Intégration de connaissances de domaine et de typage pour l'apprentissage d'automates.

F. Coste, D. Fredouille

IRISA, 35000 Rennes, France. {fcoste,dfredoui}@irisa.fr

Résumé : L'apprentissage d'automates est un domaine de l'inférence grammaticale consistant à apprendre des langages réguliers à partir d'exemples et éventuellement de contre-exemples. Nous présentons des méthodes permettant d'intégrer des biais d'apprentissage provenant par exemple de connaissances liées à l'application. Les algorithmes proposés permettent de traiter deux types de connaissances : des connaissances sur le domaine du langage à inférer et des connaissances de typage sur les mots de ce langage.
Mots-clés : Inférence grammaticale, automates, biais d'apprentissage, domaine et typage

1 Motivations et prérequis

1.1 Motivations

L'inférence grammaticale régulière a pour but l'apprentissage de langages formels réguliers. Le langage recherché est appelé *langage cible*. Classiquement, les données disponibles sont des mots appartenant au langage cible. Ces mots sont nommés *exemples* et sont regroupés dans l'ensemble nommé *échantillon positif* et noté S_+. On dispose parfois également de mots n'appartenant pas au langage cible, les *contre-exemples*, ceux-ci sont regroupés dans l'ensemble nommé *échantillon négatif* et noté S_-.

Le problème d'apprentissage peut alors se formuler comme la recherche d'un langage L parmi ceux contenant tous les exemples (i.e. $S_+ \subseteq L$) et ne contenant aucun contre-exemple (i.e. $S_- \cap L = \emptyset$). Un tel langage est dit *compatible*.

Les algorithmes usuels d'inférence grammaticale régulière utilisent les automates comme représentation des langages, ces automates sont généralement déterministes ((Oncina & García, 1992; Carrasco & Oncina, 1994; Lang *et al.*, 1998)), mais plus récemment des algorithmes inférant des automates non déterministes se sont développés (Coste & Fredouille, 2000; Coste & Fredouille, 2001; Denis *et al.*, 2000; Denis *et al.*, 2001).

Certains de ces algorithmes possèdent des propriétés d'identification du fait qu'ils retrouvent un automate représentant le langage cible si le couple $\langle S_+,S_- \rangle$ inclus un ensemble particulier appelé *échantillon caractéristique* (Oncina & García, 1992; Denis *et al.*, 2001). Néanmoins, on ne peut être assuré de la présence d'un échantillon

caractéristique dans S_+ et S_-. C'est pourquoi, il est important d'intégrer toutes les connaissances disponibles afin de favoriser la convergence des algorithmes. Ces connaissances peuvent venir par exemple d'un expert du domaine d'application, ou du résultat d'autres algorithmes d'apprentissage.

On formalisera deux types de connaissances pouvant être intégrées dans l'algorithme d'apprentissage : des connaissances sur le langage cible que nous qualifierons de *langagières* et des connaissances plus *sémantiques* données par un *typage* sur les mots. Du point de vue langagier, on proposera un cadre permettant de spécifier un sur-ensemble des mots pouvant appartenir au langage, ce qui peut être vu comme la spécification d'un *domaine d'apprentissage* (section 2). Les informations sémantiques fournies par le typage seront traitées par la disponibilité d'un *automate de type*. Nous reprenons alors le formalisme proposé par (Kermorvant & Higuera (de la), 2002), et proposons de prendre en compte des typages potentiellement complexes (section 3).

1.2 Prérequis

Définitions : Soit Σ un alphabet fini non vide de symboles appelés *lettres*, on note Σ^* l'ensemble des mots sur Σ. Un langage L sur l'alphabet Σ est un sous ensemble de Σ^*. Le mot vide sera noté ϵ. La longueur d'un mot $w \in \Sigma^*$ en nombre de lettres est notée $|w|$. Pour toute fonction f, on désignera son domaine de définition par $\mathcal{D}(f)$.

Nous considérons les algorithmes d'inférence de langages réguliers utilisant comme représentation les automates :

Définition 1
(Automate) *un* automate non déterministe *est un quintuplet* $A = \langle \Sigma, Q, I, \delta, F \rangle$ *tel que :* Σ *est l'*alphabet d'entrée, Q *est un ensemble fini d'états,* $I \subseteq Q$ *est l'ensemble des* états initiaux, δ *est la* fonction de transition *de l'automate définie de* $Q \times \Sigma$ *vers* 2^Q*, un triplet* $\langle q, a, q' \rangle$ *avec* $q' \in \delta(q,a)$ *est appelé* transition, F *est l'ensemble des* états finals.

On généralise classiquement la fonction δ *aux mots et aux ensembles d'états, soit de* $2^Q \times \Sigma^*$ *vers* 2^Q *par :* $\forall Q' \subseteq Q$, $\forall w \in \Sigma^*$, $\forall a \in \Sigma$, $\delta(Q',\epsilon) = Q'$, $\delta(Q',a) = \bigcup_{q \in Q'} \delta(q,a)$ *et* $\delta(Q',wa) = \bigcup_{q \in Q'} \delta(\delta(\{q\},w),a)$.

On notera la taille en nombre d'états d'un automate A par $|A|$. Le langage reconnu par un automate A, noté $L(A)$, est défini par $L(A) = \{w \mid \delta(I,w) \cap F \neq \emptyset\}$. Un automate est dit *déterministe* si $|I| \leq 1$ et $\forall\, q \in Q$, $\forall\, a \in \Sigma$, $|\delta(q,a)| \leq 1$. La classe de langages représentée par les automates (déterministes ou non) est la classe des langages réguliers. Pour un langage régulier L donné, il existe un unique automate déterministe minimal, appelé *automate canonique* et noté $A(L)$, reconnaissant ce dernier.

Algorithmes par fusion d'états : L'approche la plus utilisée afin de réaliser l'inférence d'automates est celle dite par *fusion d'états*. Elle a été appliquée tant pour l'apprentissage d'automates que pour l'inférence de transducteurs sous-séquentiels ou d'automates probabilistes. Cette approche est basée sur l'opérateur dit de *fusion d'états* consistant à unifier deux états d'un automate (ce qui correspond à l'unification de non terminaux pour l'inférence de grammaires). Cette opération augmente le nombre de séquences acceptées et est donc une opération de généralisation. Les algorithmes par fusion sont pour la plupart de type glouton, leur principe est décrit par le canevas donné al-

Algorithm 1 Principe des algorithmes par fusion.

