Inférence Grammaticale

Fabien Torre

Université de Lille

Mercredi 4 et 18 novembre 2009

Séquences et alphabet

Notations

- un alphabet Σ, ensemble fini quelconque ;
- un mot w, une séquence finie de lettres de Σ ;
- mots possibles sur Σ : Σ^* ;
- on note ϵ le mot vide.

Par exemple

- $\Sigma = \{a, b\}$;
- $+$: ϵ, aaa, b, $abbaa$, $aaaaaa$;
- $-$: bba, aab, $bbaababa$.

Objectifs : identifier un langage cible, apprendre à classer de nouveaux mots.

Obtention des exemples et évaluation

Protocoles possibles

- expérimental ;
- PAC [Valiant, 1984] ;
- à la limite [Gold, 1967] le plus souvent.

Apprentissage à la limite

- avec des positifs et des négatifs : *avec informateur* ;
- ou des exemples positifs seuls : *à partir de textes* ;
- tous les mots du langage cible sont progressivement fournis : *présentation positive* ;
- après chaque exemple, l'apprenant propose un classifieur ;
- il finit par proposer la cible et ne doit plus changer d'avis.

Des échantillons particuliers

Définition : échantillon caractéristique

Un échantillon S^c est dit *caractéristique* pour une cible $c \in \mathcal{H}$ et un apprenant L ssi il est le plus petit échantillon vérifiant :

$$\forall S : S^c \subseteq S \Rightarrow L(S) = c$$

Démarche

1. choisir $\mathcal{H}$;
2. proposer un algorithme L qui fournit des hypothèses de $\mathcal{H}$;
3. définir un échantillon caractéristique pour $c \in \mathcal{H}$ et L ;
4. prouver l'apprenabilité à la limite de $\mathcal{H}$.

... passons en revue les $\mathcal{H}$ pour les mots.

Hypothèses en inférence grammaticale

Apprendre une grammaire, un automate ou une expression régulière.

Définition : grammaire formelle

Une grammaire est donnée par un quintuplet (V, Σ, P, S) :

- V les variables, les non-terminaux ;
- Σ l'alphabet, les symboles terminaux ;
- P règles de production $\alpha \to \beta$ avec $\alpha \in (V \cup \Sigma)^* V (V \cup \Sigma)^*$ et $\beta \in (V \cup \Sigma)^*$;
- $S \in V$, symbole de départ.

Au tableau : exemple /production / reconnaissance (subsomption).

Différentes familles selon la forme des règles...

Hiérarchie de Chomsky

- Type 0 : les grammaires sans contrainte de forme ;
- type 1 : les grammaires contextuelles ;
- type 2 : les grammaires hors contextes ;
- les grammaires linéaires ;
- type 3 : les grammaires rationnelles.
- type 4 : les grammaires à choix finis.

Autant de langages d'hypothèses $\mathcal{H}$ à disposition... et même plus !

Grammaires de type 0

- Sans contrainte sur leur forme des règles de production $\alpha \rightarrow \beta$;
- puissance des machines de Turing ;
- le test de subsomption est indécidable.

Grammaires contextuelles (type 1)

- Pour les règles de production $\alpha \rightarrow \beta$, on impose que $|\alpha| \leq |\beta|$;
- exemples de langages contextuels :
  - $a^n b^n c^n$;
  - ww ;
- le test de subsomption est de complexité exponentielle.

Grammaires hors contextes (type 2)

- Pour les règles de production $\alpha \to \beta$, on impose que $\alpha \in V$;
- on les appelle aussi *algébriques* ;
- elles sont représentables par des *automates à pile* ;
- exemples de langages *context-free* :
  - les langages de Dyck ;
  - $a^n b^n$;
  - le langage des palindromes $\{w | w = w^R\}$;
- le test de subsomption est cubique dans la taille du mot.

Grammaires linéaires (entre les types 2 et 3)

- Pour les règles de production $\alpha \to \beta$, on impose que
  - $\alpha \in V$ et
  - soit $\beta \in \Sigma$;
  - soit $\beta \in \Sigma \times V \times \Sigma$.

Grammaires rationnelles (type 3)

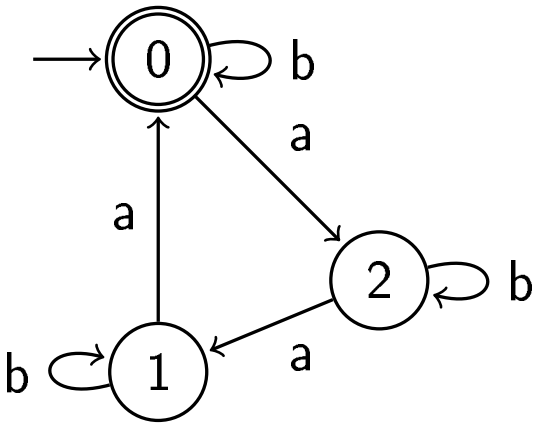
- Les grammaires linéaires droites (équivalentes aux gauches). Pour les règles de production $\alpha \to \beta$, on impose que
  - $\alpha \in V$ et
  - soit $\beta \in \Sigma$;
  - soit $\beta \in \Sigma \times V$.
- *langages réguliers* ou *grammaires rationnelles*.
- on peut les représenter par des *automates finis déterministes* ou par des *expressions régulières*.
- le test de subsomption en $|w|.|A|^2$, $|w|$ la longueur du mot w, $|A|$ le nombre d'états de l'automate A.

Les automates finis déterministes (AFD)

Définition : AFD

Un AFD est donné par $(Q, \Sigma, \delta, q_0, F)$:

- Q l'ensemble des états ; $Q = \{0, 1, 2\}$;
- Σ l'alphabet ; $\Sigma = \{a, b\}$;
- $\delta : Q \times \Sigma \to Q$ la fonction de transition ;
  $\delta(0, a) = 2$, $\delta(0, b) = 0$, $\delta(1, a) = 0$,
  $\delta(1, b) = 1$, $\delta(2, a) = 1$, $\delta(2, b) = 2$;
- $q_0 \in Q$ l'état initial ; $q_0 = 0$;
- $F \subseteq Q$ les états finaux. $F = \{0\}$.



Caractéristiques et familles d'automates

- Finis,
- minimaux,
- ambigus ou pas,
- déterministes ou non (équivalents),
- déterministes dans les deux sens (0-réversibles),
- k-TSS,
- k-réversibles,
- AFER, etc.

Vraiment beaucoup de $\mathcal{H}$ candidats !

Les expressions régulières

Définition : expression régulière

Construite à l'aide d'un alphabet Σ et des opérateurs :

- . : concaténation ;
- | : disjonction ;
- * : 0 ou plus ;
- $^+$: 1 ou plus ;
- ? : 0 ou 1.

- Exemple : $(a|b)^+.b$;
- À nouveau, plusieurs familles d'expressions régulières : CHARE et SORE, single-loop, etc.

SOREs et CHAREs [Bex et al., 2006]

Définition : Single occurrence regular expressions (SOREs)

Expressions utilisant chaque lettre de Σ au plus une fois.

Définition : *Chain regular expressions* (CHAREs)

Les SOREs qui s'expriment comme une séquence $f_1.f_2 \dots f_n$ où chaque facteur f_i est (avec $k > 0$ et $a_i \in \Sigma$) :

- soit $(a_1|a_2 \dots |a_k)$,
- soit $(a_1|a_2 \dots |a_k)?$,
- soit $(a_1|a_2 \dots |a_k)^+$,
- soit $(a_1|a_2 \dots |a_k)^*$.

Modélisations raisonnables des DTD !

Des hyperplans pour séparer les mots [Clark et al., 2006]

Hyperplans

- comptages proches de l'attributs-valeurs ;
- le langage $L = \{a^n b^n | n > 0\}$ peut être représenté par l'hyperplan : $w \ inL \Leftrightarrow (|w|_a = |w|_b) \wedge (|w|_{ba} = 0)$.

Boules de mots [Tantini, 2009] (1)

Boules : définitions

- trois opérations d'édition de coût unitaire :
  - insertion : $aab \rightarrow aa\underline{b}b$
  - suppression : $a\underline{a}b \rightarrow ab$
  - substitution : $a\underline{a}b \rightarrow a\underline{b}b$
- distance d'édition : $d(w_1, w_2)$ est le nombre minimal d'opérations qui permettent de passer de w_1 à w_2 ;
- un langage est donné par un mot-centre et un rayon : $B_2(bab)$.

Boules de mots [Tantini, 2009] (2)

Attention, beaucoup de pièges...

- les boules de mots ne sont pas homogènes (la moitié des mots de la boule sont à distance maximale du centre) ;
- une boule de mots peut être contenue dans une autre de rayon plus petit : $B_5(ab) \subset B_4(abab)$;
- deux boules de mots de même rayon et de centres de même longueur peuvent dénoter des langages de tailles différentes : $|B_2(aaaabbbb)| \neq |B_2(abababab)|$;
- les boules de mots ne sont pas convexes : $bbbaaa \notin B_4(aabb)$ alors qu'il appartient au segment entre $abbbaaa$ et $bbaaaa$, tous deux dans $B_4(aabb)$;
- représentations multiples d'un langage lorsque $|\Sigma| = 1$: $B_4(\epsilon) = B_3(a) = B_2(aa)$.

Apprendre par positifs seuls

Objectifs

- On veut apprendre des langages réguliers ;
- à partir de positifs seuls ;
- en prouvant un apprentissage à la limite.

Impossible pour la classe entière des langages réguliers...

Plan : apprentissage de sous-classes de réguliers

- un algorithme pour les expressions régulières « simple loop » ;
- un algorithme pour les expressions régulières CHAREs ;
- un algorithme pour les automates k-TSS ;
- un algorithme pour les automates 0-réversibles ;
- un algorithme pour les boules de mots.

