

CHAPITRE 3 : AUTOMATES FINIS NON-DETERMINISTES (AFN)

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

1.1 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 $\mathcal{P}(Q)$, 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.

1.2 Fonctionnement

Un automate fini non-déterministe est un automate tel que dans un état donné, il peut y avoir plusieurs transitions avec la même lettre.

Au temps initial, la machine est dans un état initial $i \in I$.

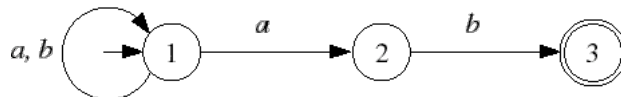
Si à l'instant t la machine est dans l'état q et lit a , alors à l'instant $t + 1$

- si $\delta(q, a) = \emptyset$, la machine se bloque,
- si $\delta(q, a) \neq \emptyset$, la machine se met dans l'un des états $\in \delta(q, a)$ (et lit le symbole suivant).

On voit que le fonctionnement n'est pas complètement « déterminé », car on ne sait pas à l'avance quel état la machine doit choisir dans $\delta(q, a)$, d'où le terme non-déterministe.

Malgré cela, les AFN peuvent être très utiles pour exprimer certains problèmes.

Exemple : ensemble des mots qui se terminent par ab



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

$I = \{1\}$,

$F = \{3\}$.

δ définie par la table de transition :

	1	2	3
a	$\{1,2\}$	\emptyset	\emptyset
b	1	3	\emptyset

1.3 Langage reconnu par un AFN

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.

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

2. Détermination d'un AFN

2.1 Algorithme de détermination d'un AFN

Un AFD est un cas particulier d'AFN, avec $\text{Card}(\delta(q, a)) \leq 1$ 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 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 détermination d'un AFN :

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

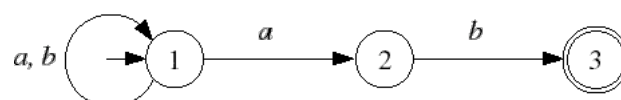
$A_{\text{det}} = (\mathcal{P}(Q), \delta_{\text{det}}, I, F_{\text{det}})$ est un AFD qui reconnaît le même langage L , avec :

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

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 :



état initial : $I = \{1\}$

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

transition par b : $\{1\}$

état $\{1, 2\}$

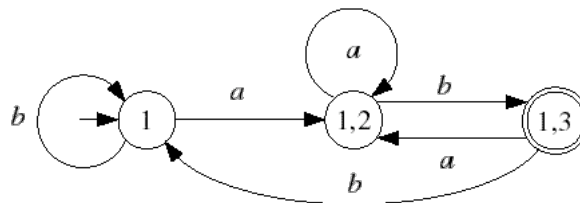
transition par $a : \{1, 2\}$

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

état $\{1, 3\}$

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

transition par $b : \{1\}$



2.2 Équivalence entre AFD et AFN

Conclusion. *Les langages reconnus par les AFN sont exactement les langages reconnus par les AFD.*

Dans l'exemple ci-dessus, le nombre d'états de l'automate déterminisé est égal à celui de l'AFN de départ. On verra que c'est le cas dans certaines applications, notamment pour la recherche de motifs.

Mais il existe des cas où l'AFD équivalent est beaucoup plus gros. En fait, si $\text{Card}(Q) = n$, la limite théorique est $\text{Card}(\mathcal{R}Q) = 2^n$. Or il existe des automates qui correspondent à ce cas limite.

Exemple : $L = \{a, b\}^* a \{a, b\}^n$

L'AFN a $n + 1$ états, alors que l'AFD équivalent a 2^n états.

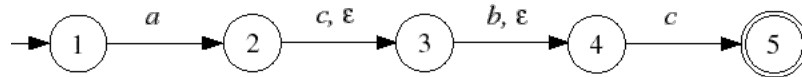
3. Automates AFN avec ε -transitions

3.1 Définition

Définition. *On généralise encore la notion d'automate en introduisant des transitions avec le mot vide ε . La fonction de transition est alors définie de $Q \times (\Sigma \cup \{\varepsilon\})$ dans $\mathcal{R}Q$, ensemble des parties de Q . Ces transitions s'effectuent sans lecture de lettre.*

Comme pour les AFN, un mot u est accepté par un AFN avec ε -transitions s'il existe un chemin d'étiquette u , partant de l'un des états initiaux, et arrivant à l'un des états finals.

Exemple :



Langage reconnu : $L = \{acbc, acc, abc, ac\}$

3.2 Transformation en AFN sans ϵ -transition

Proposition. *Tout langage reconnu par un AFN avec ϵ -transitions peut être reconnu par un AFN (sans ϵ -transitions) ayant le même nombre d'états.*

Construction de l'AFN équivalent à un automate avec ϵ -transitions, en prolongeant δ en δ_1 :

Première étape : Ajoût de ϵ -transitions par fermeture transitive.

