MMIM Méthodes mathématiques en informatique musicale

Marc Chemillier

Master Atiam (Ircam), 15 février 2008

Notions théoriques sur les langages formels

- Notion d'automate fini
  - Automate fini déterministe (AFD)
  - Automate fini non-déterministe (AFN)
- Chaînes de Markov
- Oracle des suffixes
- Reconnaissabilité dans un monoïde quelconque
- Notion de grammaire formelle

1. Définition des automates finis déterministes (AFD)

1.1 Définition générale

Définition. *Un automate fini déterministe AFD sur un alphabet* Σ *est la donnée d'un n-uplet* (Q, δ, i, F) *où* :

• Q *est un ensemble fini d'états*,

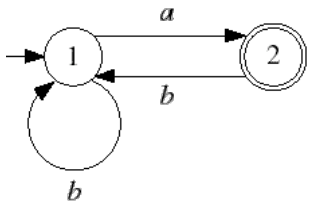
• δ *est une fonction de transition de* $Q \times \Sigma$ *dans* Q,

• i *est un état particulier de* Q *dit initial*,

• F *est une partie de* Q *d'états dits finals*.

L'automate est dit complet lorsque la fonction δ *est partout définie sur* $Q \times \Sigma$.

Exemple :



$Q = \{1, 2\}$,

$i = 1$, état noté avec une petite flèche entrante,

$F = \{2\}$, état noté avec deux cercles.

$\delta : Q \times \Sigma \to Q$

$(1, a) \to 2$

$(1, b) \to 1$

$(2, b) \to 1$

Cet automate n'est pas complet, car $\delta(2, a)$ n'est pas défini.

Pour décrire un automate, il est commode d'utiliser une table de transitions :

	1	2
a	2	
b	1	1

1.2 Prolongement de la fonction de transition

Le calcul de l'automate consiste à suivre des flèches, en partant d'un état initial et en s'arrêtant dans un état final. Le mot correspondant à ce calcul est la suite des étiquettes des flèches.

On prolonge δ par induction, en une fonction sur les mots de $Q \times \Sigma^* \to Q$:

(attention : la fonction δ est définie au départ sur $Q \times \Sigma$ et pas sur $Q \times \Sigma^*$):

$\delta(q, \varepsilon) = q$,

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

pour tous $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.

$\delta(q, w)$ = état atteint après lecture du mot w depuis un état q.

Propriété d'associativité. *Pour tous mots* $u, v \in \Sigma^*$, *on a :* $\delta(q, uv) = \delta(\delta(q, u), v)$.

Exemple : dans l'automate ci-dessus

a correspond au calcul $1 \to 2$,

aba correspond au calcul $1 \to 2 \to 1 \to 2$.

donc on écrira :

$\delta(1, aba) = 2$.

À partir de l'état 2, on ne peut lire un mot commençant par a, mais on peut lire $babb$ par exemple : $\delta(2, babb) = 1$.

La propriété d'associativité donne :

$\delta(1, ababbabb) = \delta(\delta(1, aba), babb) = \delta(2, babb) = 1$.

Le mot *aba* correspond à un calcul terminal car :

$\delta(1, aba) = 2 \in F$ terminal.

En revanche, *abababb* ne correspond pas à un calcul terminal de l'automate :

$\delta(1, abababb) = 1$ non terminal.

1.3 Langage reconnu par un AFD

Définition. *Le langage reconnu (ou accepté) par un automate AFD est l'ensemble des mots qui correspondent à un calcul de l'automate partant d'un état initial et s'arrêtant dans un état final.*

Exemple : le langage reconnu par l'automate ci-dessus est noté

$(b^*ab)^*a$

Les automates finis sont les modèles de machine les plus simples : ils n'ont aucun support de mémoire externe (comme la pile d'un automate à pile).

Leur mémoire est donc finie (espace constant), et correspond à leur nombre d'états. Par exemple, dans l'automate ci-dessus, l'état 1 permet de se souvenir qu'il faut lire un *a* pour sortir.

