite.



Treillis des partitions pour un espace des partitions tels que A_0 est un k-FSA à 4 états.

FIG. 2.9 – *Espace des partitions*

2.3.2 Espace des fusions

Dans l'espace de recherche, chaque élément est représenté par une partition qui peut être atteinte à partir de l'élément nul du treillis A_0 par une succession de fusions de paires d'états qui est l'opération élémentaire de parcours du treillis. En changeant l'ordre de ces fusions ou en considérant la fusion d'autres paires d'états dans les blocs, des chemins différents peuvent permettre d'atteindre le même résultat. Pour étudier plus précisément les conséquences et les interactions de la fusion de paires d'états, nous proposons d'introduire une représentation plus fine de l'espace de recherche basée sur les fusions de paires d'états, appelée *espace des fusions de* A_0, telle que chaque paire d'états de A_0 est un attribut à deux valeurs : $\simeq$ si les états sont *fusionnés* dans le k-FSA dérivé, $\not\simeq$ si les états sont dans des blocs différents de la partition et on parlera alors d'états *différenciés*. Nous appelerons *affectation dans l'espace des fusions de l'automate* A_0[5], ou plus brièvement *affectation sur* A_0, un choix d'attributs pour des paires d'états de A_0 (voir figure 2.10).

Cette représentation permet de travailler sur des affectations incomplètes dans lesquelles l'attribut pour une paire d'état n'est pas défini, ce qui est particulièrement utile pour spécifier de façon implicite un ensemble de k-FSA du treillis ou pour construire progressivement une affectation. Lorsqu'une valeur a été attribuée à toutes les paires d'états, l'affectation est dite *complète*.

5. Nous ne considérerons pour les affectations que des automates classifieurs dont l'ensemble des états est atteignable à partir de l'état initial par la fonction de transition. D'éventuels états non atteignables étant inutiles, une phase de suppression de ces états sera implicitement supposée.

Une affectation sur $A_0 = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ définit les relations $\simeq$ et $\not\simeq$ dans $Q \times Q$. Les paires étant par définition non ordonnées, les relations induites par une affectation $\simeq$ et $\not\simeq$ sont symétriques. On posera initialement que pour tout q de Q, $q \simeq q$ (un état est fusionné avec lui même). Pour l'exploration de l'espace des automates classifieurs dérivés de A_0, on pourra alors limiter l'exploration de l'espace des fusions de A_0 aux affectations définissant une partition des états de Q de A_0 suivant la proposition suivante.

Proposition 2 *Une affectation complète sur $A_0 = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ définit une partition si et seulement si*

$$\forall q_1,q_2,q_3 \in Q,\ q_1 \simeq q_2 \wedge q_2 \simeq q_3 \Rightarrow q_1 \simeq q_3$$

Preuve Pour définir une partition, la relation $\simeq$ doit être une relation d'équivalence. La relation $\simeq$ est déjà réflexive et symétrique, elle doit être de plus transitive. C'est ce que stipule la proposition. □

On parlera d'*affectation transitive* lorsque la relation de la proposition 2 est vérifiée. Pour explorer la partie de l'espace des fusions correspondant au treillis, on pourra se limiter aux affectations transitives en propageant les fusions selon cette relation pour chaque nouveau choix de fusion.

(a)	q_1	q_2	q_3	q_4
q_1	$(\simeq)$	$\not\simeq$	$\simeq$	?
q_2		$(\simeq)$	$\not\simeq$	$\not\simeq$
q_3			$(\simeq)$	?
q_4				$(\simeq)$

(b)	q_1	q_2	q_3	q_4
q_1	$(\simeq)$	$\not\simeq$	$\simeq$	$\not\simeq$
q_2		$(\simeq)$	$\not\simeq$	$\not\simeq$
q_3			$(\simeq)$	$\simeq$
q_4				$(\simeq)$

(c)	q_1	q_2	q_3	q_4
q_1	$(\simeq)$	$\not\simeq$	$\simeq$	$\simeq$
q_2		$(\simeq)$	$\not\simeq$	$\not\simeq$
q_3			$(\simeq)$	$\simeq$
q_4				$(\simeq)$

Affectation incomplète (a), complète (b) et complète transitive (c).

FIG. 2.10 – *Affectations.*

Pour faire le lien entre les propriétés dans l'espace de recherche sous forme de treillis et celles dans l'espace des fusions, nous introduisons la notation $B_{\simeq}(q)$ pour désigner l'ensemble des états q' tels que $q \simeq q'$ et les fonctions $\delta_{\simeq}$ et $\rho_{\simeq}$ étendant respectivement la fonction de transition δ et la fonction d'émission finale ρ de A_0 par la relation $\simeq$ (exemple en figure 2.11) :

$$\forall q \in Q, \forall a \in \Sigma,\ \delta_{\simeq}(q,a) = \{q'_e \ / \ \exists q_e \in B_{\simeq}(q), q' \in \delta(q_e,a), q'_e \in B_{\simeq}(q')\}$$

$$\forall q \in Q,\ \rho_{\simeq}(q) = \bigcup_{q' \in B_{\simeq}(q)} \rho(q')$$

Nous pouvons alors vérifier que lorsque l'affectation est complète et transitive, $B_{\simeq}(q)$ est égal au bloc de q et les fonctions $\delta_{\simeq}$ et $\rho_{\simeq}$ définissent les fonctions δ' et ρ' de l'automate dérivé suivant la partition définie par l'affectation.

Proposition 3 *Si l'affectation définit une partition π , alors $\delta_{\simeq}$ et $\rho_{\simeq}$ sont respectivement équivalentes aux fonctions de transition δ' et de classification ρ' du k-FSA $(\Sigma,\Gamma,Q',q'_0,\delta',\rho')$ dérivé de $A_0 = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ par la partition π:*

$$\forall B \in \pi, \forall q \in B,\, B = B_{\simeq}(q)$$

$$\forall B \in \pi, \forall q \in B, \forall a \in \Sigma,\, \delta'(B,a) = \delta_{\simeq}(q,a),$$
$$\forall B \in \pi, \forall q \in B,\, \rho'(B) = \rho_{\simeq}(q)$$

Preuve Remarquons que lorsque l'affectation définit une partition π, la transitivité est vérifiée et on a donc pour tout états q_1 et q_2 d'un bloc B de π :

$$B_{\simeq}(q_1) = B_{\simeq}(q_2) = B$$

$$\delta_{\simeq}(q_1,a) = \delta_{\simeq}(q_2,a)$$
$$\rho_{\simeq}(q_1) = \rho_{\simeq}(q_2)$$

La première relation est donc facilement vérifiée. De plus, la définition de $\delta_{\simeq}$ est :

$$\forall q \in Q, \forall a \in \Sigma,\, \delta_{\simeq}(q,a) = \{B_{\simeq}(q') \;/\; \exists q_e \in B_{\simeq}(q), q' \in \delta(q_e,a)\}$$

soit, en remplaçant la quantification sur tous les états par une quantification sur tous les états de tous les blocs :

$$\begin{aligned} \forall B \in \pi, \forall q \in B, \forall a \in \Sigma,\, \delta_{\simeq}(q,a) &= \{B_{\simeq}(q') \;/\; \exists q_e \in B, q' \in \delta(q_e,a)\} \\ &= \{B' \in \pi \;/\; \exists q_e \in B, \exists q' \in B',\, q' \in \delta(q_e,a)\} \\ &= \delta'(B,a) \end{aligned}$$

de même, la définition de $\rho_{\simeq}$ est :

$$\forall q \in Q,\, \rho_{\simeq}(q) = \bigcup_{q' \in B_{\simeq}(q)} \rho(q')$$

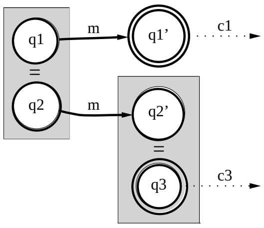
est équivalente à :

$$\begin{aligned} \forall B \in \pi, \forall q \in B, \rho_{\simeq}(q) &= \bigcup_{q' \in B} \rho(q') \\ &= \rho'(B) \end{aligned}$$

□

Nous étendons classiquement les fonctions de transition et de classification de l'espace des fusions aux mots :
$\forall q \in Q, \forall w \in \Sigma^*, \forall a \in \Sigma$,

$$\begin{aligned} \delta_{\simeq}(q,\epsilon) &= B_{\simeq}(q), \\ \delta_{\simeq}(q,wa) &= \bigcup_{q' \in \delta_{\simeq}(q,w)} \delta_{\simeq}(q',a) \end{aligned}$$



Pour cette affectation ($q_1 \simeq q_2$ et $q'_2 \simeq q_3$), $\delta_{\simeq}(q_1,m) = \{q'_1,q'_2,q_3\}$ et $\rho_{\simeq}(q'2) = \{c3\}$.

FIG. 2.11 – *Exemple de fonction $\delta_{\simeq}$ et $\rho_{\simeq}$*

On peut alors définir la fonction de classification $\gamma_{\simeq}$ de $\Sigma^* \times Q$ dans $\mathcal{P}(\Gamma)$ par :

$$\gamma_{\simeq}(q,w) = \bigcup_{q' \in \delta_{\simeq}(q,w)} \rho_{\simeq}(q')$$

Remarquons que cette définition, de par la définition de $\delta_{\simeq}$ et $\rho_{\simeq}$, peut se simplifier en :

$$\gamma_{\simeq}(q,w) = \bigcup_{q' \in \delta_{\simeq}(q,w)} \rho_(q')$$

En accord avec la proposition 3, il est facile de vérifier que si l'affectation définit une partition, les fonctions $\delta_{\simeq}$ et $\rho_{\simeq}$ coïncidant avec les fonctions de l'automate dérivé, la classification réalisée par la fonction $\gamma_{\simeq}$ est identique à celle réalisée par l'automate dérivé. Les propriétés des fonctions de transition et de classification d'un automate classifieur de l'espace de recherche dérivent des propriétés de leur contre-partie dans l'espace des fusions. Nous pouvons alors étudier à présent les liens entre la fusion de paires d'états et certaines propriétés de l'automate classifieur. Nous nous intéresserons plus particulièrement aux propriétés de déterminisme et d'univocité.

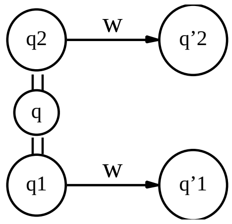
2.4 Caractérisations dans l'espace des fusions

2.4.1 Déterminisme

Dans l'espace des fusions, la proposition suivante spécifie la condition nécessaire et suffisante sur les fusions à effectuer pour que la fonction $\delta_{\simeq}$ d'une affectation sur un automate $A_0 = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ soit déterministe, assurant ainsi également le déterminisme des automates dérivés potentiels.

Proposition 4 *La fonction $\delta_{\simeq}$ d'une affectation sur un k-FSA $A_0 = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ est déterministe si et seulement si :*

$$\forall q,q_1,q_2 \in Q, \forall a \in \Sigma, \forall q'_1 \in \delta(q_1,a), \forall q'_2 \in \delta(q_2,a) \;:\; q_1,q_2 \in B_{\simeq}(q) \Rightarrow q'_1 \simeq q'_2$$



$$q \simeq q_1 \wedge q \simeq q_2 \Rightarrow q'1 \simeq q'2$$

FIG. 2.12 – *Caractérisation du déterminisme.*

Preuve La proposition peut être vue comme une traduction de la fusion déterministe utilisée dans RPNI [OG92] et peut être démontrée en partant de la définition du déterminisme de $\delta_{\simeq}$:
$\forall q \in Q, \forall a \in \Sigma, \mid \delta_{\simeq}(q,a) \mid \leq 1$
$\Leftrightarrow \forall q \in Q, \forall a \in \Sigma, \mid \{B_{\simeq}(q') \mid \exists q_e \in B_{\simeq}(q), q' \in \delta(q_e,a)\} \mid \leq 1$
$\Leftrightarrow \forall q \in Q, \forall a \in \Sigma, \forall q_1,q_2 \in Q, q_1 \simeq q, q_2 \simeq q, \forall q'_1 \in \delta(q_1,a), \forall q'_2 \in \delta(q_2,a), q'_1 \simeq q'_2$
$\Leftrightarrow \forall q \in Q, \forall a \in \Sigma, \forall q_1,q_2 \in B_{\simeq}(q), \forall q'_1 \in \delta(q_1,a), \forall q'_2 \in \delta(q_2,a), q'_1 \simeq q'_2$ □

Si l'on considère le treillis de $MCA(\mathcal{S})$, la relation d'implication de la proposition 4 détermine un ensemble initial de fusions d'états définissant, en général, une affectation incomplète. Si l'on complète cette affectation en choisissant la valeur $\not\simeq$ pour tous les attributs non définis, l'automate classifieur obtenu est le $PTA(\mathcal{S})$. L'ensemble des automates classifieurs potentiellement dérivés en essayant toutes les complétions transitives possibles de l'affectation initiale est le treillis $Lat(PTA(\mathcal{S}))$. Par conséquent, lorsque l'on considère la recherche d'affectations telles que la fonction $\delta_{\simeq}$ soit déterministe, il est plus économique en termes de nombres d'états et transitions manipulées de considérer dès le début l'espace de fusion basé sur $PTA(\mathcal{S})$. Les résultats sur l'espace des fusions basé sur $MCA(\mathcal{S})$ s'appliquant également à celui basé sur $PTA(\mathcal{S})$, nous continuerons cependant à parler de l'espace des fusions de $MCA(\mathcal{S})$ pour désigner l'espace des fusions de $MCA(\mathcal{S})$ ou de $PTA(\mathcal{S})$.

Pour les automates dérivés, la transitivité étant respectée, la proposition 4 précédente peut s'exprimer plus simplement :

Théorème 4 *Si l'affectation définit une partition π sur les états d'un automate $A_0 = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$, alors $\delta_{\simeq}$ (et par conséquent la fonction de transition δ' de l'automate dérivé) est déterministe si et seulement si :*

$$\forall q_1,q_2 \in Q, \forall a \in \Sigma, \forall q'_1 \in \delta(q_1,a), \forall q'_2 \in \delta(q_2,a),\ q_1 \simeq q_2 \Rightarrow q'_1 \simeq q'_2$$

□

Preuve Ce théorème se déduit facilement de la proposition 4 en remarquant, la transitivité étant vérifiée, que

$$\forall q, q_1, q_2 \in Q, q_1, q_2 \in B_{\simeq}(q)$$

représente le même ensemble de paires d'états q_1, q_2 que :

$$\forall q_1, q_2 \in Q, q_1 \simeq q_2$$

□

Pour un espace de fusion basé sur $MCA(\mathcal{S})$ ou $PTA(\mathcal{S})$, la structure d'arbre de l'élément nul du treillis permet de vérifier que la relation $\Rightarrow$ engendre une relation d'ordre partiel entre couples d'états. Nous noterons $\Rightarrow^*$, la fermeture transitive de $\Rightarrow$. Les deux types de parcours des k-DFA dans le treillis présentés en section 2.2.2 peuvent être caractérisés par rapport à cette relation :

- La première approche utilisant la fusion déterministe de RPNI est une approche de la gauche de la relation d'implication vers la droite de celle-ci : deux états q_1 et q_2 sont choisis puis fusionnés en fixant $q_1 \simeq q_2$ puis en fusionnant toutes les paires d'états impliquées à partir de $q_1 \simeq q_2$ par la relation $\Rightarrow^*$.
- L'approche ne considérant que des k-DFA à chaque fusion d'une paire d'états consiste à ne fusionner que des paires d'états pour lesquels la partie droite de $\Rightarrow^*$ est vide ou déjà fusionnée, c'est à dire n'impliquant aucune autre fusion.

Par exemple, pour le PTA présenté dans la figure 2.13, le déterminisme est exprimé par $q_s \simeq q_2 \Rightarrow q_1 \simeq q_3$. Dans une approche utilisant la fusion pour déterminisation comme dans RPNI, on fusionnera q_s et q_2 et donc q_1 et $q3$ pour obtenir le premier automate déterministe dérivé dans le treillis, puis seulement q_1 et q_3 pour obtenir le second automate déterministe dérivé. À l'opposé, pour ne considérer que des k-DFA par des fusions à deux états, la fusion de q_1 et q_3 doit être considérée avant la fusion q_s et q_2, on obtient alors le premier automate déterministe dérivé en fusionnant q_1 et $q3$ puis le deuxième en fusionnant en plus q_s et q_2.

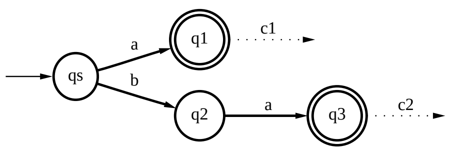


FIG. 2.13 – *Ordre partiel sur les paires d'états pour une stratégie prudente.*

Enfin, remarquons que pour l'exploration exhaustive de l'espace des k-DFA, la première approche utilisant la fusion déterministe, que l'on pourrait qualifier d'agressive, effectue deux fois la fusion $q_1 \simeq q_3$ (une fois pour la déterminisation et une fois pour atteindre le deuxième automate) alors que cette situation est impossible avec la deuxième approche, plus progressive, qui est alors optimale en nombre de fusions pour ce type de recherche.

2.4.2 Univocité

Un automate classifieur univoque classe chaque mot dans au plus une classe. Pour chaque mot, il faut donc, pour assurer l'univocité de la classification, que tous les états atteints depuis l'état initial soient : soit étiquetés par une seule classe, soit de classe indéfinie. Pour formaliser cette dernière notion, nous introduisons la notion de *classification compatible*, notée $C_1 \sim C_2$ définie par : $\forall C_1,C_2 \in \mathcal{P}(\Gamma),(C_1 \sim C_2) \Leftrightarrow (C_1 = C_2 \vee C_1 = \emptyset \vee C_2 = \emptyset)$ Sinon, les deux classification seront dites *incompatibles*, noté $C_1 \not\sim C_2$.

On peut alors exprimer dans l'espace des fusions, l'univocité d'un automate classifieur dérivé, corrélée à celle de la fonction de classification $\gamma_\simeq$:

Proposition 5 *La fonction $\gamma_\simeq$ sur les mots d'une affectation sur un k-FSA $A_0 = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ est univoque si et seulement si :*

$$\begin{cases} \forall q \in Q,\exists k \in \Gamma,\ \rho(q) = \{k\} \vee \rho(q) = \emptyset & (1) \quad \textit{unicité} \\ \forall q \in Q,\forall w \in \Sigma^*,\forall q'_1,q'_2 \in \delta_\simeq(q,w),\ \rho(q'_1) \sim \rho(q'_2) & (2) \quad \textit{compatibilité} \end{cases}$$

Preuve Par définition, $\gamma_\simeq$ univoque $\Leftrightarrow \forall w \in \Sigma^* \ : \ |\gamma_\simeq(q_0,w)\ | \ \leq 1$
$\Leftrightarrow \forall q \in Q,\forall w \in \Sigma^* \ : \ |\gamma_\simeq(q,w)\ | \ \leq 1$
$\Leftrightarrow \forall q \in Q,\forall w \in \Sigma^*,\exists k \in \Gamma,\ \gamma_\simeq(q,w) = \{k\} \ \vee \ \gamma_\simeq(q,w) = \emptyset$
$\Leftrightarrow \forall q \in Q,\forall w \in \Sigma^*,\exists k \in \Gamma,\forall q' \in \delta_\simeq(q,w),\ \rho(q') = \{k\} \ \vee \ \rho(q') = \emptyset$

Ce qui peut être décomposé en deux contraintes : une émission finale unique pour chaque état atteint et une émission compatible pour tous les états atteints par la même séquence :

$$\begin{cases} \forall q \in Q,\forall w \in \Sigma^*,\forall q' \in \delta_\simeq(q,w),\exists k \in \Gamma,\ \rho(q') = \{k\} \vee \rho(q') = \emptyset \\ \forall q \in Q,\forall w \in \Sigma^*,\forall q'_1,q'_2 \in \delta_\simeq(q,w),\ \rho(q'_1) = \rho(q'_2) \vee \rho(q'_1) = \emptyset \vee \rho(q'_2) = \emptyset \end{cases}$$

En supposant que tous les états de A_0 soient atteignables, la première relation est équivalente à $\forall q \in Q,\exists k \in \Gamma,\rho(q) = \{k\} \vee \rho(q) = \emptyset$. La deuxième relation peut être réécrite en utilisant la notation de compatibilité sur les classifications. On obtient donc le système d'équations de la proposition. □

Notons que si l'exemple d'apprentissage $\mathcal{S}$ comprend un mot classé dans deux classes differentes, alors $MCA(\mathcal{S})$ est ambigu et par conséquent tous les automates classifieurs dérivés également. L'ensemble des solutions univoques pour lesquelles $\mathcal{S}$ est structurellement complet est alors trivialement vide. Nous supposerons par la suite que chaque mot de l'échantillon d'apprentissage n'est classé que dans une seule classe ($\forall S_i,S_j \in \mathcal{S},i \neq j,S_i \cap S_j = \emptyset$) et que par conséquent $MCA(\mathcal{S})$ est univoque. Pour chaque état de $MCA(\mathcal{S})$, on a donc la propriété d'unicité d'émission finale : $\forall q \in Q,\exists k \in \Gamma,\rho(q) = \{k\} \vee \rho(q) = \emptyset$. Par conséquent, la relation $\forall q_1,q_2 \in Q,q_1 \simeq q_2,\rho(q_1) \sim \rho(q_2)$ implique la relation (1) du système d'équations. Comme cette relation est incluse dans la formulation de la relation (2) du système avec w égal au mot vide ϵ, le système peut être réduit à la relation de compatibilité uniquement. Nous pouvons alors énoncer l'univocité de la fonction de classification uniquement en termes de compatibilité :

Corollaire 3 *Si $A_0 = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ est univoque, alors la fonction $\gamma_{\simeq}$ sur les mots dans l'espace des fusions de A_0 est univoque si et seulement si :*

$$\forall q \in Q, \forall w \in \Sigma^*, \forall q_1', q_2' \in \delta_{\simeq}(q,w),\ \rho(q_1') \sim \rho(q_2')$$

□

Ce corollaire permet d'identifier l'*ensemble des fusions incompatibles* $\mathcal{F}_I$, i.e l'ensemble des paires d'états telles que tout automate obtenu par une dérivation contenant la fusion de l'une de ces paires est ambigu :

$$\mathcal{F}_I = \{\{q_1,q_2\} \ / \ q_1 \simeq q_2 \Rightarrow \exists w \in \Sigma^*,\ \exists (q_1',q_2') \in \delta_{\simeq}(q_1,w) \times \delta_{\simeq}(q_2,w),\ \rho(q_1') \not\sim \rho(q_2')\}$$

Insistons ici sur le fait que le choix $q_1 \simeq q_2$ dans l'affectation modifie l'affectation courante et par conséquent aussi la fonction $\delta_{\simeq}$ considérée. Calculer l'ensemble $\mathcal{F}_I$ revient alors à effectuer chacune des fusions pour vérifier si elles sont incompatibles.