1: **Fonction** *algorithme-glouton-par-fusion*(échantillon S_+)
2: Soit A le PTA(S_+) ou le MCA(S_+)
3: **tant que** *choix-de-deux-états*(A,q_1,q_2) **faire**
4: **essayer** $A' \leftarrow$ *fusion*(A,q_1,q_2)
5: **si exception**(automate non compatible) **alors** /* ne rien faire */
6: **sinon** $A \leftarrow A'$
7: **retourne** A

gorithme 1. Ceux-ci commencent par construire un automate reconnaissant exactement les exemples, il s'agit du PTA dans le cas de l'inférence d'automates déterministes, ou du MCA dans le cas de l'inférence d'automates non déterministes (Dupont *et al.*, 1994). On réalise ensuite itérativement des fusions, la généralisation étant arrêtée par la compatibilité de l'automate avec les contre-exemples. La recherche peut être restreinte aux automates déterministes (resp. non ambigus) en ajoutant l'appel à une procédure de *fusion pour déterminisation* (resp. *pour désambiguisation*) après l'appel de chaque fusion (Dupont *et al.*, 1994; Coste & Fredouille, 2001).

Deux fonctions utilisées dans ce type d'algorithme permettent l'intégration de biais d'inférence. Premièrement dans l'heuristique (fonction *choix-de-deux-états*), celle-ci permet en effet de se diriger dans l'espace de recherche. Deuxièmement en interdisant certaines fusions (l'exception "automate non compatible"), ce qui permet de restreindre l'espace de recherche. Les techniques que nous proposons sont inspirées de (Coste & Fredouille, 2000) et travaillent dans ce deuxième cadre : elles restreignent l'espace de recherche à la sous-partie ne contenant que des automates ne violant pas les connaissances existant sur l'automate cible.

2 Intégrer des informations de domaine

Une première façon naturelle de fournir de l'information à l'algorithme d'inférence est de forcer le langage inféré, noté par la suite L, à être inclus dans un langage plus général, noté L_G. En pratique ce langage plus général peut être défini par la connaissance d'un *domaine* maximal, i.e. la connaissance de l'ensemble des séquences dont la classification a un sens. Ce point est illustré par l'exemple 1. Ce langage peut également être vu comme un langage trop général obtenu à partir d'un expert où d'une procédure d'apprentissage, le but de l'apprentissage est alors de spécialiser ce langage.

Exemple 1
(Exemple tiré de (Kermorvant & Higuera (de la), 2002)) Dans le cadre de l'inférence d'équations booléennes définies sur l'alphabet $\Sigma = \{A,B,\ldots, \wedge , \vee ,(,),\neg\}$*, toutes les séquences de* Σ^* *ne sont pas des équations booléennes. Or, la syntaxe des équations booléennes est connue. Introduire celle-ci dans le processus d'inférence permet d'éviter l'inférence de modèles reconnaissant des mots contenant par exemple le facteur '*$\neg)$*'.*

L'utilité de l'introduction de cette connaissance est d'autant plus importante que les contre-exemples fournis dans S_- sont a priori également des équations booléennes, et donc qu'aucun contre-exemple n'indique que la suite '$\neg$)' est incorrecte.

Apprendre un langage inclus dans L_G peut également être vu comme l'inférence d'un langage étant donné un ensemble (éventuellement infini) de contre-exemples : ces contre-exemples sont les mots qui ne sont pas dans L_G. Selon ce point de vue, on dispose d'un langage de contre-exemples $L_- = \Sigma^* \setminus L_G$ en plus du (ou à la place du) traditionnel ensemble fini de contre-exemple S_- (exemple 2).

Exemple 2
Lors de l'inférence sur des séquences protéiques, on cherche à caractériser des familles de protéines possédant une même fonctionnalité biologique. Pour certaines de ces familles, une idée de la taille des protéines à caractériser est connue. On peut ainsi ajouter comme contre-exemples toutes les séquences ne respectant pas une fourchette de taille fournie par un expert.

L'équivalence du point de vue langage plus général et langage de contre-exemples est formalisée par l'équation $L \subseteq L_G \;\Leftrightarrow\; L_- \cap L = \emptyset$. Ces différentes interprétations sont évidemment non exclusives et le langage L_G ou L_- peut être construit comme une combinaison de ces différents aspects.

Nous proposons dans cette section un algorithme permettant de traiter, lorsque les langages L_G ou L_- sont réguliers, les contraintes langagières précitées. Plus précisément, nous considérons disponible un automate A_- tel que $L(A_-) = L_- = \Sigma^* \setminus L_G$, et l'algorithme garantira $L \cap L_- = \emptyset$. Si on dispose de l'automate canonique (déterministe) $A(L_G)$, A_- peut être obtenu facilement en complétant $A(L_G)$ puis en inversant états finaux et non finaux. On peut remarquer qu'il n'est pas demandé que A_- soit déterministe, ce qui permet d'utiliser des représentations compactes pour L_- (si disponibles). En particulier, A_- peut être facilement défini comme l'union des différents automates (non déterministes) représentant différentes sources d'information sans nécessiter une déterminisation potentiellement coûteuse. D'un autre côté, remarquons que ce n'est pas vrai pour L_G puisqu'un automate non déterministe pour ce langage devra être déterminisé pour pouvoir être complémenté.

Notons A l'automate courant du processus d'inférence représentant l'hypothèse courante L. Une solution directe pour savoir si $L \cap L_-$ est vide consiste à calculer l'automate produit de A et A_-, puis à vérifier que celui-ci représente le langage vide. Bien que cette solution soit simple, elle est coûteuse en pratique. C'est pourquoi nous proposons un algorithme réalisant une simulation suffisante des calculs de l'automate intersection, utilisant une relation dite de *préfixe commun* entre les états de A et A_-.

Définition 2
(Relation de préfixe commun) *Soient* $A_1 = \langle \Sigma, Q_1, I_1, \delta_1, F_1 \rangle$ *et* $A_2 = \langle \Sigma, Q_2, I_2, \delta_2, F_2 \rangle$. *Deux états* q_1 *et* q_2 *dans* $Q_1 \times Q_2$ *sont dits en* relation de préfixe commun, *si et seulement si* $\exists w \in \Sigma^* : q_1 \in \delta_1(I_1, w)$ *et* $q_2 \in \delta_2(I_2, w)$.

On montre facilement que l'intersection de $L(A_1)$ et $L(A_2)$ est non vide si et seulement si il existe un couple d'états finals $\langle q_1, q_2 \rangle$ de $F_1 \times F_2$ en relation de préfixe

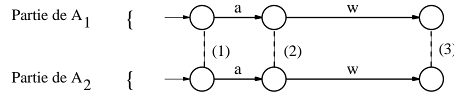

FIG. 1 – *Calcul et propagation des relations de préfixe commun : on met tout d'abord les états initiaux en relation de préfixe commun (1), puis on propage chaque relation par les transitions sortantes par une même lettre des deux états (2). Tous les états ayant un mot commun dans leur langage préfixe sont ainsi mis en relation (3).*

commun. Il est donc suffisant pour assurer la contrainte donnée par $L \cap L_- = \emptyset$ de vérifier si un état final de A est en relation de préfixe commun avec un état final de A_-. L'algorithme permettant de calculer les relations de préfixe commun entre deux automates est utilisé pour typer le MCA (ou le PTA), il est présenté au paragraphe **initialisation**. L'algorithme de maintien des relations au cours d'une inférence par fusion est donnée par le paragraphe **maintien**.

