

Chapitre I

Langages réguliers

Le cours de compilation est un prérequis pour ce chapitre.

Il suppose que vous connaissez déjà ce qu'est un automate fini (déterministe et non déterministe) et une expression régulière.

Si vous n'êtes pas familier avec ces notions, vous devez rattraper vos lacunes.

Dans cette leçon, nous effectuerons :

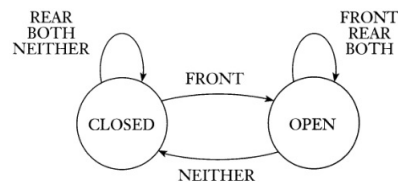
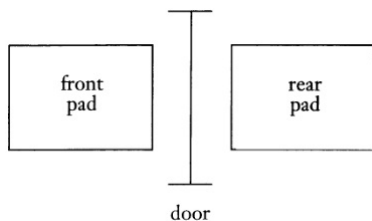
- de brefs rappels de ces modèles,
- une formalisation de ces notions,
- une étude des limites de ces modèles

Un automate fini est le modèle :

- le plus simple d'ordinateur.
- correct pour des ordinateurs avec très peu de mémoire.
- semble être un modèle très limité
- pourtant, très utilisé
automates simples, analyse syntaxique, IA, ...

Exemple : porte automatique

mécanisme électromagnétique basé sur un automate fini



1 Langage régulier

Dans notre cas, nous considérerons uniquement que le but d'un modèle de machine est de reconnaître *un langage*.

Dans ce chapitre, nous allons :

- définir la notion d'un alphabet et d'un langage
- définir la notion d'opérations régulières

— pour aboutir à la définition d'un langage régulier.

1.1 Alphabet

Donnons au préalable quelques définitions :

DÉFINITION 1 : alphabet

- Un **alphabet** Σ est un ensemble **fini** de symboles (=lettres).
- Σ^* = ensemble de toutes les chaînes possibles pouvant être générées par l'alphabet Σ .
- ϵ = la chaîne vide.

Exemples d'alphabet :

- $\Sigma = \{a, b, \dots, z\}$: alphabet des lettres (de l'alphabet)
- $\Sigma = \{0, 1\}$: alphabet binaire
- $\Sigma = \{0, 1, 2, \dots, 9\}$: alphabet des chiffres

Exemple de Σ^* :

pour l'alphabet binaire $\Sigma = \{0, 1\}$, $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$

1.2 Langage

DÉFINITION 2 : langage

Un **langage** L sur l'alphabet Σ est un sous-ensemble de Σ^* .

Autrement dit, $L \subseteq \Sigma^*$.

Exemples de langage :

- pour $\Sigma = \{a, b, \dots, z\}$, l'anglais est un langage
pour le français, il manque les symboles accentués
- $\Sigma = \{0, 1, \dots, 9\}$:
 - les nombres entre 1 et 100
 - les nombres pairs
 - les nombres premiers
- $\Sigma = \{0, 1\}$:
 - $L = \{w \mid w \text{ contient au moins 12 symboles } 0\}$
 - $L = \{0^n 1^n \mid n \geq 0\}$
 - $L = \{w \mid w \text{ a le même nombre de symboles } 0 \text{ et } 1\}$

1.2.1 Opérations régulières

DÉFINITION 3 : opérations régulières sur un langage

Soient A et B , deux langages.

On définit les opérations (dites régulières) suivantes :

- **Union** : $A \cup B = \{x \mid x \in A \text{ ou } x \in B\}$
- **Concaténation** : $A \circ B = \{xy \mid x \in A \text{ et } y \in B\}$
- **Etoile** : $A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ et } x_i \in A \text{ pour tout } i\}$

Exemple : soit $A = \{\alpha, \beta\}$ et $B = \{u, v, w\}$

- $A \cup B = \{\alpha, \beta, u, v, w\}$
- $A \circ B = \{\alpha u, \alpha v, \alpha w, \beta u, \beta v, \beta w\}$
- $A^* = \{\epsilon, \alpha, \beta, \alpha\alpha, \beta\alpha, \alpha\beta, \beta\beta, \alpha\alpha\alpha, \alpha\beta\alpha, \alpha\alpha\beta, \alpha\beta\beta, \dots\}$

Question : quelles sont les propriétés des langages ainsi obtenus ?