En ne considérant que les fonctions de transition et de classification de l'affectation courante sans modification, on peut introduire l'ensemble des *états incompatibles*, noté $\mathcal{E}_I$, défini par :

$$\begin{aligned} \mathcal{E}_I &= \{\{q_1,q_2\} \ / \ \exists w \in \Sigma^*,\ \exists q_1',q_2' \in \delta_{\simeq}(q_1,w) \times \delta_{\simeq}(q_2,w),\ \rho(q_1') \not\sim \rho(q_2')\} \\ &= \{\{q_1,q_2\} \ / \ \exists w \in \Sigma^*,\ \gamma_{\simeq}(q_1,w) \not\sim \gamma_{\simeq}(q_2,w)\} \end{aligned}$$

La définition de $\mathcal{E}_I$ diffère de celle de $\mathcal{F}_I$ uniquement par l'absence de la fusion préalable des deux états, ce qui permet de considérer les fonctions de transition $\delta_{\simeq}$ et de classification $\gamma_{\simeq}$ de l'affectation courante plutôt que chaque modification de celles-ci par chaque fusion.

L'ensemble $\mathcal{E}_I$ contient l'ensemble des paires d'états permettant d'atteindre dans l'affectation courante deux états d'émission définies et différentes par le même mot. L'ensemble $\mathcal{E}_I$ permet donc :

- d'une part, de détecter si l'affectation courante est ambigüe (si une paire d'états fusionnée dans l'affectation est détectée incompatible) et,
- d'autre part, de définir un ensemble de paires d'états à ne pas fusionner pour l'inférence d'automates classifieurs univoques, qui constitue en pratique une bonne approximation de $\mathcal{F}_I$.

Pour être plus précis, la différence entre les ensembles $\mathcal{E}_I$ et $\mathcal{F}_I$ est constituée des paires d'états n'appartenant pas à $\mathcal{E}_I$ et dont la fusion implique de nouvelles paires d'états dans $\mathcal{E}_I$ telles que l'une d'elles est déjà fusionnée.

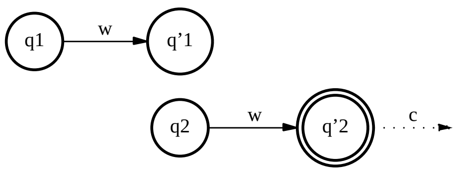
La définition de $\mathcal{E}_I$ peut être rapprochée de la définition de l'ensemble $\mathcal{E}_D$ des états distinguables d'un automate classifieur (défini par opposition aux états équivalents, voir aussi note ci-dessous) :

$$\mathcal{E}_D = \{\{q_1,q_2\} \ / \ \exists w \in \Sigma^*, \gamma_{\simeq}(q_1,w) \neq \gamma_{\simeq}(q_2,w)\}$$

L'ensemble des états incompatibles peut être calculé de façon similaire à l'ensemble des états distinguables. Comme la k-distingabilité des états, on peut définir la k-incompatibilité des états. La 0-incompatibilité d'une paire d'états $\{q,q'\}$ est définie par $\rho(q) \not\sim \rho(q')$ au lieu de $\rho(q) \neq \rho(q')$ pour la 0-distingablité. La propagation de l'incompatibilité pour k croissant est ensuite identique à celle de la distingabilité des états. On peut alors adapter facilement l'algorithme de détection des paires d'états distinguables présentés par Hopcroft et Ullman (fig 3.8, p78 [HU80]) à la détection des paires d'états incompatibles d'un automate classifieur déterministe (algorithme 2.1).

Cet algorithme de minimisation d'automates possède la particularité par rapport aux algorithmes classiques de minimisation de la littérature [Wat94] de procéder par construction de l'ensemble des états distinguables, ce qui implique, par complémentarité, les classes d'équivalence. Remarquons que les approches par construction directe des classes d'équivalence ne sont pas applicables ici, la compatibilité des états n'étant pas transitive et ne pouvant, par conséquent, pas être représentée par une partition.

Note sur la notion de distingabilité, incompatibilité et paradigme "don't care.



Dans la configuration ci-dessus, la paire d'états $\{q_1,q_2\}$ est distinguable alors qu'elle est compatible et que même la fusion de cette paire est compatible. Remarquons que si les états q'_1 et q'_2 sont fusionnés, alors pour l'affectation correspondante, les états q_1 et q_2 deviennent équivalents et ne sont donc plus distinguables. Si l'on se place dans le paradigme "don't care", considérant les états d'émission finale indéfinie comme des états pour lesquels la classe peut librement être fixée, la fusion de q'_1 et q'_2 revient à choisir pour l'état q'_1 d'émission finale indéfinie ("don't care"), l'émission $\rho(q'_1) = \{c\} = \rho(q'_2)$.
L'ensemble $\mathcal{E}_I$ peut ainsi être considéré comme l'ensemble des paires d'états distinguables dans le paradigme 'don't care". Dans notre cas, en respectant la complétude structurelle de l'échantillon d'apprentissage, l'affectation d'une émission à un état "don't care" ne pourra être effectuée que par la fusion avec un état dont la fonction d'émission finale est définie (bien qu'on puisse aussi imaginer, en considérant seulement l'hypothèse de complétude structurelle relative aux transitions de l'automate cible, la possibilité d'un parcours plus fin dans le treillis par fusion d'états et spécification d'émission finale d'états).

Prérequis M déterministe

États_Incompatibles(M = $(\Sigma,\Gamma,Q,q_0,\delta,\rho)$):
/* Recherche de l'ensemble des états incompatibles de M */
/* Initialisations */
$\mathcal{E}_I \leftarrow \emptyset$ /* ensemble des états incompatibles */
pour tout $\{q_i,q_j\} \in Q \times Q$ **faire**
    $L(q_i,q_j) \leftarrow \emptyset$ /* liste des paires d'états à ajouter à $\mathcal{E}_I$ si $\{q_i,q_j\} \in \mathcal{E}_I$ */
fin pour
/* 0-incompatibilité */
pour tout $\{q_i,q_j\} \in Q \times Q$, $\rho(q) \not\sim \rho(q_j)$ **faire**
    $\mathcal{E}_I \leftarrow \mathcal{E}_I \cup \{q_i,q_j\}$
fin pour
/* k-incompatibilité */
pour tout $\{q_i,q_j\} \in Q \times Q$, $\rho(q_i) \sim \rho(q_j)$ **faire**
    pour tout $a \in \Sigma$ **faire**
        si $\{\delta(q_i,a),\delta(q_j,a)\} \in Q \times Q$ **alors**
            si $\{\delta(q_i,a),\delta(q_j,a)\} \in \mathcal{E}_I$ **alors**
                $\mathcal{E}_I \leftarrow \mathcal{E}_I \cup \{q_i,q_j\}$
                Marque_Liste_Incompatible($L(q_i,q_j)$)
            sinon
                $L(\delta(q_i,a),\delta(q_i,a)) \leftarrow L(\delta(q_i,a),\delta(q_j,a)) \cup \{q_i,q_j\}$
            fin si
        fin si
    fin pour
fin pour
retourner $\mathcal{E}_I$

Marque_Liste_Incompatible(L):
pour tout $\{q_i,q_j\} \in L$ **faire**
    si $\{q_i,q_j\} \notin \mathcal{E}_I$ **alors**
        $\mathcal{E}_I \leftarrow \mathcal{E}_I \cup \{q_i,q_j\}$
        Marque_Liste_Incompatible($L(q_i,q_j)$)
    fin si
fin pour

ALG 2.1: États incompatibles d'un k-DFA (adaptation de Hopcroft et Ullman [HU80]).

```
États_Incompatibles_NDet(M = (Σ,Γ,Q,q_0,δ,ρ)):
/* Recherche de l'ensemble des états incompatibles de M */
/* Et détection de l'ambiguïté de M */
E_I ← ∅ /* ensemble des états incompatibles */
/* 0-incompatibilité */
pour tout {q_i,q_j} ∈ Q × Q, ρ(q_i) ≁ ρ(q_j) faire
  si {q_i,q_j} ∉ E_I alors
    Marque_Incompatible_Et_Propage_En_Arriere(q_i,q_j)
  fin si
fin pour
retourner E_I

Marque_Incompatible_Et_Propage_En_Arriere(q_1,q_2):
/* Détection Ambiguïté */
si q_1 = q_2 alors
  Retourner Exception("automate ambigu (état q_1)")
fin si
/* Mémorisation incompatibilité */
E_I ← E_I ∪ {q_1,q_2}
/* Propagation */
pour tout a ∈ Σ faire
  pour tout {p_1,p_2} ∈ δ^{-1}(q_1,a) × δ^{-1}(q_2,a) faire
    si {p_1,p_2} ∉ E_I alors
      Marque_Incompatible_Et_Propage_En_Arriere(p_1,p_2)
    fin si
  fin pour
fin pour
```

ALG 2.2: États incompatibles et ambiguïté d'un k-FSA.

La complexité de cet algorithme en fonction du nombre n d'états de l'automate classifieur et de la taille $|\Sigma|$ de l'alphabet est en $O(|\Sigma| n^2)$, chaque paire d'état ne pouvant être que dans au plus une liste, [HU80].

L'algorithme de Hopcroft et Ullman peut être adapté au traitement d'automates classifieurs non déterministes. Dans le cas non déterministe, si l'on dispose de la fonction inverse de la fonction de transition δ^{-1}, il n'y a plus de raisons de privilégier les transitions sortantes plutôt que les transitions entrantes. On peut alors se passer des listes préconisées dans le cas déterministe en utilisant un algorithme effectuant la propagation des incompatibilités "en arrière". L'algorithme 2.2 proposé permet de calculer l'ensemble des paires d'états incompatibles pour un k-FSA.

Cet algorithme permet également la détection, non triviale dans le cas non déterministe, de l'ambiguïté d'un k-FSA. En effet, l'ensemble des paires d'états incompatibles est l'ensemble des paires permettant d'atteindre par le même mot des classifications différentes. Par conséquent, si à partir du même état on peut atteindre deux classifications différentes par le même mot, alors l'automate est ambigu :

Proposition 6 *Un k-FSA est ambigu si et seulement si un état doit être marqué incompatible avec lui-même par l'algorithme 2.2.*

La complexité dans le pire des cas de l'algorithme 2.2 est en $O(|\Sigma|n^4)$: $O(n^2)$ appels à la fonction *Marque_Incompatible_Et_Propage_En_Arriere()*, dont le corps est en $O(|\Sigma|n^2)$. On peut néanmoins affiner ce résultat en fonction du nombre maximal t_a de transitions entrantes en un état par le même symbole. Le corps de *Marque_Incompatible_Et_Propage_En_Arriere()* est alors en $O(|\Sigma|t_a^2)$ et la complexité totale en $O(|\Sigma|t_a^2n^2)$, ce qui permet de situer plus pratiquement la complexité de l'algorithme entre $O(|\Sigma|n^2)$ et $O(|\Sigma|n^4)$ suivant la valeur de t_a.

L'algorithme 2.2 permet de détecter si un automate classifieur est ambigu. Il peut être utilisé à chaque pas de l'inférence pour vérifier que l'automate courant est univoque. Cependant, l'ensemble des paires incompatibles $\mathcal{E}_I$ croissant par l'opération de fusion, il est plus économique de de n'utiliser cet algorithme que pour l'initialisation de l'ensemble des paires d'états incompatibles et de maintenir de façon incrémentale cet ensemble par l'utilisation de fonctions de propagation locale des conséquences des fusions effectuées. Nous proposons d'étudier cette dynamique dans l'espace des fusions.

L'initialisation de l'espace des fusions d'un automate M peut être effectuée en ajoutant l'affectation de la valeur $\not\simeq$ aux paires d'états incompatibles détectées par l'algorithme 2.2. L'algorithme 2.3 présenté assure la détection de l'ensemble des paires d'états incompatibles et l'initialisation de l'affectation pour l'automate considéré.

Classiquement, en inférence k-régulière, l'automate initial est soit $MCA(\mathcal{S})$, soit $PTA(\mathcal{S})$. Dans ce cas, la structure d'arbre de ces deux automates assure qu'il n'existe qu'une seule transition entrante pour chaque état et le corps de la fonction *Affecte_Incompatible_et_Propage_En_Arrière()* peut alors être implémenté pour s'exécuter en temps constant. La complexité de l'algorithme 2.3 d'initialisation est alors en $O(n^2)$. Remarquons également que sous l'hypothèse que l'échantillon d'apprentissage est univoque et donc aussi l'automate initial, le test d'ambiguïté peut être omis.

La figure 2.14 montre un exemple de détection des états incompatibles sur $PTA(\mathcal{S})$ pouvant initialement être différenciés dans l'affectation.

Initialisation_États_Incompatibles_et_Affectation($M = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$):
$(\mathcal{E}_I,\mathcal{A}_{\simeq},\mathcal{A}_{\not\simeq}) \leftarrow (\emptyset,\emptyset,\emptyset)$ /* $\mathcal{E}_I$: ensemble des états incompatibles, $\mathcal{A}_{\simeq},\mathcal{A}_{\not\simeq}$: affectation */
/* Initialisation des fusions */
pour tout $q \in Q$ **faire**
    $\mathcal{A}_{\simeq} \leftarrow \mathcal{A}_{\simeq} \cup \{q,q\}$
fin pour
/* Initialisation des incompatibilités */
pour tout $\{q_i,q_j\} \in Q \times Q$, $\rho(q_i) \not\sim \rho(q_j)$ **faire**
    si $\{q_i,q_j\} \notin \mathcal{E}_I$ **alors**
        Affecte_Incompatible_et_Propage_En_Arrière(q_i,q_j)
    fin si
fin pour
retourner $(\mathcal{E}_I,\mathcal{A}_{\simeq},\mathcal{A}_{\not\simeq})$

Affecte_Incompatible_et_Propage_En_Arrière(q_1,q_2):
/* Ambiguïté */
si $\{q_1,q_2\} \in \mathcal{A}_{\simeq}$ **alors**
    Retourner Exception("Affectation automate ambigüe")
fin si
/* Mise à jour $\mathcal{E}_I$ et $\mathcal{A}_{\not\simeq}$ */
$\mathcal{E}_I \leftarrow \mathcal{E}_I \cup \{q_1,q_2\}$
$\mathcal{A}_{\not\simeq} \leftarrow \mathcal{A}_{\not\simeq} \cup \{q_1,q_2\}$
/* Propagation */
pour tout $a \in \Sigma$ **faire**
    pour tout $\{p_1,p_2\} \in \delta_{\simeq}^{-1}(q_1,a) \times \delta_{\simeq}^{-1}(q_2,a)$ **faire**
        si $\{p_1,p_2\} \notin \mathcal{E}_I$ **alors**
            Affecte_Incompatible_et_Propage_En_Arrière(p_1,p_2)
        fin si
    fin pour
fin pour

ALG 2.3: Initialisation de l'affectation.

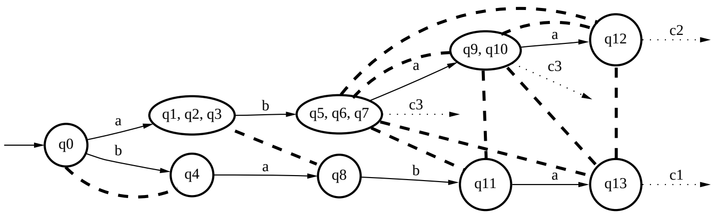
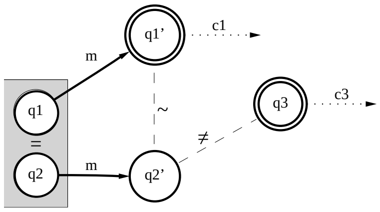


FIG. 2.14 – *Différenciations initiales $\mathcal{A}_{0\not\simeq}$ sur $PTA(\mathcal{S})$.*

Une fois l'affectation initialisée, l'ensemble des états incompatibles peut être maintenu pour chaque choix de valeur d'attributs. Le choix d'un différenciation ($\not\simeq$) ne change pas la fonction de transition et ne modifie donc pas l'ensemble des paires d'états incompatibles. Par contre par le choix de la fusion d'une paire d'états, de nouveaux chemins sont créés et de nouvelles paires d'états incompatibles peuvent apparaître. Il faut alors propager, suivant les nouveaux chemins créés, les incompatibilités de paires d'états déjà détectées. La fonction *Fusionne_et_Propage_Incompatibilités()* (algorithme 2.4) présente une proposition de stratégie de mise à jour de l'ensemble des paires d'états incompatibles suite à une fusion.

Fusionne_et_Propage_Incompatibilités(q_1,q_2):
$I_1 = \{q/\{q,q_1\} \in \mathcal{E}_I\}$ /* états incompatibles avec q_1 */
$I_2 = \{q/\{q,q_2\} \in \mathcal{E}_I\}$ /* états incompatibles avec q_2 */
/* Il est important de faire la fusion avant la propagation
pour que la propagation utilise le bonne fonction de transition $\delta_{\simeq}$*/
$\mathcal{A}_{\simeq} \leftarrow \mathcal{A}_{\simeq} \cup \{q_1,q_2\}$
/* Propagation */
pour tout $q_i \in I_1$ **faire**
  si $\{q_2,q_i\} \notin \mathcal{E}_I$ **alors**
    Affecte_Incompatible_Et_Propage_En_Arrière(q_2,q_i)
  fin si
fin pour
pour tout $q_j \in I_2$ **faire**
  si $\{q_1,q_j\} \notin \mathcal{E}_I$ **alors**
    Affecte_Incompatible_Et_Propage_En_Arrière(q_1,q_j)
  fin si
fin pour

ALG 2.4: Fusion et maintien de l'ensemble des paires d'états incompatibles



Conséquence de $q_1 \simeq q_2$ *:* $q_1 \simeq q_2 \Rightarrow \rho_{\simeq}(q_1') \sim \rho_{\simeq}(q_2') \Rightarrow q_2' \not\simeq q_3$

FIG. 2.15 – *Conséquence du choix d'une fusion*

Avant d'étudier plus particulièrement la recherche d'automates classifieurs déterministes univoques, nous esquissons ici une autre utilisation possible du corollaire 3 pour la recherche de k-FSA univoques. Nous avons vu que ce corollaire pouvait être utilisé pour détecter "en arrière" les paires d'états incompatibles. Ce corollaire peut aussi être utilisé pour propager "en avant" les conditions de compatibilité suite au choix d'une fusion.

Nous illustrons cette possibilité par un exemple sur la configuration présentée en figure 2.15. Pour une fusion $q_1 \simeq q_2$, une contrainte $\rho(q_1') \sim \rho(q_2')$ peut être ajoutée pour chaque q_1' et q_2' atteints depuis q_1 et q_2 par un même mot w de Σ^* par application du corollaire. De plus, pour chacune de ces nouvelles contraintes telles que la fonction d'émission finale est définie pour l'état q_1' alors qu'elle ne l'est pas pour q_2', l'ensemble des émissions finales potentielles de q_2' peut être limité à $\rho(q_1')$. Ceci peut être fait par filtrage des de l'ensemble des valeurs que peut prendre la fonction d'émission finale de l'état, ou en ajoutant à l'affectation pour chaque état q_3, pour lequel la fonction d'émission est définie et différente de celle de q_1', l'attribut $q_2' \not\simeq q_3$.

Dans le cas déterministe, cette propagation des compatibilités coïncide avec celle de la propagation des fusions pour déterminisation. Nous exploitons cette propriété dans la section suivante consacrée à l'inférence d'automates classifieurs déterministes univoques.

2.4.3 k-DFA univoques

En général, par souci d'efficacité d'utilisation ou d'unicité de représentation, on préfère utiliser les machines à états finies déterministes. En ajoutant la contrainte du déterminisme à la contrainte d'univocité, nous considérons à présent le problème de l'inférence d'un automate classifieur univoque et déterministe sous l'hypothèse que l'échantillon d'apprentissage est structurellement complet relativement à cet automate.

Dans ce cas, en conjugant les résultats obtenus dans les deux sections précédentes, et en remarquant que la propagation de la déterminisation et de la compatibilité peuvent être confondues, il est possible de proposer un système de contraintes simple caractérisant l'ensemble des solutions. Le système de contraintes obtenu permet de formuler sous la forme d'un *problème de satisfaction de contraintes* le problème classique de l'inférence d'automates classifieurs. Nous le désignerons par la suite sous l'appellation de *RI-CSP*[6].

Théorème 5 *(RI-CSP) Soit $\mathcal{S}$ un échantillon d'apprentissage d'un ensemble de langages réguliers univoque $\mathcal{L}$. L'ensemble des k-DFA univoques, pour lesquels $\mathcal{S}$ est structurellement complet, est l'ensemble des automates classifieurs dérivés de $PTA(\mathcal{S}) = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$ suivant les affectations complètes vérifiant le système de contraintes :*

$$\begin{cases} \forall q_1,q_2,q_3 \in Q \ : \ q_1 \simeq q_2 \wedge q_2 \simeq q_3 \Rightarrow q_1 \simeq q_3 & (1) \\ \forall q_1,q_2 \in Q \ : \ q_1 \simeq q_2 \Rightarrow \rho(q_1) \sim \rho(q_2) & (2) \\ \forall q_1,q_2,q_1',q_2' \in Q, \forall a \in \Sigma, q_1' \in \delta(q_1,a), q_2' \in \delta(q_2,a) \ : \ q_1 \simeq q_2 \Rightarrow q_1' \simeq q_2' & (3) \end{cases}$$

□

Preuve D'après le théorème 2, l'ensemble des solutions est inclus dans le treillis de $PTA(\mathcal{S})$ et peut donc être défini dans l'espace des fusions de $PTA(\mathcal{S})$ par un ensemble d'affectations complètes transitives. Suivant la proposition 3, pour assurer le déterminisme et l'univocité des automates classifieurs ainsi définis, il suffit d'assurer le déterminisme de la fonction de transition $\delta_{\simeq}$ et l'univocité de la fonction de classification $\gamma_{\simeq}$. La transitivité, le déterminisme et l'univocité des affectations peuvent s'exprimer respectivement suivant la proposition 2, le théorème 4 et le corollaire 3 par :

$$\begin{array}{ll} \forall q_1,q_2,q_3 \in Q,\ q_1 \simeq q_2 \wedge q_2 \simeq q_3 \Rightarrow q_1 \simeq q_3 & (a) \\ \forall q_1,q_2,q_1',q_2' \in Q, \forall a \in \Sigma, q_1' \in \delta(q_1,a), q_2' \in \delta(q_2,a),\ q_1 \simeq q_2 \Rightarrow q_1' \simeq q_2' & (b) \\ \forall q,q_1',q_2' \in Q, \forall w \in \Sigma^*, q_1',q_2' \in \delta_{\simeq}(q,w),\ \rho(q_1') \sim \rho(q_2') & (c) \end{array}$$

Les relations de transitivité (a) et de déterminisme (b) sont les relations (1) et (3) du système de contraintes.