Initialisation: L'ensemble des relations de préfixe commun entre deux automates $A_1 = \langle \Sigma, Q_1, I_1, \delta_1, F_1 \rangle$ et $A_2 = \langle \Sigma, Q_2, I_2, \delta_2, F_2 \rangle$ peut être calculé par l'algorithme 2. Celui-ci consiste, comme illustré figure 1, à marquer tous les couples d'états initiaux (de $I_1 \times I_2$) en relation de préfixe commun du fait de la présence du mot ϵ dans leur langage préfixe (algorithme 2, fonction *Calcul-des-états-préfixe-commun*). On propage ensuite ces relations par les transitions sortantes par un symbole identique (algorithme 2, procédure *marque-et-propage-en-avant*).
Maintien: Une fois l'ensemble des relations entre l'automate A et A_- au départ de l'inférence calculé, on maintiendra celui-ci après chaque fusion. Notons q et q' les états de A_1 fusionnés en un état q_f, l'algorithme de maintien est appelé après la fusion. Il consiste à reporter sur le nouvel état q_f les relations qui existaient entre les états de A_2 et q ou q' et à repropager ces relations (algorithme 2, fonction *maintien-préfixe-commun-après-fusion*).

Pratiquement pour calculer et maintenir les relations de préfixe commun au cours de l'inférence, on ajoute entre les lignes 2 et 3 de l'algorithme 1 l'instruction : $\mathcal{E}_p \leftarrow$ *Calcul-des-états-préfixe-commun*(A, A_-). On ajoute également entre les lignes 5 et 6 (q_{12} désigne l'état obtenu par fusion de q_1 et q_2) l'instruction : *maintien-préfixe-commun-après-fusion*$(A, A_-, q_1, q_2, q_{12}, \mathcal{E}_p)$.

Si une relation est mise entre un état final de A_- et un de A alors on lèvera l'exception d'incompatibilité. On considérera ainsi que l'automate obtenu est incompatible avec le langage de contre-exemples L_-. Cela se traduit par l'ajout, entre les lignes 2 et 3 de la procédure *marque-et-propage-en-avant* (algorithme 2), de l'instruction : **si** $\langle q_1, q_2 \rangle \in F_1 \times F_2$ **alors lever exception**(automate non compatible)

Les complexités respective des fonctions *Calcul-des-états-préfixe-commun* et *maintien-préfixe-commun-après-fusion* sont identique soit $\mathcal{O}(|A| \times |A_-| \times t_A \times t_{A_-})$. Les variables t_A et t_{A_-} désignent respectivement le nombre de transition maximum pour une lettre donnée sortant d'un état de A et d'un état de A_-. À chaque étape de l'algorithme d'inférence (à l'initialisation où à la maintenance des relations de préfixe

Algorithm 2 Algorithmes calculant et maintenant après fusion l'ensemble $\mathcal{E}_p$ contenant les paires d'états en relation de préfixe commun.

1: **Fonction** *Calcul-des-états-préfixe-commun*(A_1,A_2)
2: Soit l'ensemble de paires d'états: $\mathcal{E}_p \leftarrow \emptyset$
3: **pour tout** $q_1 \in I_1, q_2 \in I_2$ **faire**
4: *marque-et-propage-en-avant*$(A_1,A_2,q_1,q_2,\mathcal{E}_p)$
5: **retourne** $\mathcal{E}_p$

1: **Procédure** *marque-et-propage-en-avant*$(A_1,A_2,q_1,q_2,\mathcal{E}_p)$
2: **si** $\langle q_1,q_2\rangle \notin \mathcal{E}_p$ **alors**
3: $\mathcal{E}_p \leftarrow \{\langle q_1,q_2\rangle\} \cup \mathcal{E}_p$
4: **pour tout** $a \in \Sigma, q_1' \in \delta_1(q_1,a)$, $q_2' \in \delta_2(q_2,a)$ **faire**
5: *marque-et-propage-en-avant*$(A_1,A_2,q_1',q_2',\mathcal{E}_p)$

1: **Procédure** *maintien-préfixe-commun-après-fusion*$(A_1,A_2,q,q',q_f,\mathcal{E}_p)$
2: **pour** $q_1 \in \{q,q'\}$ et q_2 tel que, $\langle q_1,q_2\rangle \in \mathcal{E}_p$ **faire**
3: $\mathcal{E}_p \leftarrow \mathcal{E}_p \setminus \langle q_1,q_2\rangle$
4: *marque-et-propage-en-avant*$(A_1,A_2,q_f,q_2,\mathcal{E}_p)$

commun après chaque fusion), il est possible que la complexité obtenue soit celle du calcul de l'automate produit de A et A_-. Néanmoins, cette complexité ne peut pas être atteinte pour toutes les étapes (comme elle le serait par l'approche naïve) car le nombre de relation est majoré par $|A| \times |A_-|$. Si nous introduisons R, le nombre de relation déjà détectées, la complexité de la fonction *maintien-préfixe-commun-après-fusion* est au pire de $\mathcal{O}((|A| \times |A_-| - R + 1) \times t_A \times t_{A_-})$. Ce qui montre une répartition de la complexité dans les différentes fusions réalisées par l'inférence.

Afin de montrer l'applicabilité de la méthode, une expérience sur données artificielle a été réalisée détaillée par la figure 2. L'expérience menée est la suivante : un automate de 31 états est généré par le serveur Gowachin (Gowachin, 1998). On fournit à RPNI un ensemble d'exemple et contre-exemple de plus en plus grand (de 20 à 3000 de 20 en 20) générés par le serveur Gowachin. À cet ensemble est ajouté un ensemble minimal d'exemples permettant d'assurer que l'automate cible est dans l'espace de recherche. On fournit également un langage L_- de plus en plus grand, ce dernier est obtenu en construisant l'automate complémentaire de l'automate cible et en choisissant avec une probabilité uniforme un sous ensemble de ses états finals de plus en plus grand (ici en considérant entre 0 et 15 états finals). Ce procédé est moyenné sur 3 échantillons aléatoires et 3 choix aléatoires pour les états finals sélectionnés (soit $3000/20 * 16 * 3 * 3 = 21600$ expériences de moins d'une minute chacune sur une machine à 1GHz).