Exemples :

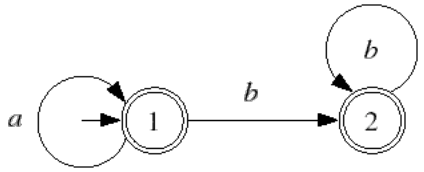
- distributeur de café : les pièces introduites sont les symboles de l'alphabet, l'état terminal est atteint quand le montant est supérieur au montant demandé,
- mécanisme contrôlant le code d'accès d'une porte : les chiffres tapés sont les symboles, l'état terminal est celui qui déclenche l'ouverture,
- rubiks cube : l'alphabet est l'ensemble des rotations des 6 faces.

1.4 Clôture par complément des langages reconnus par AFD

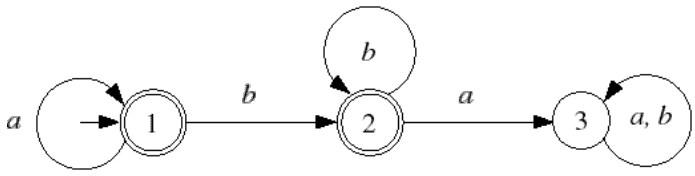
Propriété. *Si L est un langage reconnu par AFD, alors son complémentaire* $\Sigma^*\backslash L$ *l'est aussi.*

Exemple : $L = \{a^i b^j, i, j \in \mathbf{N}\}$,

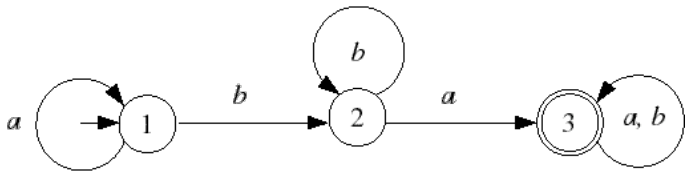
$\Sigma^*\backslash L = \{w \in \Sigma^*, ba \text{ est facteur de } w\}$



On complète l'automate :



Puis on intervertit les états finals :



Construction :

Si $A = (Q, \delta, i, F)$ est un AFD complet qui reconnaît L, alors

$A = (Q, \delta, i, Q \backslash F)$ reconnaît $\Sigma^* \backslash L$.

2. Définition des automates finis non-déterministes (AFN)

2.1 Définition générale

Définition. *Un automate fini non-déterministe AFN sur un alphabet* Σ *est la donnée d'un n-uplet* (Q, δ, I, F) *où* :

- Q *est un ensemble fini d'états*,
- δ *est une fonction de transition de* $Q \times \Sigma$ *dans* $\mathscr{P}(Q)$*, ensemble des parties de* Q,
- I *est une partie de* Q *d'états dits initiaux*,
- F *est une partie de* Q *d'états dits finals*.

Un mot u est accepté par un AFN s'il existe un chemin d'étiquette u, partant de l'un des états initiaux, et arrivant à l'un des états finals.

Un AFN est un automate dans lequel d'un état peuvent partir plusieurs flèches avec la même étiquette.

Un AFD est un cas particulier d'AFN avec pour chaque état une seule flèche pour chaque étiquette : $\mathrm{Card}(\delta(q, a)) \leq 1$ pour tous $q \in Q$, $a \in \Sigma$.

Donc tout langage reconnu par un AFD est reconnu par un AFN.

Plus surprenant, on a la réciproque.

Théorème (Rabin-Scott). *Tout langage reconnu par un AFN peut être reconnu par un AFD.*

Le principe de la construction de l'AFD correspondant est de prendre comme états de l'AFD les ensembles d'états de l'AFN. L'unique état initial de l'AFD est l'ensemble I des états initiaux de l'AFN.

Construction pour la déterminisation d'un AFN :

Si $A = (Q, \delta, I, F)$ est un AFN qui reconnaît L, alors

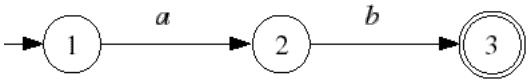
$A_{det} = (\mathscr{P}(Q), \delta_{det}, I, F_{det})$ est un AFD qui reconnaît le même langage L, avec :

- $F_{det} = \{X \in \mathscr{P}(Q), X \cap F \neq \emptyset\}$,
- $\delta_{det}(X, a) = \cup_{q \in X} \delta(q, a)$.

