



groupe
représentation
et traitement
des
connaissances

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

31, chemin Joseph Aiguier

F-13402 MARSEILLE CEDEX 9 (France)

téléphone: (33) 91 22 40 64

telex: CNRSMAR 43 0225 F

télécopie: (33) 91 71 08 08

e-mail: grtc@frmop11.bitnet

INFERENCE DE LANGAGES REGULIERS

Bernard Bel

Résumé

Cet exposé présente une méthode générale d'acquisition de connaissances dans un domaine formalisé à l'aide d'automates finis (langages réguliers). A partir d'un échantillon d'exemples le système construit un automate "presque minimal" qui n'est pas nécessairement déterministe. Cette construction peut être contrainte par des connaissances sur la segmentation des exemples que le système peut acquérir en questionnant l'informateur. Dans une deuxième phase, le système généralise l'automate à partir de propriétés connues du langage ou(et) à partir d'hypothèses validées à l'aide d'oracles. Une liste non exhaustive d'heuristiques générales est proposée. La démonstration s'appuie sur un cas réel de modélisation d'apprentissage de schémas d'improvisation musicale.

Mots clés

inférence grammaticale, inférence inductive, automates finis, langages formels, classification

GRTC / 303 / janvier 1989

Bernard Bel

Journées Françaises de l'Apprentissage (JFA 90), pp.5-27

INFERENCE DE LANGAGES REGULIERS

Bernard Bel

*Groupe Représentation et Traitement des Connaissances
Centre National de la Recherche Scientifique
31 ch. J. Aiguier, 13402 Marseille Cedex 9 (France)
E-mail: bel@frmop11*

RESUME

Cet exposé présente une méthode générale d'acquisition de connaissances dans un domaine formalisé à l'aide d'automates finis (langages réguliers). A partir d'un échantillon d'exemples le système construit un automate "presque minimal" qui n'est pas nécessairement déterministe. Cette construction peut être contrainte par des connaissances sur la segmentation des exemples que le système acquiert en questionnant l'informateur. Dans une deuxième phase, le système généralise l'automate à partir de propriétés connues du langage ou(et) à partir d'hypothèses validées à l'aide d'oracles. Une liste non exhaustive d'heuristiques générales est proposée. La démonstration s'appuie sur un cas réel de modélisation d'apprentissage de schémas d'improvisation musicale.

MOTS CLES

inférence grammaticale, inférence inductive, automates finis, langages formels, classification

1. Introduction

Etant donné un ensemble fini ou infini L , partitionné en p classes, de chaînes sur un alphabet A (un langage), on cherche à construire des caractérisations minimales de chaque classe pertinentes du point de vue d'un expert du domaine. Caractériser les classes d'un ensemble fini de chaînes est un problème de classification¹. Lorsque l'ensemble est infini (on ne connaît pas de borne supérieure de la longueur des chaînes) on dispose d'un échantillon d'exemples $\{\Sigma_1, \dots, \Sigma_p\}$, où Σ_i désigne un ensemble de chaînes appartenant à la classe i . On connaît par ailleurs (ou l'on suppose connues) des propriétés de L et de sa partition qui permettent de choisir une famille de fonctions discriminantes parmi lesquelles on cherche une caractérisation de la partition de L . Ce problème appartient à l'inférence inductive [CASE82].

¹ Une approche consiste par exemple à énumérer les plus courtes sous-chaînes de chaque classe qui ne sont sous-chaînes d'aucune autre classe, et à construire des fonctions discriminantes qui sont des disjonctions d'opérateurs de la forme "présence d'une sous-chaîne" (Guénoche 1989). Nous n'abordons pas ici un autre problème de classification qui consiste à construire une partition "optimale" d'un ensemble donné. Nous supposons ce problème préliminaire résolu, soit que la partition est donnée ou qu'elle a été construite à partir d'une fonction de similarité ou de distance entre chaînes.

Supposons ce problème résolu, et soit ϕ la fonction discriminante de la partition de L . Pour toute chaîne x de L , $\phi(x)$ retourne un entier qui est le numéro de la classe correspondante. Le résultat de $\phi(x)$ pour $x \notin L$ est ignoré. Lorsque L est infini, ce type de caractérisation ne permet donc pas de se prononcer sur une chaîne arbitraire de A^* . Il est indispensable dans ce cas de rechercher des fonctions caractéristiques: pour tout $x \notin L$, $\phi(x) \notin \{1, \dots, p\}$.

Concevoir L comme un ensemble infini revient à dire que, pour toute fonction caractéristique ϕ_{S_i} construite à partir d'un échantillon S_i de L , il peut exister une chaîne $x \notin S_i$ telle que $\phi_{S_i}(x)$ fournit une réponse incorrecte. Dans ce cas, $\phi_{S_i \approx \{x\}} \neq \phi_{S_i}$. Les méthodes qui permettent de calculer $\phi_{S_i \approx \{x\}}$ connaissant seulement ϕ_{S_i} , x et $\phi_L(x)$ (méthodes incrémentales) présentent un intérêt évident pour la construction de systèmes d'apprentissage.

Les systèmes de réécriture, plus particulièrement les grammaires formelles, permettent de caractériser des ensembles de chaînes en s'appuyant sur un formalisme susceptible de faire mettre en évidence des connaissances pertinentes pour un domaine, par exemple les catégories syntaxiques ou le lexique d'un langage. Outre les grammaires (et les automates qui leur sont équivalents, cf. [KAIN72]) il existe d'autres classes de fonctions caractéristiques récursives, par exemple les dérivations de motifs (*patterns*) [ANGL80b].

Nous commençons cet exposé avec le rappel de quelques résultats sur les classes de fonctions caractéristiques apprenables (§2). Par “apprenable” nous entendons ici “identifiable” [GOLD67] ou “fortement approchable” [BIER72] à la limite, notions qui ont été reformulées depuis d'un point de vue plus général [CASE82].

Dans un système d'apprentissage incrémental il est nécessaire d'effectuer des inférences “correctes” en présence d'exemples seuls. Nous énonçons au §3 les conditions qui permettent d'éviter toute surgénéralisation dans le cas des langages réguliers.

Nous nous intéressons enfin (§4-ff) à l'identification de langages réguliers décrits par des grammaires de type 2. Ces grammaires font apparaître des règles lexicales, dont les membres droits sont des agrégats de symboles qui peuvent avoir un “sens” pour un expert du domaine (§5-6). Les agrégats forment un lexique du langage incomplètement connu des informateurs, mais nous supposons que tout informateur est en mesure de valider toute segmentation d'une chaîne du langage proposée par le système. Le système d'apprentissage que nous avons réalisé sur ce modèle fonctionne comme suit:

- (1) Au départ le système n'a pas de connaissance du domaine, sauf certaines propriétés relatives à la classe de langage à inférer, ex. langage fini, etc.
- (2) On soumet au système un échantillon d'exemples. Le système construit un automate “presque minimal” qui reconnaît exactement les exemples et qui respecte la segmentation de chaque chaîne. Toute décision liée à la segmentation est validée au moyen de questions posées à l'informateur.
- (3) Etape facultative: le système complète la segmentation des chaînes connues en utilisant les agrégats (mots ou suites de mots) déjà connus. La nouvelle segmentation est soumise à l'informateur.
- (4) Généralisation: le système recherche une partition des états de l'automate construit en (2) afin de construire un automate quotient. L'espace de recherche (i.e. l'ensemble des partitions plausibles) est limité par des contraintes liées aux propriétés du langage, et ordonné partiellement par des heuristiques. Toute équivalence entre deux états qui ne peut être déduite d'une propriété du langage est validée par l'examen des chaînes nouvelles reconnues par l'automate quotient (production de contrexemples par oracle, §7-8).
- (5) On reprend les étapes (2), (3), (4) jusqu'à ce que tout échantillon d'exemples soumis au système soit reconnu par l'automate.

2. Paradigmes d'apprenabilité

Gold ([GOLD67],[GOLD78]) s'est intéressé au cas général: un langage fini ou infini L partitionné en p classes, ou encore le monoïde A^* partitionné en $(p+1)$ classes telles que L est la réunion de p classes. Caractériser une telle partition revient à chercher une fonction caractéristique de chaque classe à partir d'un échantillon d'exemples (des chaînes de cette classe) et de contre-exemples (des chaînes d'autres classes).

On dit qu'un langage L est identifiable à la limite s'il existe un système qui, quelle que soit la séquence d'exemples et de contre-exemples, propose toujours la même fonction caractéristique ϕ après un nombre fini d'étapes, et si par ailleurs cette fonction caractérise L :

$$\exists i_0 \in \mathbb{N} \mid \forall j \geq 0, \phi_{i_0+j} = \phi_{i_0} \\ \text{et } L(\phi_{i_0}) = L$$

Autrement dit, le système commet toujours un nombre fini d'erreurs avant de reconnaître L .

Le théorème de Gold ([GOLD67 p.452], [BIER72 p.33]) établit les limites d'apprenabilité de classes de langages dans une hiérarchie compatible² avec celle de Chomsky. Nous en rappelons une version simplifiée:

- (1) La classe la plus générale de langages formels identifiables à la limite à l'aide d'exemples ou de contre-exemples est celle des langages primitivement rékursifs³.
- (2) Seule la classe des langages finis est identifiable à la limite à partir d'exemples seuls.

Selon la proposition (2) du théorème, toute classe de langages qui contient tous les langages finis et au moins un langage infini n'est pas identifiable à la limite à partir d'exemples seuls. Certaines classes de langages non comparables avec celle des langages finis sont toutefois apprenables à la limite à partir d'exemples seuls, par exemple les langages pivots, sous-classe des langages de type 2 [FU75a p.105], les langages de motifs [ANGL80b] et les langages k-réversibles ([ANGL82a], [BERW87]).

Un autre paradigme d'apprenabilité de langage moins contraignant que celui de l'identification est celui d'approche (forte) à la limite [BIER72 p.34]. Informellement, à chaque étape le système doit proposer une fonction caractéristique de plus en plus proche de celle qui est recherchée⁴. On montre que tout système de réécriture décidable peut être approché fortement à la limite à partir d'exemples seuls.

² C'est à dire toute famille indexée de classes (C_1, \dots, C_n) telle que C_i contient strictement C_{i+1} et pour toute classe L_i de la classification de Chomsky il existe j tel que $C_j = L_i$.

³ Les langages primitivement rékursifs sont ceux générés par une classe énumérable de grammaires décidables.

