MMIM Modèles mathématiques en informatique musicale

Marc Chemillier

Master Atiam (Ircam), 2009-2010

Notions théoriques sur les langages formels (Partie I)

- Définitions générales
  - Mots, langages
  - Monoïdes
- Notion d'automate fini
  - Automate fini déterministe (AFD)
  - Automate fini non déterministe (AFN)
- Algorithme de Morris & Pratt (pattern matching)
- Oracle des suffixes (OMax)

1. Définitions générales

1.1 Mots

Un *mot* fini u est une suite de symboles. La longueur de u notée $|u|$ est le nombre de symboles du mot. Le *mot vide* noté ε est le seul mot de longueur nulle. L'ensemble des symboles noté Σ est appelé *alphabet*, et l'ensemble des mots sur l'alphabet Σ est noté Σ^*.

Pour deux mots $u = u_1 \ldots u_n$ et $v = v_1 \ldots v_m$, on définit la *concaténation* uv comme le mot obtenu en mettant les lettres de v à la suite de celles de u :

$uv = u_1 \ldots u_n v_1 \ldots v_m$.

Un mot $u \in \Sigma^*$ est *facteur* du mot $w \in \Sigma^*$ s'il existe $v, v' \in \Sigma^*$ tels que $w = vuv'$. Si $v = \varepsilon$, on dit que u est *préfixe*. Si $v' = \varepsilon$, on dit que u est *suffixe*.

Exemple : *abb* est facteur de *babba*, et *ba* est à la fois préfixe et suffixe, mais *aa* n'est pas facteur.

Un mot fini u est *périodique* si $u = x^n$ pour $n \geq 2$. Tout mot non périodique est dit *primitif*.

L'idée fondamentale de ce cours est que la notion de mot permet de représenter le principe de succession d'événements dans une séquence musicale, d'où son intérêt pour la modélisation en informatique musicale.

1.2 Langages

Les sous-ensembles de Σ^* sont appelés des *langages* (c'est-à-dire des ensembles de mots, ou de séquences musicales dans le contexte de l'informatique musicale).

Opérations sur les langages :

• opérations classiques sur les ensembles :

union $\cup$, intersection $\cap$, différence \, complémentaire,

• opérations héritées de la concaténation :

La concaténation de deux langages est définie par :

$L_1L_2 = \{uv, u \in L_1 \text{ et } v \in L_2\}$.

Exemple : $L_1 = \{a, ab\}$, $L_2 = \{c, bc\}$, $L_1L_2 = \{ac, abc, abbc\}$.

La *puissance* d'un langage L est définie inductivement :

$L^0 = \{\varepsilon\}$, $L^{n+1} = L^nL$.

L'*étoile* d'un langage L, notée L^*, est :

$L^* = \{\varepsilon\} \cup L \cup L^2 \ldots \cup L^n \cup \ldots$

Ce langage contient un nombre infini de mots, qui sont les répétitions indéfinies de mots de L.

Exemple : $L = \{ab, b\}$, L^* est l'ensemble de tous les mots tels que *aa* n'est pas facteur, et *a* n'est pas suffixe.

On note également L^+ le langage :

$L^+ = L \cup L^2 \ldots \cup L^n \cup \ldots$

1.3 Monoïdes

Un *monoïde* est un ensemble muni d'une opération

• associative,

• possédant un élément neutre 1.

Un *sous-monoïde* est un sous-ensemble fermé pour l'opération et contenant l'élément neutre. L'ensemble Σ^* des mots sur l'alphabet Σ est un monoïde pour la concaténation, dont l'élément neutre est le mot vide ε. Ce monoïde est engendré par l'ensemble de ses lettres, c'est-à-dire l'alphabet Σ.

Soient deux mots $u = u_1 \ldots u_n$ et $v = v_1 \ldots v_m$. L'égalité $u = v$ équivaut à l'égalité de toutes les lettres de u et v une à une, soit $u_i = v_i$ pour tout $i \leq n = m$. Pour cette raison, on dit que Σ^* est le *monoïde libre* sur Σ.

Remarque : Cette notion de « liberté » est similaire à celle des espaces vectoriels lorsqu'on dit qu'une famille de vecteurs est libre si l'égalité de deux combinaisons linéaires de ces vecteurs implique l'égalité de chacun de leurs coefficients.

Un sous-monoïde de Σ^* est-il toujours « libre » par rapport à l'ensemble qui l'engendre ?