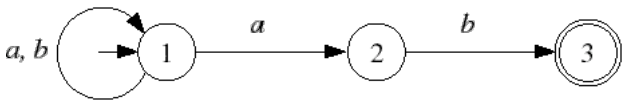
Pratiquement, on ne construit que les états accessibles à partir de I, de proche en proche :

on part de l'état initial I, puis on calcule toutes les transitions qui partent de I, puis on recommence avec les nouveaux états obtenus, etc.

Exemple : L'AFD suivant est l'écorché du mot *ab*, c'est-à-dire l'automate qui ne reconnaît que ce mot :



L'ensemble des mots qui se terminent par *ab* s'obtient très facilement avec un AFN :



$Q = \{1, 2, 3\}$,

$I = \{1\}$,

$F = \{3\}$.

état initial : $I = \{1\}$

transition par a : $\{1, 2\}$

transition par b : $\{1\}$

état $\{1, 2\}$

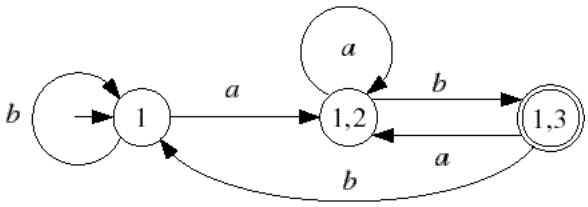
transition par a : $\{1, 2\}$

transition par b : $\{1, 3\}$

état $\{1, 3\}$

transition par a : $\{1, 2\}$

transition par b : $\{1\}$



3. AFN avec probabilités de transitions : chaînes de Markov

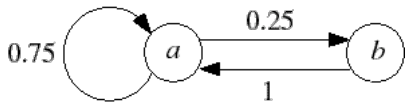
Une chaîne de Markov est un AFN dont les transitions sont munies de probabilités.

Exemple : Construction des probabilités de transitions dans un mot.

Soit le mot $w = aaabaa$

On compte le nombre de fois où chaque lettre est suivie d'une même lettre :

	a	*b*
a	3	1
b	1	0



Dans cet exemple, le contexte est de longueur 1, mais on peut étudier des contextes plus long, par exemple de longueur 2 :

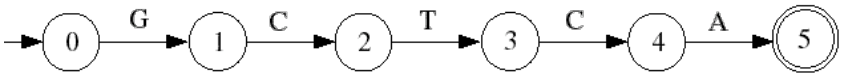
- combien de fois *aa* est suivi de *a* ? de *b* ?
- combien de fois *ab* est suivi de *a* ? de *b* ?
- etc

4. Oracle des suffixes

L'*oracle des suffixes* est un automate qui reconnaît tous les suffixes d'un mot *x*, mais avec quelques mots en plus. On peut donner directement l'AFD correspondant (sans déterminisation), par une construction très simple à mettre en œuvre.

L'*écorché* d'un mot *x* de longueur *n* est l'AFD obtenu avec $n + 1$ états, et *n* flèches correspondant aux lettres successives du mot. Dans cet automate, on voit clairement que les états ont un rôle de « mémoire » : chaque état mémorise la position dans le mot *x*.

Exemple : écorché du motif x = GCTCA



Petite remarque : pourquoi les lettres G, C, T, A ?

Le génome est fait d'ADN. Les gènes contenus dans le génome sont codés sous forme chimique le long des molécules d'ADN. Celles-ci sont constituées par l'enchaînement de "maillons" élémentaires nommés nucléotides. Les nucléotides ont une partie variable - une base, du point de vue chimique - qui peut exister sous 4 formes différentes ; ces formes sont symbolisées par les lettres A, T, G et C. Les instructions sont donc écrites dans un alphabet chimique à 4 lettres seulement.

L'une des motivations qui a conduit à définir l'oracle des suffixes est l'étude des séquences d'ADN. Il est en général utilisé de façon négative, quand on veut vérifier qu'aucun suffixe d'un motif n'apparaît dans une séquence.

Construction de l'oracle des suffixes : on part de l'écorché de x, et on rajoute des transitions à l'aide d'une fonction dite de « lien suffixiel » entre états. On construit simultanément la fonction f et les transitions de l'automate.