1.3 Expressions régulières

Vous avez tous déjà rencontrés des expressions régulières :

- caractère générique sous un shell (exemple : `ls *.txt`).
- recherche dans un éditeur de texte (exemple : `t[ao]t[ao]`).
- recherche de motif avec `grep`, `awk`, `perl`, ...
- analyseur syntaxique
- ...

Donnons-en maintenant une définition formelle.

1.3.1 Définition formelle d'une expression régulière

DÉFINITION 4 : formelle d'une expression régulière

R est une expression régulière (ER) sur un alphabet Σ si R est :

- un symbole : a où $a \in \Sigma$
- la chaîne vide : ϵ
- l'ensemble vide : \emptyset
- l'union d'ERs : $(R_1 \cup R_2)$ où R_1 et R_2 sont des ERs.
- la concaténation de ERs : $(R_1 \circ R_2)$ où R_1 et R_2 sont ERs.
- l'opération étoile sur une ER : R_1^* où R_1 est une ERs.
aussi opération + définie comme $R_1^+ = R_1 R_1^*$

Cette définition est circulaire : une expression régulière peut être formée par compositions régulières d'expressions régulières.

Exemple d'opérations de base : soit $\Sigma = \{0, 1\}$

- symbole 0 = $\{0\}$
- symbole 1 = $\{1\}$
- Σ = un symbole quelconque de l'alphabet (ici 0 ou 1).
- union : $(0 \cup 1) = \{0, 1\}$
- étoile : $0^* = \{0\}^* = \{\epsilon, 0, 00, 000, 0000, \dots\}$.
 $0^+ = \{0\}^+ = \{0, 00, 000, 0000, \dots\}$
- concaténation implicite : $01 = \{01\}$

Précédence des opérateurs : étoile, concaténation, union
utiliser les parenthèses pour changer l'ordre

Autres exemples

$$0^*10^* = \{1, 01, 10, 001, 010, 100, 0001, 0010, 0100, 1000, \dots\}$$

$$1^*(0 \cup 1)0^* = \{0, 1, 00, 10, 000, 100, 0000, 1000, 00000, 10000, \dots\}$$

Exemples :

1. $0^*10^* = \{w | w \text{ contient un seul } 1\}$
2. $\Sigma^*1\Sigma^* = \{w | w \text{ contient au moins un } 1\}$
3. $\Sigma^*001\Sigma^* = \{w | w \text{ contient } 001\}$
4. $1^*(01^+)^* = \{w | \text{tout } 0 \text{ de } w \text{ est suivi par au moins un } 1\}$
5. $(\Sigma\Sigma)^* = \{w | \text{chaîne de longueur paire}\}$
6. $(\Sigma\Sigma\Sigma)^* = \{w | \text{chaîne de longueur multiple de } 3\}$
7. $01 \cup 10 = \{01, 10\}$
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w | w \text{ commence et fini par le même symbole}\}$
9. $(0 \cup \epsilon)1^* = 01^* \cup 1^*$

10. $(0 \cup \epsilon)(0 \cup 1) = \{0, 1, 00, 01\}$
11. $1^* \emptyset = \emptyset$ (car la concaténation avec \emptyset donne \emptyset).
12. $\emptyset^* = \{\epsilon\}$ (si le langage est vide, la seule chaîne générée est la chaîne vide).

DÉFINITION 5 : langage associé à une expression régulière :

Soit R une expression régulière.

Le langage associé à l'expression régulière est l'ensemble contenant la totalité des chaînes pouvant être engendrée par cette expression régulière R .

On le note $\mathcal{L}(R)$.

Exemples :

- si $R = 1$, alors $\mathcal{L}(R) = \{1\}$.
- si $R = 1^*$, alors $\mathcal{L}(R) = \{\epsilon, 1, 11, 111, \dots\}$.
- si $R = 1^* \cup 0^*$, alors $\mathcal{L}(R) = \{\epsilon, 1, 0, 11, 00, 111, 000, \dots\}$.

Note : quelle est la différence entre $L_\emptyset = \emptyset$ et $L_\epsilon = \{\epsilon\}$?

L_\emptyset = langage qui ne contient aucune chaîne.

L_ϵ = langage qui contient la chaîne vide.