[Fernau, 2005] : regroupements et alignement

1. Échantillon : $S = \{ababb*, aabb*, ababa*, abc*\}$
2. regroupements :

$$(a)(b)(a)(bb), (aa)(bb), (a)(b)(a)(b)(a), (a)(b)(c)$$

3. alignement :

$$\begin{array}{cccccc} (a) & (b) & (a) & (bb) & & \\ (aa) & (bb) & & & & \\ (a) & (b) & (a) & (b) & (a) & \\ (a) & (b) & (c) & & & \end{array}$$

[Fernau, 2005] : alignement et généralisation

3. alignement :

$$\begin{array}{cccc} (a) & (b) & (a) & (bb) \\ (aa) & (bb) & & \\ (a) & (b) & (a) & (b) \quad (a) \\ (a) & (b) & (c) & \end{array}$$

4. généralisation (pas tout à fait une généralisation par colonne, attention) :

$$\begin{array}{llll} a^+b^+(a & & & |\epsilon|c) \\ a^+b^+(a & b^+ & & |\epsilon|c) \\ a^+b^+(a & b^+ & (\epsilon|a) & |\epsilon|c) \end{array}$$

5. résultat $a^+.b^+.(a.b^+.(\epsilon|a)|\epsilon|c)$.

Un biais ?

[Fernau, 2005] : biais

Hypothèse de l'alignement à gauche !

On aurait pu prendre :

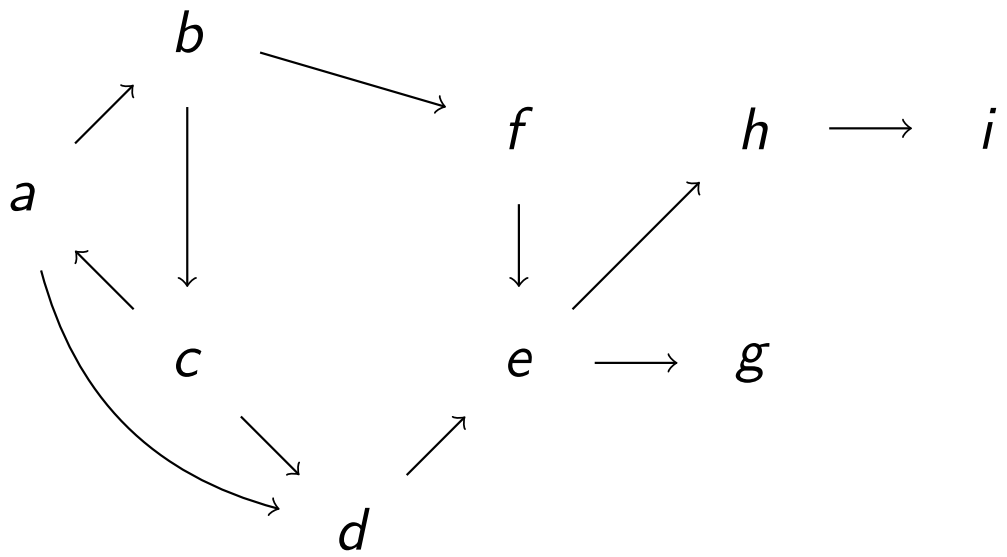
$$\begin{array}{cccccc} (a) & (b) & (a) & (bb) & & \\ & & (aa) & (bb) & & \\ (a) & (b) & (a) & (b) & (a) & \\ & & (a) & (b) & (c) & \end{array}$$

Quelle est alors l'expression apprise ?

Des liens avec la bio-informatique ?

Algorithme CRX pour les CHAREs [Bex et al., 2006]

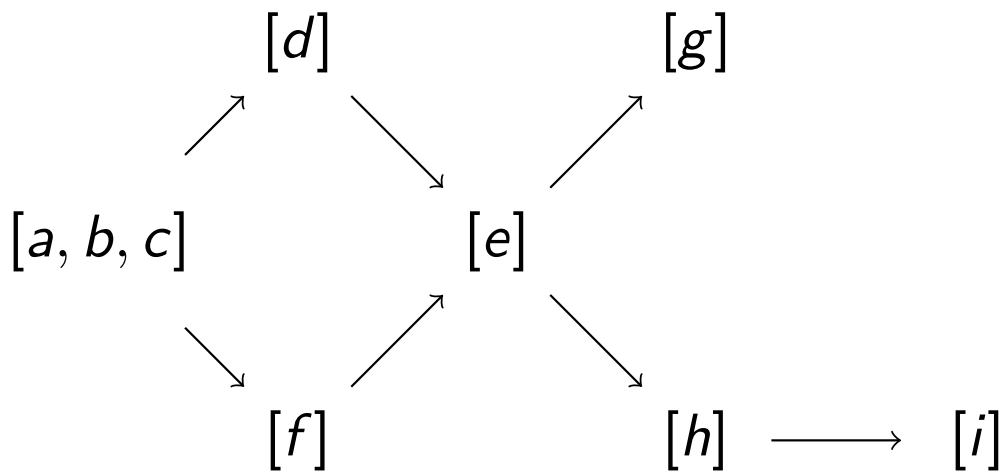
1. Échantillon : $S = \{abccde*, cccad*, bfegg*, bfehi*\}$
2. ordre sur Σ induit par S :



3. classes d'équivalences

Algorithme CRX (2)

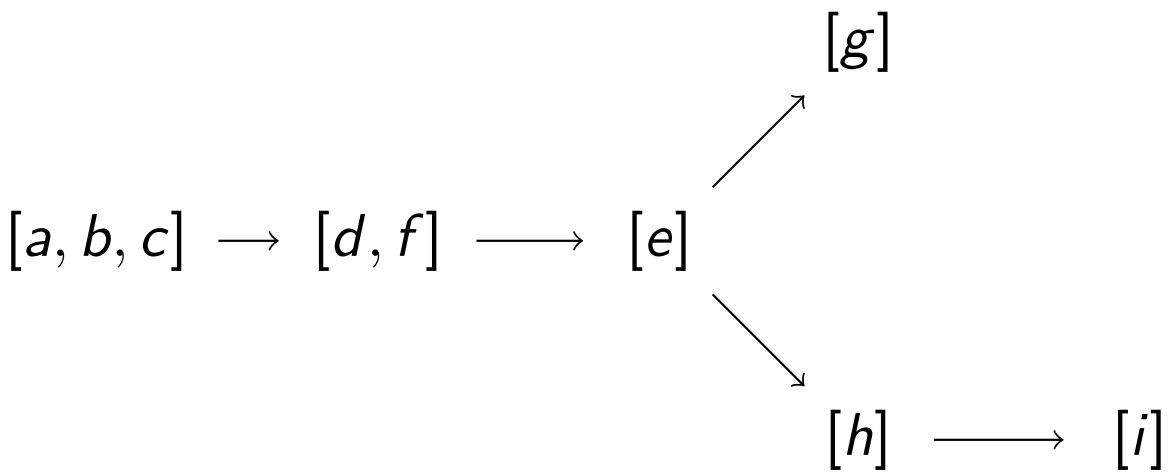
3. classes d'équivalences :



4. fusionner les classes qui ont mêmes prédécesseurs et mêmes successeurs

Algorithme CRX (3)

4. fusionner les classes si mêmes prédécesseurs et mêmes successeurs :



5. par observation de l'échantillon $\{abccde, cccad, bfegg, bfehi\}$, construire l'expression régulière :

$$(a|b|c)^+.(d|f).e?.g^*.h?.i?$$

Apprentissage de DTD avec CRX

Principes

- à partir de documents XML ;
- pour chaque nœud de chaque document, constuire la séquence des fils ;
- pour chaque élément, rassembler les séquences de fils observées ;
- pour chacun de ces échantillons, lancer CRX.

Voir exemples d'apprentissage sur un corpus de pages xhtml.

Langages k-TSS [García and Vidal, 1990]

Définition : langage k testable au sens strict ($k > 1$)

Un langage k-TSS, k étant fixé, est défini par :

- I les segments initiaux autorisés u ($|u| = k - 1$) ;
- F les segments finaux autorisés v ($|v| = k - 1$) ;
- T les segments interdits, tous de taille égale à k ;
- W les mots w du langage tels que $|w| < k - 1$) ;

Formellement :

$$L(\Sigma, I, F, T, W) = W \cup (I\Sigma^* \cap \Sigma^* F) \setminus \Sigma^* T \Sigma^*$$

... les mots w du langage tels que $|w| = k - 1$ sont dans I et F.

... voir aussi les *langages locaux* (2-TSS) et les *n-gram*.

Algorithme k-TSSI [García and Vidal, 1990] I

Clef : chaque état est identifié par une séquence de taille strictement inférieure à k.

Entrées : S un échantillon de mots positifs
Sortie : $h = (Q, \Sigma, \delta, q_0, F)$ un automate k-TSS couvrant S

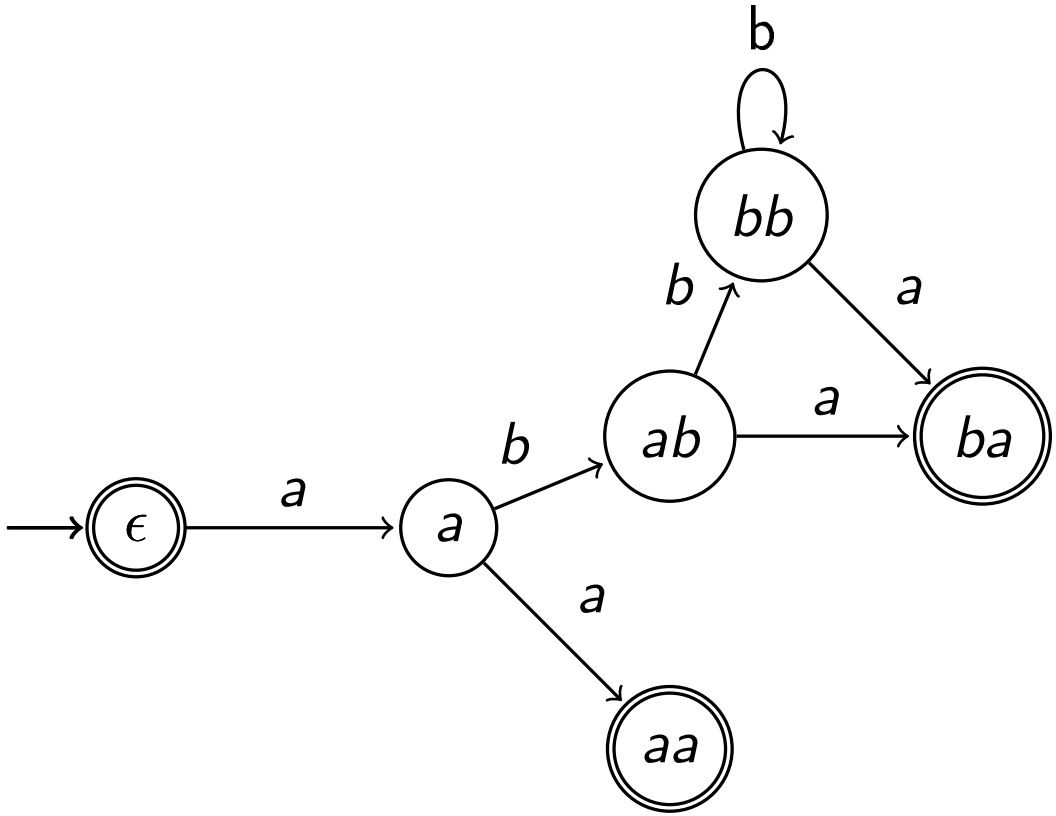
1: créer l'état $q_0 = \epsilon$
2: $Q = \{q_0\}$
3: $\delta = \emptyset$
4: $F = \emptyset$
5: **for all** $w \in S$ **do**
6: $\quad q = q_0$
7: $\quad$ **for** $i = 1$ to $|w|$ **do**
8: $\quad\quad v = q.w_i$