⁴ La distinction de ces deux paradigmes est de peu d'intérêt pratique: ni l'informateur ni le système d'apprentissage ne savent si le langage a été identifié ou non. Même si l'hypothèse reste stable on ne peut garantir qu'un nouvel exemple ou contre-exemple ne viendra pas l'infirmier.

3. Généralisation d'une fonction caractéristique de langage régulier

(Voir en annexe les définitions: accepteur fini, ... préfixe arborescent, ... canonique, etc.)

Dans la suite de cette communication nous nous intéressons au cas où le monoïde A^* est partitionné en deux classes, à savoir un langage régulier (inconnu) L et son complément. Etant donné un échantillon d'exemples S_i du langage, on peut construire l'accepteur préfixe arborescent $A_{\text{pref}}(S_i)$ qui reconnaît exactement S_i . Généraliser cette fonction caractéristique revient à trouver un accepteur fini qui reconnaisse un langage M tel que $L \in M \in S_i$.

Théorème 1

Soient S_i un échantillon d'exemples d'un langage régulier inconnu L , et $A_{\text{pref}}(S_i) = (A, E, \{\lambda\}, T, \{0,1\}, \varphi)$ son accepteur préfixe arborescent. Soit R la congruence⁵ sur A^* telle que: $R(u,v) \Leftrightarrow \text{Tail}(L,u) = \text{Tail}(L,v)$.
L'accepteur quotient $A' = A_{\text{pref}}(S_i) / R$ est isomorphe d'un sous-accepteur de l'accepteur canonique $A(L)$.

[BEL89 p.45]

Tout langage reconnu par A est aussi reconnu par A/R . Par conséquent, $L \in L(A/R) \in S_i$.
La généralisation n'est possible toutefois que si R est connue, ce qui n'est pas vrai en général.

Le théorème suivant [ANGL80a p.121] permet d'énoncer la condition à remplir pour que cette construction réalise effectivement l'identification à la limite. Rappelons que, pour tout langage L_i d'une famille énumérable, on appelle échantillon caractéristique un ensemble fini de chaînes S_c tel que $L_i \in S_c$, et pour tout $j \geq 1$, si L_j contient S_c alors L_j n'est pas inclus dans L_i .⁽⁶⁾

Théorème 2

Une famille énumérable de langages récursifs non vides (L_1, \dots, L_i, \dots) est identifiable à la limite à partir d'exemples seuls si et seulement s'il existe une procédure effective qui à toute valeur de $i \geq 1$ associe un échantillon caractéristique de L_i .

Il est clair que dans ce cas lorsque $S_i \in S_c$ alors $M = L$.

4. Accepteur "presque minimal" d'un langage fini

Le théorème 1 ne permet pas en général de réaliser un système d'apprentissage incrémental: à l'étape suivant la généralisation on ne peut pas remplacer $A_{\text{pref}}(S_i)$ par $A_{\text{pref}}(S_i)/R$, puisque la construction doit se faire sur l'accepteur préfixe arborescent qui reconnaît exactement S_{i+1} . Nous présentons maintenant une méthode incrémentale qui, de plus, tente de minimiser la représentation courante de l'accepteur reconnaissant exactement S_i . Nous appelons "presque minimal" un tel accepteur. La généralisation proprement dite est effectuée indépendamment de cette construction (voir §7).

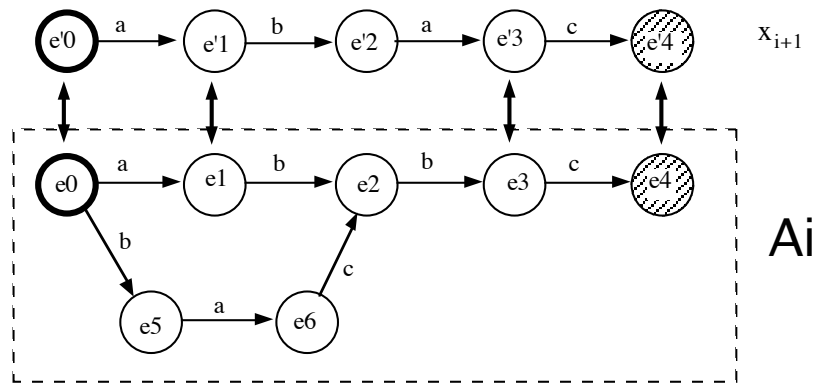
⁵ R est aussi une relation d'équivalence sur E puisque $E = \text{Pref}(L)$.

⁶ Autrement dit, L_i est le plus "petit" langage de la famille qui contienne S_c .

Soient S_i un échantillon de i exemples du langage L , et $A_i = (A, E, S, T, \{0,1\}, \varphi)$ un accepteur fini (non déterministe)⁷ reconnaissant exactement S_i . Appelons x_{i+1} le $(i+1)$ ème exemple présenté. Nous cherchons à construire un accepteur A_{i+1} qui reconnaisse exactement $S_{i+1} = S_i \cup \{x_{i+1}\}$ tout en remplissant certaines conditions de minimalité. Construisons en premier lieu l'accepteur $A' = (A, E' \approx E', S' \approx \{e'_0\}, T', \{0,1\}, \varphi')$ tel que:

$$\begin{aligned} E' &= \text{Pref}(\{x_{i+1}\}) ; E \leftrightarrow E' = \emptyset ; \\ \forall e \in E', e \neq x_{i+1} &\Leftrightarrow \varphi'(e) = 0, \text{ et } e = x_{i+1} \Leftrightarrow \varphi'(e) = 1 ; \\ \forall e \in E, \varphi'(e) &= \varphi(e) ; \\ \forall t'_j \in T', \\ \forall e \in E, t'_j(e) &= t_j(e) ; \\ \forall e \in E', t'_j(e) &= \text{Tail}(\{x_{i+1}\}, e a_j) \text{ avec } a_j \in A \end{aligned}$$

Il est facile de prouver que A' reconnaît exactement S_{i+1} . La figure ci-dessous illustre le cas où $S_i = \{abbc, bcbc\}$ et $x_{i+1} = abac$:



Pour tout état e d'un accepteur (non-déterministe) $(A, E, S, T, \{0,1\}, \varphi)$ dont f est la fonction de transition, nous appelons $H(e)$ (resp. $T(e)$) les langages générés en considérant cet état comme unique état acceptable (resp. initial), soit:

$$\begin{aligned} H(e) &= \{u \in A^* \mid \exists e_0 \in S, e \in f(e_0, u)\} \\ T(e) &= \{v \in A^* \mid \exists e_f \in f(e, v), \varphi(e_f) = 1\} \end{aligned}$$

Nous considérons maintenant la relation Rr sur l'ensemble des états E telle que:

$$\begin{aligned} \forall e_1, e_2 \in E, Rr(e_1, e_2) &\Leftrightarrow \\ Rr(e_2, e_1) \text{ ou} \\ H(e_1) = H(e_2) \text{ ou } T(e_1) = T(e_2), &\text{ et } \varphi(e_1) = \varphi(e_2) \end{aligned}$$

Il est évident qu'on a $Rr(e_2, e_1)$ pour tout couple d'états initiaux (resp. acceptables), les fonctions H (resp. T) valant $\{\lambda\}$ dans ce cas. Rr est une relation de ressemblance (pas nécessairement transitive).

Dans la figure ci-dessus, la relation Rr est indiquée à l'aide d'une flèche en gras.

⁷ On a évidemment $S_0 = \emptyset$ et A_0 est l'accepteur vide.

Nous construisons maintenant une relation d'équivalence $R0$ comme suit: les états de l'accepteur sont numérotés et l'on a:

$$\forall e_i, e_j \in E, R0(e_i, e_j) \Leftrightarrow R0(e_j, e_i) \text{ ou } i \leq j, Rr(e_i, e_j), \text{ et l'on n'a pas } R0(e_i, e_k) \text{ avec } k < j \text{ ni } R0(e_k, e_i) \text{ avec } k < i.$$

Puisqu'on ne peut avoir $R0(e_i, e_j)$ et $R0(e_j, e_k)$ avec $i \neq j, j \neq k$ et $i \neq k$, la transitivité de $R0$ est toujours vérifiée. Il n'est pas difficile de démontrer que $A'/R0$ reconnaît exactement $S_i \approx \{x_{i+1}\}$.

Pour une relation Rr donnée, $R0$ dépend de la numérotation des états. Dans le cas de l'accepteur A' , on numérote les états en commençant par E' , dans l'ordre de longueur croissante des préfixes:

$$(1) \forall e_i, e_j \in E \approx E', e_i \in E \text{ et } e_j \in E' \Rightarrow i > j$$

$$(2) \forall e_i, e_j \in E', |e_i| > |e_j| \Rightarrow i > j$$

Dans la proposition (2) il faut comprendre que e_i et e_j ne sont autres que des préfixes de x_{i+1} , c'est à dire des chaînes sur A . Avec cette numérotation on a toujours $R0(e_i, e_j)$ pour e_i, e_j états initiaux, mais pas nécessairement lorsque $\varphi(e_i) = \varphi(e_j) = 1$ (états acceptables).

Pour l'accepteur de la figure précédente on a $R0 = Rr$. Nous montrons plus loin un cas plus général.

On peut montrer que si l'on a $R0(e_j, e_k)$ avec $e_j \in E'$, l'ensemble des états nouvellement créés, on a nécessairement $e_k \in E$ et:

$$\text{Si } H(e_j) = H(e_k) \text{ alors } R0(e_j, e_k) \text{ pour } e_j, \text{ et } e_k, \text{ 1-leaders de } e_j \text{ et } e_k \text{ respectivement;}$$

$$\text{Si } T(e_j) = T(e_k) \text{ alors } R0(e_j, e_k) \text{ pour } e_j, \text{ et } e_k, \text{ 1-suiveurs de } e_j \text{ et } e_k \text{ respectivement.}$$

Par exemple, $R0(e'_1, e_1) \Rightarrow R0(e'_0, e_0)$, et $R0(e'_3, e_3) \Rightarrow R0(e'_4, e_4)$ ci-dessus. Pour calculer $R0$ il suffit donc d'énumérer les préfixes u de x_{i+1} , par longueur croissante à partir de λ , en cherchant le sous-ensemble E_u de E tel que $\forall e \in E_u, H(e) = \{u\}$. Concrètement, on cherche l'état e_m de E le plus éloigné d'un⁸ état initial de A_i tel que l'unique élément de $H(e_m)$ est un préfixe de x_{i+1} . (Si cet état est final, x_{i+1} est reconnu par A_i et on peut passer à x_{i+2} .) On cherche de même l'état e_p de E le plus éloigné d'un état acceptable de A_i tel que l'unique élément de $T(e_p)$ est un suffixe de x_{i+1} . Dans cette énumération on a trouvé tous les couples (e_j, e_k) qui satisfont la relation Rr , avec $e_j \in E'$. L'accepteur A_i étant simplifié (algorithme ci-dessous), ces couples sont les seuls qui peuvent satisfaire la relation $R0$. On construit enfin la relation $R0$ en énumérant dans l'ordre les couples satisfaisant Rr .