1.3.2 Langage régulier

DÉFINITION 6 : langage régulier

Un langage est dit régulier s'il peut être décrit sous la forme d'une expression régulière. Autrement dit, un langage L est régulier s'il existe une expression régulière R telle que $L = \mathcal{L}(R)$.

On regroupe l'ensemble des langages réguliers dans une classe.

DÉFINITION 7 : classe des langages réguliers

La classe des langages réguliers est constitué de l'ensemble des langages régulier.

Une grande partie du reste de ce chapitre a pour but de répondre aux questions suivantes :

- quels sont les propriétés de cette classe ?
- comment caractériser les langages réguliers ?
- existe-t-il des langages non réguliers ?
- si oui, comment le déterminer ?

1.3.3 Fermeture

DÉFINITION 8 : Fermeture d'une classe par une opération

Soit \times une opération définie sur une classe d'ensembles E .

E est dit fermé par \times , si $\forall A, B \in E$, alors $A \times B \in E$.

Exemple :

- \mathbb{N} est fermé par l'opérateur $+$.
pour tout couple d'entiers (a, b) dans \mathbb{N}^2 , on a $a + b \in \mathbb{N}$.
 - \mathbb{N} n'est pas fermé par l'opérateur $-$.
il existe au moins un couple d'entiers (a, b) dans \mathbb{N}^2 , tel que $a - b \in \mathbb{N}$.
- exemple :** $4 - 5 = -1 \notin \mathbb{N}$

Question : les langages réguliers sont-ils fermés par les opérations régulières ?

A savoir : soit L_1 et L_2 deux langages réguliers, est-ce-que $L_1 \circ L_2$, $L_1 \cup L_2$ ou L_1^* sont aussi des langages réguliers ?

THÉORÈME 1 : fermeture de la classe des langages réguliers

La classe des langages réguliers est fermée pour toute opération régulière.

Démonstration :

Par définition constructive des expressions régulières :

R	$\mathcal{L}(R)$
a	$\{a\}$
ϵ	$\{\epsilon\}$
$(R_1 \cup R_2)$	$\mathcal{L}(R_1) \cup \mathcal{L}(R_2)$
$(R_1 \circ R_2)$	$\mathcal{L}(R_1) \circ \mathcal{L}(R_2)$
$(R_1)^*$	$\mathcal{L}(R_1)^*$

Donc, appliquer une opération régulière entre deux langages réguliers revient à appliquer cette même opération régulière entre les deux expressions régulières associées à ces langages réguliers.

Or, par définition, ceci produit aussi une expression régulière, donc un langage régulier. \square

2 Automate fini

Un automate fini est une machine abstraite constitué d'états et de transitions dont le comportement est dirigé par un mot. Ce mot indique quelles transitions emprunter à la lecture consécutive des lettres du mot en entrée.

L'automate est dit fini car **le nombre d'états possibles dans l'automate est un ensemble fini**.

Nous allons étudier deux types d'automate fini :

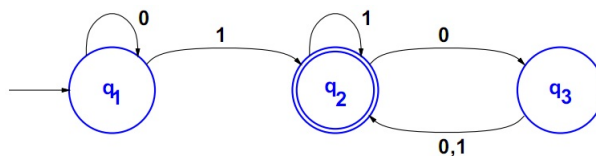
- **les automates déterministes finis** (ou ADF) : pour lesquels, la lecture d'une lettre conduit d'un état à un autre état unique
à savoir, la transition est déterministe (aucun choix possible)
- **les automates non-déterministes finis** (ou ANF) : pour lesquels, la lecture d'une lettre conduit d'un état à un ou plusieurs autres états (voir même aucun).
à savoir, la transition n'est plus déterministe (plusieurs choix possibles, voir même aucun).

Chaque automate reconnaît un langage constitué de l'ensemble des mots pour lesquels l'automate se termine dans un état dit acceptant.

2.1 Automate déterministe fini

Nous précisons maintenant la description et le fonctionnement d'un ADF.

Un ADF est une machine à état totalement défini par (sur l'exemple suivant) :



- **son alphabet** : $\{0, 1\}$ (ensemble des symboles reconnus par le langage).
- **ses états** : $\{q_1, q_2, q_3\}$
- **son état de départ (un seul)** : q_1 (marqué par une flèche entrante).
- **ses états acceptants (possiblement plusieurs)** : $\{q_2\}$ (marqués par un cercle double).
- **ses transitions d'état** : flèches entre 2 états
 - sur la flèche est indiqué le symbole qui permet la transition d'un état à l'autre,
 - à noter que pour tout état, il existe une transition (et une seule) associée à chaque symbole de l'alphabet.