Comme on pouvait s'y attendre, le nombre d'exemples et contre-exemples nécessaire à la convergence de l'algorithme RPNI diminue fortement dès l'introduction d'informations langagières. On remarque que cela est également vrai pour une information langagière "faible" (par exemple, le 95% de reconnaissance est atteint avec un échantillon

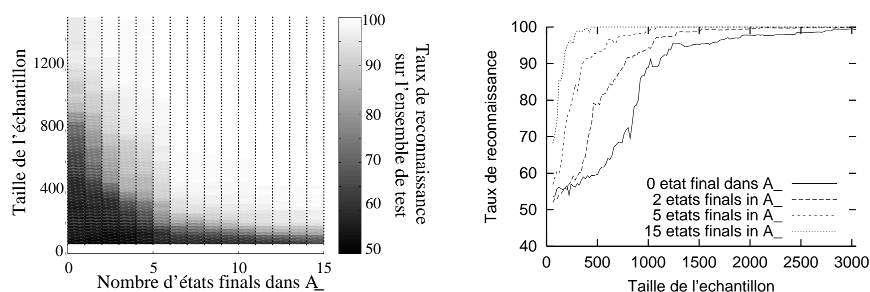

FIG. 2 – *Taux de reconnaissance sur un ensemble de test obtenu pour différentes exécutions de l'algorithme RPNI (Oncina & García, 1992) (étendu par le calcul des relations de préfixe commun), à gauche : projection de la surface 3D, à droite : sections verticales. À chaque exécution correspond une taille de l'échantillon et une quantité d'information langagière.*

de taille respectivement 740, 1000, 1160 et 1400 lorsque A_- possède respectivement 3, 2, 1 et 0 états finals). Notons néanmoins, qu'utiliser seulement des informations langagières sans fournir un échantillon de taille raisonnable ne permet pas la convergence de l'algorithme. En effet, même un langage $L_- = \Sigma^* \setminus L$ (lorsque A_- possède 15 états finals) n'est pas suffisant pour assurer la convergence. Le langage inféré est dans ce cas inclus dans le langage cible et seule la présence d'exemples supplémentaires permet la convergence.

Nous avons présenté l'introduction de connaissances langagières dans le cadre d'algorithmes d'inférence par fusions d'états. Notons que l'approche proposée est suffisamment générique pour être adaptée à d'autres algorithmes procédant par généralisation. En particulier, l'opération de généralisation de DeLeTe2 (Denis *et al.*, 2001) préservant la relation de préfixe commun, les méthodes de vérification de la compatibilité proposées sont intégrable à cet algorithme.

3 Intégrer des informations de typage

Nous avons considéré en section 2 que des informations sur le langage cible sont disponibles pour aider l'inférence. Nous considérons maintenant disponibles des informations de typage sur les lettres des mots considérés au cours de l'inférence. La notion de typage permet d'intégrer des *informations sémantiques* dans l'algorithme d'inférence : par exemple, en génomique, la structure secondaire à laquelle chaque acide aminé d'une protéine appartient peut parfois être connue et souvent prédite. Typer les exemples de protéines par ces structures secondaires permet d'inférer des automates prenant en compte celles-ci. Les composants de l'automate inféré n'auront donc plus pour seul but de représenter un langage, mais également d'associer un type (ici leur présence dans une structure secondaire particulière) aux lettres des mots acceptés. Des applications peuvent également être trouvée en linguistique, où l'on dispose d'informa-

tions d'étiquetage morpho-syntaxique (*part of speech tagging*) obtenues par un expert ou par une procédure automatique.

L'idée que nous employons ici afin d'intégrer l'information de typage au cours de l'inférence consiste à associer un type aux états de l'automate inféré, les fusions entre états de types différents étant alors interdites. On contraint ainsi non plus le langage accepté par un automate mais sa structure, cette dernière permettant de représenter la sémantique associée au typage.

Cette idée est utilisé avec succès dans (Goan *et al.*, 1996) sur des données provenant du Web afin de distinguer différents sous groupes dans les symboles de l'alphabet (chiffres, lettres, séparateurs, . . .), la mise en œuvre étant réalisée de façon ad-hoc vis à vis du problème traité. (Kermorvant & Higuera (de la), 2002) ont proposé une approche plus générique, permettant de formaliser la connaissance sur le typage dans un *automate de type*. Cette approche permet de prendre en compte une notion de *contexte* pour les lettres typées : le type d'une lettre dépend des lettres la précédant et la suivant dans un mot du langage. Néanmoins, la méthode proposée par (Kermorvant & Higuera (de la), 2002) est restreinte par plusieurs contraintes importantes sur les typages pouvant être traités : entre autre les automates de type utilisés sont obligatoirement déterministes, ce qui oblige le typage à ne dépendre que des lettres précédant la lettre typée et pas de celles lui succédant, cette méthode ne permet pas non plus de prendre en compte des typages partiels.

Nous proposons, grâce à un algorithme basé sur des techniques similaires à celles employées en section 2, un cadre formel permettant de prendre en compte des typages complexes (levant les limitations existantes pour le cadre proposé par (Kermorvant & Higuera (de la), 2002)). La sous-section 3.1 formalise la notion de typage et d'automate de type et la sous-section 3.2 présente notre algorithme. Nous détaillons ensuite deux cas particulier de fonctions de typage : la sous-section 3.3 présentera le cas où seul les exemples sont typés, et la sous-section 3.4 discutera des fonctions de typage considérées dans (Kermorvant & Higuera (de la), 2002). Nous verrons ainsi que l'inférence étant données des fonctions de typage respectant les contraintes de (Kermorvant & Higuera (de la), 2002) peut être vue comme un processus de *spécialisation d'automate*, qui implique une contrainte de spécialisation du langage reconnu.

3.1 Formalisation de la notion de typage

Afin de formaliser la connaissance disponible sur le typage des mots, nous considérons disponible au départ de l'inférence une *fonction de typage*.

Définition 3
(Fonction de typage) *Soient* Σ *et* T *deux alphabets, les symboles de* T *sont aussi nommés des* types. *Une* fonction de typage *est une fonction* f *de* $\Sigma^* \times \Sigma \times \Sigma^* \to T$ *(possiblement non totale).*

On dira du type $t = f(u,a,v)$ *qu'il est le* type du symbole a dans le contexte $\langle u,a,v \rangle$, *de* u *qu'il est le* contexte gauche *et de* v *qu'il est le* contexte droit.

Au cours de l'inférence d'automates, nous considérerons que la fonction de typage est d'une certaine manière "rationnelle". Cela signifie en pratique que nous considérerons

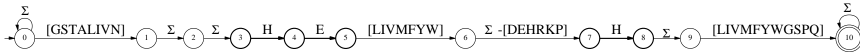

FIG. 3 – *Automate de type construit à partir du motif Prosite PS00142 (ZINC_PROTEASE) (Falquet* et al.*, 2002). Σ représente l'alphabet des acides aminés, la fonction de typage peut être définie pour retourner le numéro d'état pour les états intégrant des connaissances d'un expert du domaine. Par exemple la présence de deux acides aminés H (Histidine) dans le motif est due à ses propriétés lui permettant de fixer les molécules de Zinc.*

que celle-ci peut être réalisée par un *automate de type*. Intuitivement, un automate de type est un automate sur lequel on a ajouté des types aux états (on aurait tout aussi bien pu considérer des types sur les transitions).

Définition 4
(Automate de type) *Un* automate de type $\mathcal{T}$ *est un 7-uplet* $\langle \Sigma,Q,I,\delta,F,T,\tau \rangle$ *avec* Σ, Q, I, δ *et* F *définis de façon identique à un automate (définition 1).* T *est l'*alphabet des types *et* τ *une fonction de* $Q \rightarrow T$.

