MMIM Modèles mathématiques en informatique musicale

Marc Chemillier

Master M2 Atiam (Ircam), 2011-2012

Notions théoriques sur les langages formels

- Définitions générales
  - Mots, langages
  - Monoïdes
- Notion d'automate fini
  - Automate fini déterministe (AFD)
  - Automate fini non déterministe (AFN)

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, c'est-à-dire $u_i = v_i$ 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. Exemple : le sous-monoïde engendré par $C = \{a, ab, c, bc\}$ n'est pas libre sur C :

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

On dit qu'un sous-monoïde est libre s'il existe un ensemble générateur par rapport auquel il est libre. Exemple : le sous-monoïde engendré par $C = \{ab, cd, abcd\}$ n'est pas libre par rapport à C car $(ab)(cd) = abcd$, mais il l'est pas rapport à $C' = \{ab, cd\}$.

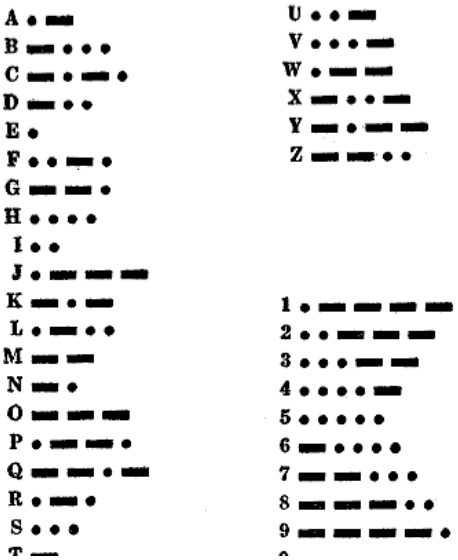
Un *code* est une partie C de Σ^* qui engendre un sous-monoïde de Σ^* libre par rapport à C. Tout élément du sous-monoïde s'écrit d'une manière unique comme suite d'éléments de C. Cela signifie qu'on peut « décoder » les éléments de C^*.

• L'ensemble Σ^n des mots de longueur n est un code, en particulier l'alphabet Σ est un code (longueur 1).

• 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.

• Représentation des rythmes par des attaques (1) et des silences (0) :

L'ensemble $C = 1\{0\}^*$ des mots commençant par 1 suivi d'une suite de 0 est un code.

Quel est le sous-monoïde libre engendré par C dans Σ^* avec l'alphabet $\Sigma = \{0, 1\}$?

= ensemble des mots soit vide, soit commençant par 1

Exemple : décodage de 1000010011101 ?

= (10000)(100)(1)(1)(10)(1)

Remarque : cette représentation ne prend pas en compte les syncopes 000001.

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 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 (de la même manière dans un espace vectoriel, on définit un morphisme en donnant l'image de tous les vecteurs de la base).

• Représentation des rythmes par des durées :

L'application qui a tout élément du code $C = 1\{0\}^*$ associe sa longueur se prolonge en un isomorphisme de C^* dans le monoïde libre sur l'alphabet (infini) **N** privé de 0 :

$\varphi(1000010011101) = 531121$

2. Notion d'automate fini

2.1 Définition des automates finis déterministes (AFD)

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*,

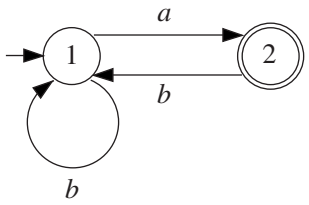
• δ *est une fonction de transition de* $Q \times \Sigma$ *dans* Q,

• 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 est-il complet ?

-> non, 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).

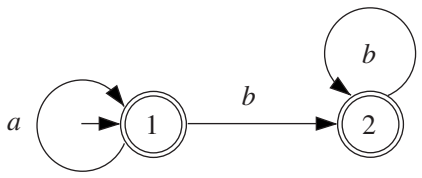
Leur mémoire est donc finie (espace constant), et correspond à leur nombre d'états. Par exemple, dans l'automate ci-dessus, l'état 1 permet de se souvenir qu'il faut lire un *a* pour sortir.