-> Non. Par exemple, le sous-monoïde engendré par $C = \{a, ab, c, bc\}$ n'est pas libre :

$(a)(bc) = (ab)(c)$

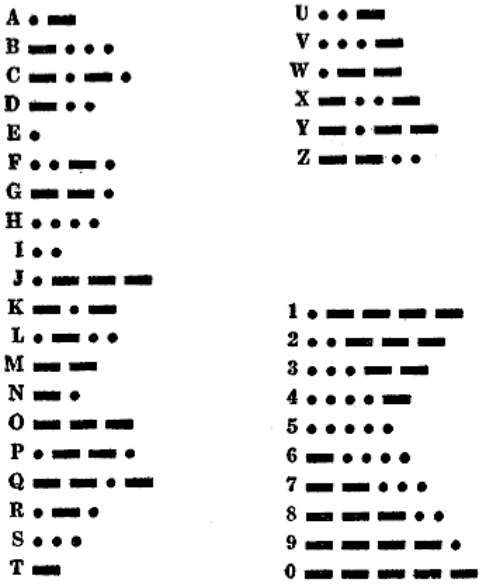
Un *code* est une partie C de Σ^* qui engendre un sous-monoïde libre de Σ^*. Tout élément du sous-monoïde s'écrit d'une manière unique comme suite d'éléments de C. Cela signifie qu'on peut « décoder » les éléments de C^*.

• L'ensemble Σ^n des mots de longueur n est un code, en particulier l'alphabet Σ est un code.

• Attention : Le code morse n'est pas un code dans le sens ci-dessus.

INTERNATIONAL MORSE CODE

1. A dash is equal to three dots.
2. The space between parts of the same letter is equal to one dot.
3. The space between two letters is equal to three dots.
4. The space between two words is equal to five dots.



On a E = « • », T = « — », et N = « —• », donc on a N = TE. Le décodage en morse se fait grâce à la présence de séparateurs entre les lettres.

Un *morphisme* de monoïde est une application φ telle que $\varphi(xy) = \varphi(x)\varphi(y)$ et $\varphi(1) = 1$.

Toute application d'un alphabet Σ dans un monoïde quelconque se prolonge dans le monoïde libre Σ^* en un unique morphisme de monoïdes. Il en résulte que pour définir un morphisme sur Σ^*, il suffit de définir l'image de toutes ses lettres.

2. Définition des automates finis déterministes (AFD)

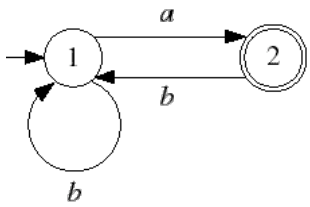
2.1 Définition générale

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.

$\delta : Q \times \Sigma \rightarrow Q$

$(1, a) \rightarrow 2$

$(1, b) \rightarrow 1$

$(2, b) \rightarrow 1$

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

2.2 Langage reconnu par un AFD

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

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.*

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,

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).

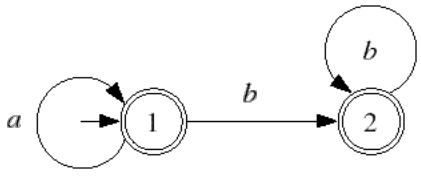
Leur mémoire est donc finie (espace constant), et correspond à leur nombre d'états. Par exemple, dans l'automate ci-dessus, l'état 1 permet de se souvenir qu'il faut lire un *a* pour sortir.

2.3 Clôture par complément des langages reconnus par AFD

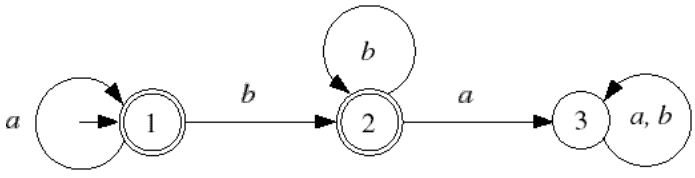
Propriété. *Si L est un langage reconnu par AFD, alors son complémentaire* $\Sigma^*\backslash L$ *l'est aussi.*

Exemple : L est l'ensemble des mots comportant une suite de a suivie éventuellement d'une suite de b, soit $L = \{a^i b^j, i, j \in \mathbf{N}\}$,

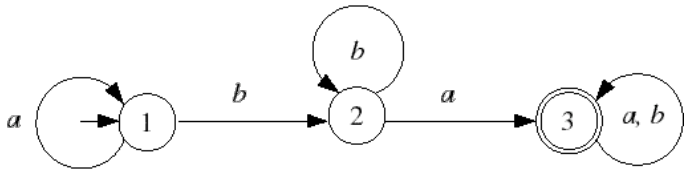
$\Sigma^*\backslash L = \{w \in \Sigma^*, ba \text{ est facteur de } w\}$



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 $\Sigma^* \backslash L$.