Supposons l'oracle construit jusqu'à l'état p, avec les liens suffixiels de tous les états jusqu'à p compris. La lettre suivante a de x donne un nouvel état $p+1 = \delta(p, a)$.

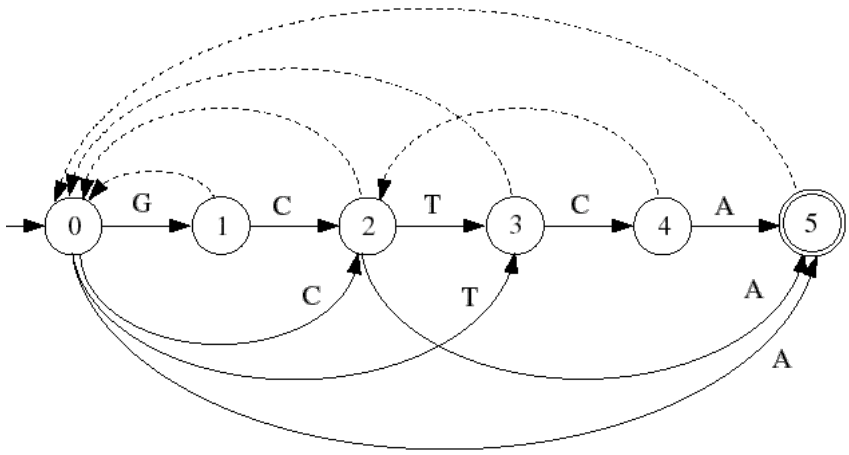
Pour rajouter les transitions, on suit les liens suffixiels déjà existants $f(p)$, $f(f(p))$, etc.

• si $\delta(f(p), a)$ non défini, on ajoute une transition $\delta(f(p), a) = p+1$, et on continue à suivre les liens,

• si $\delta(f(p), a)$ est défini pour $p \neq 0$, on stoppe (pas de nouvelle transition) et on crée le lien suffixiel de $p+1$ en posant $f(p+1) = \delta(f(p), a)$,

• si $p = 0$, on stoppe (pas de nouvelle transition) et on crée le lien suffixiel de $p+1$ en posant $f(p+1) = 0$.

Pour le motif GCTCA, l'oracle des suffixes donne l'AFD suivant (les liens suffixiels sont indiqués par des flèches en pointillé au-dessus) :



On peut vérifier que les suffixes de GCTCA sont reconnus :

GCTCA

CTCA

TCA

CA

A

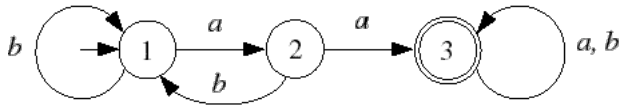
Existe-t-il d'autres mots reconnus par l'oracle qui ne sont pas suffixes de GCTCA ?

Oui, il y en a un :

GCA

5. Reconnaissabilité dans un monoïde quelconque

5.1 Automates finis et morphismes de monoïdes



Dans la table de transition d'un automate, on peut considérer que chaque lettre de l'alphabet définit une *relation* entre les états : la lettre a met en relation q_1 avec q_2 ssi $q_1.a = q_2$.
On peut alors composer ces relations : par exemple, le carré a^2 correspond à la relation définie par le mot aa, c'est-à-dire que a^2 met en relation q_1 avec q_2 ssi $q_1.aa = q_2$.

Exemple : On complète la table de transition de l'automate ci-dessus en indiquant les états obtenus pour les mots bb et aba. En comparant les relations obtenues, on en déduit des égalités vérifiées par ces relations.

	1	2	3	
a	2	3	3	
b	1	1	3	
b^2	1	1	3	$= b$
aba	2	3	3	$= a$

L'ensemble des relations sur les états {1, 2, 3} forme un monoïde fini. Si l'on complète la table de transition ci-dessus avec tous les mots de longueur 2, 3, 4 etc., on constate que cela ne donne que cinq relations distinctes. Les relations associées aux mots sur a, b forment donc un sous-monoïde à cinq éléments $\{a, b, a^2, ab, ba\}$ dont la table est :

	a	b	a^2	ab	ba
a	a^2	ab	a^2	a^2	a
b	ba	b	a^2	b	ba
a^2	a^2	a^2	a^2	a^2	a^2
ab	a	ab	a^2	ab	a
ba	a^2	b	a^2	a^2	ba