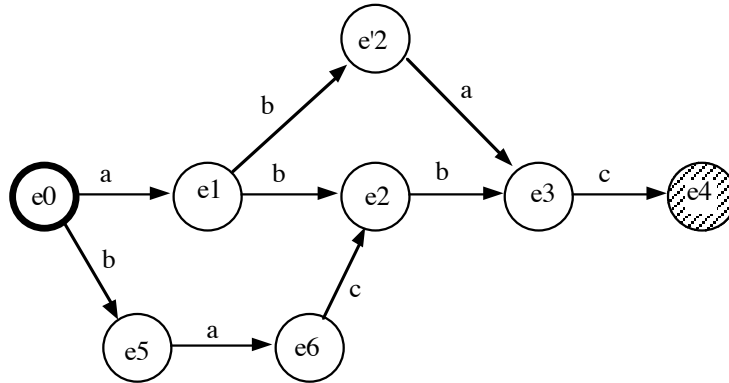
Simplification de l'accepteur

L'algorithme de simplification de l'accepteur A' est le suivant:

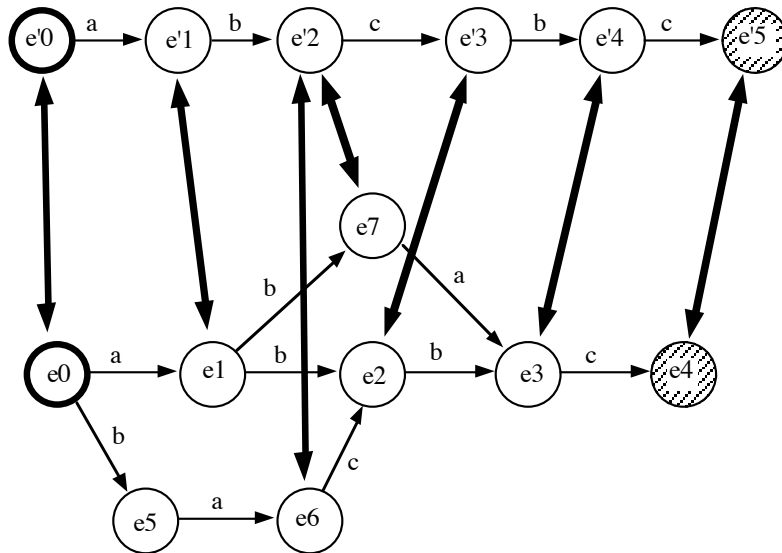
$$\text{Tant qu'il existe } e_i, e_j \in E \text{ tels que } R0(e_i, e_j) \text{ et } i \neq j, \text{ où } E \text{ est l'ensemble des états de } A', \text{ remplacer l'accepteur } A' \text{ par } A'/R0 \text{ et recalculer } R0.$$

⁸ Nous montrons plus loin que cet état initial est unique.

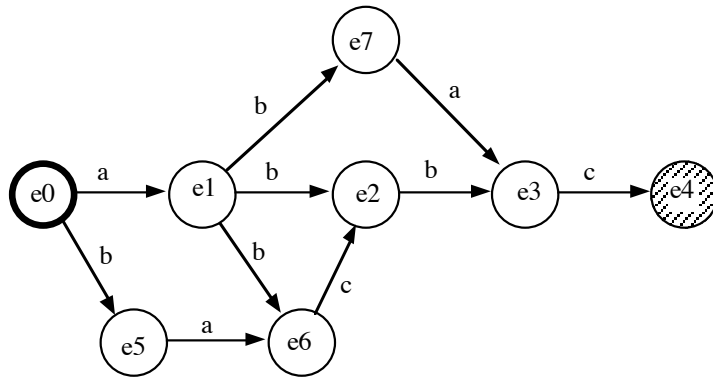
L'algorithme s'arrête nécessairement puisque $\text{card}(E)$ décroît strictement et qu'on ne peut avoir $R0(e_i, e_j)$ et $i \neq j$ lorsque $\text{card}(E) = 1$. L'accepteur fini obtenu après simplification est A_{i+1} . Pour l'accepteur A' ci-dessus on obtient A_3 :



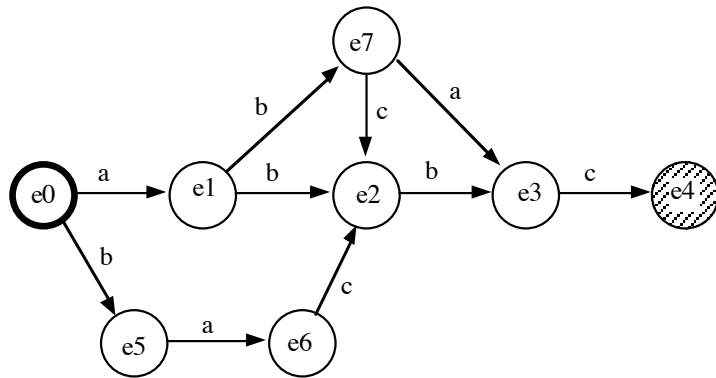
Nous montrons maintenant un cas où l'on n'a pas $Rr = R0$. Supposons que l'exemple suivant soit "abcbe". Nous montrons ci-dessous l'accepteur A' et la relation Rr :



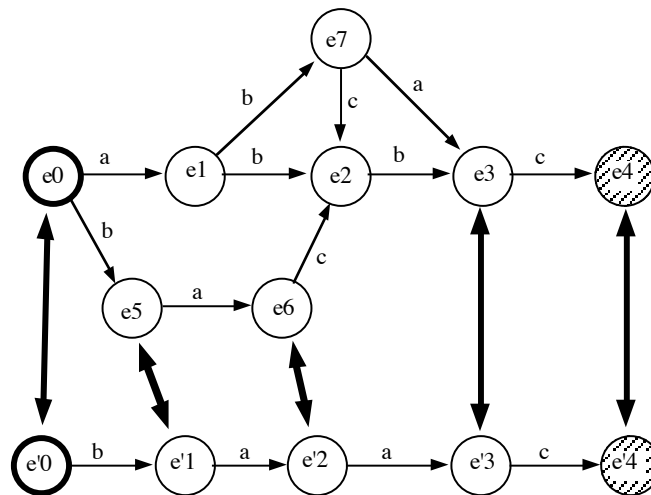
Pour construire $R0$ on examine dans l'ordre les couples (e'_0, e_0) , (e'_1, e_1) , (e'_2, e_6) , (e'_2, e_7) , (e'_3, e_2) , (e'_4, e_3) et (e'_5, e_4) . On ne peut avoir à la fois $R0(e'_2, e_6)$ et $R0(e'_2, e_7)$, c'est donc le premier couple énuméré qui est choisi. L'accepteur A_4 est dans ce cas:



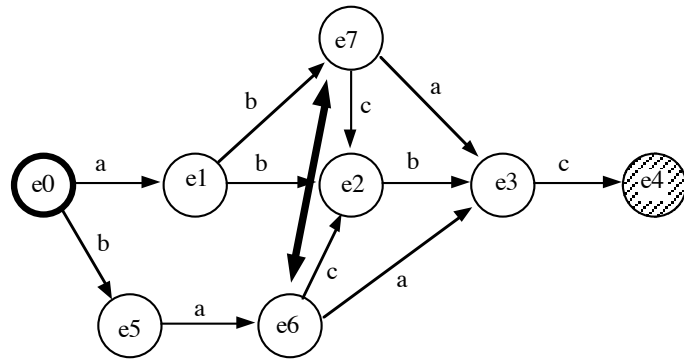
En choisissant une autre numérotation de E telle que l'ordre (e'_2, e_6) , (e'_2, e_7) soit inverse, on obtient pour A_4 :



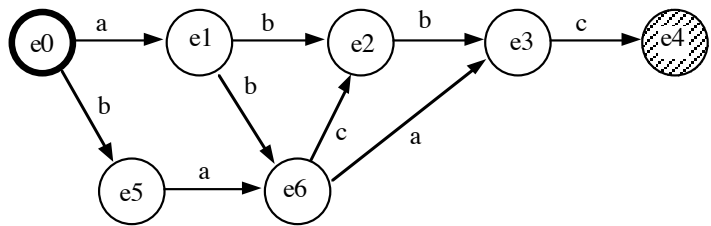
Gardons ce dernier accepteur et supposons que le prochain exemple soit "baac". On obtient alors



où $Rr = R0$, ce qui donne après simplification:



Nous avons mis en évidence un nouveau couple (e_6, e_7) vérifiant Rr et $R0$. En effet, $T(e_6) = T(e_7) = \{cbc, ac\}$. La simplification se poursuit donc et nous obtenons finalement A_5 :



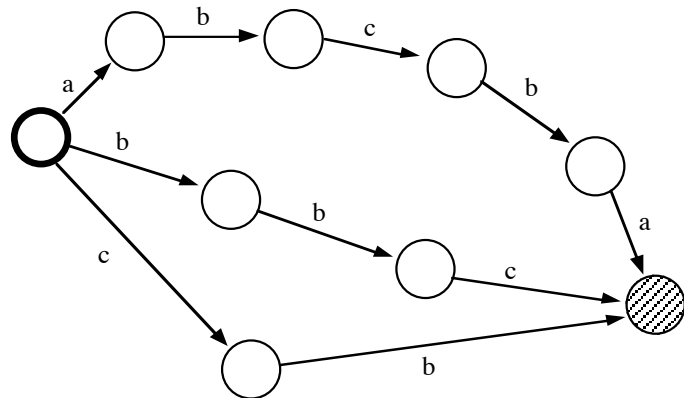
Complexité de l'accepteur construit

Dans l'exemple que nous avons traité ci-dessus, la dernière simplification n'aurait pas été possible avec le premier choix de numérotation de E . Le nombre d'états de l'accepteur construit dépend donc en général de la numérotation, c'est à dire pratiquement de l'ordre dans lequel les exemples sont fournis.