3. Définition des automates finis non déterministes (AFN)

3.1 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.

Ainsi, les AFD apparaissent comme des cas particuliers d'AFN avec pour chaque état une seule flèche pour chaque étiquette : $\mathrm{Card}(\delta(q, a)) \leq 1$ pour tous $q \in Q$, $a \in \Sigma$.

Il en résulte que tout langage reconnu par un AFD est reconnu par un AFN.

Plus surprenant, on a la réciproque, c'est-à-dire que les AFD et les AFN reconnaissent exactement les mêmes langages.

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

La déterminisation d'un AFN est la construction de l'AFD correspondant. Elle se fait en prenant comme états de l'AFD les ensembles d'états de l'AFN, c'est-à-dire $\mathscr{P}(Q)$. Exemple : construction de l'automate permettant de chercher un motif dans un texte (algorithme de Morris & Pratt).

4. Algorithme de Morris & Pratt (recherche d'un motif dans un texte)

4.1 Méthode de lecture gauche-droite avec l'AFD de $\Sigma^* x$

Problème : chercher le motif x dans le texte t. Le principe est de construire un automate complet reconnaissant $\Sigma^* x$, c'est-à-dire tous les mots qui se terminent par x. Ensuite on lit le texte t dans cet automate. Que se passe-t-il si on arrive à un état terminal ?

-> On a trouvé le motif x dans le texte t.

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.

Pour reconnaître $\Sigma^* x$, il suffit d'ajouter une boucle sur l'état initial de l'écorché de x avec toutes les lettres de l'alphabet. On obtient un AFN.

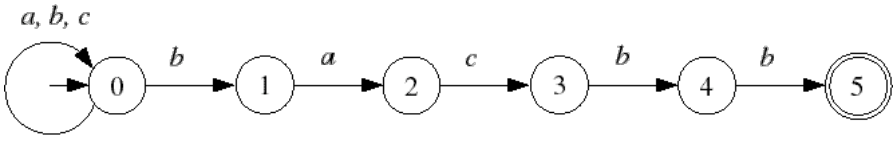
En théorie, pour un AFN ayant n états, combien d'états a l'AFD obtenu par déterminisation ?

-> 2^n états.

-> Mais dans le cas de $\Sigma^* x$, on a une propriété remarquable :

Proposition. *L'AFD obtenu en déterminisant l'écorché d'un mot x de longueur n avec une boucle sur l'état initial pour toutes les lettres, comporte au plus n états.*

Exemple : $x = bacbb$



Le principe de la déterminisation consiste à faire des « paquets d'états » de l'AFN. Le premier « paquet » est $I = \{0\}$ que l'on prend comme état initial. Puis on cherche toutes les transitions partant des états de I pour chacune des lettres et on recommence avec les nouveaux « paquets » ainsi obtenus, etc.

état initial : $I = \{0\}$, transitions par $a = \{0\}$, transitions par b = ?

-> nouveau paquet {0, 1}

On obtient ainsi les états de l'AFD que l'on renumérote :

0 = {0}

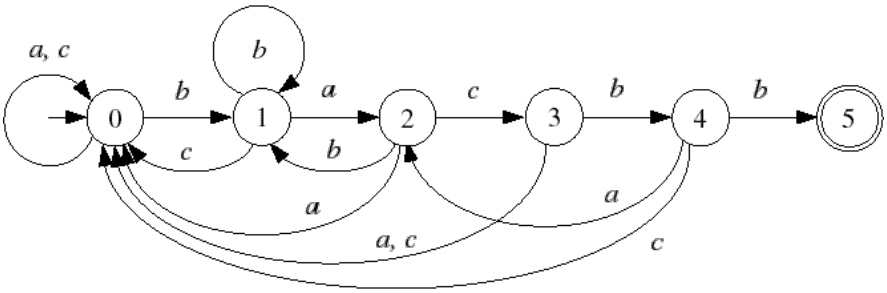
1 = {0, 1}

2 = {0, 2}

3 = {0, 3}

4 = {0, 1, 4}

5 = {0, 1, 5}



Exemple : Rechercher le motif $x = bacbb$ dans le texte $t = abacbacbba$

a b a c *b a c b b* *a*

0 1 2 3 4 2 3 4 **5** motif trouvé !!!