Les mots reconnus par l'automate sont exactement ceux qui se réduisent à a^2 dans le monoïde des relations. Soit l'application φ qui a un mot associe la relation correspondante sur les états {1, 2, 3}. C'est un morphisme du monoïde libre Σ* (infini) dans le monoïde (fini) des relations entre états. Le langage reconnu par l'automate est exactement l'ensemble $\varphi^{-1}(a^2)$. Cela conduit à effectuer la généralisation suivante :

Définition. *Une partie X d'un monoïde quelconque M est reconnaissable si et seulement s'il existe un morphisme φ de M dans un monoïde fini N tel que* $\varphi^{-1}(\varphi(X)) = X$.

1.2 Parties rationnelles et théorème de Kleene

Définition. *Les parties rationnelles d'un monoïde quelconque M forment le plus petit ensemble de parties de M*

- *contenant les parties finies,*
- *fermé par union, produit et étoile.*

Proposition. *Pour des monoïdes quelconques, l'image réciproque par un morphisme d'une partie reconnaissable est reconnaissable.*

Proposition. *Pour des monoïdes quelconques, l'image directe par un morphisme d'une partie rationnelle est rationnelle.*

Théorème (Kleene)**.** *Dans le monoïde libre Σ*, l'ensemble des parties reconnaissables est exactement égal à l'ensemble des parties rationnelles.*

Corollaire. *L'image directe et l'image réciproque par un morphisme de monoïdes libres d'une partie reconnaissable (ou rationnelle) est reconnaissable (ou rationnelle).*

6. Notion de grammaire formelle

Définition. *Une grammaire est définie par la donnée (T, N, P, S)*

• *de deux ensembles disjoints de symboles, les terminaux T que l'on note par des minuscules et les non terminaux N notés par des majuscules,* $\Sigma = N \cup T$,

• *d'un ensemble fini P de productions (ou règles de réécriture) de la forme* $u \rightarrow v$ *où u et v sont des séquences de symboles de N ou T,*

• *d'un symbole particulier S non terminal appelé axiome.*

Si $u \rightarrow v$ est une production de la grammaire, la séquence xuy peut se dériver en xvy. Cela signifie que le fragment u peut être remplacé par v à l'intérieur des séquences dans lesquelles il apparaît.

Le langage engendré par la grammaire est l'ensemble de toutes les séquences de symboles ***terminaux*** que l'on peut obtenir par dérivation à partir de l'axiome, en utilisant les productions. Les non terminaux apparaissent donc comme des variables intermédiaires qu'il faut éliminer pour obtenir les mots engendrés par la grammaire.

La hiérarchie de Chomsky distingue quatre classes de grammaires formelles, selon la forme de leurs productions :

• Type 0 : aucune restriction,

• Type 1 (contextuelles, ou context-sensitive) : productions de la forme $uAv \rightarrow uwv$ où A est un non terminal, u, v des séquences de terminaux, et w une séquence quelconque avec au moins un symbole,

• Type 2 (algébriques, ou context-free) : productions de la forme $A \rightarrow w$ où A est un non terminal, et w une séquence ***quelconque*** de terminaux ou non terminaux,

• Type 3 (régulières ou linéaires) : productions de la forme $A \rightarrow wB$ ou $A \rightarrow w$, où A et B sont des non terminaux, et w une séquence de ***terminaux exclusivement*** (donc la partie droite ne peut contenir au plus qu'un seul non terminal).

Les grammaires de type 1 sont dites « contextuelles », car les séquences u et v jouent le rôle de contextes dans lesquels on peut remplacer le symbole non terminal A par la séquence w. Notons que la condition sur le nombre de symboles de w interdit toute production dans laquelle le membre de droite serait plus court que le membre de gauche.

Les grammaires algébriques (type 2) sont caractérisées par le fait que le membre gauche des productions est réduit à un seul symbole (non terminal). Les grammaires régulières (type 3) sont des cas particuliers de grammaires de type 2, avec une restriction imposée au membre de droite w, qui ne peut contenir qu'un seul symbole non terminal B situé à la fin.