- Pour tous les états q accessibles à partir de p par ϵ -transition (en plusieurs étapes), on rajoute une ϵ -transition directe de p à q .

On prolonge ainsi la fonction δ :

$\delta_1(p, \epsilon) =$ tous les états accessibles à partir de p par ϵ -transition.

Deuxième étape : Ajoût de transitions supplémentaires et d'états initiaux supplémentaires, puis suppression des ϵ -transitions.

- Les états initiaux sont tous les états accessibles à partir des états initiaux par ϵ -transition.
- Pour toute transition de p à q par une lettre, on rajoute des transitions de p à r avec la même lettre pour toutes les ϵ -transitions de q à r .

$$\delta_1(p, a) = \delta(p, a) \cup \left(\bigcup_{q \in \delta(p, \epsilon)} \delta_1(q, \epsilon) \right).$$

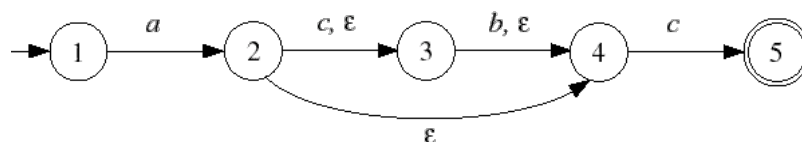
- On supprime toutes les ϵ -transitions.

On vérifie :

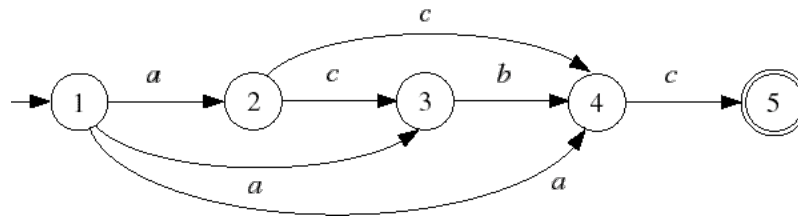
- qu'un chemin correspondant à un calcul réussi pour un mot dans l'automate avec ϵ -transitions peut facilement être transformé en chemin dans le nouvel AFN, grâce aux nouvelles transitions introduites,
- et qu'un chemin correspondant à un calcul réussi dans l'AFN peut être transformé en chemin dans l'automate avec ϵ -transitions, en remplaçant chaque nouvelle transition utilisée par les ϵ -transitions correspondantes.

Exemple :

- Fermeture transitive des ϵ -transitions



- Ajout de nouvelles transitions et suppression des ϵ -transitions



L'AFN obtenu reconnaît bien le même langage : $L = \{acbc, acc, abc, ac\}$

4. Clôture des langages reconnus par automates (AFN / AFD)

4.1 Clôture par complémentation

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

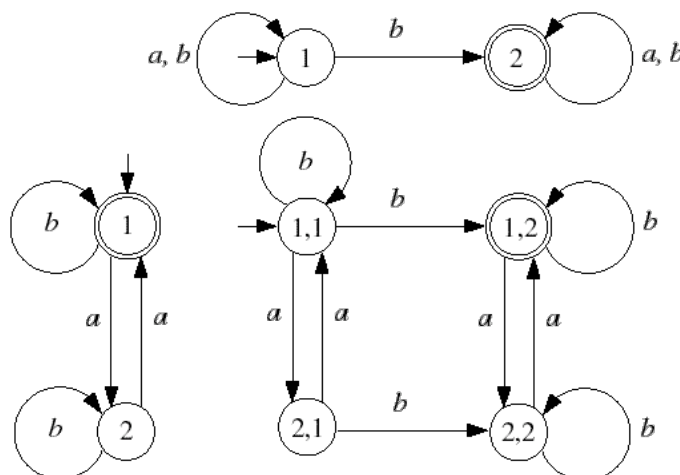
Par déterminisation, on peut construire un AFD reconnaissant le même langage L , puis appliquer la propriété de clôture vue au chapitre 2.

4.2 Clôture par intersection

Propriété. Si L_1, L_2 sont des langages reconnus par AFN, alors $L_1 \cap L_2$ l'est aussi.

Exemple : $L_1 = \{w \in \Sigma^*, w \text{ a au moins un } b\}$,

$L_2 = \{w \in \Sigma^*, w \text{ a un nombre pair de } a\}$



Construction par produit d'automates :

Si $A_i = (Q_i, \delta_i, I_i, F_i)$ reconnaît L_i pour $i = 1, 2$, alors
 $A = (Q_1 \times Q_2, \delta, I_1 \times I_2, F_1 \times F_2)$ reconnaît $L_1 \cap L_2$.

Justification :

- Si un mot correspond à un calcul réussi dans l'automate produit, on obtient une succession de couples d'états. En projetant sur les premières composantes (resp. deuxièmes), on obtient un calcul réussi dans le 1^{er} automate (resp. 2^{ème}). Donc le mot appartient à l'intersection.
- Réciproquement, si un mot appartient aux deux langages, il correspond à deux calculs réussis dans les automates, c'est-à-dire à deux chemins de même longueur. En fabriquant des couples à partir des deux séries d'états, on obtient un calcul réussi dans l'automate produit.