4.2 Fonction d'échec (ou de saut) : algorithme de Morris & Pratt

L'algorithme de Morris & Pratt revient à calculer directement l'AFD pour $\Sigma^* x$, sans passer par la déterminisation de l'AFN, grâce à une fonction d'échec.

Si p est un état de l'écorché de x (autre que 0) permettant de lire w, on définit $f(p)$ de la manière suivante :

$f(p)$ = état d'arrivée du plus long suffixe propre de w qui est aussi préfixe de w (donc de x)

Exemple : Pour l'écorché de $x = bacbb$, la fonction d'échec est :

b	ε = plus long suffixe propre qui est préfixe, donc $f(1) = 0$
b a	ε = plus long suffixe propre qui est préfixe, donc $f(2) = 0$
b a c	ε = plus long suffixe propre qui est préfixe, donc $f(3) = 0$
b *a c* ***b***	*b* = plus long suffixe propre qui est préfixe, donc $f(4) = 1$
b *a c b* ***b***	*b* = plus long suffixe propre qui est préfixe, donc $f(5) = 1$

État p	0	1	2	3	4	5
$f(p)$		0	0	0	1	1

4.3 Construction directe de l'AFD de $\Sigma^* x$ avec la fonction d'échec

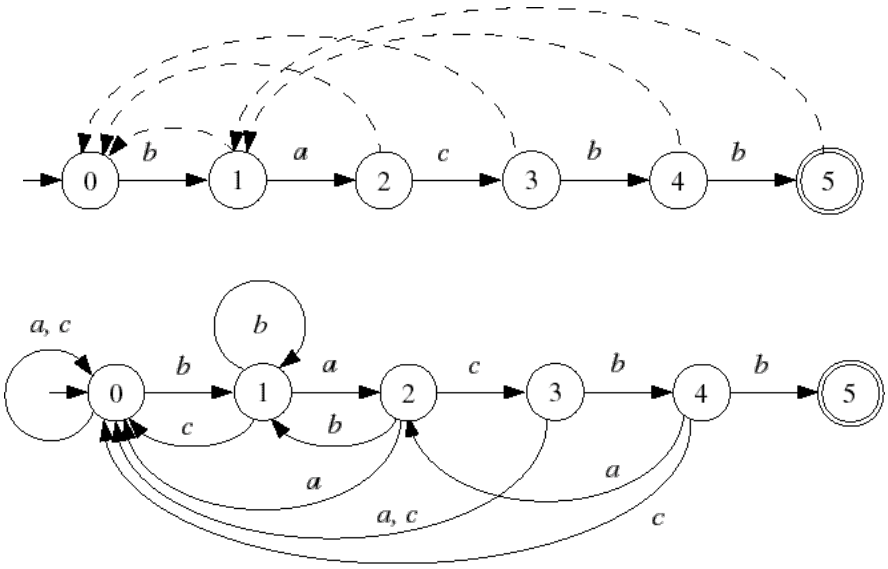
Avec la fonction d'échec, on peut construire directement l'AFD reconnaissant $\Sigma^* x$.

On complète la fonction de transition de la manière suivante :

pour toute lettre a telle que $\delta(p, a)$ non défini, on pose

- $\delta(p, a) = \delta(f(p), a)$ si $p \neq 0$,
- $\delta(0, a) = 0$.

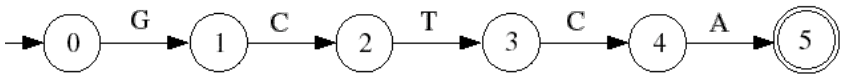
On vérifie facilement qu'en rajoutant ces transitions, on obtient exactement l'AFD déterminisé vu plus haut.



5. Oracle des suffixes (Crochemore 1999, utilisé dans OMax)

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 technique analogue à l'algorithme de Morris & Pratt.

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.

Construction de l'oracle des suffixes : on part de l'écorché de x, et on ajoute des transitions à l'aide d'une fonction f dite de « lien suffixiel » entre états, en construisant simultanément f et les transitions.

Au départ, on place un lien suffixiel de 1 vers 0. Puis on suppose l'oracle construit avec ses liens suffixiels jusqu'à l'état p compris. La lettre suivante a donne un nouvel état $\delta(p, a) = p+1$.