Algorithme k-TSSI [García and Vidal, 1990] II

9: **if** $(|v| > k-1)$ **then**
10: $v = v_{2,\ldots,|v|}$
11: **end if**
12: $nq = v$
13: ajouter nq à Q
14: ajouter (q, w_i, nq) à δ
15: **if** $(i = |w|)$ **then**
16: ajouter nq à F
17: **end if**
18: $q = nq$
19: **end for**
20: **end for**
21: **return** $h' = (Q, \Sigma, \delta, q_0, F)$

Algorithme k-TSSI : une exécution

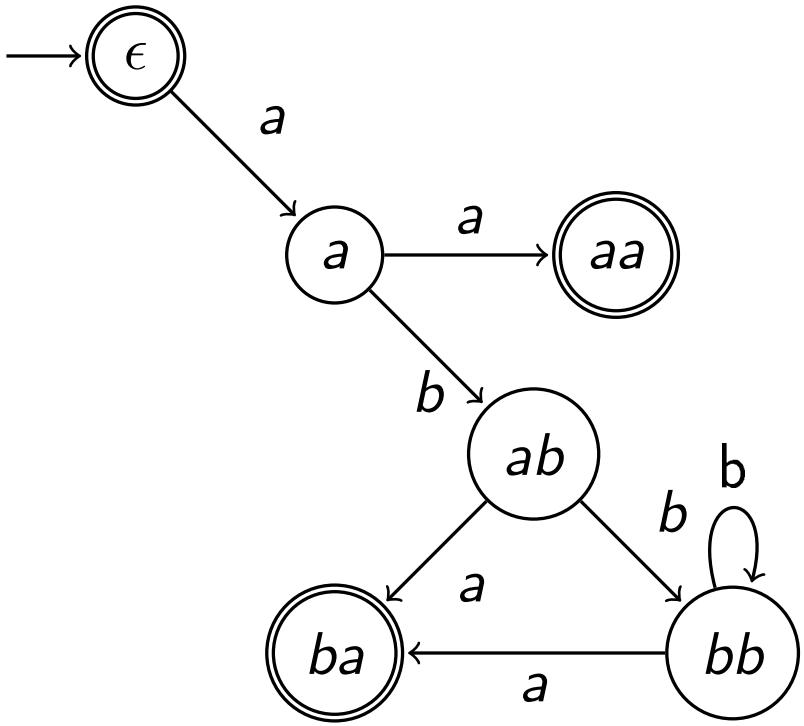
1. Échantillon $S = \{\epsilon*, aa*, aba*, abba*, abbba*\}$ et $k = 3$;
2. inférence de l'automate :



... comment revenir à (I, F, T, W) ?

Algorithme k-TSSI : résultat final

$S = \{\epsilon, aa, aba, abba, abbba\}$ et $k = 3$.



$$
\begin{aligned}
\Sigma &= \{a, b\} \\
W &= \{\epsilon\} \\
I &= \{aa, ab\} \\
F &= \{aa, ba\} \\
\overline{T} &= \{aba, abb, bba, bbb\} \\
T &= \{aaa, aab, baa, bab\}
\end{aligned}
$$

Automates 0-réversible

Définition : 0-réversible

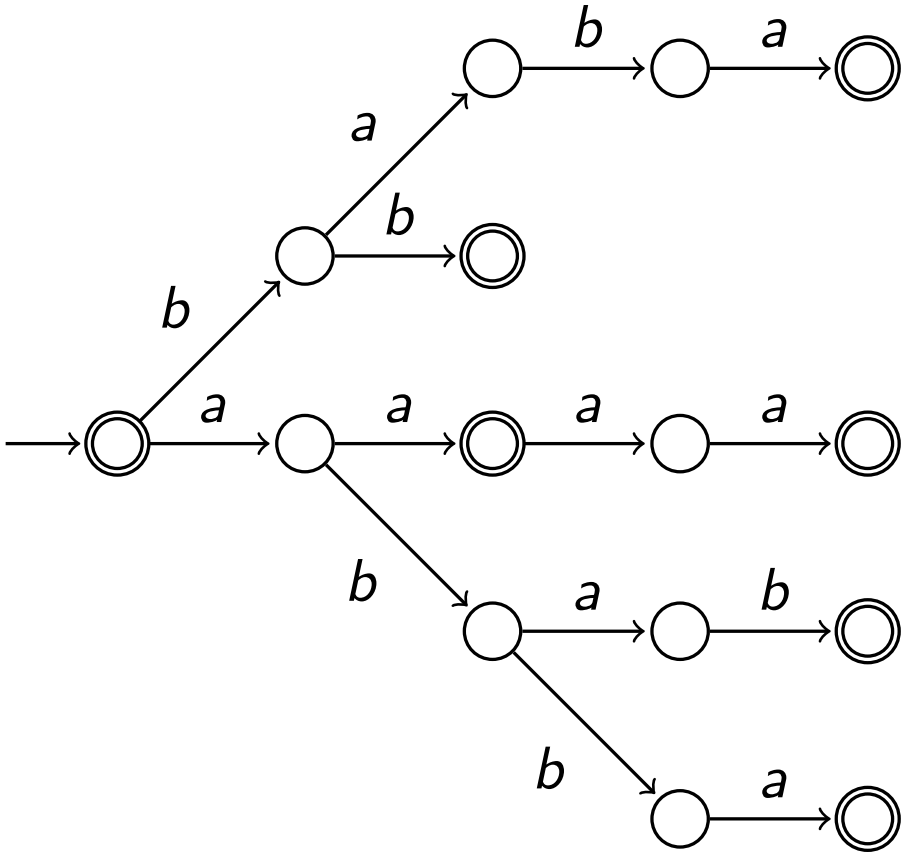
Un automate A est 0-réversible s'il est déterministe dans les deux sens : A est déterministe et son miroir A^R l'est aussi.

L'algorithme ZR de [Angluin, 1982] débute par la construction du PTA de l'échantillon.

- automate en forme d'arbre, sans boucle ;
- reconnaît strictement l'échantillon, pas un mot de plus ;
- il faudra le généraliser.

PTA : Prefix Tree Automaton

$$S = \{\epsilon*, aa*, bb*, aaaa*, abab*, abba*, baba*\}$$



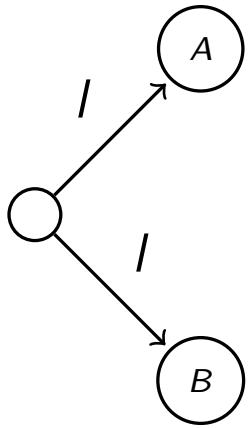
Algorithme ZR [Angluin, 1982]

Entrées : S un échantillon de mots positifs
Sortie : h un automate 0-réversible couvrant les mots de S

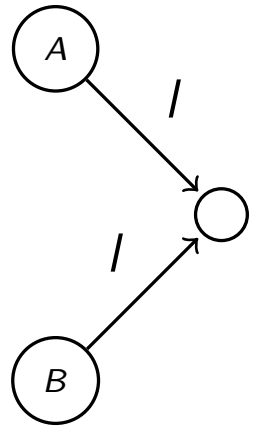
1: $h = \mathrm{PTA}(S)$
2: fusionner les états A et B dans les cas suivants ($I \in \Sigma$) :