Il reste à montrer que la relation (2), sous l'hypothèse de (1) et (3) est équivalente à l'univocité (c), ce qui revient à montrer que l'ensemble des paires d'états compatibles est

6. Regular Inference as Constraint Satisfaction Problem

le même que celui qui est fusionné. De par le déterminisme de la fonction de transition, on a :

$$\forall q,q_1',q_2' \in Q,\forall w \in \Sigma^*,q_1',q_2' \in \delta_{\simeq}(q,w) \Rightarrow q_1' \simeq q_2'.$$

Donc (2) et (3) impliquent (c). Inversement, supposons que (2) n'est pas vérifiée :

$$\exists q_1,q_2 \in Q,q_1 \simeq q_2,\, \rho(q_1) \not\sim \rho(q_2)$$

alors, en choisissant $q = q_1,q_1' = q_1,q_2' = q_2$, $w = \epsilon$, (c) n'est pas vérifiée (sous l'hypothèse que tous les états sont atteignables). La condition (2) est donc nécessaire et suffisante pour assurer l'univocité sous l'hypothèse de la transitivité (1) et du déterminisme (3). □

Ce théorème permet de caractériser, uniquement à l'aide des fonctions d'émission finale ρ et de transition δ de $PTA(\mathcal{S})$, les fusions de paires d'états pour obtenir des k-DFA univoques. L'ensemble des k-DFA univoques généralisant l'échantillon d'apprentissage sous l'hypothèse de la complétude structurelle est alors défini implicitement par l'ensemble des affectations satisfaisant le système de contraintes du théorème. Ce système de contraintes remplace avantageusement l'ensemble BS des automates classifieurs univoques les plus généraux du treillis pour représenter l'ensemble des solutions. Cette caractérisation par système de contraintes permet aussi de traiter le problème de l'inférence de k-DFA univoques à partir de données fixées comme un problème de satisfaction de contraintes sur les affectations. C'est cette approche que nous développons ici.

Le système de contraintes peut être utilisé pour tester si une affectation donnée est solution. Il suffit alors de vérifier que chaque contrainte du système est vérifiée, ce qui peut être fait en temps linéaire en fonction du nombre de contraintes. Il peut aussi être utilisé pour construire progressivement une affectation solution (ou parcourir l'ensemble des affectations solutions). Il permet alors de caractériser la dynamique de propagation des conséquences d'un choix de valeur pour un attribut lors de la contruction d'affectation déterministes univoques. Pour le choix d'une fusion, le système permet de déterminer les fusions d'états à effectuer pour assurer les propriétés de déterminisme et d'univocité. Inversement, pour le choix d'une différenciation, la propagation liée à ce choix pourra être effectuée suivant la version "contraposée" du système de contraintes :

$$\begin{cases} \forall q_1,q_2,q_3 \in Q \ : \ q_1 \not\simeq q_3 \Rightarrow q_1 \not\simeq q_2 \vee q_2 \not\simeq q_3 & (1') \\ \forall q_1,q_2 \in Q \ : \ \rho(q_1) \not\sim \rho(q_2) \Rightarrow q_1 \not\simeq q_2 & (2') \\ \forall q_1,q_2,q_1',q_2' \in Q,\forall a \in \Sigma,q_1' \in \delta(q_1,a),q_2' \in \delta(q_2,a) \ : \ q_1' \not\simeq q_2' \Rightarrow q_1 \not\simeq q_2 & (3') \end{cases}$$

Cette forme met en évidence, par la relation (2'), un ensemble initial de paires d'états à différencier parce que incompatibles. Ces différenciations peuvent être propagées par modus ponens suivant la relation (3'). Cet ensemble d'inégalités initiales $\mathcal{A}_{0\not\simeq}$ coïncide avec l'ensemble des états incompatibles de $PTA(\mathcal{S})$ présenté en section 2.4.2. Plus généralement, on peut constater que l'ensemble des paires d'états incompatibles doivent être différenciées et que la propagation des incompatibilités est identique à la propagation des différenciations assurant le déterminisme.

```
Initialisation_Affectation($M = (\Sigma,\Gamma,Q,q_0,\delta,\rho)$):
($\mathcal{A}_{\simeq}$,$\mathcal{A}_{\not\simeq}$) ← ($\emptyset$,$\emptyset$) /* affectation */
/* Initialisation des fusions */
pour tout $q \in Q$ faire
   $\mathcal{A}_{\simeq} \leftarrow \mathcal{A}_{\simeq} \cup \{q,q\}$
fin pour
/* Initialisation des différenciations dûes aux incompatibilités */
pour tout $\{q_i,q_j\} \in Q \times Q$, $\rho(q_i) \not\sim \rho(q_j)$ faire
   si $\{q_i,q_j\} \notin \mathcal{A}_{\not\simeq}$ alors
      Différencie_et_Propage($q_i$,$q_j$)
   fin si
fin pour
retourner ($\mathcal{E}_I$,$\mathcal{A}_{\simeq}$,$\mathcal{A}_{\not\simeq}$)

Différencie_et_Propage($q_1$,$q_2$):
/* Prérequis : $\{q_1,q_2)\} \notin \mathcal{A}_{\not\simeq}$ */
/* Prise en compte transitivité */
pour tout $\{q_i,q_1\} \in \mathcal{A}_{\simeq}$ faire
   pour tout $\{q_j,q_2\} \in \mathcal{A}_{\simeq}$ faire
      si $\{q_i,q_j\} \in \mathcal{A}_{\simeq}$ alors
         Retourner Exception("Affectation automate ambigüe")
      fin si
      /* Mise à jour $\mathcal{A}_{\not\simeq}$ */
      $\mathcal{A}_{\not\simeq} \leftarrow \mathcal{A}_{\not\simeq} \cup \{q_i,q_j\}$
      /* Propagation */
      pour tout $a \in \Sigma$ faire
         pour tout $\{p_i,p_j\} \in \delta^{-1}(q_i,a) \times \delta^{-1}(q_j,a)$ faire
            si $\{p_i,p_j\} \notin \mathcal{A}_{\not\simeq}$ alors
               Différencie_et_Propage($p_i$, $p_j$))
            fin si
         fin pour
      fin pour
   fin pour
fin pour
```

ALG 2.5: Différenciation d'une paire d'états et propagation

Notons que si le choix est une différenciation, alors la propagation liée à ce choix ne sera effectuée que suivant la version "contraposée" du système de contraintes et n'engendrera que des différenciations. La situation n'est cependant pas symétrique à cause de la transitivité qui implique la relation suivante :

$$\forall q_1,q_2,q_3 \in Q \; : \; q_1 \simeq q_2 \wedge q_2 \not\simeq q_3 \Rightarrow q_1 \not\simeq q_3$$

Le choix d'une fusion impliquera donc des fusions suivant les équations du système du théorème, mais également, suivant la relation ci-dessus, des différenciations et, par conséquent, la propagation de différenciations en sens inverse.

Nous proposons d'étudier à présent le maintien de l'ensemble des valeurs d'attributs déductibles de l'affectation par modus-ponens et modus-tollens sur le système de contraintes à chaque pas de la construction progressive de l'affectation. Intuitivement, ceci correspond à propager les fusions suivant la fusion pour déterminisation et à propager les différenciations à toutes les paires d'états dont la fusion pour déterminisation provoquerait la fusion d'une paire d'états différencié ou incompatible. L'initialisation de cet ensemble de valeurs d'attributs déductibles ainsi que l'affectation et propagation des différenciations peuvent être réalisées suivant le même schéma que dans l'algorithme 2.3 en confondant les ensembles de paires d'états incompatibles et différenciées et en prenant en compte la transitivité (algorithme 2.5).

Si l'espace des fusions considéré est celui de $PTA(\mathcal{S})$ (ou d'un autre automate à structure d'arbre), le corps de la fonction *Différencie_et_Propage()* peut être implémenté en temps constant et dans ce cas, la complexité de l'initialisation en fonction du nombre n d'états de de $PTA(\mathcal{S})$ est en $O(n^2)$. La fonction *Différencie_et_Propage()* peut également être utilisée lors de la construction progressive de l'affectation. Remarquons que dans ce cas, dans l'espace des fusions de $PTA(\mathcal{S})$, la complexité de l'affectation et sa propagation est proportionnelle au nombre d'attributs auxquels la valeur $\not\simeq$ a été affectée.

Pour l'affectation de valeurs $\simeq$, la situation est plus complexe puisque qu'une fusion peut entraîner des différenciations et que la propagation des différenciations dépend des fusions déjà effectuées. Pour simplifier la propagation et assurer la complétude de la propagation, nous proposons de propager les attributs en deux temps (algorithme 2.7). Dans un premier temps, les fusions sont effectuées et propagées suivant la fusion pour déterminisation à l'aide de la fonction *Fusionne()* (algorithme 2.6) et les différenciations à effectuer sont mémorisées. Dans un deuxième temps, les différenciations mémorisées sont affectées et propagées à l'aide de la fonction *Différencie_et_Propage()* déjà présentée.

<u>Fusionne</u>(q_1,q_2):
/* Prérequis : $\{q_1,q_2)\} \notin \mathcal{A}_{\simeq}$ */
/* La transitivité de $\simeq$ étant maintenue,
fusionner q_1 et q_2 revient à fusionner les blocs correspondants */
$B_1 \leftarrow \{q/\{q_1,q\} \in \mathcal{A}_{\simeq}\}$ /* États du bloc de q_1 */
$B_2 \leftarrow \{q/\{q_2,q\} \in \mathcal{A}_{\simeq}\}$ /* États du bloc de q_2 */
/* Mémorisation des différenciations à effectuer par transitivité */
pour tout $q'_1 \in B_1$ **faire**
  pour tout $q'_2 \in B_2$ **faire**
    si $\{q1_1,q'_2\} \in \mathcal{A}_{\not\simeq}$ **alors**
      Retourner Exception("Affectation automate ambigüe")
    fin si
    pour tout $\{q_i,q'_1\} \in \mathcal{A}_{\not\simeq}$ **faire**
      si $\{q'_2,q_i\} \notin \mathcal{A}_{\not\simeq}$ **alors**
        $L_{\not\simeq} \leftarrow L_{\not\simeq} \cup \{q'_2,q_i\}$
      fin si
    fin pour
    pour tout $\{q_j,q'_2\} \in \mathcal{A}_{\not\simeq}$ **faire**
      si $\{q'_1,q_j\} \notin \mathcal{A}_{\not\simeq}$ **alors**
        $L_{\not\simeq} \leftarrow L_{\not\simeq} \cup \{q'_1,q_j\}$
      fin si
    fin pour
  fin pour
fin pour
/* Transitivité et mémorisation des fusions à effectuer pour déterminisation */
pour tout $q'_1 \in B_1$ **faire**
  pour tout $q'_2 \in B_2$ **faire**
    $\mathcal{A}_{\simeq} \leftarrow \mathcal{A}_{\simeq} \cup \{q'_1,q'_2\}$
    pour tout $a \in \Sigma$ **faire**
      pour tout $\{q''_1,q''_2\} \in \delta(q'_1,a) \times \delta(q'_2,a)$ **faire**
        si $\{q''_1,q''_2\} \notin \mathcal{A}_{\simeq}$ **alors**
          $L_{\simeq} \leftarrow L_{\simeq} \cup \{q''_1,q''_2\}$
        fin si
      fin pour
    fin pour
  fin pour
fin pour

ALG 2.6: Fusion d'une paire d'états et mémorisation des propagations à effectuer

Fusionne_et_Propage(q_1,q_2)
$L_{\simeq} \leftarrow \{q_1,q_2\}$
tant que $L_{\simeq} \neq \emptyset$ **faire**
  Fusionne(Tête($L_{\simeq}$))
  $L_{\simeq} \leftarrow$ Queue($L_{\simeq}$))
fin tant que
tant que $L_{\not\simeq} \neq \emptyset$ **faire**
  Différencie_et_Propage(Tête($L_{\not\simeq}$))
  $L_{\not\simeq} \leftarrow$ Queue($L_{\not\simeq}$))
fin tant que

ALG 2.7: Fusion d'une paire d'états et propagation

Les versions des fonctions présentées ici sont des versions n'utilisant pas pleinement la propriété de transitivité dans un but d'illustration, sans préjuger de la méthode de satisfaction de contraintes utilisée, de la propagation des contraintes dans l'espace des fusions. Pour bénéficier pleinement de la puissance de la représentation implicite par espace des fusions, notamment pour un parcours dans le treillis, la propriété de transitivité peut être utilisée.

L'espace des fusions peut alors être représenté à l'aide de deux structures : une matrice de booléens pour les différenciations qui ne sont pas transitives et une structure de données de représentation de partition pour représenter les fusions. Pour les fusions, nous préconisons notamment l'utilisation de structure de données du type *disjoint set union* introduit par Tarjan [Tar72, Tar75]. La place requise par ces structures n'est que linéaire en fonction du nombre d'états. Les avantage de ces structures sont multiples pour le parcours de l'espace de recherche.

Citons en particulier les avantages suivants. Le temps de parcours de l'ensemble des états d'un bloc est linéaire par rapport à la taille du bloc. La fusion de deux blocs peut être effectuée en temps constant. Un état canonique par bloc peut être désigné, ce qui permet de concentrer les relations de fusions et de différenciation de tous les états du bloc sur un seul états (en effet, par transitivité, tous les états d'un bloc sont différenciés et fusionnés avec les même états, un seul état du bloc peut donc être utilisé pour mémoriser les relations des états du bloc avec les états (canoniques) des autres blocs)[7]. Enfin, ces structures sont particulièrement utiles pour une recherche dans l'espace de recherche avec "backtrack" par l'introduction d'une opération de *de-union* [WT89].

L'utilisation de ces deux types de structure de données dans l'espace des fusions fournit alors un outil puissant d'exploration implicite du treillis combinant les avantages de l'espace des partitions et de l'espace des fusions.

7. Notons que la propagation des contraintes devra cependant être effectuée sur l'ensemble des états du bloc

2.4.3.1 Application à l'inférence régulière d'automate déterministe

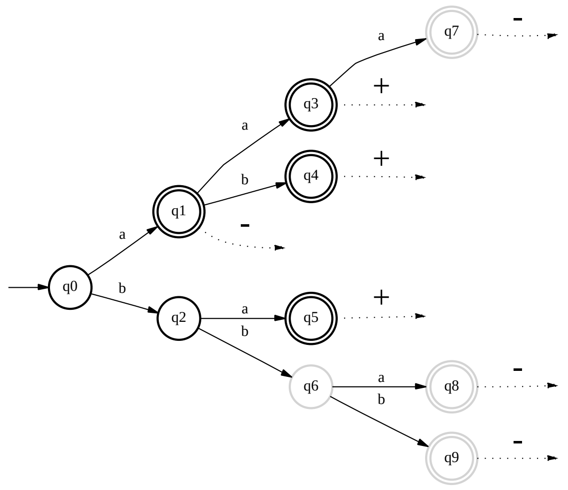


FIG. 2.16 – *Restriction du choix d'états à fusionner. Si l'on ne considère que la complétude structurelle sur l'exemple positif, il suffit de considérer les choix de fusions entre les états $q0$, $q1$,$q2$,$q3$,$q4$ et $q5$ les contraintes devant être tout de même propagées sur l'ensemble des paires d'états de $PTA(\{S_+,S_-\})$.*

Nous proposons dans cette section d'étudier l'utilisation de l'espace des fusions et du système de contraintes pour l'inférence régulière classique d'automate déterministe compatible. L'échantillon d'apprentissage est alors composé d'un échantillon positif et négatif d'un langage. La première approche consiste en l'inférence symétrique du langage et du langage complémentaire. Les définitions et propriétés déjà énoncées peuvent être utilisées, avec en plus la possibilité d'exploiter la complémentarité des langages à inférer pour introduire des biais d'apprentissage [AS95] ou des critères de succès visant la complétude de la classification par le 2-DFA inféré.

Si la symétrie de présentation des échantillons positifs et négatifs n'est pas avérée, et que l'inférence doit plutôt être considérée comme la généralisation du langage de l'échantillon positif sous le contrôle de l'échantillon négatif, l'approche classique de l'inférence régulière d'un automate compatible est à privilégier avec l'emploi des définitions usuelles. Notamment la complétude structurelle ne doit plus être considérée relativement à l'ensemble de l'échantillon positif et négatif mais seulement relativement à l'échantillon positif. Le théorème suivant nous permet d'utiliser l'espace de fusions et le système de contraintes dans ce cadre.

Théorème 4 *Soient un échantillon d'apprentissage* $\mathcal{S} = \{S_+,S_-\}$*, l'arbre accepteur des préfixes associé* $PTA(\mathcal{S}) = (\Sigma,\{+,-\},Q,q_0,\delta,\rho)$ *et l'arbre accepteur des préfixes associé à l'échantillon positif* $PTA(\{S_+\}) = (\Sigma,\{+\},Q_+,q_0,\delta_+,\rho_+)$ *obtenu par restriction de* $PTA(\mathcal{S})$ *au langage positif.*

Pour toute partition π_+ *sur* Q_+*, l'automate* $PTA(\{S_+\})/\pi_+$ *est déterministe et compatible avec* S_+ *et* S_- *si et seulement si il existe une partition* π *sur* Q *telle que la projection de* π *sur* Q_+ *est* π_+ *et l'automate classifieur* $PTA(\mathcal{S})/\pi$ *est déterministe et univoque.*

Remarquons que plusieurs solutions peuvent exister dans le treillis de $PTA(\mathcal{S})$ pour une seule solution correspondante dans le treillis de $PTA(\{S_+\})$. Ces solutions peuvent être construites à partir de la solution dans le treillis de $PTA(\{S_+\})$ en considérant différents choix pour des fusions faisant intervenir un état de $Q - Q_+$ à partir de la solution sur Q_+. Étant donné la propagation des contraintes d'inégalité en sens inverse des contraintes d'égalité, toute affectation consistante sur Q_+, avec propagation des contraintes sur Q, pourra être complétée de façon consistante (en choisissant pour toutes les paires non affectées de $(Q - Q_+) \times Q$ l'attribut inégalité). L'espace de recherche peut donc être limité en ne considérant les choix de fusions que entre des états de Q_+, mais en continuant à propager les contraintes sur Q suivant le système.

Pour la recherche d'automates compatibles pour lesquels S_+ est structurellement complet, l'espace implicite des fusions et le système de contraintes pourront donc être utilisés. Les algorithmes d'inférence de k-DFA univoques, présentés par la suite, pourront ainsi être facilement adaptés le cas échéant à l'inférence d'automates compatible. Il suffira alors de considérer une restriction du choix des fusions aux états de Q_+ avec une propagation des contraintes dans tout l'espace des fusions de $PTA(\{S_+,S_-\})$, les solutions étant dérivées de $PTA(\{S_+\})$ par la restriction à Q_+ des affectations consistantes obtenues.

Nous nous placerons par la suite dans le cadre général de l'inférence de k-DFA univoques sans oublier que les apports de cette approche s'appliquent également à l'inférence régulière classique d'automates compatibles.

Chapitre 3

Inférence de C-DFA univoques (minimaux)

Le système de contraintes *RI-CSP*, obtenu dans le chapitre précédent, permet une représentation implicite de l'ensemble des k-DFA univoques généralisant l'échantillon d'apprentissage (sous l'hypothèse de sa complétude stucturelle). Expliciter cet ensemble revient à trouver l'ensemble des affectations satisfaisant les contraintes du système et peut être abordé par plusieurs méthodes du domaine de la satisfaction de contraintes. En général, étant donné le nombre de solutions potentielles, on ne cherchera pas à les exhiber toutes mais plutôt à trouver la meilleure (ou les meilleures). Une fonction d'évaluation des solutions est alors généralement introduite. Nous nous intéresserons ici à la recherche d'un automate classifieur déterministe univoque du treillis minimisant une fonction d'évaluation *Évaluation*($\mathcal{A}$) pour une affectation complète $\mathcal{A}$. Classiquement, bien que d'autres fonctions à optimiser puissent être considérées, le but en inférence grammaticale est de minimiser le nombre d'états de l'automate.

Le problème peut alors être reformulé sous la forme d'un *problème d'optimisation sous contraintes*. Plusieurs types de résolutions de ce type de problème peuvent être envisagés. Les espaces de recherche implicites des fusions et/ou des partitions sous le contrôle du système de contrainte nous paraîssent particulièrement adapté aux parcours de recherche méta-heuristiques du type Algorithmes Génétiques, Tabou, Recuit simulé ou par réparation d'erreurs. Il nous semble cependant important d'étudier en premier lieu la recherche exacte des solutions optimales à ce problème afin de disposer d'une référence pour l'évaluation des autres types d'approche et aussi mieux comprendre la topologie de l'espace de recherche.

Nous proposons donc dans ce chapitre d'exploiter cette nouvelle formulation du problème pour la recherche exacte de la solution optimale dans le cadre de l'inférence d'automates classifieurs. Cette formulation nous permettant d'utiliser le schéma classique de *Branch and Bound* pour effectuer la recherche, nous pouvons alors proposer deux nouveaux schémas d'algorithmes par fusions et différenciations d'états. Nous présentons ces deux schémas d'algorithmes de façon générale en section 3.1 avant d'étudier leur application à la recherche exacte de k-DFA univoques de taille minimale en section 3.2 et de présenter les expérimentations menées dans ce cas en section 3.3.

3.1 Algorithmes Branch and Bound

L'algorithme générique généralement utilisé pour l'optimisation exacte d'une fonction d'évaluation sous contraintes en recherche opérationnelle et en intelligence artificielle est l'algorithme de séparation-évaluation : *Branch and Bound* (B&B) [Tsa93]. Cet algorithme nécessite une fonction (sous-)estimant pour une affectation incomplète $\mathcal{A}'$ la meilleure valeur d'évaluation que l'on peut obtenir en complétant cette affectation. Nous supposerons qu'une telle fonction est disponible et la noterons *Évaluation_partielle($\mathcal{A}'$)*. Le principe d'un algorithme B&B consiste à explorer entièrement l'espace de recherche en attribuant, à chaque pas, une valeur à un nouvel attribut et en effectuant un retour en arrière dès lors qu'une contrainte est violée ou que la fonction d'évaluation partielle retourne pour l'affectation courante une valeur supérieure à l'évaluation de la meilleure solution déjà trouvée.