Dans le cas le plus défavorable, la relation Rr n'est vérifiée que pour les états initiaux et les états finaux. On obtient alors un accepteur trivial A_T tel que:

$$\forall e \in E, \\ e \notin S \text{ et } \varphi(e) = 0 \iff \text{card}(H(e)) = \text{card}(T(e)) = 1$$

comme par exemple:



Le nombre d'états de l'accepteur trivial A_T est $2+n-i$, où i est le nombre de chaînes et n la somme des longueurs des chaînes. A_T est nécessairement déterministe: il n'a qu'un état initial, et si l'on avait deux transitions de même étiquette de cet état initial vers deux états distincts e_1 et e_2 , on aurait alors $H(e_1) = H(e_2)$ et $T(e_1) \in T(e_2)$, soit $Rr(e_1, e_2)$ et $RO(e_1, e_2)$; la simplification permettrait donc de fusionner e_1 et e_2 . De manière similaire on peut montrer que l'accepteur inverse A_T^{-1} est aussi déterministe.

Théorème 3

Tout accepteur trivial construit par l'algorithme est canonique.

Preuve

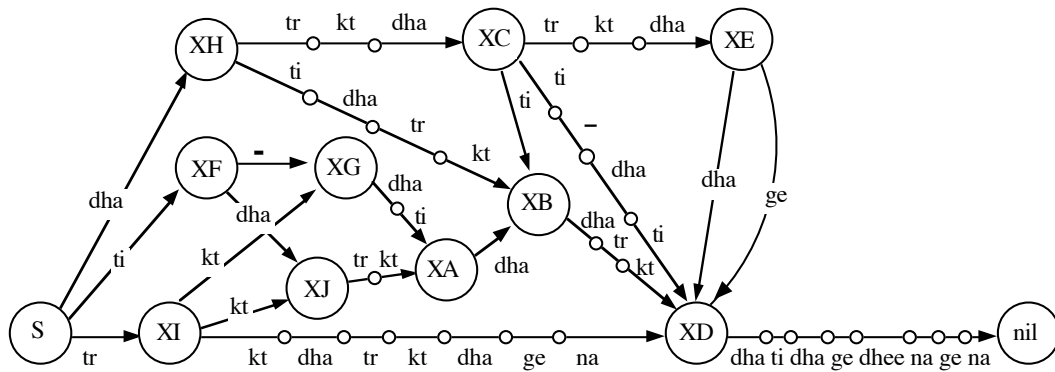
Soit $A_{\text{pref}}(S_i)$ l'accepteur préfixe arborescent du langage S_i . Puisque A_T^{-1} est déterministe la congruence R_t sur A^* telle que $R_t(u, v) \iff \text{Tail}(S_i, u) = \text{Tail}(S_i, v)$ n'est vérifiée que pour $u = v = \lambda$ ou bien $u, v \in S_i$. L'ensemble des états de $A_{\text{pref}}(S_i)$ est par définition l'ensemble des préfixes de S_i , que nous notons $\text{Pref}(S_i)$. La relation R_t n'est vraie, dans l'ensemble des états de $A_{\text{pref}}(S_i)$, que pour l'état initial $\{\lambda\}$ avec lui-même, et pour l'ensemble des états finaux S_i . L'accepteur quotient $A_{\text{pref}}(S_i)/R_t$ est donc exactement l'accepteur trivial A_T . D'après le théorème 1, cet accepteur est un sous-accepteur de l'accepteur canonique du langage S_i . Puisque A_T reconnaît S_i , il s'agit de l'accepteur canonique. ■

Nous venons de montrer que l'algorithme ne peut construire l'accepteur trivial que lorsque cet accepteur est lui-même canonique et donc minimal. Ce résultat ne dépend pas de l'ordre de présentation des exemples. En dehors de ce cas particulier l'accepteur construit peut se "rapprocher" de l'accepteur canonique de S_i lorsque les exemples ont des préfixes ou des suffixes communs.

A titre d'exemple nous donnons l'accepteur construit à partir de dix chaînes qui représentent les premières lignes de dix variations d'un *qa'ida*, un type de composition utilisé par les percussionnistes de l'Inde [KIPP89 pp.204-5]. Chaque syllabe (en fait une onomatopée) est un symbole terminal. Le tiret est un silence. L'échantillon d'exemples est le suivant:

1. tr kt dha ti dha dha tr kt dha ti dha ge dhee na ge na
2. ti dha tr kt dha dha tr kt dha ti dha ge dhee na ge na
3. dha tr kt dha ti dha tr kt dha ti dha ge dhee na ge na
4. dha tr kt dha tr kt dha ge dha ti dha ge dhee na ge na
5. dha tr kt dha tr kt dha dha dha ti dha ge dhee na ge na
6. dha tr kt dha ti - dha ti dha ti dha ge dhee na ge na
7. ti - dha ti dha dha tr kt dha ti dha ge dhee na ge na
8. dha ti dha tr kt dha tr kt dha ti dha ge dhee na ge na
9. tr kt tr kt dha dha tr kt dha ti dha ge dhee na ge na
10. tr kt dha tr kt dha ge na dha ti dha ge dhee na ge na

et l'accepteur résultant:



5. Connaissances lexicales

Sur l'accepteur précédent seuls les états correspondant à des noeuds du graphe coloré ont été étiquetés. Cette disposition suggère d'écrire la grammaire sur deux "niveaux":

S → TE1 XI	XH → TF4 XB
XI → TA7 XD	XH → TA3 XC
XD → TA8	XC → TE4 XD
XI → TF1 XJ	XC → TA3 XE
XJ → TC2 XA	XE → TA1 XD
XA → TA1 XB	XE → TC1 XD
XB → TB3 XD	XC → TB1 XB
XI → TF1 XG	S → TB1 XF
XG → TB2 XA	XF → TA1 XJ
S → TA1 XH	XF → TD1 XG

TA7 → ktdhatrkt dhagena	TE4 → ti-dhati
TC2 → trkt	TC1 → ge
TE1 → tr	TB3 → dhatrkt
TF1 → kt	TA8 →
TF4 → tidhatrkt	dhatidhagedheenagena
TD1 → -	TA3 → trktdha
TB2 → dhati	TB1 → ti
	TA1 → dha

dans lesquels les règles de type 2 (celles qui contiennent deux variables dans l'argument droit) sont des règles structurelles, et les règles de type 3 (celles du bas) sont des règles lexicales. Ce mode de représentation est rigoureusement équivalent à celui d'un accepteur fini, mais il met en évidence des agrégats de symboles pertinents au domaine, et que nous allons définir formellement.

Considérons la partition de A^* induite par la relation d'équivalence R_m telle que:

$$R_m(x,y) \Leftrightarrow \forall u,v \in A^*, uxv \in L \Leftrightarrow uyv \in L .$$

On appelle monoïde syntaxique de L l'ensemble des classes de A^*/R_m . Deux cas peuvent se présenter:

(1) Si x et y ne sont facteurs d'aucune chaîne de L , on a $uxv \notin L$ et $uyv \notin L$ pour tout u,v , et par conséquent $R_m(x,y)$. Appelons C_0 la classe qui contient toutes ces chaînes n'apparaissant dans aucune chaîne de L .

(2) Considérons x et y , deux facteurs de chaînes de L vérifiant $R_m(x,y)$ et appartenant à la classe C_j ($j \neq 0$) de A^*/R_m . Appelons e_0 l'état initial de l'accepteur canonique $A_{can}(L)$ et f sa fonction de transition. Posons $f(e_0,u) = \{e_1\}$, $f(e_0,ux) = \{e_2\}$, $f(e_0,uy) = \{e_3\}$.

L'accepteur canonique étant déterministe, les images de f sont des ensembles vides ou de cardinal 1. Nous pouvons imposer, pour satisfaire la condition du théorème 2, que L possède un échantillon caractéristique S_c . D'après le théorème 1, $A_{can}(L)$ n'est autre que $A_{pref}(S_i)/R$ avec $S_i \in S_c$, où $A_{pref}(S_i)$ est l'accepteur préfixe arborescent de S_i et R la relation d'équivalence sur A^* telle que: $R(x,y) \Leftrightarrow Tail(L,x) = Tail(L,y)$.

Par définition, la relation $R_m(x,y)$ impose $Tail(L,ux) = Tail(L,uy)$, et donc $R(ux,uy)$. Par conséquent, e_2 et e_3 appartiennent à la même classe de $A_{pref}(S_i)/R$, ce sont deux états fusionnés de l'automate préfixe arborescent.

A chaque classe C_j ($j \neq 0$) de A^*/R_m on peut donc associer deux états: $e(j) = e_1$ et $e'(j) = e_2$ (pas nécessairement distincts) de $A_{can}(L)$. Il est facile de montrer qu'à toute paire d'états de $A_{can}(L)$ correspond au plus une seule classe C_j , par conséquent le monoïde syntaxique d'un langage régulier est fini. Tous les états ainsi associés à des classes C_j sont des noeuds du graphe représentant $A_{can}(L)$. Chaque classe contient un nombre fini ou infini de chaînes de A^* ; appelons X_j l'ensemble de chaînes de la classe C_j :

$$X_j = \{x \in A^* \mid f(e(j),x) = \{e'(j)\}\}$$

Appelons X la réunion des X_j pour $j = 1, \dots, n$ où n est le nombre de classes de A^*/R_m . Nous appelons vocabulaire de L la partie V de X telle que:

$$\forall x \in V, x = uv \Rightarrow u \notin V \text{ ou(et) } v \notin V.$$

Tout élément de V est appelé un mot. On peut montrer que V est un code⁹ sur l'alphabet A et que L est un ensemble de messages dans V . Concrètement, V est représenté par l'ensemble des plus courts chemins (orientés) qui relient deux noeuds du graphe sans traverser un troisième noeud.

Si nous représentons $A_{can}(L)$ sous la forme d'une grammaire de type 2, comme ci-dessus, dans laquelle chaque noeud est désigné par une variable, le vocabulaire est l'ensemble des arguments droits des règles lexicales. Nous appelons segmentation de $x \in L$ la factorisation (unique) $x = u_1 \dots u_{n(x)}$ avec $u_j \in V$ pour tout j .