Un automate de type $\mathcal{T}$ représente la fonction $f_{\mathcal{T}}$ de $\Sigma^* \times \Sigma \times \Sigma^* \rightarrow 2^T$ définie par : $\forall u,v \in \Sigma^*, \forall a \in \Sigma,\ f_{\mathcal{T}}(u,a,v) = \{\tau(q)\ /\ q \in \delta(I,ua),\ F \cap \delta(q,v) \neq \emptyset,\ q \in \mathcal{D}(\tau)\}$. On ne considérera que les automates dont la fonction est *non ambiguë*, i.e. tels que : $\forall u,v \in \Sigma^*,\ a \in \Sigma,\ |f_{\mathcal{T}}(u,a,v)| \leq 1$, on pourra ainsi assimiler cette dernière à une fonction de typage de $\Sigma^* \times \Sigma \times \Sigma^* \rightarrow T$.

Les fonctions de typage réalisables par de tels automates permettent de prendre en compte les contextes gauche et droit, ainsi que des fonctions de typage partielles. Plus précisément, on peut typer un sous-ensemble des mots $w \in \Sigma^*$, et pour chaque mot typé seulement un sous-ensemble des contextes (i.e. des tuples $\langle u,a,v \rangle$ avec $w = uav$) puisque la fonction τ n'est pas forcément totale. Comme illustré par la figure 3, des automate de type non triviaux sont disponibles, par exemple, pour l'inférence sur des séquences protéiques.

Le problème de l'inférence d'automates étant donné une fonction de typage peut être posé comme l'inférence à la fois d'un langage et d'une fonction de typage associée. La sémantique fournie par la fonction de typage représentant les connaissances liées au domaine d'application sera donc projetée sur l'automate inféré A, qui devient un automate de type. La fonction de typage de l'automate inféré devra alors être *compatible* avec la fonction de typage fournie.

Définition 5
(Compatibilité de fonctions de typage [1]**)** *Deux fonctions de typage* f_1 *et* f_2 *seront dites* compatibles *si elles sont identiques aux points où elles sont définies. Plus formellement*

1. Notons que cette définition de compatibilité est symétrique (f_1 compatible avec f_2 est équivalent à f_2 compatible avec f_1). Nous nous différencions ici de la définition originale proposée par (Kermorvant & Higuera (de la), 2002), cette dernière ajoutant une contrainte d'inclusion entre les domaines des deux fonctions et mélangeant de ce fait une contrainte langagière et une contrainte de typage.

Algorithm 3 Algorithmes permettant de reporter les types de l'automate $\mathcal{T}$ sur l'automate inféré.

Procédure *projection-type*$(q \in Q, q_\mathcal{T} \in Q_\mathcal{T})$
si $q_\mathcal{T} \in \mathcal{D}(\tau_\mathcal{T})$ **alors**
  si $q \notin \mathcal{D}(\tau)$ **alors**
    $\tau(q) \leftarrow \tau_\mathcal{T}(q_\mathcal{T})$ /* mise à jour de τ*/
  sinon si $\tau_\mathcal{T}(q_\mathcal{T}) \neq \tau(q)$ **alors**
    lancer exception(automate non compatible)

si pour tout tuple $\langle u,a,v\rangle \in (\mathcal{D}(f_1) \cap \mathcal{D}(f_2))$ *:* $f_1(u,a,v) = f_2(u,a,v)$*. Dans le cas contraire,* f_1 *et* f_2 *seront dites* incompatibles.

Nous chercherons au cours de l'inférence à utiliser au maximum l'information donnée par la fonction de typage fournie $f_\mathcal{T}$. Cela implique que si un triplet $\langle u,a,v\rangle$ est typé par $f_\mathcal{T}$ et que uav appartient au langage inféré on considérera que $f_A(u,a,v) = f_\mathcal{T}(u,a,v)$.

3.2 Inférence étant donné un automate de type

Nous considérons dans cette section que la fonction de typage fournie à l'algorithme est représentée par un automate de type noté $\mathcal{T} = \langle \Sigma, Q_\mathcal{T}, I_\mathcal{T}, \delta_\mathcal{T}, F_\mathcal{T}, T, \tau_\mathcal{T}\rangle$. L'inférence de l'automate de type A est réalisée par les algorithmes d'inférence d'automates classiques, en y ajoutant la détection de l'incompatibilité des types avec la fonction de typage fournie $f_\mathcal{T}$. Notons que la fonction $f_\mathcal{T}$ étant non ambiguë, la fonction de typage inférée f_A sera non ambiguë sur $\mathcal{D}(f_\mathcal{T}) \cap \mathcal{D}(f_A)$, mais celle-ci sera potentiellement ambiguë sur $\mathcal{D}(f_A) \setminus \mathcal{D}(f_\mathcal{T})$. Si on désire assurer que, en plus de la compatibilité avec $f_\mathcal{T}$, la fonction de typage inférée soit non ambiguë sur tout $\mathcal{D}(f_A)$, il suffit de vérifier aussi la compatibilité de f_A avec elle-même.

La construction de la fonction τ de l'automate inféré A est réalisée par une procédure de *projection de type* entre l'automate $\mathcal{T}$ et A. Cette fonction est donnée par l'algorithme 3. Elle est appelée pour tous les couples $\langle q,q_\mathcal{T}\rangle$ de $Q \times Q_\mathcal{T}$ permettant de typer un triplet commun $\langle u,a,v\rangle$. Elle permet ainsi de prendre en compte, en projetant le type existant de $q_\mathcal{T}$ à q, le fait que tous les contextes $\langle u,a,v\rangle$ typés par $\mathcal{T}$ et tels que $uav \in L(A)$ doivent aussi être typés par A. Si la projection est impossible (q est déjà typé de façon différente de $q_\mathcal{T}$) alors l'automate inféré A et l'automate de type $\mathcal{T}$ sont incompatibles. En effet cela signifie qu'ils typeront la lettre a dans le contexte $\langle u,a,v\rangle$ de façon différente.

Afin d'appeler la procédure *projection-type* pour tous les couples d'états de A et $\mathcal{T}$ permettant le typage d'une lettre dans le même contexte, on considère les couples d'états $\langle q,q_\mathcal{T}\rangle$ à la fois en relation de préfixe commun afin de prendre en compte le contexte gauche, et de *suffixe commun* afin de prendre en compte le contexte droit.

Définition 6
(Relation de suffixe commun) *Soient* $A_1 = \langle \Sigma, Q_1, I_1, \delta_1, F_1, T, \tau_1\rangle$ *et* $A_2 =$

Algorithm 4 Algorithmes calculant et maintenant après fusion l'ensemble $\mathcal{E}_s$ contenant les paires d'états en relation de suffixe commun.