Pour savoir si un mot (= chaîne de symboles définis sur l'alphabet de l'automate) est reconnu par l'automate, on procède de la manière suivante :

Initialisation :

- sur l'ADF, on se place sur l'état de départ (ici q_1),
- sur le mot, on place un curseur de lecture sur le premier caractère du mot.

Exécution :

1. transition :

- sur l'ADF, emprunter la transition associée au symbole pointé par le pointeur de lecture, et passer à l'état suivant.
 - sur le mot, avancer le curseur
2. répéter 1 jusqu'à arriver à la fin du mot (le curseur de lecture a lu le dernier symbole).
 3. le mot est **accepté** (= appartient au langage reconnu par l'ADF), si l'état courant est acceptant, et **refusé** (n'y appartient pas) sinon.

Que se passe-t-il avec les chaînes suivantes ?

1101	$q_1 q_2 q_2 q_3 \mathbf{q_2}$	accepte (q_2 acceptant)
0010	$q_1 q_1 q_1 q_2 \mathbf{q_3}$	rejette (q_3 non acceptant)
01100	$q_1 q_1 q_2 q_2 q_3 \mathbf{q_2}$	accepte (q_2 acceptant)

Quel est le langage reconnu par cet automate ?

Cet automate accepte les chaînes qui :

- soit se terminent par 1.
 - soit contiennent au moins un 1 et se terminent par un nombre pair de 0.
- et rien d'autre !

2.1.1 Formalisation

On note :

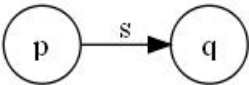
- Σ l'alphabet du langage reconnu par l'ADF.
- Q l'ensemble des états dans lequel un ADF peut se trouver.

Une **fonction de transition** δ est une fonction qui décrit :

- lorsque l'on se trouve dans un état q ,
- et que le symbole courant est s dans la chaîne d'entrée,
- donne dans quel état p on va se retrouver.

Donc, c'est une fonction $\delta : Q \times \Sigma \rightarrow Q$
 $(q, s) \mapsto p$

Autrement dit, l'état q résultant est obtenu à partir de l'état initial p et du symbole s , avec la fonction δ comme : $q = \delta(p, s)$.

Représentation graphique d'une transition : 

Rappel : $Q \times \Sigma \rightarrow Q$ signifie que pour chaque état $q \in Q$, la fonction $\delta(q, s)$ est définie pour tout symbole $s \in \Sigma$ et donne un état unique.

On peut donc maintenant définir :

DÉFINITION 9 : Description d'un automate déterministe fini (ADF)

Un automate déterministe fini est un 5-uple $(Q, \Sigma, \delta, q_0, F)$ où :

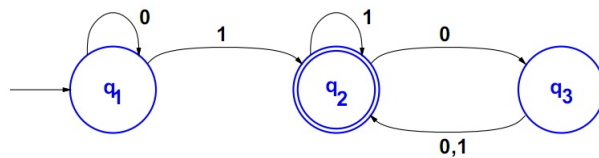
- Q est un ensemble fini d'états (ensemble des états).
- Σ est un ensemble fini de symboles (alphabet).
- $\delta : Q \times \Sigma \rightarrow Q$ est la fonction de transition.
- $q_0 \in Q$ est l'état de départ.
- $F \subseteq Q$ est l'ensemble des états acceptants.

Le 5-uple $(Q, \Sigma, \delta, q_0, F)$ est une définition formelle complète d'un ADF.

Remarque : pour un ADF $M = (Q, \Sigma, \delta, q_0, F)$

- $F = \emptyset \Leftrightarrow$ l'ADF n'accepte aucun mot, et reconnaît donc le langage vide \emptyset (qui ne contient aucun mot).
- $F = Q \Leftrightarrow$ l'ADF accepte tous les mots ($= \Sigma^*$).

sous l'hypothèse que tous les états sont atteignables depuis l'état de départ.