2.3 Clôture par complément des langages reconnus par AFD

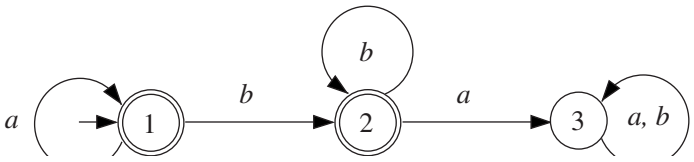
Propriété. *Si L est un langage reconnu par AFD, alors son complémentaire* $\Sigma^* \backslash L$ *l'est aussi.*

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

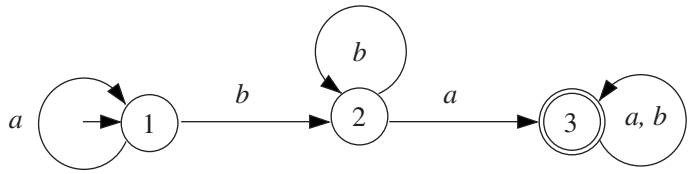
$\Sigma^* \backslash L = \{w \in \Sigma^*, ba \text{ est facteur de } w\}$



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. Généralisation aux automates finis non déterministes (AFN)

3.1 Définition des AFN, équivalence avec les AFD et langages réguliers

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 $\mathscr{P}(Q)$.

Définition. *On appelle langage régulier tout langage de* Σ^* *reconnu (ou accepté) par un automate fini, qu'il soit AFD ou AFN.*

3.2 Clôture par union des langages reconnus par AFN

S'il y a équivalence des langages reconnus par AFD et AFN, quel est l'intérêt des AFN ?

-> Ils peuvent faciliter certaines constructions

Construction d'un AFN pour l'union de deux langages réguliers :

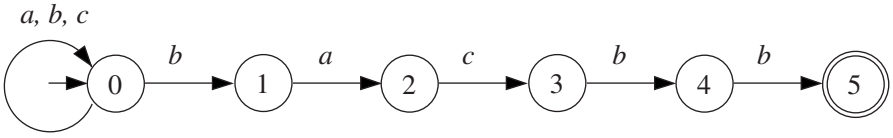
Si $A_1 = (Q_1, \delta_1, I_1, F_1)$ et $A_2 = (Q_2, \delta_2, I_2, F_2)$ sont des AFN disjoints reconnaissant L_1 et L_2, alors $A = (Q_1 \cup Q_2, \delta_1 \cup \delta_2, I_1 \cup I_2, F_1 \cup F_2)$ est un AFN reconnaissant $L_1 \cup L_2$.

Pourquoi l'automate union A n'est pas un AFD ?

-> Il n'a pas un seul état initial

3.3 Construction d'un AFN pour l'ensemble $\Sigma^* x$ des mots se terminant par x

L'écorché d'un mot x de longueur n est l'AFD de $n + 1$ états avec n flèches correspondant aux lettres successives du mot. Pour reconnaître $\Sigma^* x$, il suffit d'ajouter une boucle sur l'état initial avec toutes les lettres de l'alphabet. Exemple : $x = bacbb$



-> On obtient un AFN :

il y a deux flèches avec b partant de l'état initial.

Application à la recherche d'un motif x dans un texte t :

• on déterminise l'AFN pour obtenir un AFD reconnaissant $\Sigma^* x$

• on lit le texte t dans cet automate. Que se passe-t-il si on arrive à l'état terminal ?

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

Algorithme de Morris & Pratt : calcul direct de l'AFD pour $\Sigma^* x$, **sans passer par la déterminisation de l'AFN**. On utilise une fonction d'échec définie sur les états p de l'écorché de x. Soit w le mot lu de l'état 0 à l'état p :

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

Exemple : Fonction d'échec pour l'écorché de $x = bacbb$:

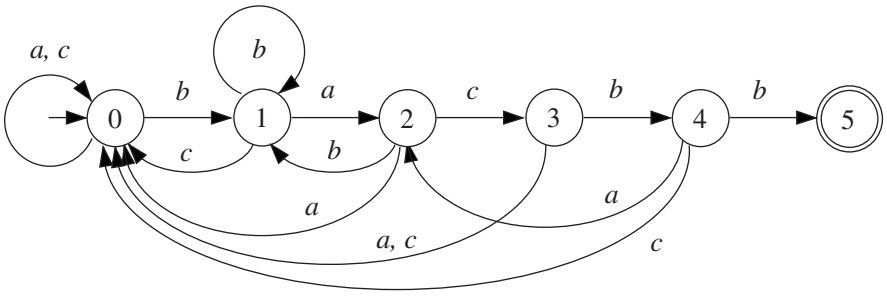
État p	0	1	2	3	4	5
$f(p)$		0	0	0	1	1

On ajoute de nouvelles transitions dans l'écorché de x 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 $p \neq 0$,

• $\delta(0, a) = 0$.



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 !!!

Références

• ouvrages et articles généraux

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

Berstel Jean, *Transductions and context-free languages*, Teubner, 1979 (biblio).

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

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