1: **Fonction** *Calcul-des-états-suffixe-commun*(A_1,A_2)
2: Soit l'ensemble de paires d'états: $\mathcal{E}_s \leftarrow \emptyset$
3: **pour tout** $q_1 \in F_1,q_2 \in F_2$ **faire**
4: *marque-et-propage-en-arrière*$(A_1,A_2,q_1,q_2,\mathcal{E}_s)$
5: **retourne** $\mathcal{E}_s$

1: **Procédure** *marque-et-propage-en-arrière*$(A_1,A_2,q_1,q_2,\mathcal{E}_s)$
2: **si** $\langle q_1,q_2\rangle \notin \mathcal{E}_s$ **alors**
3: $\mathcal{E}_s \leftarrow \{\langle q_1,q_2\rangle\} \cup \mathcal{E}_s$
4: **pour tout** $a \in \Sigma,q_1' \in \delta_1^{-1}(q_1,a)$, $q_2' \in \delta_2^{-1}(q_2,a)$ **faire**
5: *marque-et-propage-en-arrière*$(A_1,A_2,q_1',q_2',\mathcal{E}_s)$

1: **Procédure** *maintien-suffixe-commun-après-fusion*$(A_1,A_2,q,q',q_f,\mathcal{E}_s)$
2: **pour** $q_1 \in \{q,q'\}$ et q_2 tel que, $\langle q_1,q_2\rangle \in \mathcal{E}_s$ **faire**
3: $\mathcal{E}_s \leftarrow \mathcal{E} \setminus \langle q_1,q_2\rangle$
4: *marque-et-propage-en-arrière*$(A_1,A_2,q_f,q_2,\mathcal{E}_s)$

$\langle \Sigma,Q_2,I_2,\delta_2,F_2,T,\tau_2\rangle$. *Deux états* q_1 *et* q_2 *dans* $Q_1 \times Q_2$ *sont dits en* relation de suffixe commun *si et seulement si* $\exists w \in \Sigma^*\ : F_1 \cap \delta_1(q_1,w) \neq \emptyset$ *et* $F_2 \cap \delta_2(q_2,w) \neq \emptyset$.

La relation de préfixe commun assure l'existence d'un mot ua[2] commun aux langages préfixe des deux états, et la relation de suffixe commun assure l'existence d'un mot v commun aux langages suffixe des deux états.

Les algorithmes de calcul et maintien après fusion de l'ensemble des relations de suffixe commun sont similaires et de complexités identiques aux algorithmes présentés pour la relation de préfixe commun. Le principe est le même si ce n'est que les relations initiales sont mise entre les états finals, et la propagation est réalisée par les transitions entrantes (algorithme 4). Comme pour la relation de préfixe commun, on calcule ces relations en deux temps. À l'initialisation l'instruction $\mathcal{E}_s \leftarrow$ *Calcul-des-états-suffixe-commun*$(A,\mathcal{T})$ est exécutée, puis on maintient l'ensemble des relations obtenu en appellant après chaque fusion la fonction *maintien-suffixe-commun-après-fusion*$(A,\mathcal{T},q,q',q_f,\mathcal{E}_s)$.

En pratique, on détecte que des états sont à la fois en relation de préfixe et suffixe commun en ajoutant, lors du calcul de la relation de suffixe commun, entre les lignes 3 et 4 de la fonction *marque-et-propage-en-arrière*, les instructions : **si** $\langle q_1,q_2\rangle \in \mathcal{E}_p$ **alors** *projection-type*(q_1,q_2). Et en ajoutant, lors du calcul de la relation de préfixe commun,

2. Notons que l'algorithme 2 fait qu'il peut exister une relation de préfixe commun due au mot vide alors que $|ua| > 0$. Ce problème peut être levé soit en considérant que le mot vide peut effectivement être typé, soit en propageant les premières relations non pas à partir des couples d'états initiaux mais à partir de leurs successeurs par une même lettre.

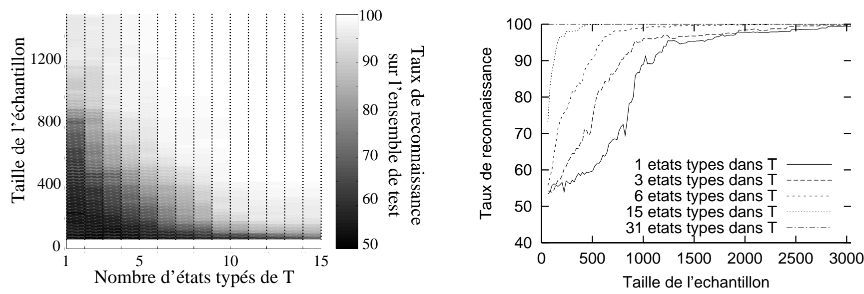

FIG. 4 – *Taux de reconnaissance sur un ensemble de test obtenu pour différentes exécutions de l'algorithme RPNI (Oncina & García, 1992) (étendu par la projection de type), à gauche : projection de la surface 3D, à droite : sections verticales. À chaque exécution correspond une taille de l'échantillon et une quantité d'information de typage.*

entre les lignes 3 et 4 de la fonction *marque-et-propage-et-avant*, les instructions : **si** $\langle q_1,q_2\rangle \in \mathcal{E}_s$ **alors** *projection-type*(q_1,q_2).

Notons que les différentes opérations à mener dès lors qu'un couple d'états est détecté en relation de préfixe commun et de suffixe commun sont en $\mathcal{O}(1)$. Le coût de l'algorithme de vérification du typage est donc celui du calcul et du maintien des relations de préfixe et suffixe commun, soit comme pour la détection de contraintes sur le langage inféré de la section 2 en $\mathcal{O}(|A| \times |\mathcal{T}| \times t_A \times t_{\mathcal{T}})$, avec t_A et $t_{\mathcal{T}}$ le nombre maximum de transition entrantes ou sortantes par une lettre donnée d'un état de A et de $\mathcal{T}$.

Une exception d'inconsistance à l'initialisation de l'algorithme signifie que la fonction de typage ne peut être réalisée par la classe d'automates inferée. Par exemple, si la classe est restreinte aux automates déterministes, l'automate de départ est le PTA; celui-ci ne peut réaliser de fonction de typage pour laquelle il existe deux exemples uav_1 et uav_2 de même préfixe ua et pour lesquels on a $f_{\mathcal{T}}(u,a,v_1) \neq f_{\mathcal{T}}(u,a,v_2)$.

De façon similaire à l'expérience menée pour l'utilisation de contraintes langagières, nous avons réalisé une expérimentation sur des données artificielles afin de vérifier l'applicabilité de nos algorithmes. Celle-ci est donnée par la figure 4. L'expérimentation suit le même protocole que l'expérimentation de la figure 2. Plutôt qu'un langage plus général, on considère ici une fonction de typage représentée par un automate. Celui-ci est construit à partir de l'automate cible en typant un sous ensemble des états de plus en plus grand (ici typant entre 1 et 31 états). Ce procédé est moyenné sur 3 échantillons et 3 choix pour les états typés sélectionnés choisis uniformément parmi les 31 états (soit $3000/20 * 31 * 3 * 3 = 41850$ expériences de moins d'une minute chacune sur une machine à 1GHz). De même que lors de l'introduction d'informations langagières, le nombre d'exemples et contre-exemples nécessaire à la convergence de l'algorithme RPNI diminue fortement dès l'introduction d'informations de typage. La convergence est plus rapide que dans le cas de l'introduction de connaissances langagières. En effet, l'information fournie est plus importante et élague une plus grande partie de l'espace

de recherche.