EXEMPLE 1: description d'un ADF

Cet ADF M est défini comme $(Q, \Sigma, \delta, q_1, F)$:

- l'ensemble des états est $Q = \{q_1, q_2, q_3\}$
- l'alphabet est $\Sigma = \{0, 1\}$
- la fonction de transition δ (ou table de transition) est :

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- q_1 est l'état de départ,
- l'ensemble des états acceptants est $F = \{q_2\}$

2.1.2 Modèle de calcul

Nous donnons maintenant une description formelle du traitement d'une chaîne de symboles par un ADF.

DÉFINITION 10 : chaîne acceptée par un ADF

Soient un ADF $M = (Q, \Sigma, \delta, q_0, F)$ et une chaîne de symboles $w = w_1 w_2 \cdots w_n$ de Σ^* . M accepte la chaîne de symboles w ssi il existe une suite d'états $\{r_0, \dots, r_n\} \in Q^{n+1}$ telle que :

- $r_0 = q_0$ (= le premier état est l'état de départ).
- pour $0 \leq i < n$, $r_{i+1} = \delta(r_i, w_{i+1})$ (\Rightarrow la suite des états permet d'accepter la suite des symboles de w).
- $r_n \in F$ (= l'état final est un état acceptant).

DÉFINITION 11 : langage accepté par un ADF

Le langage reconnu par un ADF M (noté $\mathcal{L}(M)$) est l'ensemble des chaînes qu'il accepte, à savoir $\mathcal{L}(M) = \{w \in \Sigma^* \mid M(w) \text{ accepte}\}$.

Un ADF peut être considéré comme un modèle simple d'ordinateur pour lequel :

- Q = états possibles de la mémoire.
- Σ = instructions reconnues par l'ordinateur.

- δ = effet d'une instruction sur la mémoire
définie pour toutes les instructions et tous les états de la mémoire.
- F = état de la mémoire en fin d'exécution.
état dans laquelle la terminaison du programme est normale.

En terme d'exécution de programme :

- la chaîne d'entrée est le programme à exécuter.
- le langage reconnu par l'ADF est l'ensemble des programmes qui s'exécutent.

2.1.3 Remarques

REMARQUE 1 :

1. Un ADF est déterministe car :
 - il n'y a qu'une seule transition possible par symbole Σ .
c'est le sens du mot déterministe.
 - il y a autant de transitions que de symboles dans Σ .
par définition de la fonction de transition.
2. pourquoi dit-on qu'un ADF possède de la mémoire ?
 - l'état présent peut être enregistré en prenant un chemin particulier.
 - ne permet de stocker qu'un petit nombre d'états (finis et prédéterminés).

EXEMPLE 2: (mémoire d'un ADF)

- un ADF peut se souvenir que les 3 derniers caractères rencontrés sont 000 ou 111.
- si une chaîne qui précède un état peut être de longueur quelconque, l'ADF auquel il appartient ne peut pas se souvenir si cette chaîne contient autant de 0 que de 1.

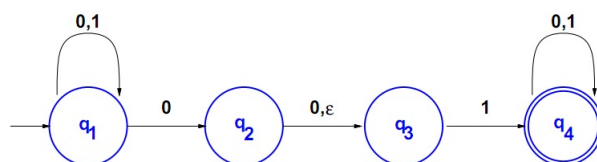
2.2 Automates non déterministes finis

Un automate non-déterministe fini (ANF) est un ADF modifié pour lequel, depuis un état, on modifie les transitions possibles de la manière suivante :

- un même symbole peut être associé à plusieurs transitions, voir même aucune.
- un symbole supplémentaire (noté ϵ) permet une transition sans lecture de symbole sur la chaîne d'entrée.

L'automate n'est alors plus déterministe car plusieurs choix peuvent être possibles lors d'une transition, ou aucun !

EXEMPLE 3: ANF



- transitions supplémentaires : q_1 a deux transitions associées à 1.
- transition manquante : q_2 n'a pas de transition pour 1 (aussi q_3 pour 0).
- transition ϵ : la transition entre q_2 et q_3 peut se faire sans lecture de symbole.

Un ANF s'exécute de la façon suivante :

- si plusieurs transitions sont possibles :
 - la machine se découpe en des copies multiples.
 - chaque copie (= une branche d'exécution) suit une possibilité.
 - l'ensemble des branches continuent et suivent toutes les possibilités.