L'efficacité d'un algorithme B&B est liée à sa capacité à élaguer au mieux l'espace de recherche et dépend essentiellement de deux facteurs : la qualité de l'évaluation partielle (garante de grandes bornes inférieures) et la rapidité d'obtention d'une bonne solution (fournissant une petite borne supérieure). Les deux facteurs sont clairement de nature heuristique. La qualité de l'évaluation partielle ainsi que les stratégies à mettre en oeuvre pour l'obtention rapide d'une bonne solution dépendent de la fonction à optimiser. Pour les algorithmes B&B avec satisfaction de contraintes, l'élagage peut aussi être provoqué par la détection de l'inconsistance de l'affectation partielle courante. L'algorithme B&B peut alors être amélioré par l'intégration des techniques employées en satisfaction de contraintes de retour en arrière intelligent (*Back-Jump*) avec une éventuelle mémorisation des causes du conflit (*Backmark*) et/ou de propagation d'attributs (*Look-Ahead*) [AS94].

Le retour arrière intelligent et la mémorisation des causes de conflit font partie des schémas rétrospectifs. C'est à dire que l'exploration arborescente déjà effectuée durant l'exécution est mise à profit pour calculer à moindre coût un sous ensemble de l'affectation ne pouvant pas aboutir à une solution. Le backtrack intelligent utilise immédiatement cette détection pour effectuer un retour arrière jusqu'à la première affectation d'attribut à l'origine de la situation détectée. La mémorisation des causes du conflit peut également être utilisée pour éviter cette situation dans l'exploration ultérieure.. Un judicieux compromis doit alors être trouvé entre la mémoire consommée et les effets bénéfiques de la mémorisation.

La propagation d'attributs fait partie des schémas prospectifs. Elle consiste à propager les conséquences du choix d'un attribut suivant le système de contraintes.

Divers types de consistance locale, plus ou moins complètes et, par conséquent, plus ou moins coûteuses à propager, peuvent être envisagées. Les techniques avec propagation des attributs sont, en général, les plus efficaces et sont employées par la majorité des langages et systèmes logiciels de satisfaction de contraintes, le maintien de la consistance étant alors souvent proche de *l'arc consistance totale* [AS94]. Des études menées en satisfaction des contraintes suggèrent que l'hybridation des deux types d'approche peut apporter des gains importants [Pro93]. Mais le gain n'est pas systématique.

Nous proposons dans les sections 3.1.1 et 3.1.2, d'utiliser ce schéma pour la recherche de l'affectation "minimale" dans l'espace implicite des fusions sous le contrôle du système de contrainte *RI-CSP*.

3.1.1 Algorithmes par fusion d'états

Dans cette section, nous proposons un algorithme de type B&B de recherche de la solution optimale sur l'espace des fusions de $PTA(\mathcal{S})$. Celui-ci constitue une extension des algorithmes classiques d'inférence régulière par fusion d'états à la prise en compte des fusions impossibles (algorithme 3.1). Le parcours de l'espace de recherche est effectué en construisant progressivement l'affectation $\mathcal{A}$ par une procédure récursive d'attribution des valeurs possibles ($\simeq$ puis $\not\simeq$) à une paire d'états choisie par la fonction *Choix_paire_d_états()*. Cette attribution est ensuite propagée suivant le système de contraintes par la fonction *Propagation()* (mécanisme de Look-Ahead). À chaque appel de la fonction récursive, un élagage peut être provoqué soit par le test de consistance effectué par la fonction *Consistante()* qui vérifie qu'aucune contrainte du système n'est violée par l'affectation courante, soit par une évaluation partielle plus grande que le meilleur résultat déjà trouvé. La récursivité termine lorsque l'affectation est complète (test par la fonction *Complète()*) et l'affectation courante est alors évaluée par rapport à la meilleure solution courante pour y être substituée au cas échéant. Dans la version présentée ici, seule une des meilleures solutions est recherchée pour simplifier la présentation, mais l'extension à la recherche de l'ensemble des meilleures solutions est facile. D'autre part, cette version calquée sur les algorithmes par fusion, n'intègre pas de mécanisme de retour en arrière intelligent, mais il est évident qu'un tel mécanisme pourrait également être intégré pour une approche plus générale. Notons aussi que l'ordre d'instantiation des valeurs $\simeq$ et $\not\simeq$ pourrait éventuellement être choisi dynamiquement à chaque pas.

Ce schéma général d'algorithme permet de parcourir l'ensemble des k-DFA univoques et de définir les variantes possibles du schéma classique des algorithmes par fusions en explicitant les fonctions utilisées. Les fonctions *Consistante()*, *Complète()* et *Évaluation()* peuvent être implémentées efficacement *a priori* et n'affectent que peu l'efficacité de l'algorithme qui dépend essentiellement de la portion de l'espace de recherche parcourue et donc de sa capacité à élaguer les branches inutiles. L'élagage étant lié à la qualité de l'évaluation partielle, la rapidité d'obtention d'une bonne solution et la rapidité de détection d'inconsistance dans une affectation partielle, l'efficacité de l'algorithme dépend donc du choix des fonctions *Évaluation_partielle()*, *Choix_paire_d_états()* et *Propagation()*. Comme pour toute heuristique de parcours d'espace de recherche, un juste équilibre doit être trouvé entre l'élagage permis par le choix de ces fonctions et le coût calculatoire qu'elles impliquent.

Notons que pour les algorithmes classiques, seules les fusions possibles sont considérées, la propagation des fusions étant effectuée par la fusion pour déterminisation, ce qui revient dans l'espace des fusions à propager les attributs $\simeq$ suivant les relations 1 et 3 du système de contrainte. Cette propagation suffit à détecter l'inconsistance d'une affectation partielle si une contrainte de type 2 est violée ou si la propagation n'est pas possible à cause d'une affectation préalable d'un attribut $\not\simeq$. Cette propagation permet de détecter immédiatement une inconsistance de l'affectation pour un coût proportionnel au nombre d'affectations effectuées si l'on utilise des structures de données adaptées.

Pour une recherche (semi)-exhaustive dans l'espace des fusions, nous proposons dans cet algorithme de marquer et propager également des attributs $\not\simeq$ [CN98a]. L'avantage du marquage est d'éviter de considérer plusieurs fois le même automate. L'avantage de la propagation est la détection plus rapide des inconsistances et le maintien d'un ensemble de fusions impossibles. On diminue ainsi d'autant le nombre de paires d'états à considérer et le temps de calcul de l'heuristique d'ordonnancement pour le choix des paires d'état à considérer . Parmi les autres avantages, on peut citer une meilleure information disponible sur l'espace des fusions (qui peut être exploitée par l'heuristique de choix des paires d'états) [CN98b] ou, comme nous allons le voir dans la section 3.2 portant sur l'inférence d'automates minimaux, pour calculer une évaluation partielle.

Le maintien d'un ensemble de fusions incompatibles permet également de voir le problème comme un problème de coloriage dynamique de graphe. Ce point de vue est utilisé dans la section suivante pour proposer un autre schéma d'algorithme de type B&B pour l'inférence des automates classifieurs univoques.

<u>Main</u>($\mathcal{S}$) :

/* Entrée : échantillon d'apprentissage $\mathcal{S}$ */

/* Sortie : k-DFA optimal suivant la fonction d'évaluation */

PTA $\leftarrow$ Arbre_accepteur_des_préfixes($\mathcal{S}$)

$\mathcal{A} \leftarrow$ Initialisation_Affectation(PTA) /* Affectation courante */

$\mathcal{A}_{Sol} \leftarrow \emptyset$ /* Meilleur affectation trouvée */

Cisma($\mathcal{A}$)

Sortie $\leftarrow$ Dérivation(PTA, $\mathcal{A}_{Sol}$)

<u>Cisma</u>($\mathcal{A}$) :

si Consistante($\mathcal{A}$) **alors**

  si Complète($\mathcal{A}$) **alors**

    /* Solution ? */

    si Évaluation($\mathcal{A}$) < Évaluation($\mathcal{A}_{Sol}$) **alors**

      $\mathcal{A}_{Sol} \leftarrow \mathcal{A}$

    fin si

  sinon

    /* Affectation incomplète */

    si Évaluation_partielle($\mathcal{A}$) < Évaluation($\mathcal{A}_{Sol}$) **alors**

      /* Choix de la paire d'états à considérer */

      (q_1, q_2) $\leftarrow$ Choix_paire_d_états($\mathcal{A}$)

      /* Fusion */

      $\mathcal{A}' \leftarrow \mathcal{A} \cup \{ q_1 \simeq q_2 \} \cup$ Propagation($q_1 \simeq q_2$)

      Cisma($\mathcal{A}'$)

      si Évaluation_partielle($\mathcal{A}$) < Évaluation($\mathcal{A}_{Sol}$) **alors**

        /* Différenciation */

        $\mathcal{A}' \leftarrow \mathcal{A} \cup \{ q_1 \not\simeq q_2 \} \cup$ Propagation($q_1 \not\simeq q_2$)

        Cisma($\mathcal{A}'$)

      fin si

    fin si

  fin si

fin si

ALG 3.1: Cisma, Compatible and Incompatible State Merging Algorithm

3.1.2 Algorithmes par coloriage de graphe

L'ensemble $A_{\not\simeq}$ est un ensemble de couples d'états. Cet ensemble peut donc être représenté par un graphe dont les sommets sont les états de $PTA(\mathcal{S})$ et dont les arcs joignent des états incompatibles.

Le problème de la recherche d'un k-DFA univoque peut donc être vu comme le problème de trouver une k-coloration du graphe respectant le système de contraintes.

Définition 28 *Un graphe $G = (Q,I)$, où Q désigne l'ensemble des sommets et I l'ensemble des arcs, est k-coloriable s'il existe une application ϕ de Q dans l'ensemble $\{1,2,\dots,k\}$ telle que $(u,v) \in I$ entraîne $\phi(u) \neq \phi(v)$, une telle fonction est appelée k-coloration.* □

Une k-coloration sur le graphe des états incompatibles définit donc une partition des états tels que les états coloriés de la même couleur sont compatibles et peuvent être fusionnés entre eux.

Nous proposons de traiter le problème de l'inférence de k-DFA univoques comme un problème de coloriage de graphe sous un système de contraintes. Beaucoup de travaux ont été effectués dans le domaine du coloriage de graphes, qui peut ainsi constituer une bonne source d'inspiration pour l'inférence grammaticale.

Motivé par la recherche de k-DFA minimaux, nous avons choisi de nous inspirer de l'algorithme de recherche des k-colorations minimales DSATUR [Bré79]. Cet algorithme est à la fois performant et basé sur des principes génériques simples pouvant être adaptés facilement à l'inférence d'automates classifieurs. C'est encore une fois un algorithme de type B&B dont nous présentons l'adaptation au coloriage des états de l'espace des fusions sous le contrôle du système de contrainte *RI-CSP* introuduit en section 2.4.3 dans l'algorithme 3.2.

L'algorithme présenté est l'extension à l'espace des automates classifieurs de l'algorithme initialement proposé dans [CN97b] pour l'inférence d'DFA compatibles. Le parcours de l'espace de recherche s'effectue en construisant progressivement le coloriage $\mathcal{C}$ par une procédure récursive attribuant successivement chacune des couleurs possible à un état de Q choisi par la fonction *Choix_état()*.

Par rapport à l'algorithme B&B par fusions, la principale différence, outre l'introduction du coloriage $\mathcal{C}$, est que, à chaque pas, un seul état de Q est considéré au lieu de deux, alors que le nombre de valeurs possibles est plus élevé (choix pour un état d'une des couleurs possibles, au lieu du choix pour un couple entre $\simeq$ et $\not\simeq$).

```
Main(𝒮) :
/* Entrée : échantillon d'apprentissage 𝒮 */
/* Sortie : k-DFA optimal suivant la fonction d'évaluation */
PTA ← Arbre_accepteur_des_préfixes(𝒮)
𝒜 ← Initialisation_Affectation(PTA) /* Affectation courante */
𝒞 ← ∅ ; 𝒞_Sol ← ∅ /* Coloriage courant et meilleur coloriage trouvé */
/* Coloriage d'une clique initiale */
Clique ← Grande_Clique(𝒜)
pour tout q ∈ Clique faire
  (𝒜, 𝒞) ← Affecte_Nouvelle_Couleur(𝒜, 𝒞, q)
fin pour
ColorIG(𝒜, 𝒞)
Sortie ← Dérivation(PTA, 𝒞_Sol)

ColorIG(𝒜, 𝒞) :
si Consistante(𝒜) alors
  si Complet(𝒞) alors
    /* Solution ? */
    si Évaluation(𝒞) < Évaluation(𝒞_Sol) alors
      𝒞_Sol ← 𝒞
    fin si
  sinon
    /* Affectation incomplète */
    si Évaluation_partielle(𝒜,𝒞) < Évaluation(𝒞_Sol) alors
      /* Choix de l'état à colorier */
      q ← Choix_état(𝒜,𝒞)
      /* Boucle sur les couleurs existantes */
      pour tout couleur ∈ 𝒞 faire
        (𝒜', 𝒞') ← Affecte_Couleur(𝒜, 𝒞, q, couleur)
        ColorIG(𝒜',𝒞')
        si Évaluation_partielle(𝒜,𝒞) < Évaluation(𝒞_Sol) alors
          Sortie de la procédure
        fin si
      fin pour
      /* Nouvelle couleur */
      (𝒜', 𝒞') ← Affecte_Nouvelle_Couleur(𝒜, 𝒞, q)
      ColorIG(𝒜',𝒞')
    fin si
  fin si
fin si
```

ALG 3.2: ColorIG, Inférence Grammaticale par Coloriage de graphes

Deux fonctions sont chargées de l'affectation de la couleur d'un état : *Affecte_ Nouvelle_ Couleur()* qui affecte à l'état considéré une nouvelle couleur n'existant pas encore dans le coloriage et *Affecte_ Couleur()* qui affecte une couleur du coloriage courant. Ces fonctions sont chargées de modifier à la fois le coloriage courant $\mathcal{C}$ et l'affectation courante $\mathcal{A}$ suivant la relation :

$$\forall q,q_c, \in Q,\mathcal{C}(q_c) \neq \emptyset,\, q \simeq q_c \Leftrightarrow \mathcal{C}(q) = \mathcal{C}(q_c)$$

et donc également :

$$\forall q,q', \in Q,\mathcal{C}(q) \neq \emptyset,\mathcal{C}(q') \neq \emptyset,\, q \not\simeq q' \Leftrightarrow \mathcal{C}(q) \neq \mathcal{C}(q')$$

Une propagation des attributs de l'affectation peut alors être effectuée suivant le sytème de contraintes.

Comme pour l'algorithme B&B par fusions, un retour arrière intelligent et un ordre de considération des couleurs déterminé dynamiquement peut être ajouté à l'algorithme par coloriage. Au cas échéant, nous désignerons par *Choix_ Couleur($\mathcal{C}$)*, une fonction retournant successivement toutes les couleurs de $\mathcal{C}$ suivant un ordonnancement dynamique. Remarquons que le nombre de choix de couleurs et donc d'appels récursifs pour un état donné dépend de la coloration des états adjacents. Le choix de ce type d'algorithme permet donc d'utiliser la stratégie de saturation des couleurs, qui a donné son nom à l'algorithme DSATUR et qui consiste à choisir comme état à colorier celui qui est adjacent au plus grand nombre de couleurs, pour limiter l'espace de recherche à parcourir.

Par rapport à l'algorithme B&B par fusions, une étape de recherche de clique et de coloriage peut aussi être ajoutée à l'initialisation. En effet, pour un coloriage si l'on peut détecter une clique, i.e. un ensemble d'états tel que chaque état est en relation avec tous les autres états de la clique, tous les éléments de la clique doivent être coloriés par des couleurs différentes. Disposant d'un ensemble de fusions impossibles grâce à l'initialisation de l'affectation, la recherche de la plus grande clique peut donc être effectuée sur ce graphe des fusions impossibles et les états de la clique coloriés de couleurs différentes. La recherche de la clique maximale étant elle même un problème NP-Complet, on se contentera d'une approximation donnée par la fonction *Grande_ Clique()*.

L'approche par coloriage de graphe revient à choisir des états "représentatifs" qui représentent les états de l'automate résultat et à fusionner sur ces états des états "auxiliaires". On peut exprimer dans ce paradigme les algorithmes RPNI [OG92, Lan92], l'algorithme avec heuristique data driven [HOV96], et l'exploration *Blue-Fringe* proposée par Hugues Juillé [JP98] pour la compétition Abbadingo One [LPP98]. Nous détaillons ici les liens avec l'approche Blue-Fringe qui présente de façon visuelle les principes utilisés tout en étant représentative des autres méthodes.

L'approche *Blue-Fringe* propose de répartir les états dans trois ensembles, Rouge, Bleu et Blanc caractérisés par :

– Les états de Rouge ne sont pas fusionnables entre eux ;
– Les fils d'états de Rouge qui ne sont pas dans Rouge sont dans Bleu ;
– Les états qui ne sont ni dans Rouge, ni dans Bleu sont dans Blanc.

Initialement, l'ensemble Rouge est constitué de l'état initial, l'ensemble Bleu est l'ensemble des fils de l'état initial et l'ensemble Blanc contient tous les autres états.

À chaque pas de l'algorithme, si il existe un état de Bleu qui n'est fusionnable avec aucun état de Rouge, il est ajouté à Rouge, sinon, un état de Bleu est fusionné avec un état de Rouge. Les ensembles Bleu et Blanc sont mis à jour en conséquence. Cette méthode permet de limiter le nombre d'états à considérer à chaque pas puisque seuls les paires d'états Rouge $\times$ Bleu sont évaluées. En outre, cet ordonnancement permet de conserver l'invariant que les états de Bleu sont les racines de sous arbres, ce qui permet de faciliter la fusion d'états (pas de tests de boucles à faire).

Ce schéma peut être adopté par l'algorithme B&B par coloriage en restreignant la clique initiale à l'état initial. Les états Rouge étant alors ceux qui sont coloriés, on peut restreindre la fonction de choix d'état à ne sélectionner que des états successeurs d'un état colorié, privilégiant ceux qui sont complètement saturés, c'est à dire ceux qui ne peuvent être fusionnés avec aucun état colorié. L'algorithme B&B par coloriage permet alors de parcourir le même espace de recherche que l'algorithme par fusion Blue-Fringe, il offre cependant la possibilité supplémentaire pour une recherche complète d'ajouter un état à Rouge après avoir testé toutes les possibilités de fusions de cet état avec les états de Rouge.

La différence fondamentale de l'algorithme B&B de coloriage par rapport à l'algorithme Blue-Fringe est la possibilité de ne pas se restreindre à un ordre à partir de l'état initial et de pouvoir choisir les états "représentatifs" parmi tous les états de $PTA(\mathcal{S})$.

3.2 Inférence de k-DFA univoques minimaux

En inférence grammaticale, de par la définition de l'automate canonique et par application du principe du rasoir d'Occam, le critère à optimiser est généralement la taille de l'automate inféré. Pour une présentation complète à données fixée, le problème est bien connu sous la dénomination de la recherche du plus petit DFA compatible et a été montré NP-complet [Gol78]. Nous proposons de considérer dans cette section la contre-partie de ce problème dans l'espace des automates classifieurs : l'inférence de k-DFA univoques minimaux.

Dans les algorithmes génériques de B&B présentés dans les sections précédentes, le choix du critère d'optimisation revient à choisir comme fonction *Évaluation(),* une fonction retournant le nombre d'états du k-DFA courant. Nous abordons dans les prochaines sections les principaux facteurs de l'efficacité de l'algorithme dans ce cas : l'évaluation partielle en section 3.2.1 et l'ordonnancement des choix d'états en section 3.2.2.

3.2.1 Évaluation partielle

Pour la recherche d'automates minimaux, l'évaluation partielle doit fournir une estimation du nombre d'états minimal que l'on peut obtenir à partir de l'affectation courante. Si l'on dispose d'un (sous-)ensemble de fusions incompatibles, une telle borne peut être obtenue en considérant les cliques maximales du graphe des états incompatibles. Comme pour la recherche de clique initiale, on pourra se contenter d'une approximation de la taille de la plus grande clique pour éviter la NP-Complétude inhérente à ce problème.

Remarquons que pour l'algorithme B&B par coloriage, chaque couleur représentant un état de l'automate en cours de construction, l'évaluation partielle peut également être donnée par le nombre de couleurs employées dans le coloriage courant. En fonction des stratégies employées pour colorier les états, l'élagage pourra intervenir plus ou moins tôt. Une bonne stratégie consiste à colorier immédiatement d'une nouvelle couleur tout état fusionnable avec aucun des états déjà coloriés. Dans ce cas, la recherche de clique est incorporée dans le processus de coloriage et peut être remplacé par celui-ci. Remarquons que ce type de stratégies s'améliore avec le nombre d'états considérés à chaque fois pour le choix de l'état à colorier. Pour des stratégies du type Blue-Fringe ne considérant qu'une fraction des états pour le choix de l'état à colorier, une mémorisation pour chaque état du nombre de couleurs adjacentes (par la relation états incompatibles) peut également être employée pour détecter les états à colorier avec une nouvelle couleur.

3.2.2 Ordonnancement

En inférence régulière, les algorithmes classiques par fusions d'états de la littérature sont des algorithmes gloutons n'effectuant qu'une seule recherche en profondeur d'abord et s'arrêtant à la première feuille de l'arbre de recherche, seules les fusions possibles étant ainsi considérées. L'aspect privilégié dans ces algorithmes a donc été l'obtention rapide (gloutonne) d'une bonne solution et les différentes approches se distinguent par le choix de la fonction *Choix_paire_d_états()*.

Dans l'algorithme par fusion proposé par Trakhenbrot et Bardzin [TB73], l'échantillon est supposé complet, ce qui implique que les seules fusions possibles sont celles qui permettent d'obtenir la solution. L'ordre du choix des fusions n'est alors pas important et pourrait être aléatoire.

Pour RPNI [OG92, Lan92], les états sont ordonnés suivant l'ordre standard appliqué au plus court mot permettant de les atteindre. L'ordre standard sur les mots de Σ^* correspond à un ordonnancement suivant la longueur des mots puis, pour une longueur donnée, à l'ordre lexicographique. En considérant les états de l'arbre accepteur des préfixes, cela correspond à ordonner les états suivant un parcours en largeur d'abord avec un ordonnancement des transitions suivant leur étiquette. Le choix des paires d'états à fusionner est alors effectué suivant l'application de cet ordre aux paires d'états. Cet ordre est statique, (il peut être calculé au début de l'algorithme) et prédéfini (il ne tient pas compte des informations fournies par les échantillons d'apprentissage).

Lors de la compétition Abbadingo One [LPP98], les meilleurs résultats ont été obtenus à l'aide d'heuristiques sur le choix des paires d'états à fusionner calculées dynamiquement à chaque pas de la récursivité. La meilleure heuristique actuelle pour le type de problème considéré pour cette compétition[1] consiste à fusionner prioritairement les paires d'états dont la fusion provoque la fusion pour déterminisation du plus grand nombre d'états accepteurs (positifs et négatifs). Plus exactement, si on note n_a le nombre d'états accepteurs avant la fusion, et n'_a le nombre d'états après la fusion déterministe (fusion, puis fusion pour déterminisation) d'une paire d'états, les paires d'états sont choisie prioritairement suivant le score de leur fusion donné par $n_a - n'_a$ [Lan97, LPP98]. Nous noterons par la suite ce score *DD* (pour *Data Driven*).