deux finaux

non déterminisme

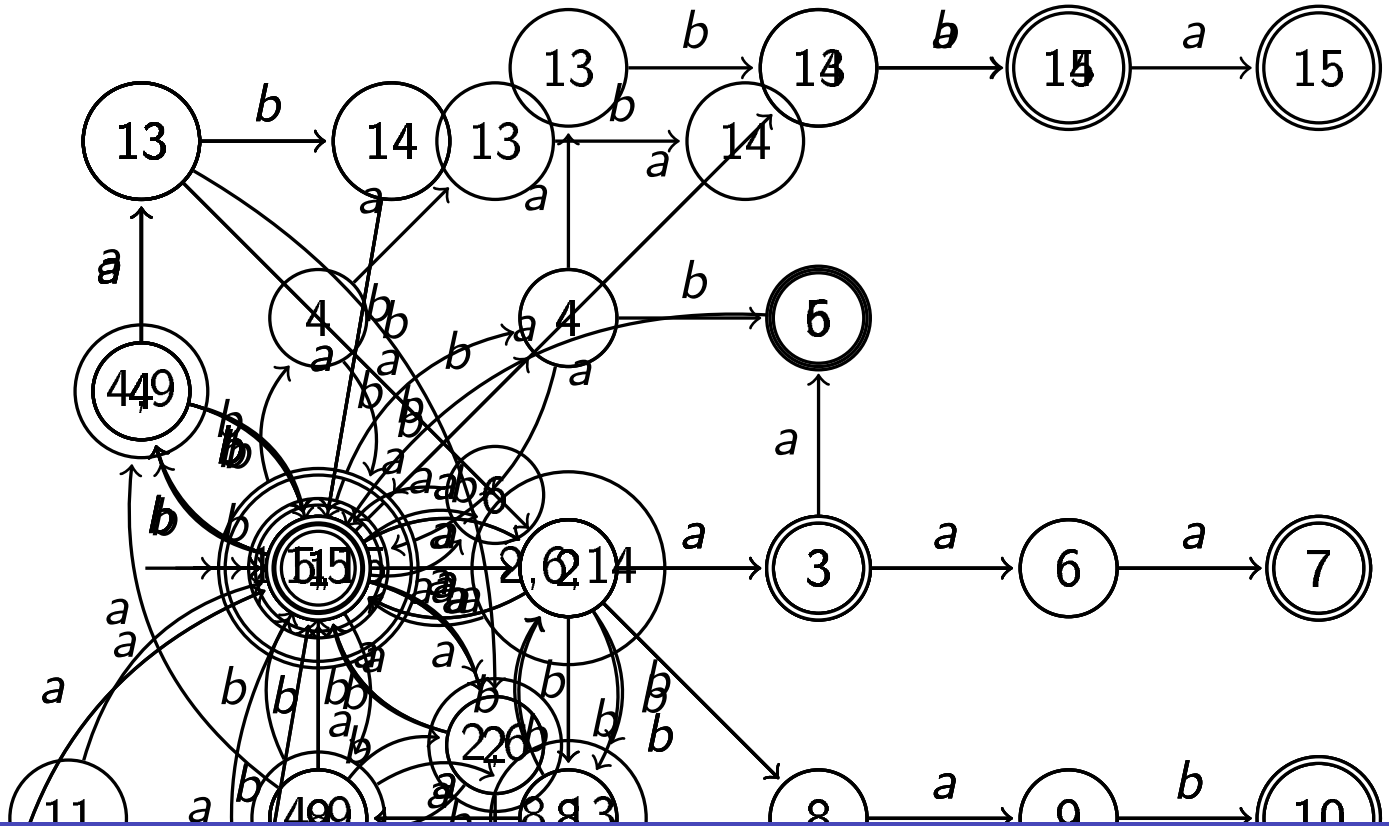


non réversible



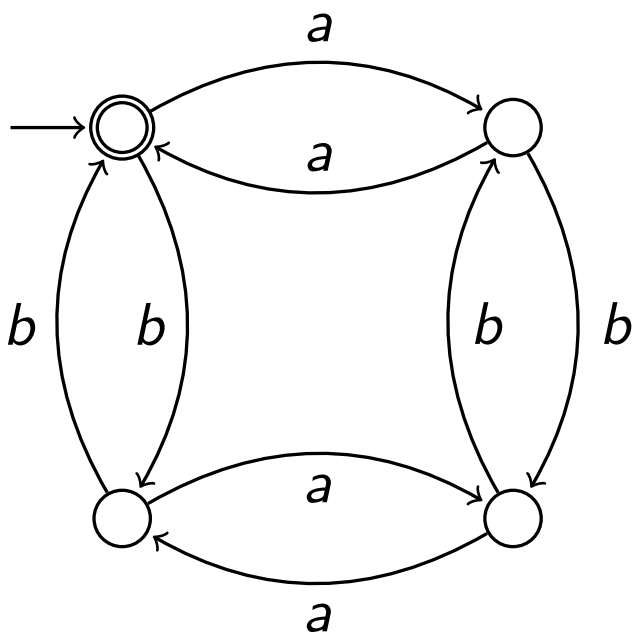
3: **return** h

ZR : une exécution (fusion de 1 et 5) (fusion de 1 et 15) (fusion de 1 et 7, 10, 12) (fusion de 1 et 3) (fusion de 2 et 6) (fusion de 2 et 14) (fusion de 2 et 11) (fusion de 4 et 9) (fusion de 8 et 13)



ZR : une exécution (bilan)

$$S = \{\epsilon, aa, bb, aaaa, abab, abba, baba\}$$



Esquisse de la preuve d'apprenabilité des 0-réversibles

1. Montrer que quel que soit le langage cible L, il existe un échantillon S^c tel que L est le plus petit langage 0-réversible contenant S^c.
2. Montrer qu'à partir d'un échantillon S, l'algorithme ZR fournit un automate reconnaissant le plus petit langage 0-réversible qui contient S.
3. Montrer que la présentation des exemples va finir par inclure S^c.
4. Conclure : à partir d'une certaine étape de l'apprentissage S contient S^c (3), le plus petit langage 0 réversible qui contient S est donc la cible L (1) et ZR va fournir un automate qui reconnait exactement L (2).

Proposition d'échantillon pour les 0-réversibles

Pour chaque état q de la cible A

On place $u.v$ dans l'échantillon avec u et v définis comme les plus petits mots tels que u conduit de q_0 à q et que v fait passer de q à un état final de A.

Pour toute transition de A entre q et q' par l

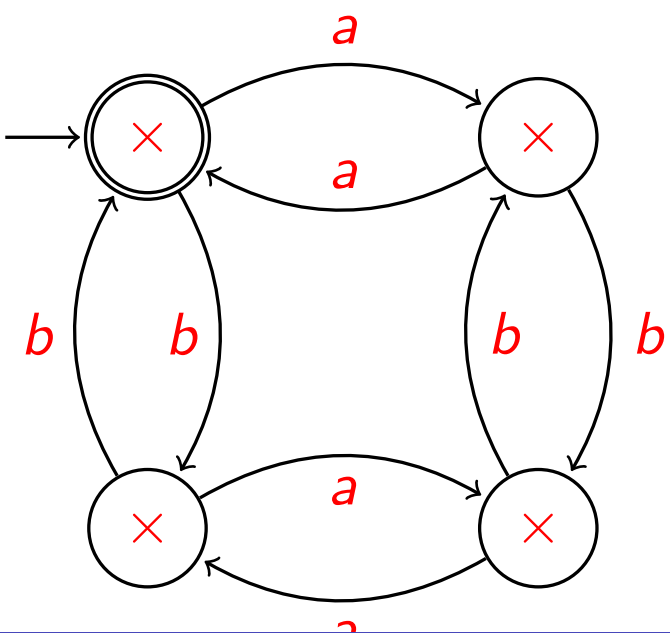
On place $u.l.v$ dans l'échantillon avec u et v définis comme les plus petits mots tels que u conduit de q_0 à q et que v fait passer de q' à un état final de A.

Taille de cet échantillon en fonction de $|A|$?

... notion proche de la ▸ complétude structurelle

Échantillon pour les 0-réversibles : un exemple

Pour chaque état q de la cible A, on place $u.v$ dans l'échantillon avec u et v définis comme les plus petits mots tels que u conduit de q_0 à q et que v fait passer de q à un état final de A.
Pour toute transition de A entre q et q' par l, on place $u.l.v$ dans l'échantillon avec u et v définis comme les plus petits mots tels que u conduit de q_0 à q et que v fait passer de q' à un état final de A.



$$\begin{aligned} \epsilon.\epsilon &= \epsilon \\ a.a &= aa \\ b.b &= bb \\ ab.ab &= abab \\ ab.b.a &= abba \\ b.a.ba &= baba \end{aligned}$$

Preuve pour l'échantillon construit

Montrer que quel que soit le langage cible L, l'échantillon S^c de L que l'on vient de définir est tel que L est le plus petit langage 0-réversible contenant S^c.

Preuve de minimalité de ZR

Montrer qu'à partir d'un échantillon S, l'algorithme ZR fournit un automate reconnaissant le plus petit langage 0-réversible qui contient S.

Apprenabilité des 0-réversibles : conclusion

Montrer que la présentation des exemples va finir par inclure S^c.

Conclure : à partir d'une certaine étape de l'apprentissage S contient S^c (3), le plus petit langage 0 réversible qui contient S est donc la cible L (1) et ZR va fournir un automate qui reconnait exactement L (2).

S^c un échantillon caractéristique pour un langage L 0-réversibles et l'algorithme ZR.

Les 0-réversibles sont apprenables à la limite par positifs seuls.

Apprendre à la limite les boules de mots [Tantini, 2009]

Observation

Pour toute boule $B_r(o)$ définie sur $\Sigma = \{a, b\}$, les mots $a^r o$ et $b^r o$ appartiennent à la boule.

Algorithme

À l'étape t,

1. on cherche dans les t mots reçus les deux plus longs mots de la forme $a^n v$ et $b^n v$;
2. on prédit $B_n(v)$.

Pour poursuivre

1. Est-ce que les algorithmes vus sont *minimaux* ? Autrement dit, est-ce que chacun propose le plus petit langage de sa classe compatible avec l'échantillon ?
2. Pour chaque classe de langages étudiée, quelle est sa dimension de Vapnik-Chervonenkis ?

RPNI [Oncina and García, 1992] : motivations

- Les langages réguliers ne sont pas apprenables à la limite par positifs seuls, des sous-classes oui (k-TSS, k-réversibles, etc.).
- On vise maintenant toutes la classes des AFD mais à partir d'exemples et de contre-exemples, pour un langage cible on va en particulier chercher son *automate canonique*.

Définition : automate canonique

Pour un langage L, l'*automate canonique* ou l'*automate déterministe minimal* est l'automate déterministe le plus petit en nombre d'états qui reconnaît L.

Intuition de RPNI

On commence par calculer le PTA des mots positifs, si l'échantillon est *complet structurellement* il suffit de bien fusionner les états... les négatifs vont nous aider à contrôler les fusions.

Complétude structurelle et RPNI

Définition : complétude structurelle

Un échantillon S de mots positifs est dit *structurellement complet* vis-à-vis d'un automate cible A ssi :

- toute transition de A est utilisée dans la reconnaissance d'au moins un mot de S ;
- tout état final de A est activé pour au moins un mot de S.

Si l'échantillon est suffisant, RPNI découvre le canonique.

RPNI [Oncina and García, 1992] : esquisse

Recherche en largeur d'abord pour trouver un état à fusionner avec un état déjà visité. Validation des fusions vis-à-vis des négatifs : si un exemple de S^- est reconnu, la fusion est rejetée.

- F : les nœuds fermés, F sont les états les plus proches de l'état initial, on a échoué à les faire disparaître par fusion, ils resteront dans l'automate final ;
- O : les nœuds ouverts, c'est-à-dire candidats à la fusion, les états de O suivent ceux de F, ils constituent une frontière qui sépare F des autres états ;
- $Q \setminus O \setminus F$ sont les états les plus profonds de l'automate.

On cherche à fusionner un état de F avec un état de O. Si un état de O ne peut être fusionné, il passe dans F et ses successeurs vont dans O.

RPNI [Oncina and García, 1992] : procédures

Il faut savoir :

- calculer le PTA d'un ensemble d'exemples ;
- fusionner des états ;
- maintenir le déterminisme ;
- tester vis-à-vis des négatifs.