Il est clair qu'une partie importante de l'acquisition de connaissances sur un langage fini ou infini L consiste à trouver la segmentation de L . Toutefois il n'est pas garanti que la segmentation, telle que nous venons de la définir, soit entièrement compatible avec celle qu'un informateur pourrait proposer. Nous étudions donc maintenant le cas où un informateur est chargé de valider toute décision du type: "x se segmente uv" pour $x = uv$. Appelons $Seg(x,u,v)$ la décision. Evidemment, on a toujours $Seg(x,\lambda,x)$, $Seg(x,x,\lambda)$, et l'on peut supposer que les deux règles de transitivité sont vérifiées:

$$\begin{aligned} Seg(x,u,v) \text{ et } Seg(u,w,z) &\Rightarrow Seg(x,w,zv) \\ Seg(x,u,v) \text{ et } Seg(v,w,z) &\Rightarrow Seg(x,uw,z) \end{aligned}$$

⁹ Toute chaîne de A^* possède au plus une factorisation dans V .

Nous appelons accepteur étoile $Ae(S_i)$ l'accepteur fini non-déterministe $(A, E, \{e_0\}, T, \{0, 1\}, \varphi)$ reconnaissant exactement S_i tel que:

$$\forall e \in E, \text{card}(H(e)) = 1, \text{ et } \text{card}(T(e)) = 1 \text{ si } e \neq e_0$$

Il est facile de prouver que l'accepteur préfixe arborescent $A_{\text{pref}}(S_i)$ est $Ae(S_i)/R_{S1}$, où R_{S1} est la relation d'équivalence sur E telle que:

$$\forall e_1, e_2 \in E, R_{S1}(e_1, e_2) \Leftrightarrow H(e_1) = H(e_2)$$

Chaque état e de $A_{\text{pref}}(S_i)$ est étiqueté par $H(e)$. L'accepteur canonique $A_{\text{can}}(L)$ est $A_{\text{pref}}(S_i)/R$, avec $S_i \in S_c$, où:

$$R(u_1, u_2) \Leftrightarrow \text{Tail}(L, u_1) = \text{Tail}(L, u_2) \\ \text{ et } u_1, u_2 \text{ sont les étiquettes de deux états } e_1, e_2 \text{ de } A_{\text{pref}}(S_i).$$

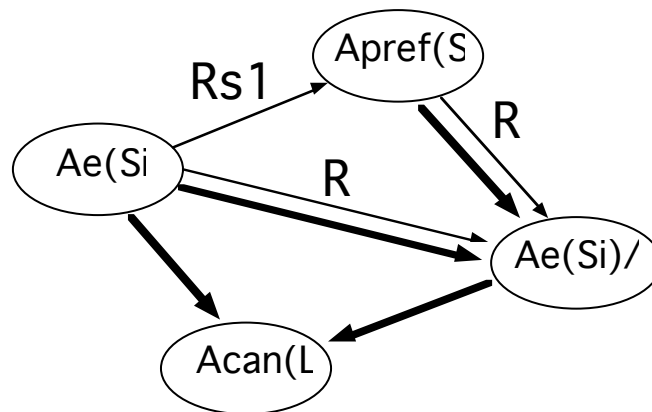
On peut donc écrire:

$$R(e_1, e_2) \Leftrightarrow \text{Tail}(L, H(e_1)) = \text{Tail}(L, H(e_2))$$

Par ailleurs, il est évident que:

$$\forall e_1, e_2 \in E, R_{S1}(e_1, e_2) \Rightarrow R(e_1, e_2)$$

Par conséquent $A_{\text{pref}}(S_i)/R$ est aussi $Ae(S_i)/R$, où R est étendue à E . Pour S_i quelconque on peut représenter ainsi les accepteurs et les quotients:



où $Ae(S_i)/R$ est isomorphe d'un sous-accepteur de $A_{\text{can}}(L)$, d'après le théorème 1. On a indiqué en gras la relation: "le langage de l'accepteur X est inclus dans celui de l'accepteur Y".

Appelons maintenant R' la relation d'équivalence sur les états de $Ae(S_i)$ telle que:

$$\forall e_1, e_2 \in E, R'(e_1, e_2) \Leftrightarrow R(e_1, e_2) \text{ et } \text{Seg}(H(e_1).T(e_1), H(e_1), T(e_1)) \text{ et } \text{Seg}(H(e_2).T(e_2), H(e_2), T(e_2)).$$

Selon R' , on ne peut fusionner deux états que si R est vérifiée et que chacun d'eux induit une segmentation acceptable sur la chaîne (unique) qui le traverse. Appelons $A_{\text{seg}}(S_i)$ l'accepteur quotient (en général non déterministe) $Ae(S_i)/R'$. On vérifie facilement que, puisque la partition

induite par R' affine celle induite par R , le langage L_{seg_i} reconnu exactement par $A_{\text{seg}(S_i)}$ est inclus dans le langage L_{nseg_i} reconnu exactement par $A_{\text{e}(S_i)}/R$, par conséquent:

$$L \in L_{\text{nseg}_i} \in L_{\text{seg}_i} \in S_i$$

et dans le cas où $S_i \in S_c$,

$$L = L_{\text{nseg}_i} \in L_{\text{seg}_i} \in S_i$$

6. Construction sous contraintes de A_i

Soit A_i n'importe quel accepteur fini (non déterministe) à un seul état initial qui reconnaît exactement S_i . On peut toujours concevoir A_i comme le quotient $A_{\text{e}(S_i)}/R_{s2}$, où R_{s2} est une relation d'équivalence convenablement choisie¹⁰ sur les états de $A_{\text{e}(S_i)}$.

Faire une inférence inductive correcte sur A_i revient à trouver une relation d'équivalence $R_{\text{inf}(i)}$ telle que le langage exactement reconnu par $A_i/R_{\text{inf}(i)}$ soit inclus dans L et que la segmentation induite par $A_i/R_{\text{inf}(i)}$ soit acceptable. Si nous appelons $R'(i)$ la relation d'équivalence sur E (l'ensemble des états de $A_{\text{e}(S_i)}$), telle que $A_i/R_{\text{inf}(i)} = A_{\text{e}(S_i)}/R'(i)$, il suffit d'avoir

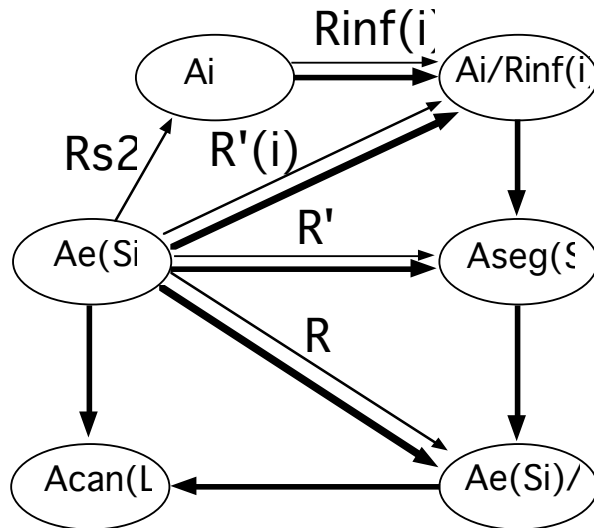
$$\forall e_1, e_2 \in E, R'(i)(e_1, e_2) \Rightarrow R'(e_1, e_2)$$

pour que le langage reconnu par $A_i/R_{\text{inf}(i)}$ soit inclus dans L_{seg_i} , donc a fortiori dans L , et que la segmentation soit acceptable. Comme on a par ailleurs $R_{s2}(e_1, e_2) \Rightarrow R'(i)(e_1, e_2)$, on doit choisir R_{s2} puis $R_{\text{inf}(i)}$ de sorte que:

$$\forall e_1, e_2 \in E, R_{s2}(e_1, e_2) \Rightarrow R'(e_1, e_2) \quad \text{et} \quad R_{\text{inf}(i)}(e_1, e_2) \Rightarrow R'(e_1, e_2)$$

La première condition est réalisée en contraignant la construction de A_i , mais pour la seconde il faudra disposer de contre-exemples puisque la relation R n'est pas connue. Les accepteurs ainsi construits, les quotients, et la relation d'inclusion de langages (en gras) peuvent être représentés ainsi:

¹⁰ Nous appelons R_{s1} et R_{s2} des relations de "simplification" de $A_{\text{e}(L_i)}$.



La contrainte sur la construction de A_i se traduit par une condition supplémentaire dans la relation d'équivalence $R0$ qui sert à simplifier l'automate A' . Nous allons établir cette condition.

Lorsqu'on cherche à construire $R0$ pour simplifier A' , on énumère les couples d'états (e_1, e_2) , où e_2 est un état de A_{i-1} , et pour $e = e_1$ puis pour $e = e_2$ on cherche à vérifier:

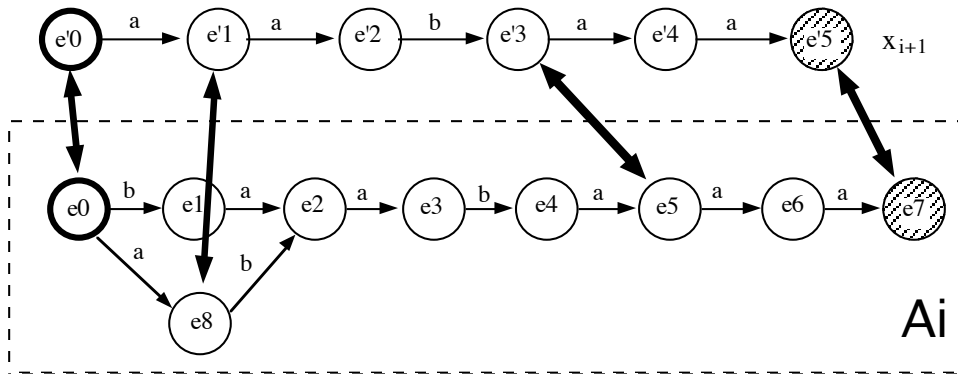
- (1) ou bien que e est déjà un noeud de A_{i-1} ;
- (2) ou encore que toute chaîne x dont le chemin $E_{A'}(x)$ contient e se segmente uv tel que:
 $e \in f(e_0, u)$ où e_0 est un état initial de A' , et $Seg(x, u, v)$.