Pour une recherche gloutonne, si l'on dispose d'une bonne heuristique pour le choix des paires à fusionner, il semble que le meilleur résultat sera obtenu par l'utilisation d'un algorithme par fusion laissant toute liberté pour le choix de la paire d'états à fusionner. En revanche, si l'on souhaite effectuer une exploration plus complète du treillis, il devient alors important de diminuer également la portion de l'espace de recherche explorée.

1. Inférence d'automates compatibles minimaux à partir d'exemples, la génération des automates et des exemples étant effectuée suivant un processus aléatoire (voir section 3.3)

3.3 Implémentation et comparaisons expérimentales

Nous décrivons à présent le travail d'implémentation et d'expérimentation réalisé sur les idées présentées dans les sections précédentes.

3.3.1 Plate forme logicielle d'expérimentation

Dans cette section, nous présentons la plate-forme logicielle d'expérimentation pour l'apprentissage de machines à états finies utilisée pour les expérimentations. La plate-forme n'étant pas encore figée, nous ne décrivons ici que la philosophie de conception sans entrer dans les détails techniques de l'implémentation. La plate-forme étant auto-documentée, nous renvoyons le lecteur à celle-ci pour une documentation technique à jour.

Les objectifs de la plate-forme sont le test, la validation et la comparaison de différentes approches d'apprentissage de machines à états finies. Le but poursuivi étant de pouvoir rapidement implémenter un algorithme et de tester ses différentes variantes dans des conditions similaires, la modularité, la simplicité de modification et de réutilisation ont été privilégiées. Nous avons donc choisi d'adopter une approche objet pour la conception de la plate-forme. Le langage utilisé est C++, permettant ainsi une bonne portabilité.

Actuellement, la plate-forme logicielle peut être découpée en trois modules principaux (voir figure 3.1, un mémento des notations du type UML/OMT utilisées est présenté en annexe A). La base de la plate-forme, appelée FSMTL (Finite State Machine Template Library), gère la création et la manipulation de différents types de machines à états finies. Utilisant cette base, le module LATFSM (LATice of FSM) permet de gérer et parcourir (dans l'espace des partitions ou dans l'espace des fusions) le treillis des machines dérivées d'une machine à états finise. Enfin, le module SMARI (State Merging Algorithms for Rational Inference) regroupe les principaux algorithmes d'apprentissage par fusion d'état et utilise les deux modules précédents.

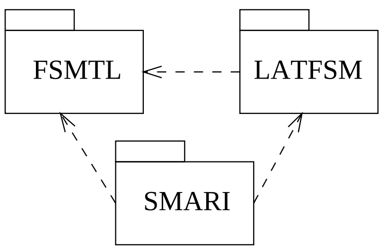


FIG. 3.1 – *Composantes principales de la plate-forme logicielle*

3.3.1.1 FSMTL

Le module FSMTL a été conçu comme un complément de la librairie STL (Standard Template Library), qui fait à présent partie du standard C++. La librairie STL est une implémentation flexible et puissante de structures de données (containers) et d'algorithmes standards. Le module FSMTL propose une implémentation des machines à états finies dans l'esprit de la librairie STL dont il reprend les principaux traits : *templates* pour la souplesse et l'efficacité, *containers*, accès par itérateurs et algorithmes génériques.

La principale particularité de cette librairie est la nette séparation entre aspects structurels et aspects fonctionnels d'une machine à états finie. En effet une même structure de données peut servir à implémenter différentes machines suivant la définition considérée de la fonction d'entrées-sorties (la relation de transduction) réalisée par la machine.

Au niveau structurel, l'analyse des différences entre les divers types de machine permet d'identifier quatre types de structures de données pour l'implémentation des machines à états finies, suivant que l'étiquetage des transitions par les symboles d'entrée suffit ou qu'une information supplémentaire est nécessaire sur les états, les transitions ou les deux à la fois. Cette différentiation introduit une hiérarchisation des structures qui permet de définir les principales classes du module FSMTL ainsi que leur hiérarchie.

La figure 3.2 présente le graphe d'héritage des quatre principales classes de FSMTL : *FsmTL, MooreTL, MealyTL, TransducerTL.*

La classe *FsmTL* est la classe de base de la hiérarchie, elle permet de gérer la structure d'une machine à états finie avec des transitions entre états étiquetées. Elle est paramétrée par la classe *TransInput* des symboles d'entrée étiquetant les transitions. Les services rendus par cette classe sont essentiellement la création, la destruction et l'accès sur les états et les transitions.

Les classes *MooreTL, MealyTL* constituent une spécialisation de cette classe par l'ajout d'une information de sortie, respectivement sur les états et les transitions (de classes respectives *StateOutputTL* et *TransOutputTL*). Elle fournissent alors respectivement une fonction d'émission d'états ρ et une fonction d'émission de transition λ.

Enfin, pour les machines nécessitant une information de sortie sur les états et les transitions, la classe *TransducerTL* permet d'instancier des objets héritant à la fois des propriétés de *MooreTL* et de *MealyTL*.

Notons que le schéma présenté ici est une version simplifiée de la hiérarchie du module *FSMTL* dans lequel nous avons notamment masqué la possibilité de choisir le type d'implémentation.

Dans la bibliothèque réalisée, il est possible de paramétrer les classes par le type d'implémentation à utiliser, ce qui permet de choisir la structure de données à utiliser (double liste d'adjacence, matrice, tableaux associatifs, tables de hachage, ...). L'implémentation doit être choisie en fonction du niveau de dynamisme requis par l'application, des contraintes de mémoire et des propriétés des machines manipulées (déterminisme par exemple). L'utilisateur peut alors choisir une implémentation fournie par la bibliothèque ou créer lui même sa classe d'implémentation en respectant l'interface minimale de la partie «*Structure*», définie pour chaque type de structures de machines.

Différentes interfaces de la partie «*Structure*» sont possibles, classées en fonction du niveau de dynamisme requis pour l'implémentation. Le niveau de dynamisme d'une implémentation correspond à la complexité des opérations de création et de destruction d'éléments d'une machine. Ces niveaux vont de "statique", qui correspond à une machine destinée à être construite uniquement par copie, à dynamique qui correspond à une machine à laquelle on peut ajouter et enlever des noeuds ou transitions à volonté. Le niveau intermédiaire correspond à des machines auxquelles des noeuds et des transitions peuvent être facilement ajoutés mais pas supprimés.

En complément de la gestion de la structure interne, les classes *FsmTL, MooreTL, MealyTL, TransducerTL* proposent également les principales fonctions génériques de parsing et de production applicables à chaque niveau (partie «*Services*»). Un type de machine particulier peut alors être rapidement défini par soit en utilisant directement la classe correspondante de *FSMTL* et ses fonctions, soit par dérivation de cette classe et définition de la fonction de transduction réalisée, à l'aide des fonctions génériques.

Ainsi, par exemple, la classe *MooreTL* permet d'implémenter facilement les automates, les automates classifieurs et les machines de Moore alors que *TransducerTL* peut être utilisé pour l'implémentation de transducteurs ou de transducteurs sous-séquentiels. Si l'on décide d'implémenter les automates avec la classe *MooreTL*, on peut choisir comme classe de sortie d'états (*StateOutput*) la classe des booléens, ce qui permet de définir l'acceptation d'un mot w par $\rho(\delta(q_0,w))$. Pour avoir une utilisation plus intuitive et plus en rapport avec les habitudes du domaine, on pourra spécialiser la classe *MooreTL* en une classe *Automate* pour ne pas avoir à se soucier du paramétrage du type de production des états et définir, par exemple, une fonction *Est_Accepté(w)*.

3.3.1.2 LATFSM

Le module LATFSM permet de représenter économiquement des machines à états finies du treillis d'une de ces machines par utilisation des espaces implicites des partitions et/ou des fusions définis en section 2.3.

La technique employée pour l'utilisation d'une machine à états finie du treillis est une technique d'évaluation paresseuse. Dans l'espace des partitions, une machine du treillis sera représentée uniquement par la partition qui permet de l'obtenir. Les transitions de la machine dérivée ne seront calculées à partir de l'élément nul du treillis que lors de leur utilisation effective.

Cette approche permet un gain de place lorsque l'on considère plusieurs éléments du treillis simultanément et une économie de calcul si l'on n'a pas besoin de considérer toutes les transitions de chaque élément dérivé. De plus, une partition peut être implémentée par une structure de type *disjoint set union* [Tar72, Tar75]. Cette structure permet la fusion de deux blocs en temps constant et également une fission efficace si les fissions sont réalisées dans l'ordre inverse des fusions. Cette approche favorise aussi l'efficacité de l'exploration du treillis. De façon similaire à l'espace des partitions, une machine du treillis sera représentée dans l'espace des fusions par l'affectation qui la définit.

Le schéma 3.3 présente les classes *PartitionFsm* et *AffectationFsm* représentant une machine dérivée d'une instance de *FsmTL* respectivement dans l'espace des partitions et l'espace des fusions. Le schéma d'héritage sur les types de machine de la figure 3.2 peut être appliqué à chacune de ces classes pour ajouter les fonctionalités des classes Moore, Mealy et Transducer. Nous illustrons en figure 3.4, l'exemple du schéma de dérivation de la classe *AffectationFsm* en *AffectationMoore*, *AffectationMealy* et *AffectationTransducer*.

Si l'on ne considère que des affectations transitives, il est plus efficace de mémoriser les égalités de l'affectation à l'aide d'une partition. Dans ce cas, la classe *TransitiveAffectationFsm* est alors particulièrement adaptée à une propagation efficace des attributs dans l'espace des fusions suivant le système de contraintes *RI-CSP* exposé en section 2.4.3. Notons ici, que le système de contrainte est exprimé uniquement en fonction des fonctions de transition et de production de $PTA(\mathcal{S})$ et qu'il peut être obtenu immédiatement à partir de celui-ci. Il est alors possible de représenter économiquement le système par $PTA(\mathcal{S})$ si celui-ci est déjà disponible. Pour l'inférence d'automates classifieurs à partir d'exemples, le $PTA(\mathcal{S})$ étant l'élément nul du treillis, la propagation suivant le système de contrainte ne nécessite donc pas d'espace mémoire supplémentaire pour le stockage de *RI-CSP*.

3.3.1.3 SMARI

Le module SMARI contient les algorithmes d'inférence de machines à états finies (voir figure 3.5). Il est actuellement limité à l'inférence de machine de Moore, mais la généralité de l'approche et la hierarchisation de FsmTL permet d'envisager son extension aux autres types de machines.

L'ossature de ce module est constitué de deux classes implémentant respectivement le schéma d'inférence par fusion de la section 3.1.1 et le schéma d'inférence par coloriage de la section 3.1.2. Les algorithmes sont considérés comme des containers de solutions et peuvent donc être parcourus par itérateur. Le paramétrage de ces classes permet d'instancier les différentes variantes d'algorithmes.

Ainsi une instance d'algorithme de classe *Cisma* sera défini par le choix de :

- la classe de la machine à apprendre ;
- la classe de la représentation implicite dans le treillis utilisée, ce qui permet divers degrés de propagation des contraintes et de détection des inconsistances ;
- la classe proposant les fonctions d'évaluation d'une solution complète ou partielle ;
- la classe proposant la fonction de choix de la paire d'états considérée à chaque pas.

De même, l'instanciation d'un objet algorithme de la classe *Colorig* nécessitera la spécification de :

- la classe de la machine à apprendre
- la classe de la représentation implicite dans le treillis utilisée, ce qui permet divers degrés de propagation des contraintes et de détection des inconsistances
- la classe proposant les fonctions d'évaluation d'une solution complète ou partielle
- la classe proposant la fonction de choix de l'état colorié à chaque pas
- la classe proposant la fonction d'ordonancement des couleurs

Les classes des machines à apprendre et de la représentation implicite sont à choisir respectivement dans les modules FSMTL et LATFSM. Le module SMARI propose des classes implémentant diverses politiques d'évaluation (Taille de l'automate, complétude du classement pour un automate classifieur, ...) et d'ordonnancement (ordre préfixe, ordonancement dynamique avec calcul de scores, ...).

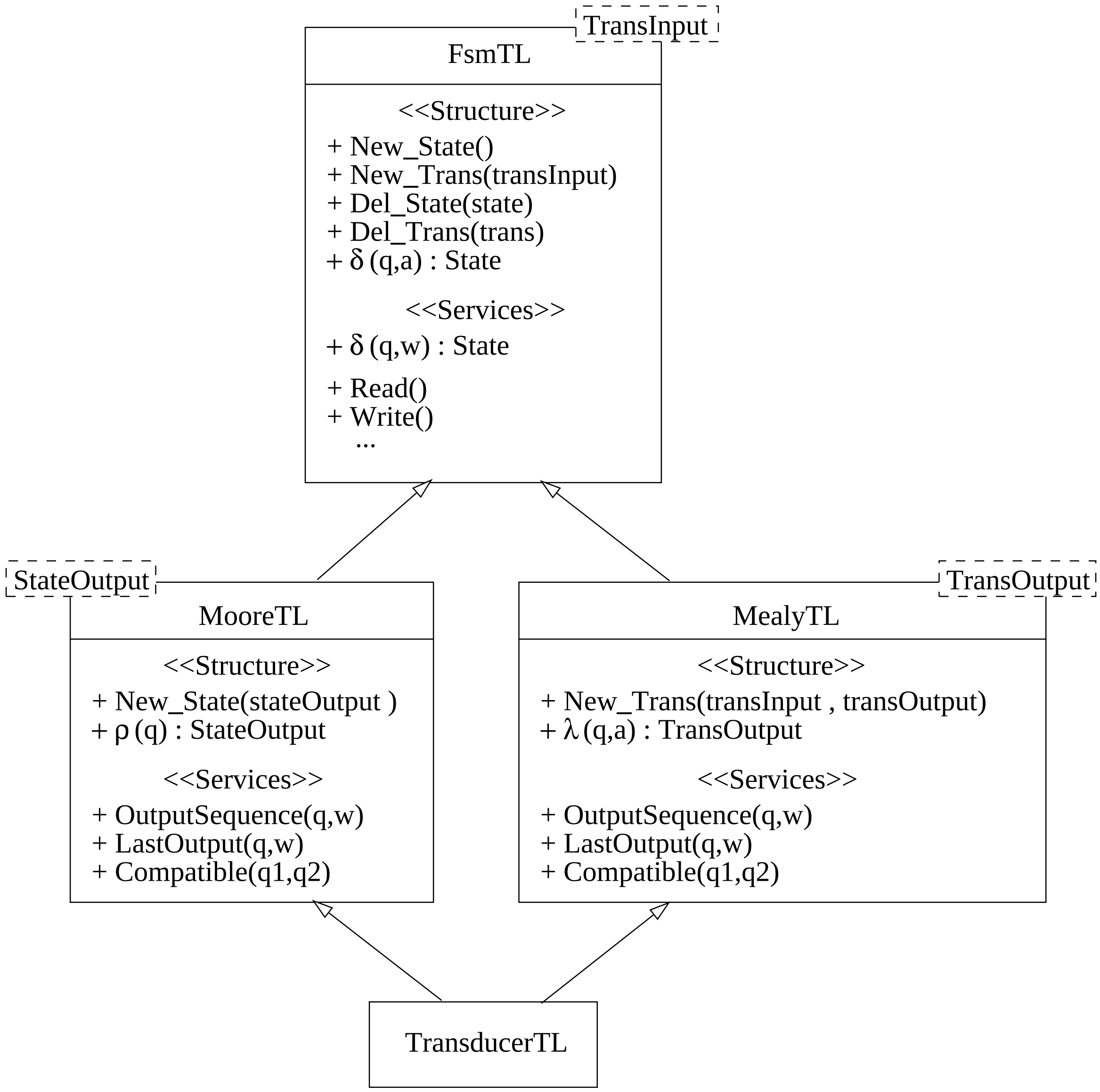


FIG. 3.2 – *Hiérarchie des structures de machines à états finies*

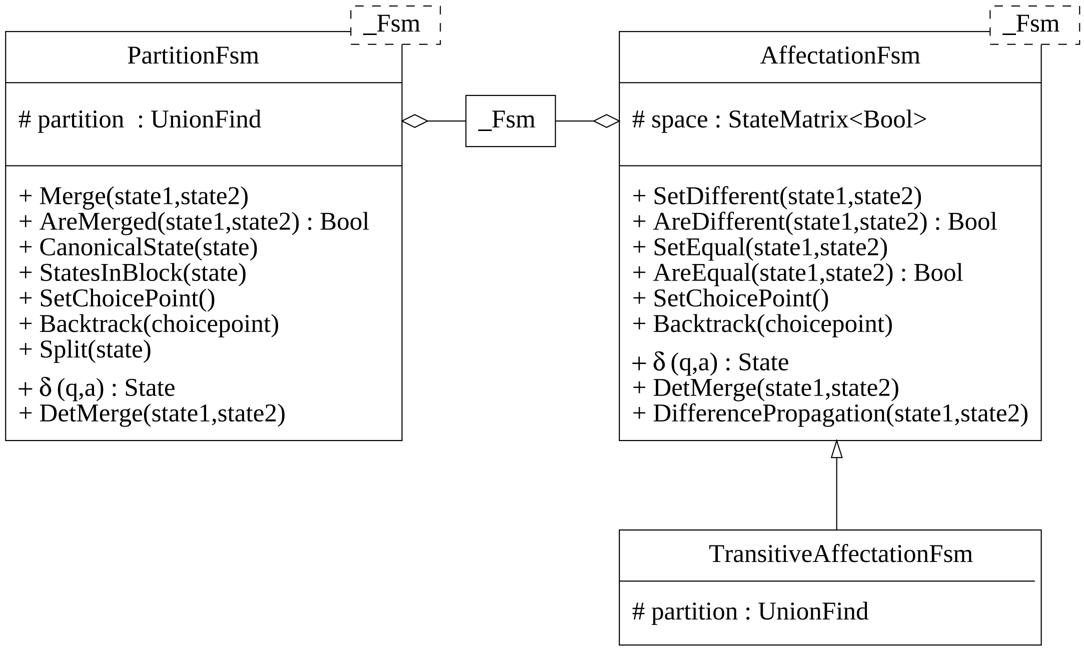


FIG. 3.3 – *Représentations implicites d'un Fsm dans l'espace des partitions et des fusions*

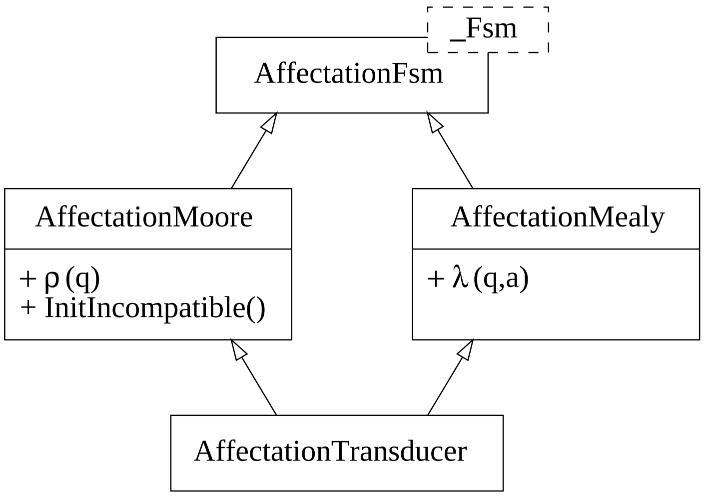


FIG. 3.4 – *Spécialisation de AffectationFsm aux autres types de machines*

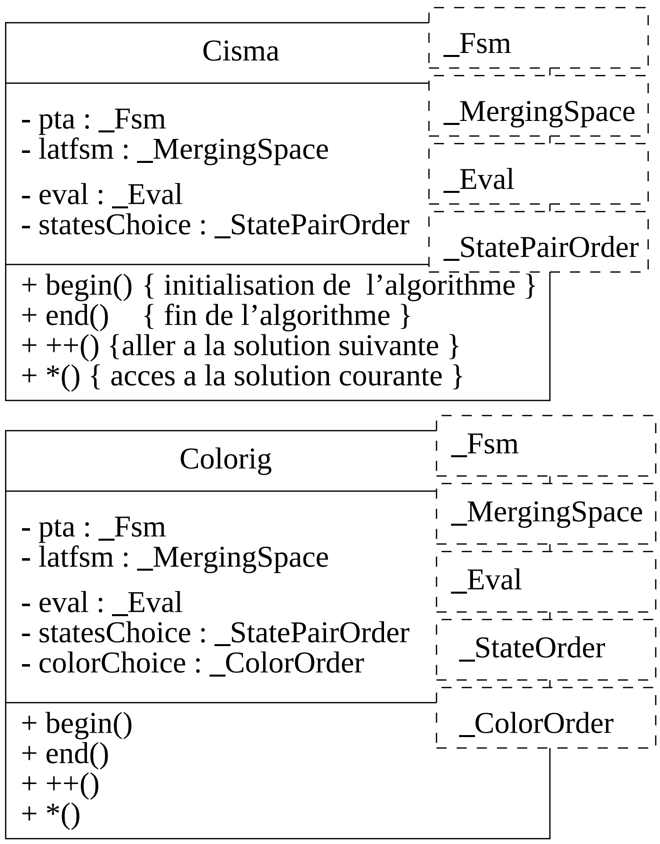


FIG. 3.5 – *Classes d'algorithmes d'inférence*

3.3.2 Comparaison expérimentale des ordonnancements

Nous proposons dans cette section de comparer et illustrer les propriétés des deux types de schémas B&B proposés dans la section 3.1 sur le problème de la recherche exacte d'un k-DFA univoque de taille minimale introduit en section 3.2.

Notons ici que ce problème est NP-Complet en général. Les conditions pour pouvoir résoudre ce problème reposent alors soit sur une quantité d'information suffisante pour guider la recherche grâce à l'utilisation d'heuristiques, soit sur une taille de l'espace de recherche suffisamment restreint pour pouvoir être parcouru par une stratégie efficace. Dans le dernier cas de figure, il est intéressant de connaître le domaine d'application et les limites d'une telle approche qui est la seule à pouvoir garantir l'optimalité de la solution obtenue. Une telle étude permet en outre de disposer d'une base d'évaluation des recherches heuristiques et peut donner lieu à des versions dégradées d'exploration pseudo-exhaustive de l'espace de recherche.

Dans cette section mous nous consacrons plus particulièrement au facteur principal d'efficacité : l'ordonnancement des attributs considérés.