Fusion de plusieurs états I

Algorithme Fusion(A, E)

Entrées : $A = (Q, \Sigma, \delta, q_0, F)$, $E \subseteq Q, q_0 \notin E_{[2,\ldots,|E|]}$

Sortie : A' l'automate résultat de la fusion des états de E dans A

1: $q_1 = E[1]$
2: $E = E_{[2,\ldots,|E|]}$
3: $Q' = Q \setminus E$
4: **if** $E \cap F \neq \emptyset$ **then**
5: $\quad F' = F \setminus E \cup \{q_1\}$
6: **else**
7: $\quad F' = F$
8: **end if**
9: $\delta' = \emptyset$
10: **for all** $(q_a, l, q_b) \in \delta$ **do**
11: $\quad$ **if** $q_a \in E$ **then**

Fusion de plusieurs états II

12: $q'_a = q_1$
13: **else**
14: $q'_a = q_a$
15: **end if**
16: **if** $q_b \in E$ **then**
17: $q'_b = q_1$
18: **else**
19: $q'_b = q_b$
20: **end if**
21: $\delta' = \delta' \cup \{(q'_a, l, q'_b)\}$
22: **end for**
23: **return** $A' = (Q', \Sigma, \delta', q_0, F')$

La fusion de deux états peut provoquer une perte de déterminisme au niveau de l'état résultat...

RPNI [Oncina and García, 1992] : fusion déterministe

On va propager la fusion jusqu'au retour du déterminisme.

Algorithme FusionD(A, E)

Entrées : $A = (Q, \Sigma, \delta, q_0, F)$, $E \subseteq Q$, $q_0 \notin E_{[2,\ldots,|E|]}$

Sortie : A' résultat de la fusion de q_1 et q_2 dans A et déterministe

1: $A' = (Q', \Sigma, \delta', q_0, F') = \text{Fusion}(A, E)$
2: $q_1 = E[1]$
3: nondet = **true**
4: **while** nondet **do**
5: **if** $\exists E' \subseteq Q, |E'| > 1, \exists l \in \Sigma, \forall q \in E' : (q_1, l, q) \in \delta'$ **then**
6: $A' = \text{FusionD}(A', E')$
7: **else**
8: nondet = **false**
9: **end if**
10: **end while**

RPNI [Oncina and García, 1992] : test de correction

Algorithme isCorrect(A, S^-)

Entrées : A un automate et S^- un échantillon de n mots négatifs

Sortie : **true** si A ne reconnaît aucun mot de S^-, **false** sinon

```
1:  correct = true
2:  i = 0
3:  while correct and i < n do
4:      if A ≽ S⁻[i] then
5:          correct = false
6:      else
7:          i = i + 1
8:      end if
9:  end while
10: return correct
```

RPNI [Oncina and García, 1992] : algorithme I

Algorithme RPNI(S)

Entrées : S un échantillon de mots (positifs et négatifs)

Sortie : A un AFD consistant avec S

1: $A = (Q, \Sigma, \delta, q_0, F) = \mathrm{PTA}(S^+)$
2: $F = [q_0]$
3: $O = [q \in Q | \exists l \in \Sigma : q \in \delta(q_0, l)]$
4: **while** $O \neq \emptyset$ **do**
5: $\quad q_O = O[1]$
6: $\quad$ encore = **true** ; $i = 0$
7: $\quad$ **while** encore **and** $i < |F|$ **do**
8: $\quad\quad q_F = F[i]$
9: $\quad\quad A' = \mathrm{FusionD}(A, \{q_F, q_O\})$
10: $\quad\quad$ **if** isCorrect(A', S^-) **then**
11: $\quad\quad\quad A = A'$

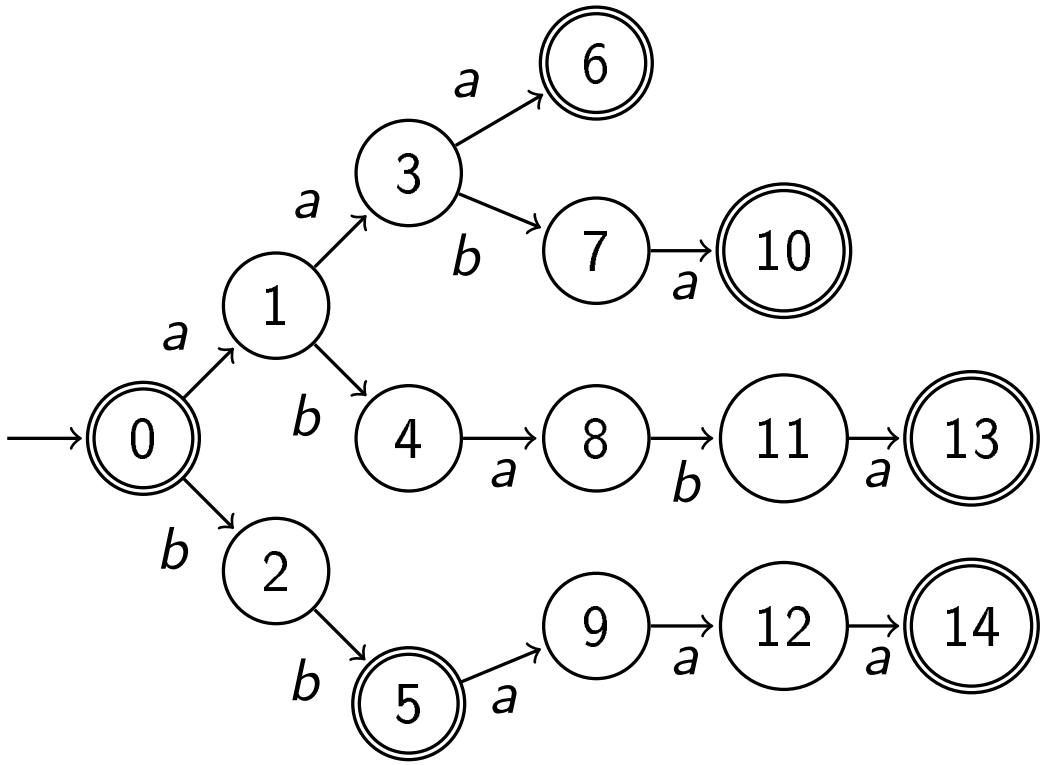
RPNI [Oncina and García, 1992] : algorithme II

12: encore = **false**
13: **end if**
14: $i = i + 1$
15: **end while**
16: $O = O_{[2,\ldots,|O|]}$
17: **if** encore **then**
18: $F = F \cup \{q_O\}$
19: $O = \emptyset$
20: **for all** $q_F \in F$ **do**
21: $O = O \cup \{q \in Q \setminus F | \exists l \in \Sigma : q \in \delta(q_F, l)\}$
22: **end for**
23: **end if**
24: **end while**
25: **return** A

RPNI : calcul du PTA

$$S = \{ \quad (\epsilon, +), (bb, +), (aaa, +), (aaba, +), (ababa, +), (bbaaa, +), \\ (aa, -), (ab, -), (ba, -), (aaaa, -)\}$$

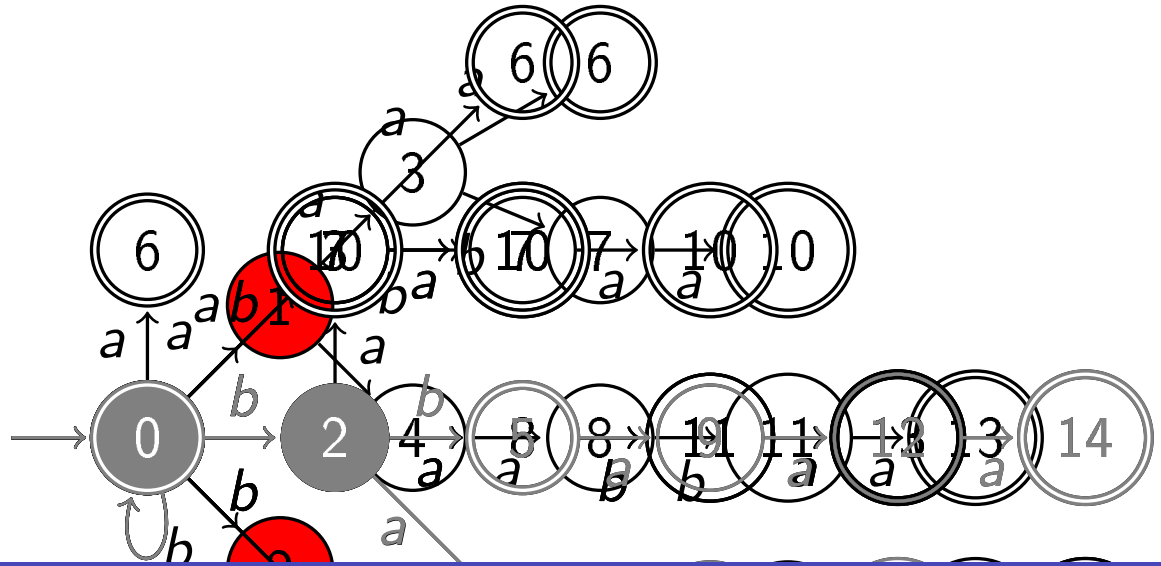
On calcule le PTA de $S^+ = \{\epsilon, bb, aaa, aaba, ababa, bbaaa\}$.



RPNI : première tentative de fusion

Tentative de fusion de 0 et 1.
Perte de déterminisme, fusion de 0 et 3. Perte de déterminisme, fusion de 0 et 6. Perte de déterminisme, fusion de 2, 4 et 7. Perte de déterminisme, fusion de 8 et 10. Automate déterministe.
Vérification de la correction vis-à-vis de $S^- = \{aa, ab, ba, aaaa\}$.
$(aa, -)$ est reconnu, l'automate et la fusion $(0, 1)$ sont rejetés.