Appelons $P(A', e)$ cette propriété. Il est clair que $P(A', e) \Rightarrow Seg(H(e), T(e), H(e), T(e))$. Nous remplaçons donc, pour simplifier A' , la relation $R0$ par R'_0 sur E , l'ensemble des états de A' , telle que:

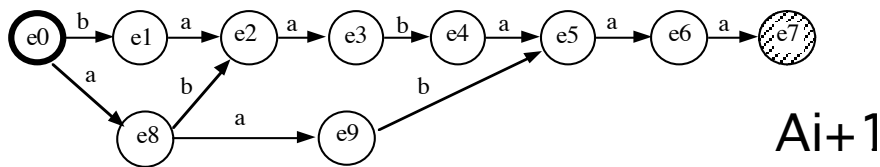
$$\forall e_1, e_2 \in E, \\ R'_0(e_1, e_2) \Leftrightarrow R0(e_1, e_2) \text{ et } P(A', e_1) \text{ et } P(A', e_2).$$

Puisque $R'_0(e_1, e_2) \Rightarrow R0(e_1, e_2)$ on construit encore un accepteur qui reconnaît exactement $S_i \approx \{x_{i+1}\}$, mais avec une segmentation correcte. On peut montrer que la simplification de A' aboutit à un accepteur $A_i = Ae(S_i)/Rs2$, où $Rs2$ respecte la condition: $Rs2(e_1, e_2) \Rightarrow R'(e_1, e_2)$.

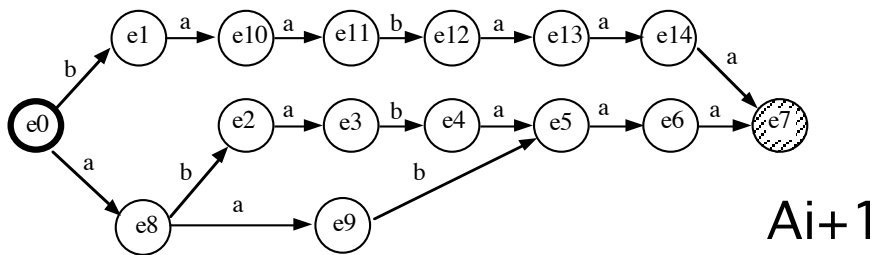
Dans la définition de la propriété P , nous avons souligné "toute" parce qu'il est souvent fastidieux de vérifier toutes les chaînes. Une approche réaliste consiste à vérifier $P(A', e)$ pour un petit nombre de chaînes dont le chemin contient e , au risque de segmenter incorrectement d'autres chaînes. On peut accepter ce type d'erreur dans la mesure où elle n'introduit pas de mot nouveau incorrect. Elle peut du reste être corrigée lorsqu'on demande à l'accepteur de générer toutes ou une partie des chaînes qu'il reconnaît et qu'on fait apparaître leur segmentation. Supposons par exemple que $S_i = \{baabaaa, ababaaa\}$ et que $x_{i+1} = aabaa$:



On a accepté $P(A', e'_1)$ parce que l'informateur accepte de segmenter $a/abaa$, $P(A', e'_3) \dots aab/aa$, $P(A', e_8) \dots a/babaaa$, mais pour $P(A', e_5)$ on a vérifié $ababa/aa$ seulement. On obtient ainsi A_{i+1} :



mais on s'aperçoit alors (ou plus tard) que $baaba/aa$ est une segmentation inacceptable. Il n'est pas très difficile de corriger A_{i+1} :



tout en gardant en mémoire l'information $P(A_{i+1}, e_{10})$.

La construction sous contraintes de A_i introduit donc un dialogue entre la machine et l'informateur, dialogue qui porte sur des segmentations autorisées. Le système acquiert ainsi une connaissance de X . Il peut ensuite utiliser cette connaissance pour prendre seul une décision sur la propriété P : c'est pourquoi le système pose de moins en moins de questions à l'informateur.

Il est possible de réduire le questionnement à l'aide de contraintes supplémentaires spécifiques au domaine. Une contrainte que nous utilisons pour les langages rythmiques est qu'aucun mot n'est de longueur inférieure à 2, sauf peut-être le silence "-".

Construire une table de vérité de $P(A_i, e)$ revient à segmenter complètement toutes les chaînes de S_i . Cette opération n'est possible qu'à condition de connaître tout le vocabulaire. Si l'on ne connaît qu'une partie de l'ensemble X des séquences possibles de mots, on peut toutefois, pour

n'importe quel A_i , énumérer les états pour lesquels $P(A_i, e)$ est plausible, et faire valider cette hypothèse par l'informateur. L'opération revient à examiner toute chaîne w de la partie connue de X , et si $w = uv$ avec $u \in X$ et $v \notin X$ on pose la question sur v . Inversement, si $u \notin X$ et $v \in X$, on pose la question sur u . A noter que l'informateur peut différer sa réponse: en pratique on construit une table de vérité de $P(A_i, e)$ à trois valeurs: vrai, faux et inconnu.

A partir d'un certain moment l'utilisateur peut décider que le système connaît entièrement le vocabulaire V . La construction de A_i se poursuit alors sans dialogue puisque le système est jugé assez compétent pour calculer lui-même la propriété P .

Il existe d'autres contraintes que l'on peut imposer à la construction de A_i , contraintes qui dépendent de propriétés du langage mais qui permettent par la suite de réaliser des inférences sans surgénéralisation. C'est le cas des langages k -réversibles [ANGL82a p.750]: tout échantillon d'un langage k -réversible L est reconnu exactement par un accepteur k -réversible, et d'autre part l'accepteur canonique de L est k -réversible. On peut donc limiter la recherche de A_i à l'ensemble des accepteurs k -réversibles. Pour cela, d'une part, A_i doit être déterministe, et d'autre part deux états e_1, e_2 ne peuvent vérifier $R^0(e_1, e_2)$ que s'ils ont un k -leader commun.

Nous effectués la construction sous contraintes de l'accepteur reconnaissant exactement les dix exemples de variations rythmiques ci-dessus (en limitant la recherche aux mots de longueur supérieure à 1, sauf “-”) [KIPP89 p.209]. Nous donnons ci-dessous la segmentation obtenue et la partie de X correspondante:

1. trkt / dhati / dhadrtrkt / dhatidhagedheenagena
2. tidha / trkt / dhadrtrkt / dhatidhagedheenagena
3. dhadrtrkt / dhati / dhadrtrkt / dhatidhagedheenagena
4. dhadrtrkt / dhadrtrkt / dhage / dhatidhagedheenagena
5. dhadrtrkt / dhadrtrkt / dhadrtrkt / dhatidhagedheenagena
6. dhadrtrkt / dhati-dhati / dhatidhagedheenagena
7. ti- / dhati / dhadrtrkt / dhatidhagedheenagena
8. dhatidhadrtrkt / dhadrtrkt / dhatidhagedheenagena
9. trkt / trkt / dhadrtrkt / dhatidhagedheenagena
10. trktdhadrtrktdhagena / dhatidhagedheenagena

Mots reconnus:

trktdhadrtrktdhagena	dhage
tidha	dhadrtrkt
dhatidhadrtrkt	dhadrtrkt
ti-	dhatidhagedheenagena
dhati-dhati	trkt
dhagedheenagena	trktdhatidhagedheenagena
dhati	dhadrtrkt

Arrivé à ce stade on a cherché à établir la table de vérité de $P(A_i, e)$ en posant des questions du type: “couper trktdhadrtrkt/dhagena?”, ce qui a abouti à la segmentation:

1. trkt / dhati / dhadrtrkt / trkt / dhati / dhagedheenagena
2. tidha / trkt / dhadrtrkt / trkt / dhati / dhagedheenagena
3. dhadrtrkt / dhati / dhadrtrkt / dhati / dhagedheenagena
4. dhadrtrkt / dhadrtrkt / dhage / dhati / dhagedheenagena
5. dhadrtrkt / dhadrtrkt / dhadrtrkt / dhati / dhagedheenagena
6. dhadrtrkt / dhati-dhati / dhati / dhagedheenagena
7. ti- / dhati / dhadrtrkt / trkt / dhati / dhagedheenagena
8. dhatidhadrtrkt / dhadrtrkt / dhati / dhagedheenagena
9. trkt / trkt / dhadrtrkt / trkt / dhati / dhagedheenagena
10. trkt / dhadrtrkt / dhagena / dhati / dhagedheenagena

7. Généralisation de l'accepteur presque minimal

Nous supposons maintenant connus l'accepteur presque minimal A_i (construit sous contraintes) qui reconnaît exactement S_i , et la propriété $P(A_i, e)$. Soit E l'ensemble des états de A_i . Nous choisissons $Rinf(i)$ telle que:

$$\forall e_1, e_2 \in E, Rinf(i)(e_1, e_2) \Rightarrow P(A_i, e_1) \text{ et } P(A_i, e_2),$$

afin de restreindre l'espace de recherche des couples d'états qui satisfont $Rinf(i)$ aux couples d'états correspondant à des noeuds sur le graphe, ou marqués comme vérifiant P . Il est facile de montrer que dans ce cas:

$$\forall e_1, e_2 \in E, Rinf(i)(e_1, e_2) \Rightarrow \text{Seg}(H(e_1).T(e_1), H(e_1), T(e_1)) \text{ et } \text{Seg}(H(e_2).T(e_2), H(e_2), T(e_2)).$$

Si de plus on a $R(e_1, e_2)$ alors on aura bien réalisé la condition $Rinf(i)(e_1, e_2) \Rightarrow R'(e_1, e_2)$. Pour vérifier $R(e_1, e_2)$ il suffit de vérifier que l'ensemble des chaînes nouvelles reconnues par l'accepteur quotient $A_i/Rinf(i)$ est inclus dans L . Le système génère donc cet ensemble et le soumet à l'informateur pour validation (oracle).

Une première coupure de l'espace de recherche de $Rinf(i)$ peut être imposée par des propriétés spécifiques du domaine, par exemple:

(1) Langage fini: on ne peut avoir $Rinf(i)(e_1, e_2)$ s'il existe $u \in A^*$ tel que $f(e_1, u) \in \{e_2\}$, où f est la fonction de transition de A_i . Dans le cas contraire, en effet, si $[e]$ est la classe d'équivalence contenant e_1 et e_2 , on aurait $f([e], u) \in \{[e]\}$ où f est la fonction de transition de $A_i/Rinf(i)$, et l'accepteur reconnaîtrait des mots contenant un facteur u^k pour tout entier k .

(2) Dans le cas des langages rythmiques, les chaînes ont des longueurs identiques, ou plus généralement des longueurs prenant à un très petit nombre de valeurs connues. Dans ce cas, $Rinf(i)(e_1, e_2) \Rightarrow \forall u_1 \in H(e_1), \forall u_2 \in H(e_2), |u_1| = |u_2|$.

La propriété (2) implique la propriété (1), et dans ce cas on a avantage à affecter, dès sa création, à chaque état e un entier qui représente lul pour tout $u \in H(e)$.