3.3.2.1 Stratégies d'ordonnancement évaluées

Nous considérons dans cette étude plusieurs types d'ordonnancements choisis pour leur représentativité et leurs performances.

Pour le schéma d'algorithmes B&B par fusions, nous comparons quatre stratégies de choix des paires d'états à fusionner :

– *StdSma* : Les paires d'états sont considérées dans l'ordre standard. Cet algorithme peut être considéré comme l'extension de RPNI pour une recherche complète.

– *MSma* (MergeSma) : Les paires d'états sont ordonnées et choisies suivant le nombre de fusions de blocs qu'elles entraînent. Cette stratégie correspond à une approche du type *hill-climbing* pour la diminution du nombre d'états.

– *DDSma* (DataDriven) : Les paires d'états sont ordonnée et choisies suivant le score *DD*, comptant le nombre d'états accepteurs fusionnés entre eux (voir section 3.2.2). C'est la stratégie gagnante de la compétition Abbadingo One.

– *BlueFringeSma* : est l'implémentation de la méthode Blue Fringe dans le schéma par fusions d'états (cf section 3.1.2). À chaque pas, la paire d'états choisie est celle, composée d'un état de Rouge et d'un états de Bleu, de plus grand score *DD*.

Pour le schéma d'algorithmes B&B par coloriage, nous avons considéré cinq stratégies de choix d'états à colorier, le choix dynamique de l'ordre d'affectation des couleurs étant fixé et effectué suivant le score *DD* pour tous les algorithmes :

- *StdColor* : L'état à colorier est choisi suivant l'ordre standard. Cet algorithme peut être considéré comme l'extension de RPNI pour une recherche complète avec choix dynamique des couleurs suivant l'heuristique DD.

- *BlueFringeColor* : est l'implémentation de la méthode Blue Fringe dans le schéma par coloriage (cf section 3.1.2). À chaque pas, la paire d'états, composée d'un état colorié (Rouge) et d'un états de Bleu, dont la fusion maximise la fonction *DD* est recherchée. L'état à colorier de cette paire est alors celui choisi par la fonction de choix d'états. La différence avec *BlueFringeSma* est que une fois choisi, l'état, dans le cadre d'une recherche complète, est colorié successivement avec toutes les couleurs.

- *DDColor* : L'état est choisi parmi tous les états non coloriés suivant le score *DD*.

- *ColorSatColor* : L'état choisi est celui avec le plus grand nombre de couleurs adjacentes différentes. Cet ordonnancement est l'adaptation de l'heuristique proposée par Brèlaz pour le coloriage de graphes et est utilisé dans [CN97b] [2]. Le nombre de couleurs possible pour l'état étant plus faible, l'espace de recherche est réduit. De plus, cette stratégie réduit le nombre de mauvais choix.

- *FnokSatColor* : L'état choisi est celui adjacent avec le plus grand nombre d'états, ce qui correspond à une stratégie de saturation brute de l'espace.

Pour comparer les ordonnancements, les autres paramètres d'implémentation ont été fixés de sorte à être aussi neutres que possible. Nous utilisons bien entendu l'initialisation et la propagation des contraintes suivant le système de contrainte *RI-CSP*, comme présenté en section 2.4.3 (i.e. la transitivité est maintenue, l'ensemble des paires d'états incompatibles est maintenu et la propagation des fusions est effectuée par fusion pour déterminisation). La fonction d'évaluation partielle est implémentée pour les algorithmes par fusion par une recherche gloutonne de clique, avec ordonnancement des états suivant leur degré. Pour les algorithmes de coloriage, l'évaluation partielle est directement donnée par le nombre de couleurs utilisées. Sans l'utilisation de ces techniques, ces expérimentation intensives sur la recherche exacte de solutions minimales n'auraient pas pu être effectuées.

2. L'ordonnancement utilisé dans [CN97b] est en fait une version améliorée départageant les *ex-aequo* en favorisant l'état ayant le plus grand nombre d'états adjacents suivant en celà l'implémentation de DSATUR [Bré79]. Mais pour comparer l'influence propre de chaque stratégie, nous avons préféré ne pas considérer ici les combinaisons de celles-ci.

Précisons encore que le nombre de calculs de scores a été limité à chaque pas aux 10000 premières paires d'états fusionnables suivant l'ordre standard, suivant en cela l'observation empirique de Lang [Lan97] que l'on pouvait limiter le calcul des scores à une profondeur limitée (paramètre *window*). D'autre part, le calcul d'une clique initiale pour un premier coloriage n'a pas été effectué pour ne pas fausser les comparaisons d'ordonnancement et de schémas. Nous avons ainsi choisi de ne colorier initialement que l'état initial.

3.3.2.2 Données d'apprentissage

Nous avons basé cette étude expérimentale sur le jeu de test utilisé par Dupont [Dup96] et un jeu de test aléatoire généré à la demande par le serveur Gowachin.

Le jeu de test de Dupont est classique au sein de la communauté. Il comporte un ensemble de 300 échantillons générés à partir d'un ensemble de 15 langages considérés comme représentatifs de la classe des langages réguliers (voir figure 3.6). La taille des automates cibles est suffisamment petite pour pouvoir faire des expérimentations intensives, la contrepartie étant que la taille des automates limite la significativité des résultats et ne permet pas forcément de préjuger des résultats sur des problèmes de plus grande taille.

Le serveur Gowachin est un serveur Web (http://www.irisa.fr/Gowachin/) de problème d'inférence grammaticale à la demande. Il a été conçu par l'auteur, Barak Pearlmutter et Kevin Lang pour faire suite au succès de la compétition Abbadingo One (http://abbadingo.cs.unm.edu/). Les problèmes sont générés suivant le même processus que pour les problèmes de la compétition, l'utilisateur du serveur pouvant spécifier ses propres paramètres : taille du problème, nombre d'exemples, probabilité de bruit sur l'étiquetage et probabilité de bruit sur les exemples. Nous avons généré 10 problèmes à 10 états, avec des probabilités de bruit nulles.

Pour pouvoir comparer les différentes stratégies en fonction de la quantité d'information relative disponible , nous avons utilisé suivant l'idée du protocole expérimental de Lang [Lan92], des échantillons de 50 à 500 exemples. Lang a en effet mis en évidence une plage de difficulté pour l'inférence d'automates selon la valeur du ratio des tailles de l'échantillon d'apprentissage et de l'automate. En dessous de cette plage, la taille de l'automate minimal compatible est inférieure à celle de l'automate cible, ce qui montre que l'échantillon est insuffisant pour espérer retrouver l'espace cible. Au dessus de cette plage, un algorithme glouton suffit pour trouver la solution minimale. Cette plage de valeurs correspond donc à la zone d'inférence difficile, qui n'est pas traitée de façon satisfaisante par les approches actuelles.

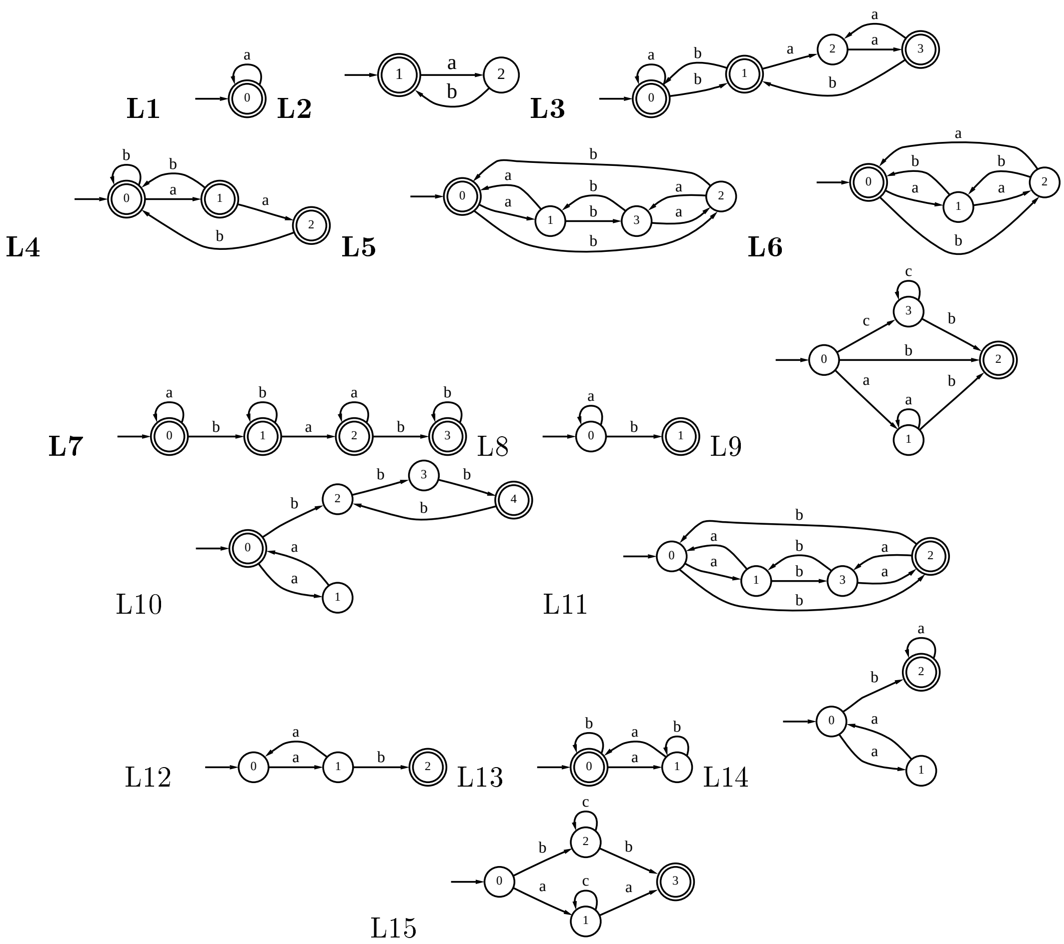


Fig. 3.6 – *Automates pour la génération des échantillons d'apprentissage [Dup96]*

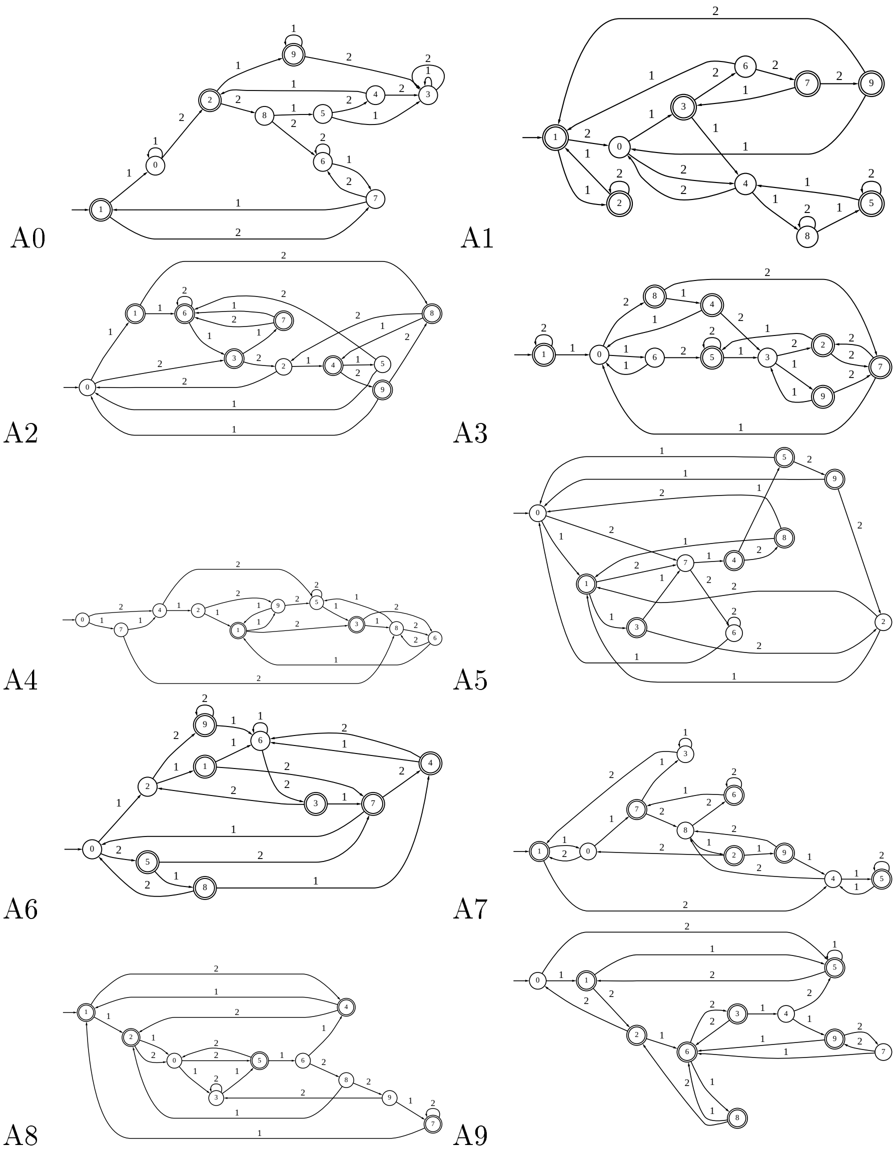


FIG. 3.7 – *Automates cibles générés par Gowachin*

3.3.2.3 Sélection des meilleures stratégies

Nous avons d'abord utilisé le jeu de test de Dupont pour faire une première sélection des stratégies les plus prometteuses et discerner les grandes tendances. Chaque stratégie a été confrontée à l'ensemble des problèmes, le nombre maximal de fusions étant fixé à 500000. Par rapport aux problèmes traités, cette valeur est suffisamment haute pour que la recherche ait été complète pour presque toutes les stratégies sur l'ensemble des problèmes. Nous avons étudié quatre cas de figure.

Dans une première expérimentation, nous proposons de comparer les performances pures de chacune des stratégies pour la recherche exacte de solution minimale. La figure 3.8 synthétise les résultats obtenus sous la forme d'un graphique présentant le nombre moyen de fusions nécessaires pour trouver la solution ainsi que pour effectuer la recherche complète. Trouver une bonne solution rapidement permettant de mieux élaguer l'arbre de recherche, les résultats de la recherche complète sont corrélés à l'obtention rapide de la solution. La différence de performance entre approches d'ordonnancement *Data Driven* et approches d'ordonnancement statiques, constatée pour les recherches gloutonnes [Lan97], se retrouve donc logiquement pour les recherches complètes.

Les résultats permettent de distinguer nettement trois stratégies moins efficaces : *StdSma*, *StdColor* mais aussi la stratégie par saturation des couleurs *ColorSatColor*. Les autres stratégies obtiennent des résultats plus comparables, avec un très bon comportement de *DDSma* mais aussi de *BlueFringeColor* (malgré le nombre moins grand de fusions possibles à chaque pas) et, en queue de peloton, *MSma* et *FnokSatColor*, handicapé par sa recherche de première solution. On peut également remarquer, en comparant les résultats de *BlueFringeColor* et *DDColor*, que la stratégie de choix dans le *Blue Fringe* n'est pas seulement une stratégie permettant de limiter le nombre de calculs de scores de paires d'états, mais constitue en elle-même une heuristique pour trouver plus rapidement la solution.

Nous avons comparé les stratégies dans une deuxième expérimentation, amenuisant l'importance de la recherche de la première solution. Nous avons exécuté les algorithmes en leur fournissant une borne supérieure de la taille de la solution, trouvée par l'algorithme glouton suivant l'heuristique *DD*. Ce scénario correspond à une situation où l'on essaye d'améliorer ou vérifier le résultat donné par un algorithme rapide de nature heuristique. La figure 3.9 présente le nombre moyen de fusions effectuées pour trouver la solution (0 si la solution est trouvée par l'algorithme glouton) et pour effectuer la recherche complète. Cette situation ne change pas fondamentalement les résultats pour les stratégies suivant l'ordre standard. La recherche de la première solution est améliorée pour la stratégie*ColorSatColor*, mais le temps d'exploration de l'espace de recherche reste alors quand même très largement supérieur aux autres stratégies. Le plus grand changement concerne la stratégie *FnokSatColor* qui devient alors la stratégie la plus performante.

Nous avons ensuite évalué les performances des stratégies pour la vérification de l'optimalité de la taille d'un automate (figure 3.10). La borne supérieure fournie aux algorithmes est alors la taille réelle de l'automate cible. Dans ce cas, *StdColor* et *ColorSatColor* restent de loin les moins bonnes stratégies alors que *StdSma* redevient une stratégie acceptable. La différence entre *StdSma* et *StdColor* est alors explicable par la différence d'évaluation partielle utilisée : la recherche de clique sur les paires d'états incompatibles, même gloutonne, permet alors d'élaguer beaucoup plus fortement l'espace de recherche que le compte d'états coloriés. Les autres méthodes présentent des résultats du même ordre de grandeur. Les bonnes performances de *DDSma* et *DDColor* sont à noter pour une tâche où les stratégies par saturation semblaient logiquement plus adaptées. Il semblerait que considérer les fusions les plus prometteuses en premier puis de les différencier soit aussi une bonne stratégie pour l'élagage de l'espace de recherche, mais cela reste à confirmer sur des problèmes de plus grande taille.

Enfin, nous présentons dans la figure 3.11 le nombre de fusion moyen pour effectuer la recherche de toutes les solutions de taille minimale. L'exploration du treillis est alors logiquement plus coûteuse. La stratégie *ColorSatColor* figure encore une fois parmi les moins efficaces, rejointe dans ce cas de figure par la stratégie *MSma*. Les meilleures stratégies sont alors *BlueFringeColor* et *FnokSatColor*.

De cette étude sur le jeu de test de Dupont, nous retenons surtout la bonne performance de l'approche BlueFringe dans tous les cas de figure. Cette approche réalise un très bon compromis entre le choix d'états suffisament proches de la racine pour être impliqués dans l'acceptation d'un grand nombre d'exemples (et ainsi n'être fusionnables que si les états de l'automate cible correspondant sont effectivement fusionnés) et l'aspect dynamique de l'exploration de l'espace de recherche. Nous pouvons également retenir les bonnes performances de l'approche *FnokSatColor* pour la vérification de solutions, là où *ColorSatColor* était attendue. Nous pensons que cette différence s'explique par l'aspect dynamique du coloriage de graphe considéré, dû à la propagation des contraintes. La stratégie *FnokSatColor* obtient le comportement escompté pour *ColorSatColor* par la propagation des contraintes. Fusionner un état colorié avec un état incompatible avec beaucoup d'états revient à affecter par transitivité les incompatibilités à l'état colorié et ainsi restreindre l'espace de recherche et les choix possibles. Enfin, la stratégie *DDSma*, grâce à l'introduction de la prise en compte des états incompatibles et de l'évaluation partielle par recherche de clique, est apparu très polyvalente et a montré qu'elle ne devait pas être réservée uniquement à l'inférence gloutonne.