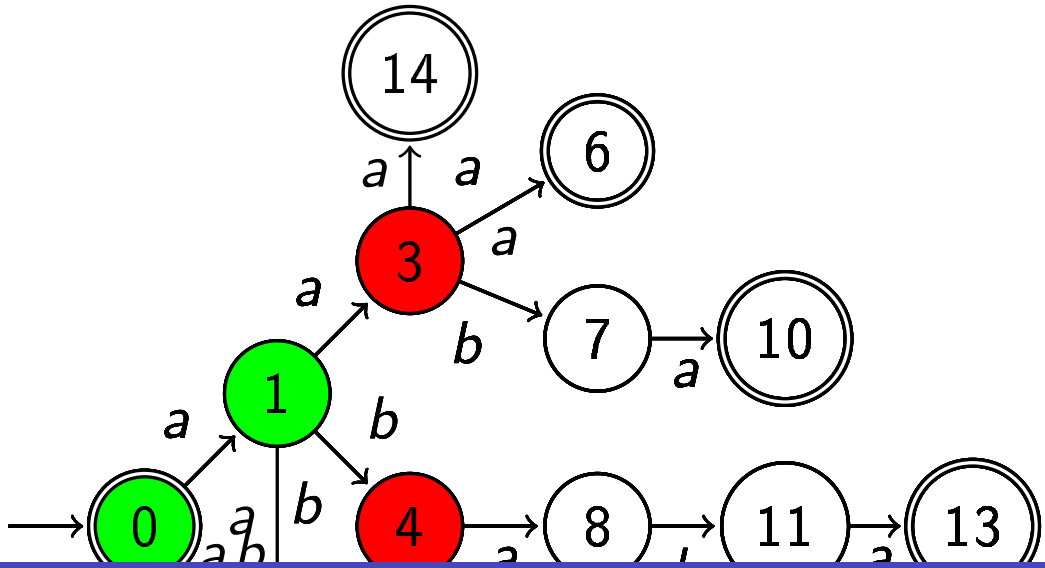
$F = \{0\}$
$O = \{1, 2\}$



RPNI : deuxième tentative de fusion

1 rejoint l'ensemble F, 3 et 4 vont dans O.
Tentative de fusion de 0 et 2.
Perte de déterminisme, fusion de 0 et 5. Perte de déterminisme, fusion de 1 et 9. Perte de déterminisme, fusion de 3 et 12. Perte de déterminisme, fusion de 6 et 14. Automate déterministe.
Vérification de la correction vis-à-vis de $S^- = \{aa, ab, ba, aaaa\}$.
L'automate est correct et la fusion $(0, 2)$ est validée.

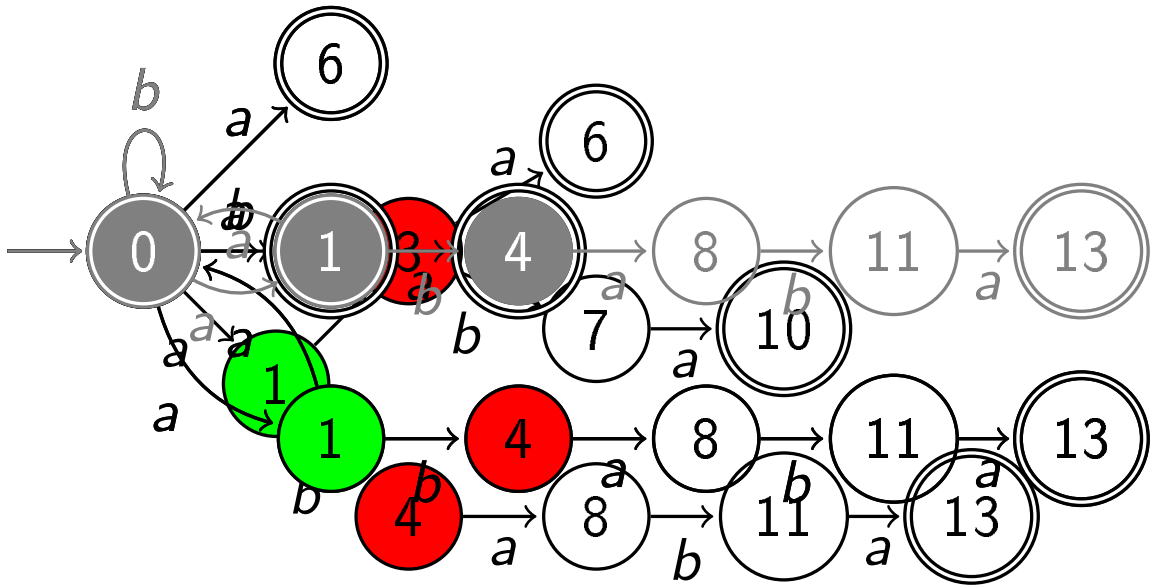
$F = \{0, 1\}$
$O = \{2, 3, 4\}$



RPNI : troisième tentative de fusion

Tentative de fusion de 0 et 3.
Perte de déterminisme, fusion de 0 et 7. Perte de déterminisme, fusion de 1, 6 et 10. Automate déterministe.
Vérification de la correction vis-à-vis de $S^- = \{aa, ab, ba, aaaa\}$.
$(aa, -)$ est reconnu, l'automate et la fusion $(0, 3)$ sont rejetés.

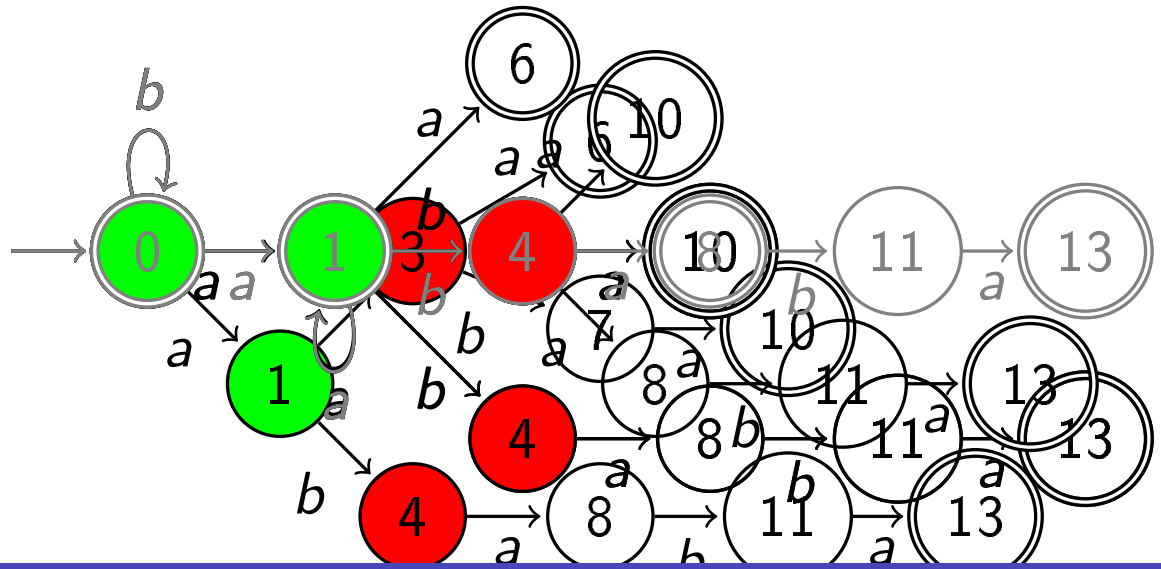
$F = \{0, 1\}$
$O = \{3, 4\}$



RPNI : quatrième tentative de fusion

Tentative de fusion de 1 et 3.
Perte de déterminisme, fusion de 1 et 6. Perte de déterminisme, fusion de 4 et 7. Perte de déterminisme, fusion de 8 et 10.
Automate déterministe.
Vérification de la correction vis-à-vis de $S^- = \{aa, ab, ba, aaaa\}$.
$(aa, -)$ est reconnu, l'automate et la fusion $(1, 3)$ sont rejetés.

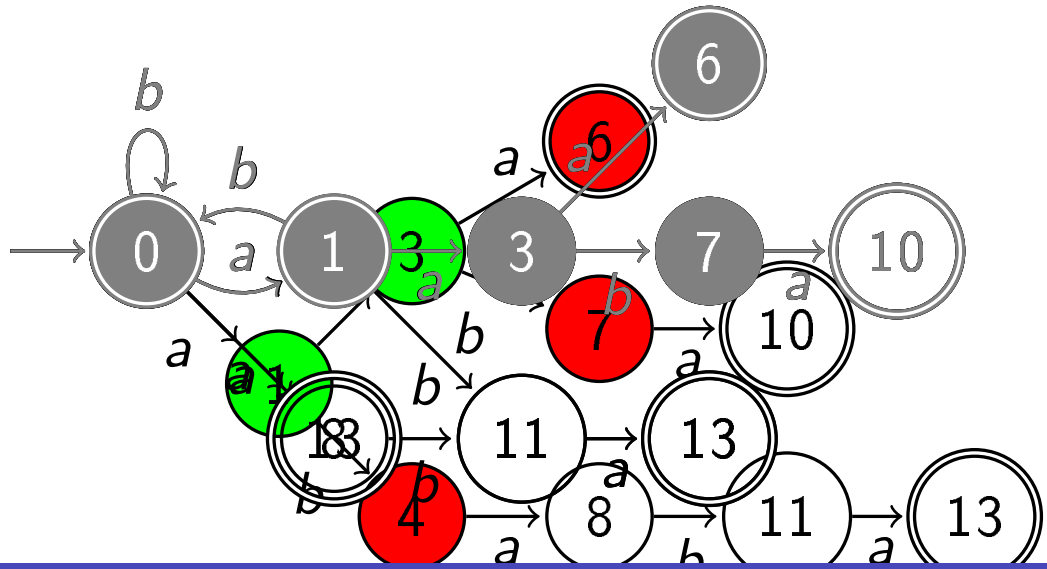
$F = \{0, 1\}$
$O = \{3, 4\}$



RPNI : cinquième tentative de fusion

3 rejoint l'ensemble F, 6 et 7 vont dans O.
Tentative de fusion de 0 et 4.
Perte de déterminisme, fusion de 1 et 8. Perte de déterminisme, fusion de 0 et 11. Perte de déterminisme, fusion de 1 et 13.
Automate déterministe.
Vérification de la correction vis-à-vis de $S^- = \{aa, ab, ba, aaaa\}$.
$(ab, -)$ est reconnu, l'automate et la fusion $(0, 4)$ sont rejetés.