Il peut exister une propriété $Rspec$ telle que: $Rspec(e_1, e_2) \Rightarrow Tail(L, H()) = Tail(L, H())$ et donc $R(e_1, e_2)$. C'est le cas par exemple d'une propriété caractéristique des langages k -réversibles [ANGL82a p.750]:

$$\forall u_1, u_2, v, w \in A^* \text{ tels que } u_1 v w \in L, u_2 v w \in L \text{ et } |v| = k, \\ Tail(L, u_1 v) = Tail(L, u_2 v)$$

Par conséquent, si deux états e_1, e_2 d'un accepteur k -réversible sont tels que $T(e_1) \leftrightarrow T(e_2) \neq \emptyset$ et qu'ils ont en commun un k -leader v , alors $R(e_1, e_2)$ et dans ce cas $Rinf(i)(e_1, e_2)$ ne produit aucune surgénéralisation. La classe des langages k -réversibles est pour cette raison une classe de langages infinis identifiable à la limite à partir d'exemples seuls.

La condition de k -réversibilité entraîne d'autre part que l'accepteur quotient $A_i/Rinf(i)$ n'est pas ambigu. Dans le cas général, si l'on veut construire un accepteur non ambigu il suffit d'appliquer la règle:

$\forall e_1, e_2 \in E$, s'il existe $u \in H(e_1)$ et $v \in T(e_2)$ tels que uv est reconnu par A_i , alors on ne peut pas avoir $Rinf(i)(e_1, e_2)$.

8. Heuristiques de généralisation

Des heuristiques générales permettent d'ordonner partiellement l'espace de recherche de $Rinf(i)$. Supposons que pour deux états distincts e_1, e_2 on ait $P(A_i, e_1)$ et $P(A_i, e_2)$ et que par ailleurs ces états satisfassent les conditions spécifiques du domaine (langage fini, k-réversible, etc.). La relation $Rinf(i)(e_1, e_2)$ est plausible dans les cas suivants, par ordre de plausibilité décroissante:

- (1) $T(e_1) \in T(e_2)$ ou $H(e_1) \in H(e_2)$;
- (2) $T(e_1) \leftrightarrow T(e_2) \neq \emptyset$ ou $H(e_1) \leftrightarrow H(e_2) \neq \emptyset$;
- (3) e_1 et e_2 possèdent un k-leader (ou un k-suiveur) commun, pour des valeurs décroissantes de k.

Il faut ajouter des heuristiques dépendantes du domaine, par exemple celles basées sur des relations de ressemblance entre chaînes.

9. Conclusion

Nous avons abordé l'inférence de langages réguliers sous un angle général tout en considérant la prise en compte de contraintes relatives à divers domaines. La méthode proposée ici ne présuppose pas que la classe de langages considérée soit apprenable à la limite à l'aide d'exemples seuls, mais elle permet de limiter considérablement l'espace de recherche des généralisations (pour lesquelles une validation par oracles est nécessaire). Une originalité de la méthode est d'alterner les phases d'acquisition non inductive de connaissances à partir d'exemples seuls et celles où le système peut généraliser ces connaissances en produisant, le cas échéant, des contrexemples¹¹.

Les travaux actuels portent sur la construction de fonctions discriminantes d'un monoïde A^* partitionné en $(p+1)$ classes avec $p > 1$ (voir §2): il s'agit de construire, non plus un accepteur, mais un automate fini dont la fonction de sortie retourne le numéro de la classe¹².

10. Références

- [ANGL80a] Inductive inference of formal languages from positive data. Dana Angluin. *Information & Control*, Vol.45, N°2, 1980, pp.117-35.
- [ANGL80b] Finding patterns common to a set of strings. Dana Angluin. *Journal of Computer and System Sciences* 21, 1980, pp.46-62.

¹¹ Cette approche est une modélisation idéalisée de la méthode d'enseignement dans le domaine étudié.

¹² C'est dans ce but que nous avons jugé utile d'introduire le formalisme de la fonction de sortie, au lieu d'associer à tout automate l'ensemble de ses états acceptables.

- [ANGL82a] Inference of reversible languages. Dana Angluin. *Journal of the ACM*, Vol.29, N°3, 1982, pp.741-65
- [ANGL82b] A note on the number of queries to identify regular languages. Dana Angluin. *Information & Control*, Vol.51, 1982, pp.76-87.
- [ANGL83] Inductive inference: theory and methods. Dana Angluin. *Computing Surveys*, Vol.15, N°3, 1983, pp.237-69.
- [BEL89] Inférence d'un langage formel. Bernard Bel. Note interne GRTC 380, 1989.
- [BERW87] Learning syntax by automata induction. Robert C. Berwick & Sam Pilato. *Machine Learning*, Vol.2, N°1, 1987, pp.9-38.
- [BIER72] A survey of results in grammatical inference. A.W. Biermann & J.A. Feldman. *Frontiers of Pattern Recognition*, 1972, pp.31-54.
- [CASE82] Machine inductive inference and language identification. John Case & Christopher Lynes. *Intern. Coll. on Algorithms, Languages and Programming*, 1982, pp.107-15.
- [FU75a] Grammatical inference: introduction and survey - part I. King Sun Fu & Taylor L. Booth. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-5, N°1, 1975, pp.95-111.
- [FU75b] Grammatical inference: introduction and survey - part II. King Sun Fu & Taylor L. Booth. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-5, N°4, 1975, pp.409-22.
- [GOLD67] Language identification in the limit. E. Mark Gold. *Information and Control*, 10, 1967, pp.447-74.
- [GOLD78] Complexity of automation identification from given data. E. Mark Gold. *Information and Control*, 37, 1978, pp.302-20.
- [GUEN89] Algorithmes d'apprentissage dans les chaînes de caractères. Alain Guénoche. Journées Françaises de l'Apprentissage, St Malo, 1989.
- [KAIN81] *Automata theory: machines and languages*. Richard Y. Kain. Krieger, Malabar, 1981 (Edition originale 1972).
- [KIPP89] The identification and modelling of a percussion “language”, and the emergence of musical concepts in a machine-learning experimental set-up. Jim Kippen & Bernard Bel. *Computers & Humanities* 23.3, 1989, pp.199-214.

Annexe: définitions et notations

Préfixe, suffixe, tête et queue d'un langage

On appelle préfixe (resp. suffixe) d'une chaîne u sur l'alphabet A toute chaîne v (resp. w) telle que $u = v w$. Nous appelons préfixe (resp. suffixe) d'un langage L l'ensemble des préfixes (resp. suffixes) des chaînes de L .

Nous appelons tête (resp. queue) d'une chaîne u dans un langage L , et nous notons $\text{Head}(L,u)$ (resp. $\text{Tail}(L,u)$) l'ensemble des chaînes v de A^* telles que " $v u$ " (resp. " $u v$ ") est une chaîne de L .

Remarque: $\text{Head}(\emptyset,u) = \text{Tail}(\emptyset,u) = \emptyset$.

Automate fini, accepteur fini

Nous appelons automate fini le sextuplet (A,E,S,T,Y,φ) tel que:

$A = \{a_1, \dots, a_p\}$ est un alphabet;
 $E = \{e_1, \dots, e_n\}$ est un ensemble fini d'états;
 $S = \{s_1, \dots, s_q\}$ est une partie non vide de E appelée ensemble des états initiaux;
 $T = \{t_1, \dots, t_p\}$ est un ensemble fini de correspondances de E dans lui-même appelé ensemble des transitions;
 φ est une application univoque de E dans un ensemble quelconque Y .

On appelle fonction de sortie l'application φ .

L'automate est déterministe si les deux conditions suivantes sont remplies:

$$(1) \forall i \leq p, \forall e_j \in E, \text{card}(t_i(e_j)) < 2 \quad (2) \text{card}(S) \leq 1$$

L'inégalité dans la condition (2) permet de classer l'automate vide parmi les automates déterministes. A partir de l'ensemble des transitions $T = \{t_1, \dots, t_p\}$ on peut construire la fonction de transition $E \times A^* \rightarrow \mathcal{P}(E)$ que nous définissons récursivement:

$\forall u \in A^*$
(1) si $u = a_i$ avec $a_i \in A$, alors $\forall e_j \in E, f(u,e_j) = t_i(e_j)$;
(2) si $u = a_i v$ avec $a_i \in A$ et $v \in A^*$, alors $\forall e_j \in E,$
 $f(u,e_j) = \{e_k \mid \exists l, e_l \in t_i(e_j) \text{ et } e_k \in f(v,e_l)\}$

Cette fonction fait correspondre, à tout état e_j et à toute chaîne u , l'ensemble des états que l'on peut atteindre en parcourant l'automate à partir de e_j de sorte qu'à la k -ième étape on ne franchisse une transition que si elle est étiquetée par le k -ième symbole de u .

A tout automate fini (A,E,S,T,Y,φ) on peut associer un graphe p-coloré dont les sommets sont les états $\{e_1, \dots, e_n\}$ étiquetés par $\varphi(e_j)$, et dont les arcs de couleur q sont le graphe de la correspondance $t_q \in T$.

On appelle accepteur fini (*finite-state acceptor*) tout automate fini (déterministe ou non) (A,E,S,T,Y,φ) tel que $Y = \{0,1\}$. Tout état $e_j \in E$ tel que $\varphi(e_j) = 1$ est un état acceptable ou final (*accepting state*).

Une chaîne $x = x_1 \dots x_m$ de A^* est acceptée (reconnue) par l'accepteur \mathcal{A} si et seulement s'il existe une suite d'états de $\mathcal{A} : (e_1, \dots, e_m)$ telle que:

(1) $e_1 \in S$;
(2) $\forall k \in (1, \dots, m), e_{k+1} \in t_i(e_k)$ pour i tel que $a_i = x_k$ et $t_i \in T$;
(3) $\varphi(e_m) = 1$, c'est à dire e_m est un état acceptable de \mathcal{A} .

Nous appelons chemin de x dans \mathcal{A} , et nous notons $E_{\mathcal{A}}(x)$, la suite (e_1, \dots, e_m) .

Un langage L sur l'alphabet A est accepté (reconnu) par \mathcal{A} si et seulement si toute chaîne du langage est acceptée par \mathcal{A} . Si de toute chaîne de A^* acceptée par \mathcal{A} appartient à L , on dit que \mathcal{A} reconnaît exactement L . On note $L(\mathcal{A})$ le langage exactement reconnu par \mathcal{A} .