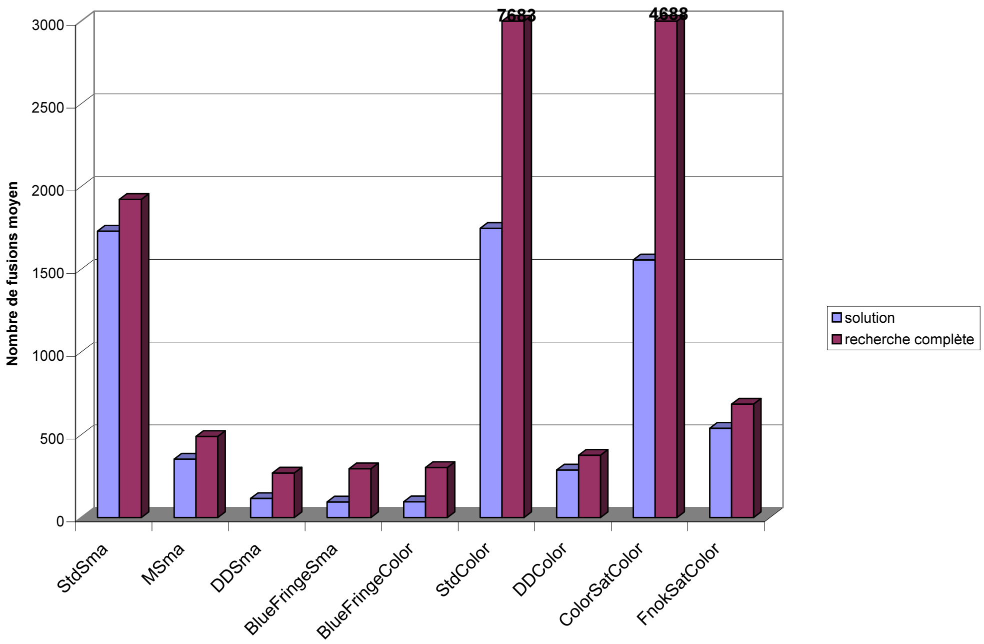


FIG. 3.8 – *Recherche complète*

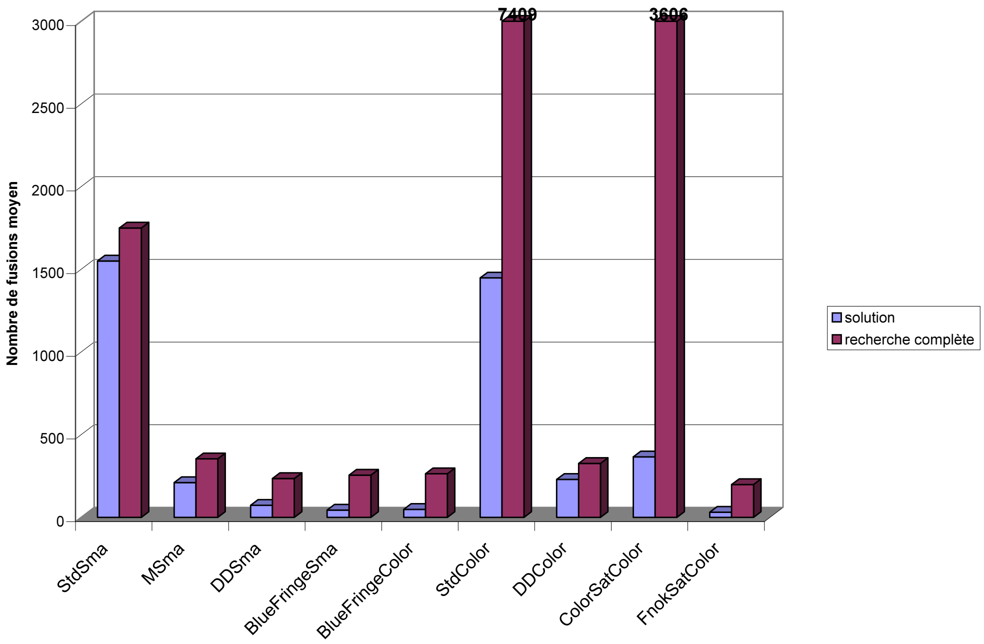


FIG. 3.9 – *Recherche complète avec borne supérieure par algorithme glouton*

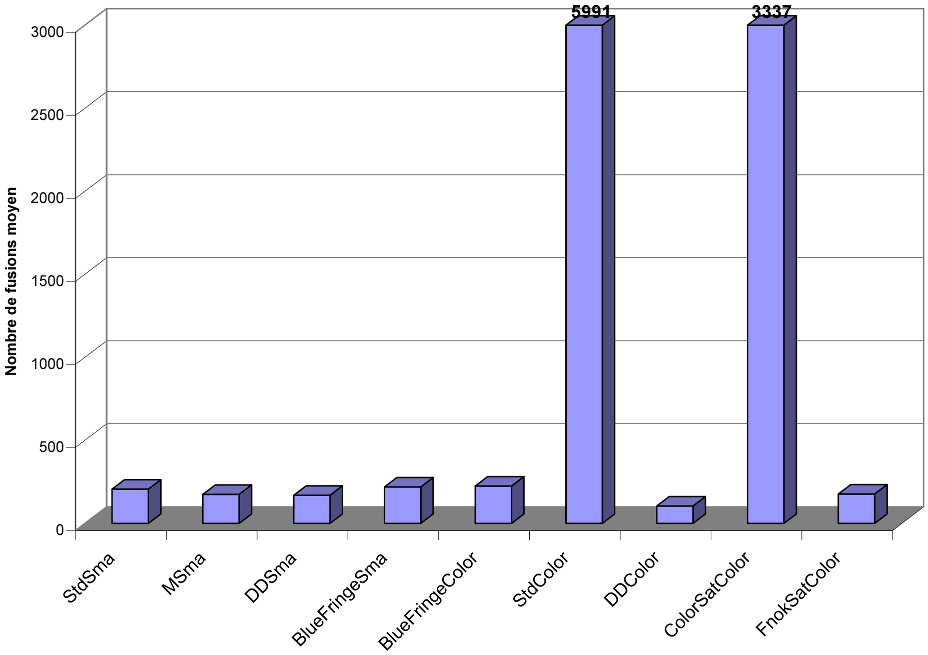


FIG. 3.10 – *Vérification de la taille*

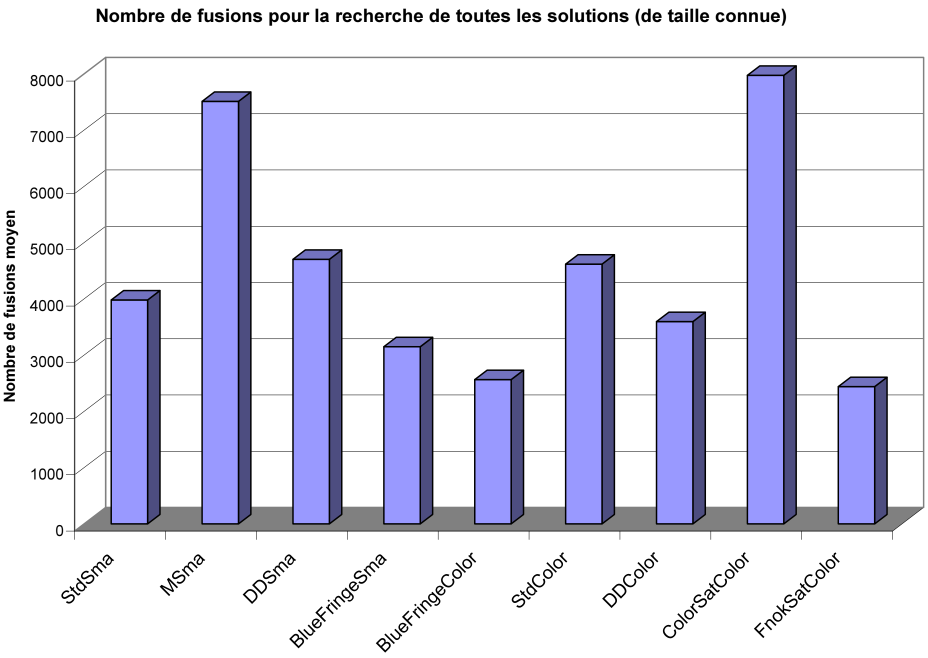


FIG. 3.11 – *Recherche de toutes les solutions (de taille connue)*

3.3.2.4 Étude des trois meilleures stratégies

Après avoir considéré la recherche complète de solutions, nous proposons d'étudier les limites d'applicabilité des meilleures méthodes pour des problèmes de plus grande taille. Pour avoir une idée du comportement de ces méthodes dans le cas où l'espace de recherche est trop grand, nous avons limité le nombre de fusion au nombre volontairement faible de 200000 fusions et nous avons observé les résultats en fonction du nombre d'exemples disponibles, sur le jeu de test généré par Gowachin. Au vu des expérimentations précédentes, trois stratégies ont été sélectionnées : *BlueFringeColor*, *FnokSatColor* et *DDSma*. Comme dans la deuxième expérimentation sur le jeu de données de Dupont, une recherche gloutonne de la borne supérieure de la taille de l'automate a été effectuée avant de lancer la recherche.

En ce qui concerne le nombre de recherches complétées (*Complet*, figure 3.12), les résultats obtenus par *BlueFringeColor* et *FnokSatColor* sont comparables, avec un léger avantage à *BlueFringeColor* ; alors que *DDSma* ne complète la recherche que dans quelques cas. Nous avons également fait figurer sur le graphe, le nombre de recherches incomplètes mais pour laquelle une solution de taille inférieure à la taille proposée par l'algorithme glouton a été trouvée (*Meilleur*, figure 3.12). Les résultats obtenus semblent alors plus intéressants, mais doivent être relativisés par la taille de l'automate trouvé qui est alors proche de la taille proposée par l'algorithme glouton (voir figure 3.13, Taille des solutions). Les résultats obtenus sont toutefois légèrement favorables à *DDSma*.

La faible différence entre le premier résultat trouvé et la solution trouvée en un temps limité indique que l'algorithme n'a pas pu revenir sur un choix crucial effectué tôt dans l'arbre de recherche et plaide pour des parcours remettant en cause plus vite les choix peu informés effectués au début de la recherche. Cette courbe et la courbe du nombre de fusions nécessaires permettent aussi d'illustrer la difficulté du problème abordé en fonction du nombre d'exemples disponibles.

On peut notamment remarquer que la complexité de l'exploration de l'espace de recherche ne croît pas suivant le nombre d'exemples. Au contraire, la complexité de l'exploration diminue au fur et à mesure que l'échantillon devient de plus en plus informé. On peut également remarquer que cette complexité n'est pas uniquement corrélée à la taille de la borne supérieure, elle dépend aussi du nombre de solutions dans le treillis, comme le montre la complexité de la recherche pour les échantillons de 50 états alors que la borne trouvée par l'algorithme glouton est proche de l'optimum.

Dans ce cas peu informé, trouver une solution de petite taille compatible avec les données d'apprentissage sera relativement facile, par contre, l'identification d'un automate cible sera particulièrement difficile.

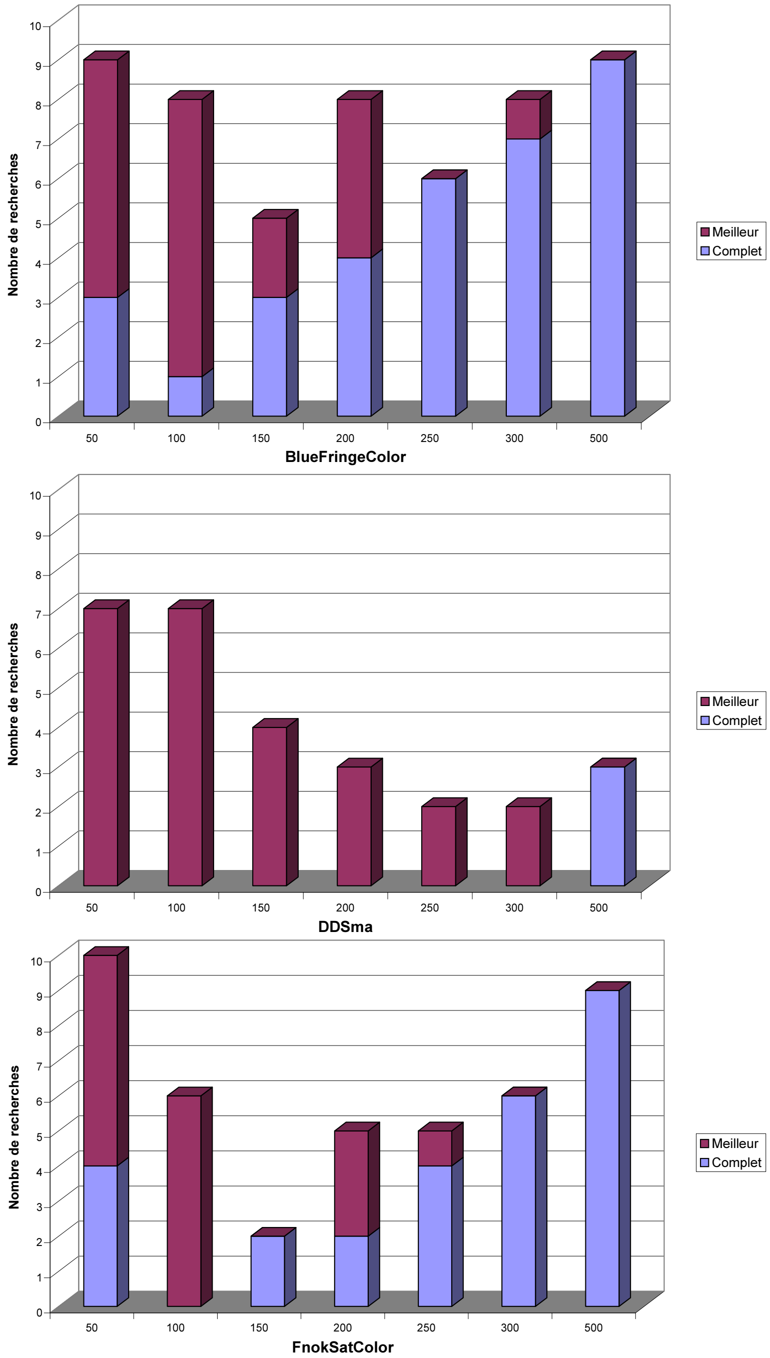


FIG. 3.12 – *Complétude de la recherche ou amélioration de la solution*

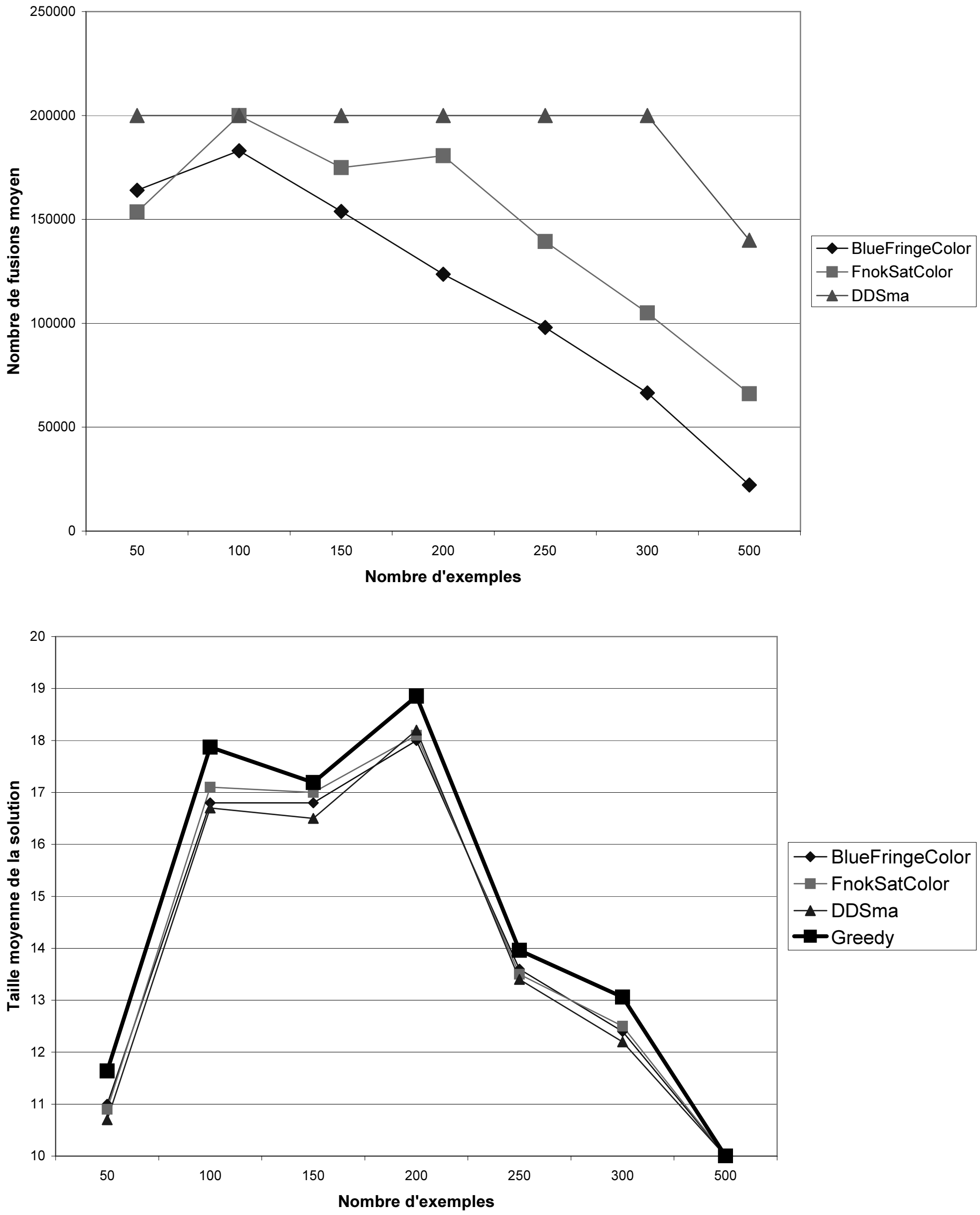


FIG. 3.13 – *Nombre de fusions et solution obtenue*

Dans cette section, nous avons étudié plusieurs stratégies d'ordonnancement pour le problème difficile de la recherche exacte de solutions de taille minimale. L'ordonnancement des états constitue un des facteurs principaux d'efficacité des algorithmes.

L'expérimentation présentée permet de mettre en évidence les caractéristiques fondamentales de chacun des principaux types d'ordonnancement. On peut ainsi préconiser trois types d'ordonnancement en fonction des applications considérées. Nous disposons alors de quatre algorithmes pouvant être utilisés séparément pour chacune de leur tâche de prédilection ou pouvant être enchainés pour une recherche efficace de la (ou les) solution(s) minimale(s) :

1. Pour la recherche d'une borne supérieure, la stratégie *DDSma* permet d'atteindre rapidement une bonne solution (on peut utiliser une version gloutonne ou plus exhaustive grâce à la détection de cliques).
2. Pour compléter l'exploration, on choisira une extension à la recherche complète de l'approche Blue Fringe. Le choix entre la déclinaison *BlueFringeSma* ou *BlueFringeColor* dépend de qualité supposée de la borne supérieure obtenue. Pour une bonne borne supérieure, *BlueFringeColor* est parfaitement adapté, alors que *BlueFringeSma* est plus coûteux mais permet d'escompter une diminution plus rapide de la valeur de la borne supérieure.
3. Pour vérifier la minimalité d'une solution ou pour la recherche de l'ensemble des solutions minimales, l'approche *FnokSatColor* est alors la stratégie la plus indiquée.

L'étude a permis de dégager ces trois types d'ordonnancement en fonction du but visé. Il reste cependant beaucoup de paramètres à étudier pour obtenir le meilleur algorithme : par exemple le choix de la clique initiale pour le coloriage, l'influence de l'approche Blue-Fringe et du paramètre *window* ou les possibilités de dégradation de la complétude de la propagation des incompatibilités. Il est également essentiel de réaliser des implémentations optimisées pour comparer en termes temporels les différentes approches, et vérifier que le surcoût de travail effectué n'est pas supérieur au temps qui aurait été passé à explorer l'espace de recherche ainsi élagué (particulièrement pour des automates de petite taille).

Ce type d'étude à effectuer reste toutefois tributaire du type de données traitées. Nous garderons à l'esprit que ces expériences ont été réalisées sur des jeux de test artificiels. Bien que ceux-ci soient généralement considérés comme plus difficiles, les résultats obtenus sur ce genre de jeux de test ne permet que de se faire une idée du comportement des programmes pour des données réelles. Il est à présent souhaitable de tester et confronter les programmes et stratégies de recherche sur les différents types de données réelles disponibles.

Conclusion et perspectives

Dans cette étude, nous avons introduit et étudié l'inférence d'automates classifieurs par fusion d'états. Considérer cette extension des automates permet d'identifier et utiliser la notion d'univocité (c'est à dire la classification des mots dans, au plus, une classe) comme moyen de limiter la sur-généralisation lors de l'apprentissage. Au cours de cette étude, nous nous sommes donc plus particulièrement intéressé à l'inférence d'automates classifieurs univoques. Le problème étudié peut alors être considéré, suivant les points de vue, comme l'apprentissage d'une fonction de classification "rationnelle" discriminante, comme l'inférence k-régulière d'un ensemble de langages deux à deux disjoints ou comme l'identification d'un automate compatible à partir d'instances positives et négatives.

Nous avons choisi d'aborder ici le problème suivant l'approche, classique en inférence grammaticale, par fusions d'états. Notre premier apport à cette approche a été l'extension de l'espace de recherche des automates aux automates classifieurs, illustrant ainsi la différence entre l'approche acceptation et l'approche classification. Nous avons notamment pu proposer une définition de la complétude structurelle pour l'approche classification, nécessitant potentiellement moins d'exemples que l'approche acceptation pour définir l'espace de recherche. Nos autres apports à l'approche par fusion concernent plus particulièrement l'inférence d'automates classifieurs univoques. Dans ce cadre, le choix de la représentation par automates classifieurs utilisée permet alors d'identifier efficacement l'ensemble des paires d'états incompatibles, c'est à dire qui ne doivent pas être fusionnées. Nous avons alors proposé d'étendre l'approche par fusion en introduisant également l'opération duale de différenciation, pour la prise en compte des fusions impossibles, et nous avons introduit un nouvel espace de recherche implicite sur les fusions (et différenciations) de paires d'états. Nous avons caractérisé dans cet espace la dynamique de l'ensemble des états incompatibles et avons proposé des algorithmes efficaces pour le maintien de cet ensemble. Dans le cas de l'inférence de k-DFA univoques, cette étude nous a permis de reformuler le problème sous la forme d'un problème de satisfaction de contraintes, dépendant uniquement de l'automate initial.

Nous avons alors étudié le problème de la recherche exacte dans cet espace d'une solution minimale. Pour ce problème d'optimisation sous contraintes, nous avons exhibé deux schémas algorithmiques, du type *Branch and Bound*, offrant un cadre général unifié pour exprimer, dans l'espace implicite, l'extension aux automates classifieurs des approches par fusions existantes, et présentant l'originalité de pouvoir prendre en compte les fusions impossibles. L'application de ces schémas nous a permis d'implémenter des algorithmes efficaces pour la recherche complète des solutions minimales et nous a permis de comparer expérimentalement l'influence du choix de l'ordonnancement des attributs pour cette tâche. D'après nos expérimentations, les algorithmes par coloriage semblent plus appropriés, les stratégies les plus efficaces étant alors du type Blue-Fringe ou par saturation des incompatibilités.

L'ensemble de ces travaux permettent d'établir les bases de l'inférence d'automates classifieurs univoques, mais de nombreuse pistes de recherches restent à envisager.

Tout d'abord, les algorithmes proposés n'épuisent pas l'ensemble des études possibles sur le sujet. En particulier, d'autres stratégies de propagation des contraintes, éventuellement moins complètes mais avec intégration de retour en arrière intelligent, pourraient être considérées. Un grand nombre d'heuristiques d'ordonnancement alternatives peuvent aussi être imaginées. Globalement, l'interdépendance entre élagage, propagation des contraintes et ordonnancement du choix des attributs reste à étudier finement. Cependant, même en améliorant ces algorithmes, étant donné la nature du problème, la recherche exacte de solutions restera cantonnée à des automates de "petite" taille, à moins de disposer d'échantillons d'apprentissage très informant. Pour des problèmes de taille plus conséquente, il faudra envisager de n'explorer qu'une partie de l'espace de recherche, sans garantie d'optimalité. Une bonne heuristique étant disponible (EDSM), les techniques de recherche à divergence limitée (*Limited discrepency search*) et en profondeur d'abord entrelacée (*Interleaved depth-first search*) semblent alors parfaitement indiquées. Mais de nombreuses autres méthodes peuvent être envisagées. Nous n'avons notamment pas exploité ici toutes les propriétés du nouvel espace de recherche par fusion introduit. Sa relation de voisinage simple, la représentation économique d'un élément du treillis permettant d'envisager efficacement une population de solutions et la possibilité d'établir des scores sur le nombre de contraintes violées ou sur les propriétés de propagation de contraintes doivent pouvoir être exploitées et semblent, par exemple, particulièrement adaptées à une exploration de l'espace par des méthodes méta-heuristiques du type algorithmes génétiques, tabou, recuit simulé ou par réparation d'erreur.

Nous avons implémenté l'ensemble des algorithmes présentés au sein d'une bibliothèque dédiée à l'inférence de machines à états finies. La conception de cette bibliothèque a été guidée par le souci de son extensibilité à d'autres types de machines. L'extension du travail sur l'univocité à d'autres types de machines semble particulièrement prometteuse. Les transducteurs sous-séquentiels [Sch77] utilisés pour le traitement de la langue naturelle [CVO93, CGV94, Moh97] sont par exemple très proches des k-DFA et peuvent être considérés comme combinant les aspects classification (émission sur les états finaux) et les aspects traduction (émission sur les transitions utilisées). Nous pensons aussi poursuivre l'étude sur l'inférence d'automates non déterministes et l'appliquer à la discrimination de séquences génétiques. D'une manière générale, le non déterminisme est une composante fondamentale pour les sciences du vivant, qui doit par conséquent être prise en compte en bioinformatique. Dans cette application, nous espérons qu'autoriser le non déterminisme nous permettra d'inférer des automates plus petits et par conséquent plus "lisibles" par les biologistes, pour fournir des indications intuitives sur le fonctionnement de certaines mécanismes de fabrication de protéines. Enfin, il serait intéressant de savoir si l'approche peut être étendue au cas stochastique.