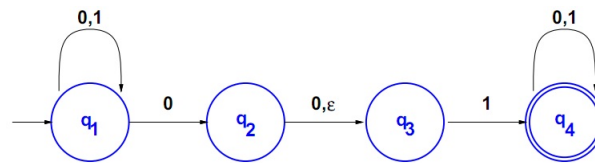
- une transition ϵ correspond à une transition sans lecture de symbole.
- si aucune transition n'est possible
la branche d'exécution ne peut pas se poursuivre : elle s'arrête (=rejet)

La chaîne d'entrée est :

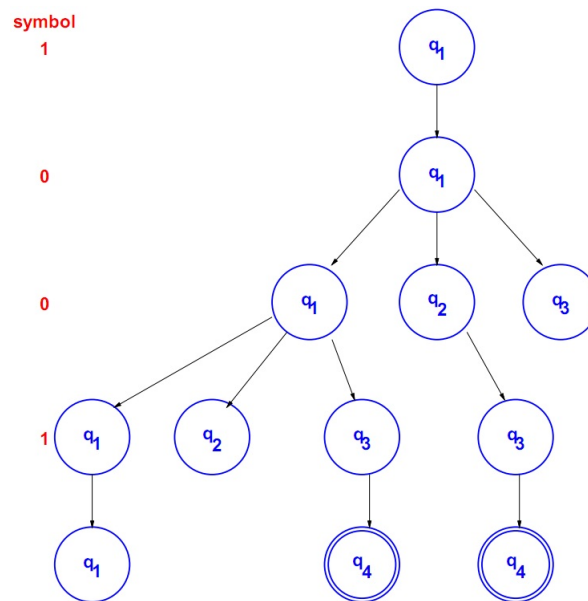
- **acceptée** si au moins l'une de ses branches d'exécution accepte (= est dans un état acceptant à la fin de la lecture de la chaîne).
- **rejetée** si toutes les branches d'exécution ont rejeté ou se sont arrêtées avant la fin de la chaîne.

EXEMPLE 4:

Automate non déterministe



Exécution



Note : pour une transition ϵ , $q_1 \xrightarrow{\epsilon} q_2 \xrightarrow{1} q_3$ devient $q_1 \xrightarrow{1} q_3$.

2.2.1 Formalisation

Avant de formaliser, nous avons au préalable besoin de définir la notion suivante :

DÉFINITION 12 : Ensemble des parties d'un ensemble

L'ensemble des parties d'un ensemble dénombrable E est défini comme étant l'ensemble de tous les sous-ensembles possibles contenu un ensemble (= union de l'ensemble des parties de E).

Notation : $\mathcal{P}(E)$.

Autrement dit : $\mathcal{P}(E) = \{A \mid A \subset E\}$.

$\mathcal{P}(E)$ est donc un ensemble d'ensembles (= contenant des ensembles).

Exemple : si $Q = \{q_1, q_2, q_3\}$ alors

$$\mathcal{P}(Q) = \{\{\}, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_1, q_3\}, \{q_2, q_3\}, \{q_1, q_2, q_3\}\}$$

Autre notation : $E_\epsilon = E \cup \{\epsilon\}$

ensemble E augmenté du symbole ϵ .

DÉFINITION 13 : Fonction de transition

La fonction de transition δ est définie de $Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$

où Q est l'ensemble des états et Σ l'alphabet du langage.

EXEMPLE 5:

On définit l'ensemble des états $Q = \{q_1, q_2, q_3\}$ et l'alphabet $\Sigma = \{0, 1\}$.

- $\mathcal{P}(Q)$ contient bien l'ensemble de toutes les transitions possibles entre un état et tous les autres (voir exemple ci-avant).
- Σ_ϵ est l'ensemble des symboles provoquant une transition (= symboles de l'alphabet + transition ϵ).
- $\delta(q, s)$ est défini pour tout $q \in \mathcal{P}(Q)$ et $s \in \Sigma_\epsilon$ (i.e. sur le produit cartésien $\mathcal{P}(Q) \times \Sigma_\epsilon$).