3.3 Typage des exemples seulement

Nous discutons dans cette section du cas particulier où l'information de typage n'est disponible que pour les mots de S_+. Plus formellement : $\forall \langle u,a,v \rangle \in \mathcal{D}(f_\mathcal{T}), uav \in S_+$. Ce cas se produit lorsque les exemples sont typés “à la main” par un expert du domaine d'application, sa connaissance n'ayant pu être formalisée dans un automate de type. Ce cas permet également d'utiliser une partie de l'information disponible dans une fonction de typage nécessitant un pouvoir d'expression supérieur à celui des automates de type.

Un automate de type $\mathcal{T}$ peut, dans ce cadre, facilement être construit à partir des exemples typés (avec $L(\mathcal{T}) = S_+$), le cadre formel de la section 3.2 pouvant alors être utilisé. Néanmoins, on peut constater que toute l'information de typage de $\mathcal{T}$ sera, dès l'initialisation, entièrement projetés sur le MCA ou le PTA. La propagation des préfixe et suffixe commun ne sert donc plus qu'à éviter que de nouveaux chemins existent après une fusion dans l'automate inféré pour un mot w de S_+, ce nouveau chemin typant un tuple $\langle u,a,v \rangle$, avec $w = uav$, différemment du chemin existant initialement dans le MCA ou le PTA.

Si l'inférence est restreinte aux automates *non ambigus*[3], il ne peut exister qu'un seul chemin pour un mot dans l'automate inféré et la fonction de typage représentée par un tel automate sera également non ambiguë. La compatibilité peut donc être assurée en interdisant la fusion de deux états de types différents (la propagation des relations de préfixe et suffixe commun n'est plus nécessaire). Les automates déterministes étant une sous classe des automates non ambigus, cette remarque est également valable pour leur inférence. Notons que la détection d'une mauvaise fusion se fait ainsi en $\mathcal{O}(1)$ mais que pour assurer la propriété de non ambiguïté ou de déterminisme, des fusions pour déterminisation ou désambiguisation doivent potentiellement être réalisées (Dupont *et al.*, 1994; Coste & Fredouille, 2001) : le coût de détection de la compatibilité devient donc égal au coût de ces opérations.

3.4 Spécialisation d'automates

Nous considérons dans cette sous-section des fonctions de typage particulières, proposées par (Kermorvant & Higuera (de la), 2002), et pour lesquelles la contrainte de typage peut être respectée en $\mathcal{O}(1)$. Nous montrons ici que l'inférence étant donné de telles fonctions peut être vue comme la spécialisation d'un automate, impliquant une inclusion du langage inféré dans celui de l'automate de type.

Le cadre proposé par (Kermorvant & Higuera (de la), 2002) considère des automates de type $\mathcal{T}$ et des automates inférés A tels que :

(1) Pour tous les états de $\mathcal{T}$, la fonction τ est définie et que pour chaque état le type est différent, i.e.: $\mathcal{D}(\tau_\mathcal{T}) = Q_\mathcal{T}$ et $\forall q,q' \in Q_\mathcal{T} \ : q \neq q' \Leftrightarrow \tau_\mathcal{T}(q) \neq \tau_\mathcal{T}(q')$.

(2) L'automate de type permet de typer entièrement les exemples, i.e. tous les tuples $\langle u,a,v \rangle$ de $\Sigma^* \times \Sigma \times \Sigma^*$ avec $uav \in S_+$ sont dans $\mathcal{D}(f_\mathcal{T})$.

3. Formellement, un automate non ambigu peut être décrit par la propriété $\forall u,v \in \Sigma^* \ : |\{\delta(I,u) \cap \delta^{-1}(F,v)\}| \leq 1$, avec $\delta^{-1}(F,v) = \{q \ / \ \delta(q,v) \cap F \neq \emptyset\}$.

(3) L'automate de type $\mathcal{T}$ et l'automate inféré A sont déterministes.

Sous ces conditions, (Kermorvant & Higuera (de la), 2002) ont montré que le typage initial de A (alors égal au PTA) et l'interdiction de fusionner deux états de types différents sont des conditions suffisantes pour assurer la compatibilité de A et de $\mathcal{T}$. La détection de la non compatibilité se fait donc en $\mathcal{O}(1)$ à chaque fusion après une initialisation en $\mathcal{O}(|PTA|)$ [4].

La preuve de (Kermorvant & Higuera (de la), 2002) se déroule en deux temps. Premièrement on considère l'automate de type $\mathcal{T}'$ obtenu en élaguant $\mathcal{T}$ des tous les composants ne servant pas à la reconnaissances de S_+. Cet élagage est possible grâce à la condition (2). On peut montrer que $L(\mathcal{T}') \subseteq L(\mathcal{T})$ et que $\mathcal{T}'$ est compatible avec $\mathcal{T}$. Deuxièmement, on démontre (grâce à la condition (1)) que $\mathcal{T}'$ est identique à l'automate obtenu en fusionnant tous les états du PTA de même type. Pour tout automate A obtenu par fusion d'états de même type de PTA, des fusions supplémentaires sur A permettent donc d'obtenir $\mathcal{T}'$. Le nombre de tuple typés ne pouvant qu'augmenter par fusion, les automates A sont compatibles avec $\mathcal{T}'$ et donc $\mathcal{T}$. De même, le langage reconnu par un automate ne pouvant qu'augmenter par fusion, on a $L(A) \subseteq L(\mathcal{T}') \subseteq L(\mathcal{T})$ [5].

À partir de leur démonstration, on peut voir que tous les automates de l'espace de recherche respectant la contrainte de compatibilité peuvent être obtenus en *spécialisant l'automate de type*, i.e. obtenus en réalisant des suppressions de transitions dans $\delta_\mathcal{T}$, d'états dans $Q_\mathcal{T}$, $I_\mathcal{T}$ et $F_\mathcal{T}$, ainsi que des fissions (l'opération complémentaire de la fusion). Cette spécialisation peut être intérprétée par l'intérmédiaire de la condition (1), celle-ci est une bijection entre les états de $Q_\mathcal{T}$ et les types : l'information fournie à l'algorithme peut ainsi être considérée comme la structure de $\mathcal{T}$ plutôt qu'un typage des lettres.

Notons que spécialiser un automate $\mathcal{T} = A_G$ implique spécialiser $L(A_G)$. Néanmoins, il ne s'agit pas d'une équivalence : lors de la spécialisation de A_G des automates représentant des langages inclus dans le langage $L(A_G)$ seront élagués de l'espace de recherche. Ce point est illustré par la figure 5. Utiliser la technique de spécialisation d'automates afin de réaliser une contrainte langagière pose ainsi le problème important de nécessiter plus d'exemples dans S_+ pour assurer la présence, dans l'espace de recherche, d'un automate reconnaissant le langage cible. Si peu d'exemples sont disponibles, cette technique réduit donc les chances d'identification.