Enfin, le dernier axe de développement qu'il nous semble nécessaire d'étudier est l'introduction de mesures qualitatives des solutions en fonction de l'application. En effet, même si la taille de l'automate trouvé est importante, ce critère ne doit pas être le seul pris en compte pour évaluer une solution et guider la recherche. Dans un cadre général, on peut étudier l'inférence pour des critères issus du domaine de la classification. On peut ainsi imaginer d'étudier l'inférence d'automates classifieurs séparant au mieux, selon un critère et une distance à définir, les exemples de l'échantillon d'apprentissage, ou alors, l'inférence d'un automate classifieur univoque minimisant le nombre de séquences non classées. Nous pensons aussi par exemple à exploiter des informations du type distance ou mesure de similarité entre les lettres de l'alphabet pour guider l'inférence. La prise en compte de ce type d'information, ainsi que le développement de techniques permettant de les utiliser au cours de l'apprentissage, permettrait alors d'utiliser les technique de l'inférence régulière sur des données symboliques pouvant correspondre à une discrétisation de valeurs numériques ou, plus simplement, d'indications de tendances.

Annexe A

Notations OMT/UML utilisées

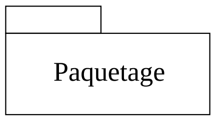
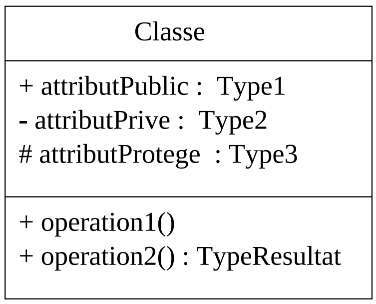
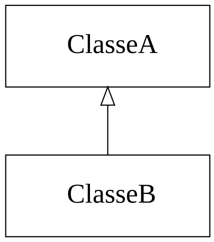
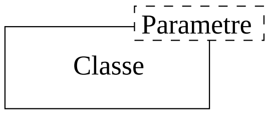
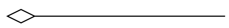
<table>
<tr><td>Paquetage</td><td>
</td></tr>
<tr><td>Classe</td><td>
</td></tr>
<tr><td>Héritage (B hérite de A)</td><td>
</td></tr>
<tr><td>Classe paramétrée</td><td>
</td></tr>
<tr><td>Lien de référence</td><td></td></tr>
</table>

Table des matières

Table des figures

Algorithmes

Index

Bibliographie

[Alq97] Alquézar (R.). – *Symbolic and connectionist learning techniques for grammatical inference.* – Thèse de PhD, Universitat Politecnica de Catalunya, mars 1997.

[Ang78] Angluin (D.). – On the complexity of minimum inference of regular sets. *Information and Control*, vol. 29, n3, 1978, pp. 741 – 765.

[Ang81] Angluin (D.). – A note on the number of queries needed to identify regular languages. *Information and Control*, vol. 51, 1981, pp. 76–87.

[Ang82] Angluin (D.). – Inference of reversible languages. *Communications of the ACM*, vol. 29, 1982, pp. 741–765.

[Ang87] Angluin (D.). – Queries and concept learning. *Machine Learning*, vol. 2, 1987, pp. 319–342.

[Ang88] Angluin (D.). – *Identifying Languages from Stochastic Examples.* – Rapport technique nYALEU/DCS/RR-614, Yale University, March 1988.

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

[AS94] Alliot (J.-M.) et Schiex (T.). – *Intelligence Artificielle & Informatique Théorique.* – Cépaduès Éditions, 1994.

[AS95] Alquézar (R.) et Sanfeliu (A.). – Incremental grammatical inference from positive and negative data using unbiased finite state automata. *In :* Shape, Structure and Pattern Recognition*, Proc. Int. Workshop on Structural and Syntactic Pattern Recognition, SSPR'94, Nahariya (Israel)*, pp. 291–300. – 1995.

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

[BBP75] Biermann (Alan W.), Baum (Richard I.) et Petry (Frederick E.). – Speeding up the synthesis of programs from traces. *IEEE Transactions on Computers*, vol. C-24, 1975, pp. 122–136.

[Ber79] Berstel (Jean). – *Transductions and Context-Free-Languages.* – Stuttgart, B.G. Teubner, 1979.

[BF72] Biermann (A. W.) et Feldmann (J. A.). – On the synthesis of finite-state machines from samples of their behaviour. *IEEE Transactions on Computeurs C 21*, 1972, pp. 592 – 597.

[BJVU98] Brazma (A.), Jonassen (I.), Vilo (J.) et Ukkonen (E.). – Pattern discovery in biosequences. *Lecture Notes in Computer Science*, vol. 1433, 1998, pp. 257–??

[Boo67] Booth (T. L.). – *Sequential Machines and Automata Theory.* – New York, John Wiley and Sons, 1967.

[Bré79] Brélaz (D.). – New methods to color the vertices of a graph. *Communications of the ACM*, vol. 22, 1979, pp. 251–256.

[CGL97] Castro (J.), Guijarro (D.) et Lavín (V.). – Learning nearly monotone k-term DNF. *In : Proceedings of the 3rd European Conference on Computational Learning Theory*, éd. par Ben-David (Shai). pp. 162–170. – Berlin, mars 17–19 1997.

[CGV94] Castellanos (A.), Galiano (I.) et Vidal (E.). – Application of OSTIA to machine translation tasks. *In : Grammatical Inference and Applications, ICGI'94*. pp. 93–105. – Springer Verlag, 1994.

[CN97a] Coste (F.) et Nicolas (J.). – Inférence grammaticale régulière : caractérisation des solutions canoniques dans l'espace des fusions. *In : 12èmes Journées Francophones sur l'Apprentissage Automatique (JFA'97).* – Roscoff, France, 1997.

[CN97b] Coste (F.) et Nicolas (J.). – Regular inference as a graph coloring problem. *In : Workshop on Grammar Inference, Automata Induction, and Language Acquisition (ICML' 97).* – Nashville, TN., USA, 1997.

[CN98a] Coste (F.) et Nicolas (J.). – How considering incompatible state mergings may reduce the dfa induction search tree. *In : Fourth International Colloquium on Grammatical Inference (ICGI-98), Ames, Iowa, USA*, éd. par Honavar (V.) et Slutzki (G.). pp. 199–210. – Berlin, juillet 1998.

[CN98b] Coste (F.) et Nicolas (J.). – Inference of finite automata: Reducing the search space with an ordering of pairs of states. *In : 10th Eutopean Conference on Machine Learning (ECML'98)*, éd. par Nédellec (C.) et Rouveirol (C.). pp. 37–42. – Chemnitz, Germany, avril 1998.

[CO94] Carrasco (R.) et Oncina (J.). – Learning stochastic regular grammars by means of a state merging method. *In : Grammatical Inference and Applications, ICGI'94*. pp. 139–150. – Springer Verlag, 1994.

[Cos99] Coste (F.). – *State merging inference of finite state classifiers.* – Rapport technique nINRIA/RR-3695, IRISA, septembre 1999.

[CVO93] Castellanos (A.), Vidal (E.) et Oncina (J.). – Language understanding and subsequential transducer learning. *In : International Colloquium on Grammatical Inference*, pp. 11/1–11/10. – 1993.

[DDG96] Denis (F.), D'Halluin (C.) et Gilleron (R.). – PAC learning with simple examples. *In : 13th Symposium on Theoretical Aspects of Computer Science, STACS'96*, pp. 231–242. – 1996.

[Den98] Denis (F.). – Apprentissage PAC par exemples positifs. *In : Journées Francophones sur l'Apprentissage, JFA'98*, pp. 86–99. – 1998.

[DM98] Dupont (P.) et Miclet (L.). – *Inférence grammaticale régulière : fondements théoriques et principaux algorithmes.* – Rapport technique, IRISA, Juillet 1998.

[DME94] Dupont (P.), Miclet (L.) et E.Vidal. – What is the search space of the regular inference? *ICGI'94, Grammatical inference and Applications*, 1994, pp. 25–37. – Springer Verlag.

[Dup96] Dupont (P.). – *Utilisation et apprentissage de modèles de langages pour la reconnaissance de la parole continue.* – Thèse de PhD, Ecole Nationale Supérieure des Télécommunications, 1996.

[FB75] Fu (K. S.) et Booth (T. L.). – Grammatical inference: Introduction and survey — part I and II. *IEEE-Transactions on Systems, Man and Cybernetics*, vol. 5, 1975, pp. 95–111 and 409–423.

[Gio95] Giordano (J.Y.). – L'inférence grammaticale comme problème d'optimisation combinatoire : application de la méthode Tabou a l'inférence de grammaires régulières. *In : Actes des 10 èmes Journées Francophones d'Apprentissage.* – avril 1995.

[Gol67] Gold (E. M.). – Langage indentification in the limit. *Information and control*, vol. 10, n 5, 1967, pp. 447 – 474.

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

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

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

[Hig96] Higuera (de la) (C). – Ensembles caractéristiques en inférence grammaticale. *Rapport interne : Lirmm*, 1996.

[Hir92] Hirsh (H.). – Polynomial-time learning with version spaces. *In : National Conference on Artificial Intelligence*, pp. 117–122. – San Jose, CA, juillet 1992.

[HOV96] Higuera (de la) (C.), Oncina (J.) et Vidal (E.). – Identification of dfa : data-dependent versus data-independant algorithms. *Grammatical inference Learning Syntax from Sentences, ICGI'96*, 1996, pp. 311–325. – Springer Verlag.

[HU80] Hopcroft (J.) et Ullman (J.). – *Introduction to Automata Theory, Languages, and Computation.* – N. Reading, MA, Addison-Wesley, 1980.

[JP98] Juillé (H.) et Pollack (J.B.). – A stochastic search approach to grammar induction. *In : Grammatical Inference.* pp. 126–137. – Springer-Verlag, 1998.

[KMT95] Koshiba (T.), Maekinen (E.) et Takada (Y.). – Learning strongly deterministic even linear languages from positive examples. *Lecture Notes in Computer Science*, vol. 997, 1995, pp. 41–??

[KMT97] Koshiba (Takeshi), Mäkinen (Erkki) et Takada (Yuji). – *Inferring pure context-free languages from positive data.* – Rapport technique nA-1997-14, Department of Computer Science, University of Tampere, 1997. a revived vesion will appear in Acta Cybernetica.

[KS88] Kudo (M.) et Shimbo (M.). – Efficient regular grammatical inference techniques by the use of partial similarities and their local relationship. *Pattern recognition*, vol. 21, 1988, pp. 401 – 409.

[KV89] Kearns (M.) et Valiant (L.G.). – Cryptographic limitation on learning boolean formulae and finite automata. *In : Proceedings of the Twenty First Annual Symposium on Theory of Computing*, pp. 433 – 444. – may 1989.

[KVBSV94] Kam (Timothy), Villa (Tiziano), Brayton (Robert) et Sangiovanni-Vincentelli (Alberto). – A fully implicit algorithm for exact state minimization. *In : Proceedings of the 31st Conference on Design Automation*, éd. par Lorenzetti (Michael). pp. 684–690. – New York, NY, USA, juin 1994.

[Lan92] Lang (K. J.). – Random dfa's can be approximately learned from sparse uniform examples. *5th ACM workshop on Computation Learning Theorie*, 1992, pp. 45 – 52.

[Lan97] Lang (K.). – *Merge Order Counts.* – Rapport technique, NEC Research Institute, September 1997.

[LPP98] Lang (K. J.), Pearlmutter (B. A.) et Price (R. A.). – Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science*, vol. 1433, 1998, pp. 1–12.

[LV91] Li (Ming) et Vitányi (Paul M. B.). – Learning simple concepts under simple distributions. *SIAM Journal on Computing*, vol. 20, n 5, octobre 1991, pp. 911–935.

[Mäk94] Mäkinen (Erkki). – *A note on the grammatical inference problem for even linear languages.* – Rapport technique n A-1994-9, Department of Computer Science, University of Tampere, 1994. revised version appeared in Fundamenta Informaticae 25 (1996) 175–181.

[MdG94] Miclet (L.) et de Gentille (C.). – Inférence grammaticale à partir d'exemples et de contre-exemples : deux algorithmes optimaux : (big et rig) et une version heuristique (brig). *JAVA94, Journées Acquisition, Validation, Apprentissage*, 1994, pp. F1–F13. – Strasbourg France.

[Mic80] Miclet (L.). – Regular inference with a tail-clustering method. *IEEE Transactions on SMC*, vol. SMC-10, 1980, pp. 737 – 743.

[Mit78] Mitchell (T.). – *Version Spaces : an approach to concept learning.* – Thèse de PhD, Standford University, dec 1978, 357–365p.

[Mäk99] Mäkinen (Erkki). – *Inferring finite transducers.* – Report, Department of Computer Science University of Tampere, Finland, 1999.

[Moh97] Mohri (Mehryar). – Finite-state transducers in language and speech processing. *Computational Linguistics*, vol. 23, n4, 1997.

[Nic93] Nicolas (J.). – Une représentation efficace pour les espaces de versions. *In : 8ièmes JFA93.* – Saint-Raphael, mars 1993.

[OE96] Oliveira (Arlindo L.) et Edwards (Stephen). – Limits of exact algorithms for inference of minimum size finite state machines,. *In : 7th International Workshop in Algorithmic Learning Theory, Lecture Notes in Artificial Intelligence 1160.* – 1996.

[OG92] Oncina (J.) et Garcia (P.). – Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis*, 1992, pp. 49 – 61.

[OGV93] Oncina (J.), García (P.) et Vidal (E.). – Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 15, 1993, pp. 448 – 458.

[Onc98] Oncina (J.). – The data driven approach applied to the ostia algorithm. *In : Fourth International Colloquium on Grammatical Inference (ICGI-98), Ames, Iowa, USA*, éd. par Honavar (V.) et Slutzki (G.). pp. 50–56. – Berlin, juillet 1998.

[OS98] Oliveira (A. L.) et Silva (J. P. M.). – Efficient search techniques for the inference of minimum size finite automata. *In : South American Symposium on String Processing and Information Retrieval.* – 1998.

[OV96] Oncina (J.) et Varó (M. A.). – Using domain information during the learning of a subsequential transducer. *In : Proceedings of the Third International Colloquium on Grammatical Inference (ICGI-96): Learning Syntax from Sentences*, éd. par Miclet (Laurent) et de la Hieguera (Colin). pp. 301–312. – Berlin, septembre 25–27 1996.

[PH97] Parekh (R.) et Honavar (V.). – Learning DFA from simple examples. *In : Workshop on Automata Induction, Grammatical Inference, and Language Acquisition, ICML-97.* – 1997.

[Pit89] Pitt (L.). – Inductive inference, DFA's, and computational complexity. *In : Analogical and Inductive Inference*, éd. par Jantke (K.P.), pp. 18–44. – Berlin, Springer-Verlag, 1989.

[PO98] Pena (J. G.) et Oliveira (A. L.). – A new algorithm for the reduction of incompletely specified finite state machine. *In : International Conference on Computer Aided Design, San Jose, CA.* – 1998.

[Pro93] Prosser (P.). – Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, vol. 3, n9, 1993, pp. 245 – 250.

[PV98] Picó (D.) et Vidal (E.). – Transducer-learning experiments on language understanding. *In : Fourth International Colloquium on Grammatical Inference (ICGI-98), Ames, Iowa, USA*, éd. par Honavar (V.) et Slutzki (G.). pp. 138–149. – Berlin, July 1998.

[PW89] Pitt (L.) et Warmuth (M.). – The minimum consistent DFA problem cannot be approximated within any polynomial. *In : 21st ACM Symposium on Theory of Computing*, pp. 421–444. – 1989.

[Ros67] Rosenberg (Arnold L.). – A machine realization of the linear context-free languages. *Information and Control*, vol. 10, n2, février 1967, pp. 175–188.

[RS97] Roche (Emmanuel) et Schabes (Yves) (édité par). – *Finite-State Language Processing.* – Cambridge, Massachusetts, The MIT Press, 1997.

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

[RV84] Richetin (M.) et Vernadat (F.). – Efficient regular grammatical inference for pattern recognition. *pattern recognition*, vol. 17, n 2, 1984, pp. 245 – 250.

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

[Sak90] Sakakibara (Y.). – *An Efficient Robust Algorithm for Learning Decision Lists.* – Rapport technique nNo. 105, International Institute for Advanced Study of Social Information Science, Fujitsu Laboratories Ltd, août 1990.

[Sak92] Sakakibara (Y.). – Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, vol. 97, 1992, pp. 23–60.

[Sak97] Sakakibara (Yasubumi). – Recent advances of grammatical inference. *Theoretical Computer Science*, vol. 185, n1, 10 octobre 1997, pp. 15–45.

[SBH+94] Sakakibara, Brown, Hughey, Mian, Sjolander, Underwood et Haussler. – Recent methods for RNA modeling using stochastic context-free grammars. *In : CPM: 5th Symposium on Combinatorial Pattern Matching.* – 1994.

[Sch61] Schützenberger (M. P.). – A remark on finite transducers. *Information and Control*, vol. 4, 1961, pp. 185–196.

[Sch77] Schützenberger (M. P.). – Sur une variante des fonctions sequentielles. *Theoretical Computer Science*, vol. 4, n1, février 1977, pp. 47–57.

[Seb94] Sebag (M.). – Une approche par contraintes de l'espace des versions. *In : RFIA'94*, pp. 17–25. – 1994.

[SG94] Sempere (J. M.) et García (P.). – A characterization of even linear languages and its application to the learning problem. *In : Proceedings of the Second International ICGI Colloqium on Grammatical Inference and Applications*, éd. par Carrasco (Rafael C.) et Oncina (Jose). pp. 38–44. – Berlin, septembre 1994.

[SN98] Sempere (José M.) et Nagaraja (G.). – Learning a subclass of linear languages from positive structural information. *In : Proceedings of the 4th International Colloquium on Grammatical Inference (ICGI-98)*, éd. par Honavar (V.) et Slutzki (G.). pp. 162–174. – Berlin, juillet 12–14 1998.

[Tak88] Takada (Y.). – Grammatical inference for even linear languages based on control sets. *Information Processing Letters*, vol. 28, n4, 1988, pp. 193–199.

[Tak94] Takada (Y.). – A hierarchy of language families learnable by regular language learners. *Lecture Notes in Computer Science*, vol. 862, 1994, pp. 16–??

[Tak95] Takada (Yuji). – A hierarchy of language families learnable by regular language learning. *Information and Computation*, vol. 123, n1, 15 novembre 1995, pp. 138–145.

[Tar72] Tarjan (Robert Endre). – *On the Efficiency of a Good but Not Linear Set Union Algorithm.* – Technical Report n TR72-148, Cornell University, Computer Science Department, novembre 1972.

[Tar75] Tarjan (R. E.). – Efficiency of a good but not linear set merging algorithm. *Journal of the ACM*, vol. 22, 1975, pp. 215–225.

[TB73] Trakhenbrot (B.) et Barzdin (Ya.). – Finite automata : Behavior and synthesis. *Amsterdam, North Holland Pub. Comp*, 1973.

[Tsa93] Tsang (E.). – *Foundations of Constraint Satisfaction.* – London, Academic Press, 1993.

[Val84] Valliant (L.). – A theory of the learnable. *Comunication of the ACM*, vol. 27, n11, 1984, pp. 1134 – 1142.

[VR84] Vernadat (F.) et Richetin (M.). – Regular inference for syntactic pattern recognition: a case study. *In : 7th Conf. on Pattern Recognition*, pp. 1370–1372. – 1984.

[Wat94] Watson (B. W.). – *A taxonomy of finite automata minimization algorithms.* – Report, Department of Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands, 1994.

[WT89] Westbrook (Jeffery) et Tarjan (Robert E.). – Amortized analysis of algorithms for set union with backtracking. *SIAM Journal on Computing*, vol. 18, n1, février 1989, pp. 1–11.

[Yok95] Yokomori (T.). – On polynomial-time learnability in the limit of strictly deterministic automata. *Machine Learning*, vol. 19, 1995, pp. 153–179.

Résumé

L'apprentissage automatique occupe aujourd'hui une place majeure au sein de l'Intelligence Artificielle. Un grand intérêt est notamment accordé à l'apprentissage par induction qui consiste à inférer des concepts à partir d'exemples. La plupart des méthodes d'apprentissage travaillent sur une description des objets sous la forme d'un vecteur de valeurs d'attributs. Pour traiter la classification de suites de symboles, nous introduisons et étudions l'apprentissage par induction d'automates classifieurs de séquences. Cette inférence peut être formellement définie comme une extension du problème d'inférence régulière et permet de prendre en compte l'interaction entre les langages au cours de l'apprentissage. Nous nous sommes notamment intéressés à l'inférence de langages réguliers deux à deux disjoints, ce qui inclut, en particulier, la généralisation d'un langage limitée par des exemples négatifs. Le problème peut alors être posé en termes d'inférence d'automates classifieurs univoques par fusion d'états et la représentation choisie peut alors être utilisée pour détecter efficacement l'ensemble des paires d'états incompatibles, c'est à dire ne devant pas être fusionnées. L'introduction d'un espace de recherche implicite et la caractérisation de la dynamique de l'ensemble des paires d'états incompatibles, nous ont alors permis de reformuler le problème de l'inférence d'automates classifieurs univoques déterministes sous la forme d'un problème de satisfaction de contraintes. Si une contrainte de minimalité est ajoutée, le problème devient un problème d'optimisation sous contraintes qui peut être traité par des algorithmes du type Branch and Bound, introduisant ainsi un nouveau type d'algorithme d'inférence efficace considérant les fusions, mais aussi les fusions impossibles.

Mots-clé : Apprentissage par induction, théorie des langages, machines à états finies, classification de séquences, problème de satisfaction de contraintes, optimisation sous contraintes, espaces de recherche implicites.

Abstract

Machine Learning has become a foremost topic within the field of Artificial Intelligence. In particular, a great deal of effort has been devoted to inductive inference which consists in learning concepts from examples. Usually the objects are described by a finite set of valued attributes. To deal with sequences classification, we introduce and study the finite state classifiers inference problem. This problem may be formally defined as an extension of the regular inference problem, allowing to handle interactions between languages during the inference process. More precisely, we address here the inference of a set of disjoint languages problem, which includes, as a particular case, the compatible automata identification problem. Thanks to the use of finite state classifier, we are able to compute efficiently the set of incompatible pairs of states, i.e. the states that should not be merged to ensure compatibility. We introduce a new implicit search space allowing to characterize the dynamic of this set of incompatible states. In this search space the problem may then be stated as a constraint satisfaction problem. When looking for the smallest deterministic finite state classifier, this problem becomes a constrained optimization problem and may be tackled by Branch and Bound algorithms leading to new and efficient inference algorithm schemes considering not only the possible state mergings, but also the impossible ones.

Key-words: Inductive machine learning, formal languages, finite state machines, sequences classification, constraint satisfaction problem, constrained optimization, implicit search space.