Remarque : Si les automates sont des AFD, alors l'automate produit obtenu pour l'intersection est aussi un AFD.

4.3 Clôture par union

Propriété. Si L_1, L_2 sont des langages reconnus par AFN, alors $L_1 \cup L_2$ l'est aussi.

Il suffit de faire l'union des deux AFN.

4.4 Clôture par concaténation et étoile

Propriété. Si L_1, L_2 sont des langages reconnus par AFN, alors $L_1 L_2$ l'est aussi.

Pour cette propriété, il est très commode d'utiliser des ϵ -transitions :

on rajoute des ϵ -transitions de tous les états initiaux de l'AFN pour L_1 , vers tous les états finals de l'AFN pour L_2 .

Propriété. Si L est un langage reconnu par AFN, alors L^* l'est aussi.

De même, on rajoute des ϵ -transitions de tous les états finals de l'AFN pour L , vers tous ses états initiaux.

4.5 Théorème de clôture pour les langages reconnus par AFN ou AFD

Théorème. Si L, L' sont des langages reconnus par AFN, alors il en est de même :

- du complémentaire $\Sigma^* \setminus L$,
- de l'intersection $L \cap L'$,
- de l'union $L \cup L'$,

- de la concaténation LL' ,
- de l'étoile L^*

Les langages reconnus par AFD étant les mêmes que les langages reconnus par AFN, ces propriétés de clôture sont vraies aussi pour les langages reconnus par AFD.

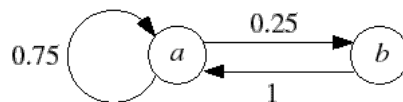
5. Chaînes de Markov

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

Exemple : 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

? (créer-table "Bonjour, ceci est le cours sur les automates, c'est-à-dire une structure de calcul très simple, mais qui a beaucoup de propriétés intéressantes")

```

((#\é 3 (#\r 1) (#\s 1) (#\t 1)) (#\q 1 (#\u 1)) (#\p 4 (#\r
2) (#\Space 1) (#\l 1)) (#\è 1 (#\s 1)) (#\d 3 (#\e 2) (#\i
1)) (#\à 1 (#\ - 1)) (#\ - 2 (#\d 1) (#\à 1)) (#\' 1 (#\e 1))
(#\m 3 (#\p 1) (#\a 2)) (#\a 7 (#\n 1) (#\Space 1) (#\i 1)
(#\l 1) (#\t 1) (#\u 2)) (#\l 5 (#\Space 1) (#\c 1) (#\e 3))
(#\t 10 (#\é 2) (#\u 1) (#\r 2) (#\ - 1) (#\e 2) (#\o 1)
(#\Space 1)) (#\s 13 (#\a 1) (#\s 1) (#\i 1) (#\ , 1) (#\u 1)
(#\Space 5) (#\t 3)) (#\i 7 (#\n 1) (#\é 1) (#\s 1) (#\m 1)
(#\r 1) (#\Space 2)) (#\e 15 (#\a 1) (#\ , 1) (#\Space 6) (#\s
6) (#\c 1)) (#\c 8 (#\u 1) (#\a 1) (#\t 1) (#\' 1) (#\o 2)
(#\i 1) (#\e 1)) (#\Space 21 (#\i 1) (#\p 1) (#\b 1) (#\q 1)
(#\m 1) (#\t 1) (#\d 2) (#\u 1) (#\a 2) (#\s 3) (#\l 2) (#\e
1) (#\c 4)) (#\ , 3 (#\Space 3)) (#\r 10 (#\i 1) (#\o 1) (#\è
1) (#\u 1) (#\e 3) (#\Space 1) (#\s 1) (#\ , 1)) (#\u 11 (#\p
1) (#\i 1) (#\l 1) (#\c 2) (#\n 1) (#\t 1) (#\r 4)) (#\j 1
(#\o 1)) (#\n 4 (#\t 2) (#\e 1) (#\j 1)) (#\o 6 (#\p 1) (#\m
1) (#\u 3) (#\n 1)) (#\B 2 (#\e 1) (#\o 1)))
  
```

? (markov 5000)

"'es s complcai ma truromai cturestéste des quronjop caless s
lc'e dint s primp e, les cop comaure qurouciés cop satés
uctér, e qunjounjomantérésaiétés quront e ciés ciére s s
près mp esss auci curoulesainjompres cureaintéres, c'e
dintériétététuc'ecuimanjompr a couprèsalcturesi di ecoi près
le aiésuc'estr térintrèsa b"

? (markov 5000)

" i pre al ce c'esulestérsururs ines aur sa t-à-à-dess cout-
à-de les e, diromales p cintes, cis le ul destétre s
trestéroure b"