Notons également que la condition (3) posée par (Kermorvant & Higuera (de la), 2002) n'est pas nécessaire, la spécialisation d'automates non déterministes peut être réalisée par la même technique (i.e. typage des états du MCA, et restriction aux fusions entre états de même type). Par contre la condition (1) est indispensable, comme illustré par la figure 6.

---

4. En effet, la propriété de déterminisme permet de réaliser le typage initial en $\mathcal{O}(|PTA|)$ plutôt que $\mathcal{O}(|PTA| \times |\mathcal{T}|)$.

5. Cette propriété est utilisée dans l'algorithme OSTIA pour l'inférence de transducteurs (Oncina & Varó, 1996) : en effet, elle permet d'intégrer une notion de domaine de transduction à l'inférence grâce au même algorithme que celui proposé par (Kermorvant & Higuera (de la), 2002).

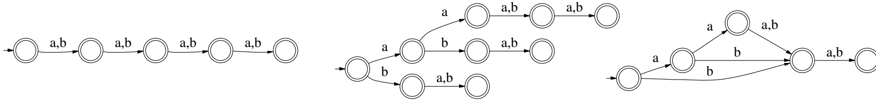

FIG. 5 – *Soient n, $m \in \mathbb{N}$ deux entiers fixés (sur la figure $n = m = 2$). Soit $\mathcal{T} = A_G$ l'automate minimal acceptant $L(A_G) = \{w \in \Sigma \ / \ |w| \leq n + m\}$ (à gauche). Spécialiser A_G implique que deux états ne peuvent être fusionnés que si ils sont à la même profondeur dans le MCA. Considérons le langage $L = \{a^x\Sigma^y \mid x \leq n, y \leq m\}$, un automate pour L spécialisé de A_G possède au moins $(n + 1) * (m + 1)$ états (au centre). Or, un automate acceptant L et possédant seulement $n + m + 1$ états existe (à droite), ce dernier a donc besoin de moins d'exemples pour être présent dans l'espace de recherche.*

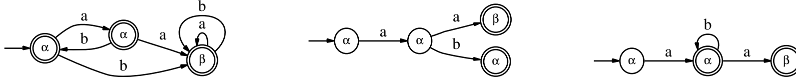

FIG. 6 – *$\mathcal{T}$ (à gauche), A à l'initialisation pour l'inférence d'automates déterministes avec $S_+ = \{aa,ab\}$ (milieu) et A obtenu après fusion des états source et destination de la transition par b (à droite) : même si les états fusionnés ont le même type, la compatibilité n'est pas respectée. En effet, après fusion $f_A(ab,a,\epsilon) = \beta$ est différent de $f_{\mathcal{T}}(ab,a,\epsilon) = \alpha$. De même on remarque que A n'est pas une spécialisation de $\mathcal{T}$.*

Conclusion

Ce papier apporte des méthodes permettant d'intégrer, dans les algorithmes classiques d'inférence d'automates, des connaissances sur le langage à inférer et des connaissances de typage - potentiellement complexes - sur les séquences du langage inféré.

Les algorithmes proposés ont été testés sur des benchmarks de données artificielles et ont ainsi prouvé leur applicabilité. Ils doivent maintenant être utilisés pour traiter des problèmes réels. Nous sommes plus particulièrement intéressés par l'inférence de modèles permettant de réaliser la classification de séquences de type biologique (ADN, ARN, protéines) et désirons utiliser les connaissances formalisées dans les modèles existants (motifs de la banque Prosite (Falquet *et al.*, 2002)) afin d'améliorer ces derniers.

Du point de vue théorique, de nombreux problèmes restent ouvert. Il semble premièrement intéressant de formaliser la quantité d'information apportée par les méthodes proposées pour pouvoir comparer cette quantité à celle amenée par la présence de nouveaux mots dans l'échantillon. Outre ce premier point, les méthodes proposées devraient pouvoir être étendues dans deux directions : passer de l'inférence d'automates à celles de grammaires possédant un plus fort pouvoir d'expression, et considérer l'intégration d'autres types de connaissances au cours de l'inférence.

Notons enfin que les informations de typage peuvent être utilisées non seulement pour restreindre l'espace de recherche, comme réalisé dans ce papier, mais également comme

heuristique afin de se guider dans cet espace. De telles heuristiques restent à développer et sont, à notre avis, prometteuse de par leur nature basée sur les connaissances liées à l'application.

Remerciements

Nous tenons à remercier Colin de la Higuera et Christopher Kermorvant pour leurs remarques et discussions.

Références

CARRASCO R. & ONCINA J. (1994). Learning stochastic regular grammars by means of a state merging method. In *Grammatical Inference and Applications, ICGI'94*, number 862 in Lecture Notes in Artificial Intelligence, p. 139–150: Springer Verlag.

COSTE F. & FREDOUILLE D. (2000). Efficient ambiguity detection in C-NFA, a step toward inference of non deterministic automata. *Grammatical Inference: Algorithms and Applications, ICGI'00*, p. 25–38.

COSTE F. & FREDOUILLE D. (2001). Inférence d'AFNs : restriction de l'espace de recherche aux automates non ambigus. *Conférence d'apprentissage CAp'01*.

DENIS F., LEMAY A. & TERLUTTE A. (2000). Apprentissage de langages réguliers à l'aide d'automates non déterministes. In *Conférence d'apprentissage CAp'00*.

DENIS F., LEMAY A. & TERLUTTE A. (2001). Learning regular languages using RFSA. In *Proceedings of the 12th International Conference on Algorithmic Learning Theory, ALT'01*, p. 348–363.

DUPONT P., MICLET L. & VIDAL E. (1994). What is the search space of the regular inference ? *Grammatical inference and Applications, ICGI'94*, p. 25–37. Springer Verlag.

FALQUET L., PAGNI M., BUCHER P., HULO N., SIGRIST C., HOFMANN K. & BAIROCH A. (2002). Protein data bank. *Nucleic Acid Research*, **30**, 235–238. `http://www.expasy.ch/prosite/`.

GOAN T., BENSON N. & ETZIONI O. (1996). A grammar inference algorithm for the world wide web. In *AAAI spring Symp. on Machine Learning in Information Access*.

Gowachin (1998). Gowachin, benchmarks for grammatical inference. `http://www.irisa.fr/Gowachin/`.

KERMORVANT C. & HIGUERA (DE LA) C. (2002). Learning languages with help. *Grammatical Inference: Algorithms and Applications, ICGI'02*, p. 161–173.

LANG K. J., PEARLMUTTER B. A. & PRICE R. A. (1998). Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science*, **1433**, 1–12.

ONCINA J. & GARCÍA P. (1992). Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis*, p. 49 – 61.

ONCINA J. & VARÓ M. A. (1996). Using domain information during the learning of a subsequential transducer. In *Grammatical Inference: Learning Syntax from Sentences, ICGI-96*, volume 1147, p. 301–312: Springer, Berlin.