Exemples de transitions :

- $\delta(q_1, \epsilon) = \{q_2\}$: transition simple $q_1 \xrightarrow{\epsilon} q_2$
- $\delta(q_2, 0) = \{q_1, q_3\}$: transitions multiples $q_3 \xleftarrow{0} q_2 \xrightarrow{0} q_1$
- $\delta(q_1, 1) = \{\}$: pas de transition pour le symbole 1 depuis q_1

DÉFINITION 14 : description d'un automate non-déterministe fini (ANF)S

Un automate non-déterministe fini est un 5-uple $(Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'états(ensemble des états).
- Σ est un ensemble fini de symboles (alphabet).
- $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ est la fonction de transition.
- $q_0 \in Q$ est l'état de départ.
- $F \subseteq Q$ est l'ensemble des états acceptants.

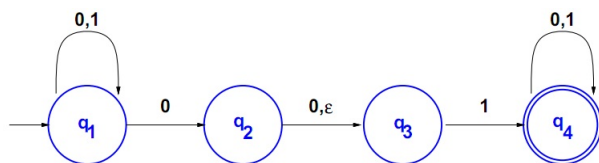
Le 5-uple $(Q, \Sigma, \delta, q_0, F)$ définit complètement l'automate non-déterministe fini.

On remarque que par rapport à la définition formelle d'un ADF seule la définition de la fonction de transition indique que l'on a affaire à un ANF.

Qu'en est-il du modèle de calcul ?

EXEMPLE 6: description formelle d'un ANF

Soit l'ANF suivant :



$N = (Q, \Sigma, \delta, q_1, F)$ où :

- états : $Q = \{q_1, q_2, q_3, q_4\}$
- langage : $\Sigma = \{0, 1\}$
- fonction de transition : $\delta(q, s) =$

	0	1	ϵ
q_1	$\{q_1, q_2\}$	$\{q_1\}$	\emptyset
q_2	$\{q_3\}$	\emptyset	$\{q_3\}$
q_3	\emptyset	$\{q_4\}$	\emptyset
q_4	$\{q_4\}$	$\{q_4\}$	\emptyset

- état de départ : q_1
- états acceptants : $F = \{q_4\}$

2.2.2 Modèle de calcul

DÉFINITION 15 : chaîne acceptée par un ANF

Soient :

- un ANF $M = (Q, \Sigma, \delta, q_0, F)$
 - une chaîne de symboles $w = w_1 w_2 \cdots w_n$ de Σ_ϵ^* (i.e. $\forall i, w_i \in \Sigma_\epsilon$)
 - soit u la chaîne de Σ^* obtenue en supprimant toutes les occurrences ϵ de la chaîne w .
- M accepte la chaîne de symboles w ssi il existe une suite d'états $\{r_0, \dots, r_n\} \in Q^{n+1}$ telle que :
- $r_0 = q_0$ (= le premier état est l'état de départ)
 - pour $0 \leq i < n$, $r_{i+1} \in \delta(r_i, w_{i+1})$ (= la suite des états permet d'accepter la suite des symboles de w).
 - $r_n \in F$ (l'état final est un état acceptant)

Alors le mot u est accepté par l'ANF.

Noter que la condition $r_{i+1} \in \delta(r_i, w_{i+1})$ correspond à l'existence d'une branche d'exécution conduisant à un état acceptant.

2.2.3 Intérêt du non déterministe

Quel est l'intérêt du non déterministe ? Dans un premier temps, comparons l'ADF et l'ANF qui reconnaissent un même langage.

EXEMPLE 7:

Soit le langage qui accepte toutes les chaînes ayant un 1 en 3^{ème} position avant la fin.
i.e. : de la forme $xx \dots xx1xx$ où chaque x est 0 ou 1.
Quels sont l'ANF et l'ADF correspondants ?

Solution pour l'ADF :

Solution pour l'ANF :

En conséquence, comparativement à un ADF, un ANF :

- a visiblement une expressivité plus grande,
- par son non-déterministe, paraît être en mesure de reconnaître des langages qu'un ADF n'est pas capable de reconnaître.

2.3 Equivalence ANF-ADF

THÉORÈME 2 : équivalence entre ANF et ADF

les ADFs et ANFs acceptent **exactement** le même ensemble de langage.

Ce qu'il faut démontrer :

\Rightarrow pour tout ADF M , il existe un ANF N tel que $\mathcal{L}(M) = \mathcal{L}(N)$.

Evident, car les ANFs sont une généralisation des ADFs.