Si l'on désigne par $\mathscr{L}_0$, $\mathscr{L}_1$, $\mathscr{L}_2$, $\mathscr{L}_3$ les ensembles de langages engendrés respectivement par les grammaires de types 0, 1, 2, 3, on a la suite d'inclusions suivante :

$$\mathscr{L}_0 \supset \mathscr{L}_1 \supset \mathscr{L}_2 \supset \mathscr{L}_3.$$

Ainsi, tout langage régulier (appartenant à $\mathscr{L}_3$) est aussi un langage algébrique (appartenant à $\mathscr{L}_2$), car les grammaires régulières sont des cas particuliers de grammaires algébriques. Lorsqu'on parle du « type » d'un langage, on désigne le plus petit ensemble qui contient ce langage, dans la suite d'inclusions ci-dessus. Ce type correspond à la grammaire la plus simple permettant de décrire ce langage.

Les ensembles étant emboîtés les uns dans les autres, un même langage peut être décrit par plusieurs grammaires de types différents, qui ne correspondent pas toutes au type du langage lui-même.

Exemple : Les ensembles finis de séquences sont des langages de type régulier, comme le langage {*ababb*} réduit à une séquence unique. Pourtant, ce langage peut être engendré par la grammaire $S \rightarrow AB$, $A \rightarrow ab$, $B \rightarrow abb$ qui est algébrique, sa première production contenant deux non-terminaux en partie droite. Mais la grammaire régulière $S \rightarrow abB$, $B \rightarrow abb$ engendre le même langage, qui peut donc être engendré par deux grammaires de types différents.

Les langages de type algébrique sont caractérisés par des phénomènes de récursion infinie, comme le langage {*ab*, *aabb*, *aaabbb*, etc.} engendré par la grammaire $S \rightarrow aSb$, $S \rightarrow ab$. Les langages de type régulier ne comportent pas ce phénomène, et c'est la raison pour laquelle la grammaire algébrique précédente peut être remplacée par une grammaire régulière.

Références

• références générales

Gross M., Lentin A., *Notions sur les grammaires formelles*, Paris, Gauthier-Villars, 1967.

Chemillier M., Structure et méthode algébriques en informatique musicale, Thèse Université Paris 7, LITP, 1990.

Chemillier M., Synchronisation of musical words, *Theoretical Computer Science* **310** (2003) 35-60.

• oracle des facteurs (simulation stylistique)

Allauzen, Cyril & Maxime Crochemore, Mathieu Raffinot, Factor oracle : A new structure for pattern matching, *SOFSEM '99*: *Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science, Springer-Verlag, 1999, p. 291-306 (présentation).

Mancheron Alban , Moan Christophe, Combinatorial Characterization of the Language Recognized by Factor and Suffix Oracles, *International Journal of Foundations of Computer Science*, 16 (6) (2005) 1179-1191.

Assayag, Gérard, Shlomo Dubnov, Olivier Delerue, Guessing the Composer's Mind: Applying Universal Prediction to Musical Style, *Proceedings of the ICMC (Int. Computer Music Conf.)*, 1999, p. 496-499 (pdf).

Assayag, Gérard, Shlomo Dubnov, Using factor oracles for machine improvisation, *Soft Computing*, special issue on Formal Systems and Music, G. Assayag, V. Cafagna, M. Chemillier (eds.), 8 (9) (2004) 604–610 (pdf).

• grammaire de substitution harmonique

Steedman, Mark, A Generative Grammar for Jazz Chord Sequences, *Music Perception*, vol. 2 (1) (1984) 52-77.

Steedman, Mark, The Blues and the Abstract Truth: Music and Mental Models, A. Garnham, J. Oakhill, (eds.), *Mental Models In Cognitive Science*, Mahwah, NJ: Erlbaum 1996, p. 305-318 (PostScript).

Chemillier, Marc, Grammaires, automates et musique, J.-P. Briot, F. Pachet (éds.), *Informatique musicale*, Traité IC2, Hermès, Paris, 2004, chap. 6, p. 195-230 (exemples).

Chemillier M., Toward a formal study of jazz chord sequences generated by Steedman's grammar, *Soft Computing*, special issue on Formal Systems and Music, G. Assayag, V. Cafagna, M. Chemillier (eds.) **8** (9) (2004) 617-622.