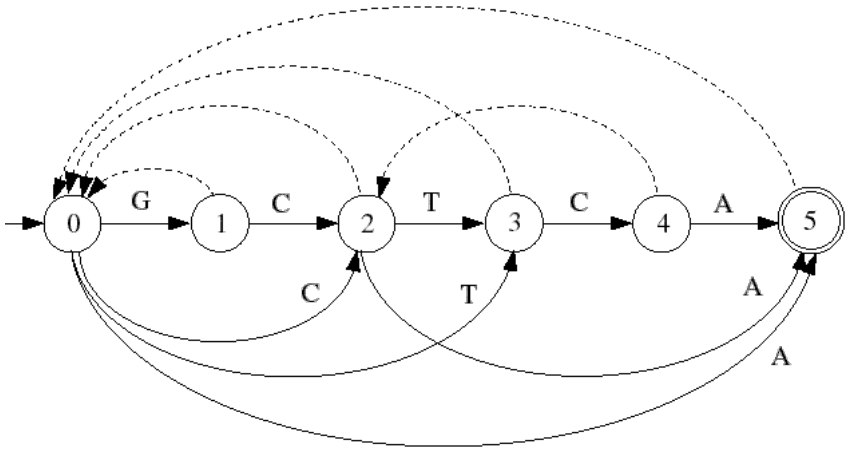
Pour ajouter les transitions, on suit les liens suffixiels déjà existants $p' = f(p)$, puis $p' = f(f(p))$, etc.

• si $\delta(p', a)$ est non défini, on ajoute une transition $\delta(p', a) = p+1$

- si $p' \neq 0$, on continue à suivre les liens,
- si $p' = 0$, on stoppe et on crée un lien suffixiel en posant $f(p+1) = 0$

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

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

Caractérisation des mots reconnus par l'oracle : cf. Mancheron & Moan 2005.

Remarque : Dans les applications bioinformatiques, l'oracle est utilisé de façon négative. Si un mot n'est pas reconnu par l'oracle, on est sûr qu'il n'est pas suffixe.

Proposition. *Si p est l'état d'arrivée d'un préfixe w de x, le lien suffixiel f(p) correspond à l'état d'arrivée du plus long suffixe de w qui est répété à gauche, c'est-à-dire qui est facteur non suffixe de w.*

Attention : Il ne faut pas confondre

- la fonction de saut de Morris & Pratt,
- la fonction correspondant aux liens suffixiels de l'oracle.

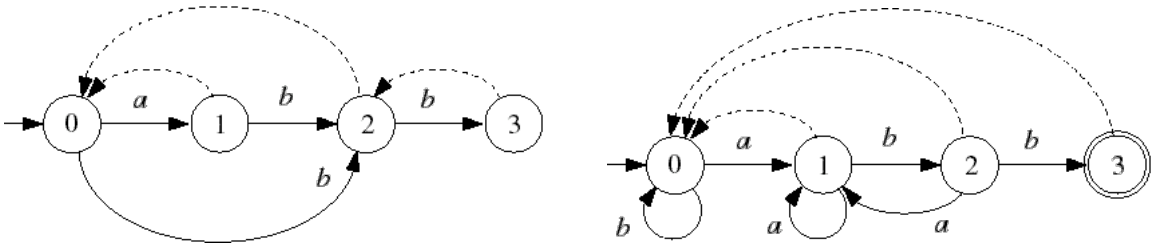
Si *p* est l'état permettant de lire le préfixe *w*, la fonction de Morris & Pratt est définie par:

f(*p*) = état d'arrivée du plus long suffixe propre de *w* qui est aussi **préfixe** de *w* (donc de *x*)

alors que la fonction correspondant aux liens suffixiels est définie par:

f(*p*) = état d'arrivée du plus long suffixe propre de *w* qui est aussi **facteur** de *w* (donc de *x*)

Exemple : Construction des deux automates (oracle des suffixes, AFD de Morris & Pratt) pour le mot *abb*



Références

• références générales

Lothaire M., *Combinatorics on Words*, Encyclopedia of Mathematics, Vol. 17, Addison-Wesley, 1983 (réédité Cambridge University Press, 1997, chap. 1 disponible en ps).

Berstel Jean, *Automates et grammaires*, Cours de Licence d'informatique, Université de Marne-la-vallée, 2004-05 (pdf).

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

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

• oracle des suffixes (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).