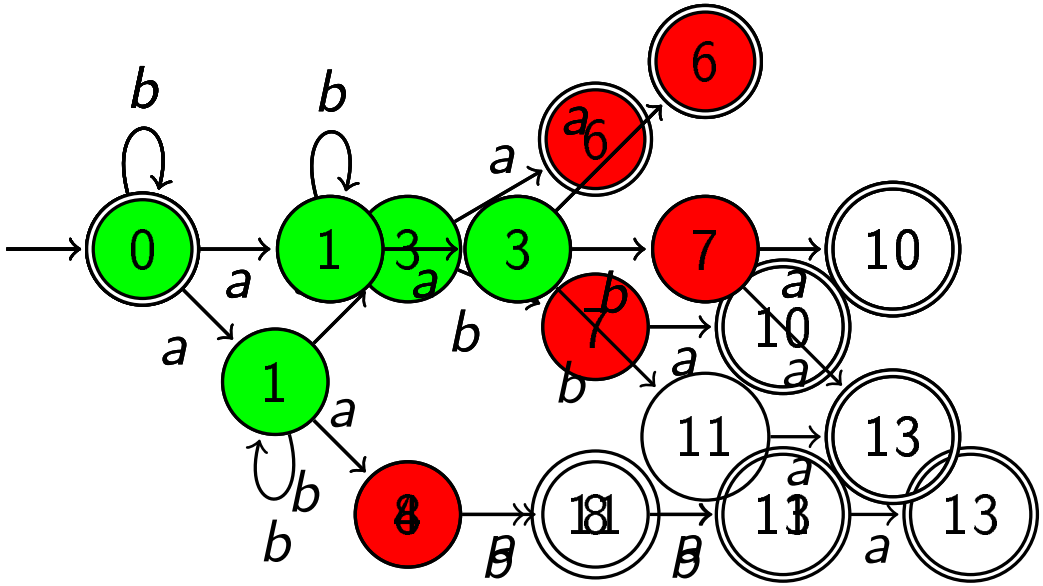
$F = \{0, 1, 3\}$
$O = \{4, 6, 7\}$



RPNI : sixième tentative de fusion

Tentative de fusion de 1 et 4.
Perte de déterminisme, fusion de 3 et 8. Perte de déterminisme, fusion de 7 et 11. Perte de déterminisme, fusion de 10 et 13.
Automate déterministe.
Vérification de la correction vis-à-vis de $S^- = \{aa, ab, ba, aaaa\}$.
L'automate est correct et la fusion $(1, 4)$ est validée.

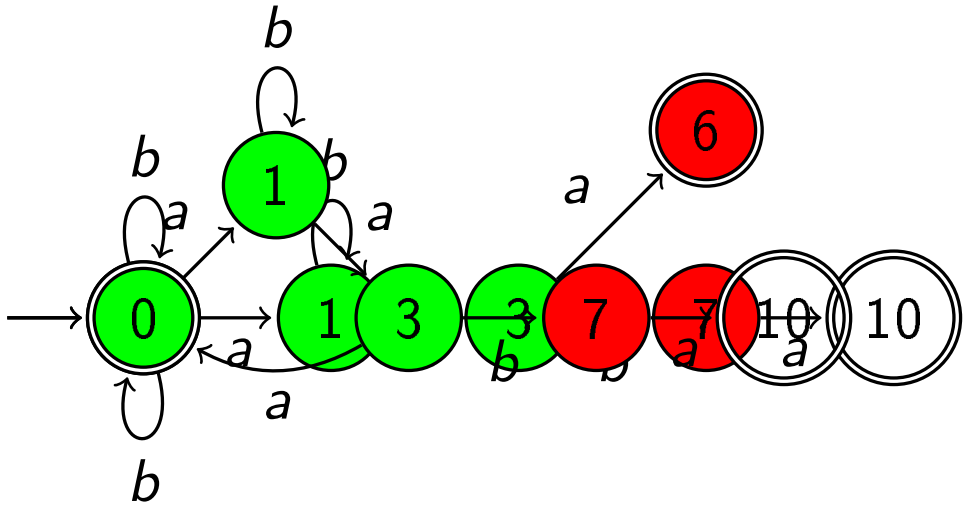
$F = \{0, 1, 3\}$
$O = \{4, 6, 7\}$



RPNI : septième tentative de fusion

Tentative de fusion de 0 et 6.
Automate déterministe.
Vérification de la correction vis-à-vis de $S^- = \{aa, ab, ba, aaaa\}$.
L'automate est correct et la fusion $(0, 6)$ est validée.

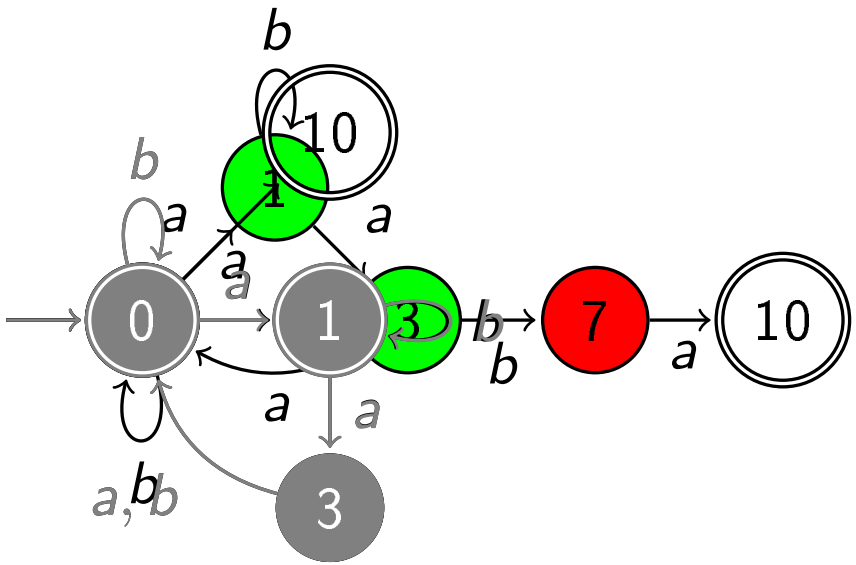
$F = \{0, 1, 3\}$
$O = \{6, 7\}$



RPNI : huitième tentative de fusion

Tentative de fusion de 0 et 7.
Perte de déterminisme, fusion de 1 et 10.
Automate déterministe.
Vérification de la correction vis-à-vis de $S^- = \{aa, ab, ba, aaaa\}$.
$(ab, -)$ est reconnu, l'automate et la fusion $(0, 7)$ sont rejetés.

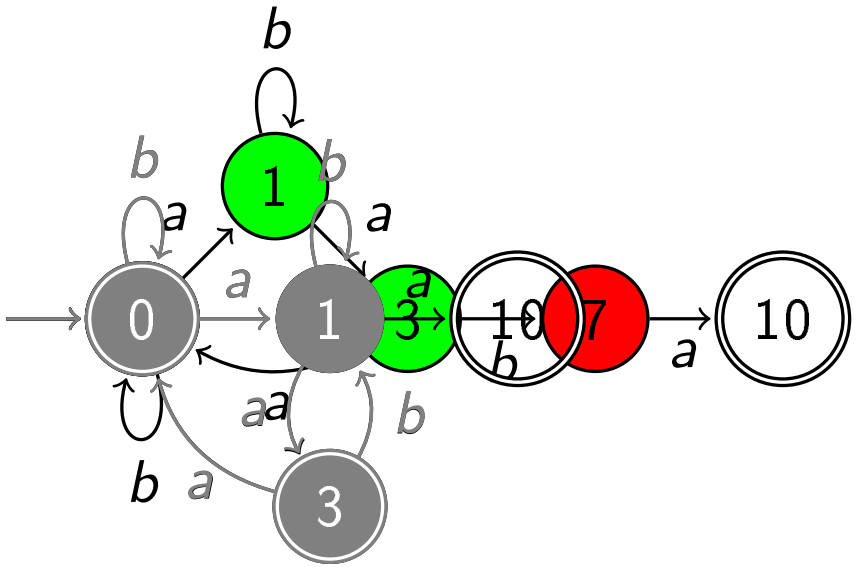
$F = \{0, 1, 3\}$
$O = \{7\}$



RPNI : neuvième tentative de fusion

Tentative de fusion de 1 et 7.
Perte de déterminisme, fusion de 3 et 10.
Automate déterministe.
Vérification de la correction vis-à-vis de $S^- = \{aa, ab, ba, aaaa\}$.
$(aa, -)$ est reconnu, l'automate et la fusion $(1, 7)$ sont rejetés.

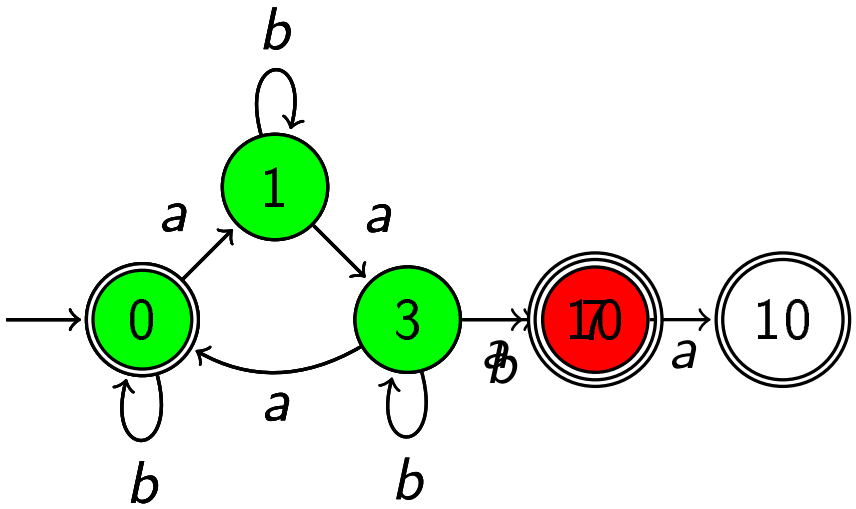
$F = \{0, 1, 3\}$
$O = \{7\}$



RPNI : dixième tentative de fusion

Tentative de fusion de 3 et 7.
Perte de déterminisme, fusion de 0 et 10.
Automate déterministe.
Vérification de la correction vis-à-vis de $S^- = \{aa, ab, ba, aaaa\}$.
L'automate est correct et la fusion $(3, 7)$ est validée.