***k*-suiveur, *k*-leader**

Soit (A, E, S, T, Y, φ) un automate fini tel que $A = \{a_1, \dots, a_p\}$, $E = \{e_1, \dots, e_n\}$, $T = \{t_1, \dots, t_p\}$. Pour tout état $e_i \in E$, un symbole a_x de A est un 1-suiveur de e_i si et seulement si:

$$\exists j \in (1, \dots, n) \mid e_j \in t_x(e_i)$$

Soit $k \in \mathbb{N}$. Une chaîne $u \in A^*$ est un k-suiveur de e_i dans un des deux cas suivants:

$$\begin{aligned} (1) & k = 0 \text{ et } u = \lambda ; \\ (2) & u = a_x v \text{ tel que } a_x \in A, v \in A^*, a_x \text{ est un } \underline{1}\text{-suiveur de } e_i, \text{ et, pour } j \text{ tel que } e_j \in t_x(e_i), k' = k-1, v \\ & \text{est un } k'\text{-suiveur de } e_j . \end{aligned}$$

Il est facile de montrer que dans tous les cas $\text{lul} = k$. On définit un k-leader de manière duale: il suffit de remplacer, dans la définition précédente, " $e_j \in t_x(e_i)$ " par " $e_i \in t_x(e_j)$ ", et " $a_x v$ " par " $v a_x$ ".

Accepteurs finis isomorphes, sous-accepteurs, accepteur minimal

On dit que deux accepteurs finis: $(A, E, S, T, \{0, 1\}, \varphi)$ et $(A', E', S', T', \{0, 1\}, \varphi')$ avec $n = \text{card}(E)$ et $p = \text{card}(A)$ sont isomorphes si et seulement s'il existe une bijection h de E vers E' telle que:

$$\begin{aligned} h(S) &= S' \\ \forall i \in (1, \dots, n), \\ \varphi'(h(e_i)) &= \varphi(e_i) \text{ et } \forall j \in (1, \dots, p), t'_j(h(e_i)) = h(t_j(e_i)) \end{aligned}$$

Informellement, les accepteurs sont identiques à l'étiquetage des états près.

$(A', E', S', T', \{0, 1\}, \varphi')$ est un sous-accepteur de $(A, E, S, T, \{0, 1\}, \varphi)$, avec $n = \text{card}(E)$ et $n' = \text{card}(E')$ si et seulement si:

$$\begin{aligned} E' &\subseteq E, S' \subseteq S \\ \forall i \in (1, \dots, n') \\ \varphi'(e_i) &= \varphi(e_i) \text{ et } \forall j \in (1, \dots, p), t_j(e_i) \in t'_j(e_i) \end{aligned}$$

On montre facilement que si \mathcal{A}' est un sous-accepteur de \mathcal{A} , $L(\mathcal{A}') \in L(\mathcal{A})$.

Soient $\mathcal{A} = (A, E, S, T, \{0, 1\}, \varphi)$ un accepteur fini et E' une partie de E avec $n' = \text{card}(E')$. On dit que $\mathcal{A}' = (A', E', S', T', \{0, 1\}, \varphi')$ est un sous-accepteur de \mathcal{A} induit par E' si et seulement si:

$$\begin{aligned} S' &= S \Leftrightarrow E' \\ \forall i \in (1, \dots, n') \\ \varphi'(e_i) &= \varphi(e_i) \text{ et } \forall j \in (1, \dots, p), t_j(e_i) = t'_j(e_i) \Leftrightarrow E' \end{aligned}$$

Un état e_i d'un accepteur fini $\mathcal{A} = (A, E, S, T, \{0, 1\}, \varphi)$ est inutile si le langage reconnu par le sous-accepteur induit par $E \setminus \{e_i\}$ est exactement celui reconnu par \mathcal{A} . Dans le cas contraire, l'état considéré est utile. Tout accepteur fini qui ne contient pas d'état inutile est appelé accepteur restreint (*stripped*). Un accepteur $(A, E, S, T, \{0, 1\}, \varphi)$ est dit minimal pour un langage L s'il reconnaît exactement L et s'il n'existe pas d'accepteur $(A', E', S', T', \{0, 1\}, \varphi')$ reconnaissant exactement L tel que $\text{card}(E') < \text{card}(E)$. Tout accepteur minimal d'un langage est nécessairement restreint, mais la réciproque n'est pas vraie en général.

Quotient d'un accepteur fini par une relation d'équivalence sur ses états

Soit un accepteur fini $\mathcal{A} = (A, E, S, T, \{0, 1\}, \varphi)$ et une relation d'équivalence totale \mathcal{R} sur l'ensemble des états E . On appelle quotient de \mathcal{A} par \mathcal{R} , et on note \mathcal{A}/\mathcal{R} l'accepteur fini $(A', E', S', T', \{0, 1\}, \varphi')$ tel que:

$$\begin{aligned} E' &\text{ est l'ensemble des classes } E/\mathcal{R}; \\ \forall e_i \in S, \forall e'_j \in E', e_i \in e'_j &\Rightarrow e'_j \in S' ; \\ \forall e_i \in E, \forall e'_j \in E', e_i \in e'_j \text{ et } \varphi(e_i) = 1 &\Rightarrow \varphi(e'_j) = 1 ; \\ \forall e_i, e_j \in E, \forall t_q \in T, \forall e'_k, e'_l \in E', \\ t'_q \in T', e_i \in e'_k, e_j \in e'_l \text{ et } e_j \in t_q(e_i) &\Rightarrow e'_l \in t'_q(e'_k) \end{aligned}$$

Il est facile de montrer que, si f est la fonction de transition de \mathcal{A} , f' celle de \mathcal{A}/\mathcal{R} , pour toute chaîne $u \in A^*$, s'il existe $e_i \in E$ tel que $f(u, e_i) \neq \emptyset$, alors il existe $e_j \in S$ tel que $f(u, e_j) \neq \emptyset$. Autrement dit, à tout parcours sur \mathcal{A}

correspond un parcours sur \mathcal{A}/\mathcal{R} étiqueté identiquement par les symboles de u . En particulier, si $e_i \in S$ et $\varphi(f(u, e_i)) = 1$ alors $u \in L$; ce qui prouve que tout langage accepté par \mathcal{A} est aussi accepté par \mathcal{A}/\mathcal{R} . La réciproque n'est généralement pas vraie. On peut donc écrire

$$L(\mathcal{A}/\mathcal{R}) \in L(\mathcal{A})$$

où $L(X)$ désigne le langage exactement reconnu par l'accepteur X .

On peut prouver [BEL89 p.40] que si \mathcal{A} est déterministe alors \mathcal{A}/\mathcal{R} l'est aussi.

Accepteur canonique d'un langage régulier

Soit L un langage régulier sur l'alphabet A . Notons $\text{Pref}(L)$ l'ensemble des préfixes de L , et $\text{Tail}(L, u)$ la queue d'une chaîne u dans L . L'accepteur canonique de L est l'accepteur fini $\mathcal{A}(L) = (A, E, S, T, \{0, 1\}, \varphi)$ avec $p = \text{card}(A)$ et $T = \{t_1, \dots, t_p\}$ tel que:

- (1) $E = \{\text{Tail}(L, u) \mid u \in \text{Pref}(L)\}$;
- (2) $S = \{\text{Tail}(L, \lambda)\}$ lorsque $L \neq \emptyset$; $S = \emptyset$ dans le cas contraire ;
- (3) $\forall e_i \in E$, $e_i = \text{Tail}(L, u)$ avec $u \in L \Rightarrow \varphi(e_i) = 1$;
- (4) $\forall e_i = \text{Tail}(L, u) \mid u \in \text{Pref}(L)$,
 $\forall a_j \in A$, $u a_j \in \text{Pref}(L) \Rightarrow t_j(e_i) \in \text{Tail}(L, u a_j)$.

Par extension, tout accepteur fini d'un langage L est appelé canonique s'il est isomorphe de l'accepteur canonique $\mathcal{A}(L)$.

On peut prouver [BEL89 p.41] que l'accepteur canonique est déterministe, reconnaît exactement L , et qu'il est minimal.

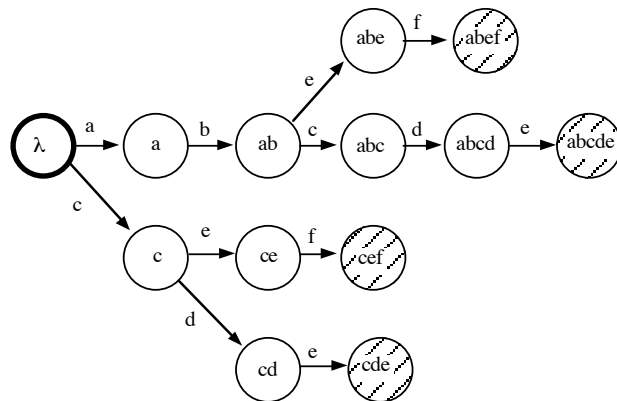
Accepteur préfixe arborescent d'un langage fini

Soit L un langage fini sur l'alphabet A . L'accepteur fini $\mathcal{A}_{\text{pref}}(L) = (A, E, S, T, \{0, 1\}, \varphi)$ tel que

- (1) $E = \text{Pref}(L)$;
- (2) $S = \{\text{Tail}(L, u) \mid u \in L\} = \{\lambda\}$;
- (3) $\forall e_i \in E$, $e_i \in L \Leftrightarrow \varphi(e_i) = 1$; sinon, $\varphi(e_i) = 0$;
- (4) $\forall e_i = \text{Tail}(L, u) \mid u \in \text{Pref}(L)$,
 $\forall a_j \in A$, $u a_j \in \text{Pref}(L) \Rightarrow t_j(e_i) \in \text{Tail}(L, u a_j)$.

est appelé accepteur préfixe arborescent de L . L'identité de la proposition (4) pour $\mathcal{A}_{\text{pref}}(L)$ et pour l'accepteur canonique de L permet de montrer que $\mathcal{A}_{\text{pref}}(L)$ est déterministe et qu'il accepte exactement L . L'ensemble des états finaux est L , qu'il ne faut pas confondre avec $\{\lambda\}$, l'état initial de l'accepteur canonique de L .

Exemple: $\mathcal{A}_{\text{pref}}(\{abcde, abef, cef, cde\})$ est:



Par convention, nous hâchurons les états e_i tels que $\varphi(e_i) = 1$ (états acceptables) et nous cerclons en gras les états initiaux.