F	$=$	$\{0, 1, 3\}$
O	$=$	$\{7\}$

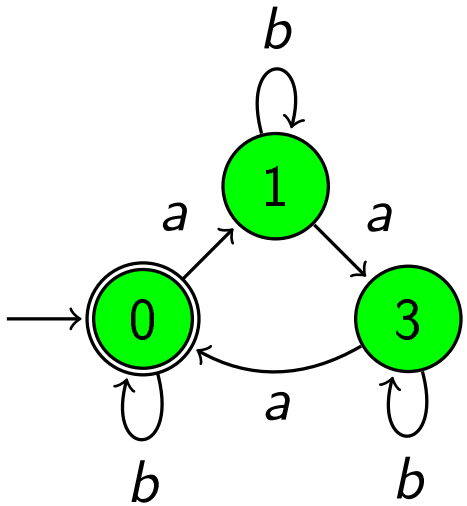


RPNI : fin

$$S = \{ \quad (\epsilon, +), (bb, +), (aaa, +), (aaba, +), (ababa, +), (bbaaa, +), \\ (aa, -), (ab, -), (ba, -), (aaaa, -)\}$$

$F = \{0, 1, 3\}$

$O = \{\}$



Algorithmes et candidats MG

Algorithmes génériques à base de MG

- DLG et GloBo ;
- BaggingMG, AdaBoostMG, GloBoost ;
- résistance au bruit.

Candidats MG dans les mots

- algorithme k-TSSI ;
- algorithme ZR ;
- apprentissage de boules de mots ?

Retour sur MGC

Algorithme MGC(E, N)

Entrées : $E = [e_1, \ldots, e_n] \subseteq \mathcal{X}$ une séquence de n exemples de même classe, N un ensemble de contre exemples.

Sortie : $h \in \mathcal{H}$ une généralisation de E, maximalement correcte par rapport à E et N.

1: $h = e_1$
2: **for** $i = 2$ to n **do**
3: $h' = \texttt{MG}(h, e_i)$ {Généralisation entre deux hypothèses.}
4: **if** $(\forall e \in N : h' \not\geq e)$ **then**
5: $h = h'$ {h' (correcte) devient la généralisation courante.}
6: **end if**
7: **end for**
8: **return** h

Que sont les combinaisons ?

Quelques questions...

- la combinaison de deux k-TSS est elle un k-TSS ?
- oui ? quel est alors l'intérêt de combiner ?
- la combinaison de deux 0-réversibles est elle un 0-réversible ?
- non ? dans quelle classe sommes-nous ? peut-on atteindre tous les langages réguliers ?
- est-ce que les poids jouent un rôle dans l'expressivité de la combinaison ?

Et à propos... quelles sont les VC-dim des classes étudiées ?

k-TSSI : forme moindre généralisée I

Algorithme MG-k-TSSI(h, w)

Entrées : $h = (Q, \Sigma, q_0, F)$ un automate k-TSS et w un mot
Sortie : h' un k-TSS généralisation minimale de h couvrant w

1: $q = q_0$
2: **for** $i = 1$ to $|w|$ **do**
3: $v = q.w_i$
4: **if** $(|v| > k - 1)$ **then**
5: $v = v_{2,\ldots,|v|}$
6: **end if**
7: $nq = v$
8: ajouter nq à Q
9: ajouter (q, w_i, nq) à δ
10: **if** $(i = |w|)$ **then**

k-TSSI : forme moindre généralisée II

11: ajouter nq à F
12: **end if**
13: $q = nq$
14: **end for**
15: **return** $h' = (Q, \Sigma, \delta, q_0, F)$

ZR : forme moindre généralisée

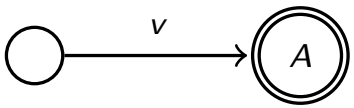
Algorithme MG-k-R(h, w)

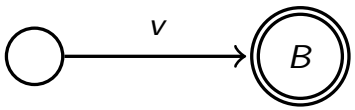
Entrées : h un automate k-réversible et w un mot

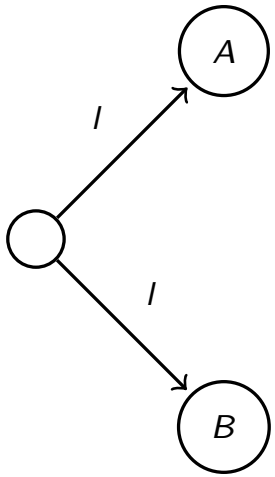
Sortie : h' un k-réversible généralisation minimale de h couvrant w

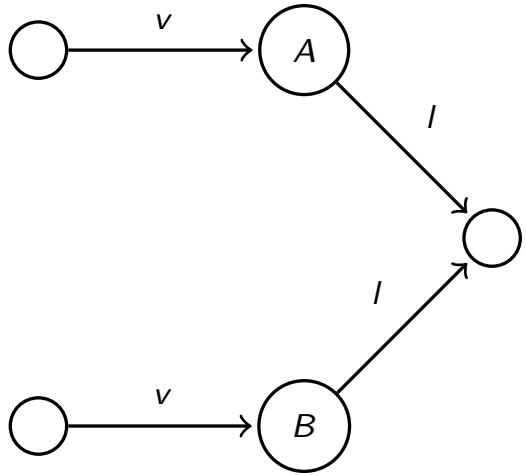
1: $h' =$ ajoutBranche(w, h)

2: appliquer les fusions des états A et B dans les situations suivantes ($I \in \Sigma$ et $|v| = k$) :









3: **return** h'

Boules de mots [Tantini, 2009] et moindres généralisés

ajout de e à $B_r(c)$

- Soit d la distance entre le c et e ;
- soit $r' = \frac{r+d}{2}$ le nouveau rayon ;
- le nouveau centre c' est sur le segment $[c; e]$, à une distance de e qui vaut r'.

ajout de *chiens* à $B_3(niche)$

- $d = \text{distance}(niche, chiens) = 5$;
- nouveau rayon : $r' = \frac{3+5}{2} = 4$;
- chemin d'édition : *niche* ; *iche* ; *che* ; *ches* ; *chens* ; *chiens* ;
- nouvelle hypothèse : $B_4(iche)$.

Premières expérimentations

- Uniquement sur `tic-tac-toe` [Blake and Merz, 1998] ;
- validations croisées 10 fois ;
- méthodes stochastiques exécutées 10 fois ;
- utilisation de volata pour les combinaisons d'automates ;
- RPNI apprend sur chaque classe ;
- repères :

majoritaire	65.34
C4.5	85.60
RPNI+	90.81
RPNI−	91.13

Combinaisons d'automates k-TSS

Combinaisons de taille 100.

k-TSSI	$k = 4$	$k = 5$	$k \in [3;5]$	$k \in [3;5]$ + Laplace
DLG	76.72	71.43	73.27	75.29
GloBo	82.78	74.69	80.47	82.36
AdaBoostMG	88.51	87.62	88.73	90.03
GloBoost	87.97	87.47	90.08	89.94

Combinaisons d'automates 0-réversibles

Combinaisons de taille 20.

k-réversibles	$k = 0$	$k = 0$ + Laplace
DLG	81.83	83.77
GloBo	85.05	87.41
AdaBoostMG	97.08	96.87
GloBoost	96.45	95.41

Variations sur la taille de l'échantillon

Proportion en apprentissage	10%	25%	50%	90%
RPNI+	66.52	74.91	80.17	90.81
RPNI−	66.89	74.15	78.60	91.13
AdaBoostMG	75.64	93.37	96.08	96.87
GloBoost	72.44	93.31	96.12	95.41

Et avec du bruit de classe

Avec 50% des données seulement.

Taux de bruit	0%	1%	5%	10%	15%
RPNI+	80.17	81.21	68.89	59.71	54.91
RPNI−	78.60	76.93	66.91	62.11	57.10
AdaBoostMG + Laplace	96.08	94.57	86.37	80.95	74.08
GloBoost	96.34	94.23	87.07	77.42	69.75
GloBoost + Laplace	96.12	94.14	88.59	79.14	69.04

Bilan de la combinaison d'automates

Au moins sur le problème *tic-tac-toe* :

- les algorithmes d'apprentissage par positifs seuls k-TSSI et ZR sont utilisables dans nos méthodes génériques ;
- et permettent d'obtenir des performances raisonnables ;
- certaines de nos méthodes rivalisent avec RPNI et le battent ;
- nos méthodes nécessitent moins d'exemples que RPNI ;
- nous résistons également mieux au bruit de classe.

Plus de résultats dans [Torre and Terlutte, 2009].
Poursuite de ce travail, intégration des boules de mots dans volata avec Frédéric Tantini et Alain Terlutte.

Bibliographie I


Angluin, D. (1982).
Inference of reversible languages.
Journal of the ACM, 29(3) :741–765.


Bex, G. J., Neven, F., Schwentick, T., and Tuyls, K. (2006).
Inference of concise dtds from xml data.
In *VLDB '06 : Proceedings of the 32nd international conference on Very large data bases*, pages 115–126. VLDB Endowment.


Blake, C. and Merz, C. (1998).
UCI repository of machine learning databases
[http ://archive.ics.uci.edu/ml/].

Bibliographie II


Clark, A., Florêncio, C. C., and Watkins, C. (2006).
Languages as hyperplanes : Grammatical inference with string kernels.
In Fürnkranz, J., Scheffer, T., and Spiliopoulou, M., editors, *ECML 2006, 17th European Conference on Machine Learning*, volume 4212 of *Lecture Notes in Computer Science*, pages 90–101. Springer.


Dupont, P. and Miclet, L. (1998).
Inférence grammaticale régulière : fondements théoriques et principaux algorithmes.
Technical Report RR-3449, INRIA.

Bibliographie III

Fernau, H. (2005).
Algorithms for learning regular expressions.
In Jain, S., Simon, H.-U., and Tomita, E., editors, *Algorithmic Learning Theory, 16th International Conference, ALT 2005, Singapore, October 8-11, 2005, Proceedings*, volume 3734 of *Lecture Notes in Computer Science*, pages 297–311. Springer.

García, P. and Vidal, E. (1990).
Inference of k-testable languages in the strict sense and application to syntactic pattern recognition.
IEEE Trans. Pattern Anal. Mach. Intell., 12(9) :920–925.

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

Bibliographie IV

Oncina, J. and García, P. (1992).
Identifying regular languages in polynomial time, pages 99–108.

World Scientific Publishing.

Tantini, F. (2009).
Inférence grammaticale en situations bruitées.
PhD thesis, Université Jean Monnet de Saint-Étienne.

Torre, F. and Terlutte, A. (2009).
Méthodes d'ensemble en inférence grammaticale : une approche à base de moindres généralisés.
In Bennani, Y. and Rouveirol, C., editors, *11ème Conférence francophone sur l'Apprentissage automatique (CAp'2009)*, pages 33–48, Hammamet (Tunisie). PUG.

Bibliographie V


Valiant, L. G. (1984).
A theory of the learnable.
Communications of the ACM, 27 :1134–1142.