Tout ADF est aussi un ANF pour lequel il y a exactement une seule transition par symbole. \square

\Leftarrow pour tout ANF N , il existe un ADF M tel que $\mathcal{L}(M) = \mathcal{L}(N)$.

1. dans un premier temps, on ignore les transitions ϵ .
on construit un ADF qui simule tous les états de l'ANF.
2. on ajoute ensuite toutes les transitions créées par les suites de transition ϵ .
3. on vérifie ensuite que le langage accepté par l'ADF est le même que celui accepté par l'ANF.

Pour la réciproque, nous allons démontrer ce principe sur un exemple. Celui-ci peut être facilement généralisé et formalisé.

Démonstration étape 1 : \Leftarrow (Construction sans transition ϵ)

Soit $N = (Q, \Sigma, \delta, q_0, F)$, un ANF qui accepte le langage A .

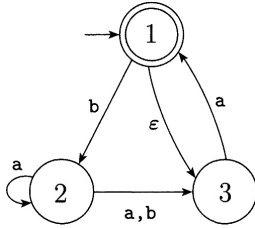
Construisons l'ADF $M = (Q', \Sigma, \delta', q'_0, F')$ qui simule toutes les branches d'exécution en même temps.

- $Q' = \mathcal{P}(Q)$ (l'ensemble des parties de Q).
 \Rightarrow si le ANF a k état, alors l'ADF a 2^k états.
- pour $R \in Q'$ et $s \in \Sigma$,
 soit $\delta'(R, s) =$ pour tous les $q \in R$, ensemble des q' tel que $q' = \delta(q, s)$ (l'état de départ est l'élément de Q' qui contient q_0).
- $q'_0 = \{q_0\}$
- $F' = \{R \in Q' \mid R \text{ contient un état acceptant de } N\}$
 \Rightarrow si la partie contient un état acceptant, elle accepte.

On remarquera que la simulation des branches fusionne automatiquement les branches qui ont abouties à un même état (*i.e.* si un état est atteint dans l'ADF équivalent, on ne connaît pas la branche qui conduit à cet état).

Exemple étape 1 : Construction sans transition ϵ (exemple)

Pour $N = (Q, \Sigma, \delta, q_0, F)$



$$\begin{aligned} Q &= \{1, 2, 3\} \\ \Sigma &= \{a, b\} \\ q_0 &= 1 \\ F &= \{1\} \end{aligned}$$

$\delta =$	a	b	ϵ
1	\emptyset	$\{2\}$	$\{3\}$
2	$\{2, 3\}$	$\{3\}$	\emptyset
3	$\{1\}$	\emptyset	\emptyset

Pour $M = (Q', \Sigma, \delta', q'_0, F')$

$$\begin{aligned} Q' &= \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \\ &\quad \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\} \\ q'_0 &= \{1\} \\ F' &= \{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\} \end{aligned}$$

$\delta' =$	a	b
\emptyset	\emptyset	\emptyset
$\{1\}$	\emptyset	$\{2\}$
$\{2\}$	$\{2, 3\}$	$\{3\}$
$\{3\}$	$\{1\}$	\emptyset
$\{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{1, 3\}$	$\{1\}$	$\{2\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$

Démonstration étape 2 : \Leftarrow (construction avec transition ϵ)

Comment intégrer les transitions ϵ ?

Pour chaque état $q \in Q$, on regarde tous les états atteignables depuis q par un nombre quelconque de transitions ϵ (zero transition compris).

On note cet ensemble d'états $E(q)$.

La fonction de transition est alors modifiée comme :

$$\delta''(R, s) = \{E(q) \mid q \in \delta'(R, s)\}$$

Autrement dit, on intègre à $\delta'(R, s)$ l'ensemble des états qui peuvent être atteints par un ou plusieurs transitions ϵ depuis les états contenus dans $\delta'(R, s)$.

L'état de départ est également modifié en : $q'_0 = E(\{q_0\})$.

Les états finaux ne sont pas modifiés (les transitions ϵ ont déjà été intégrées dans δ'').

Exemple étape 2 : Construction avec transition ϵ (exemple)

— La seule transition ϵ est la transition $1 \xrightarrow{\epsilon} 3$.

— A savoir, ajouter la transition 3 partout où il y a la transition 1.

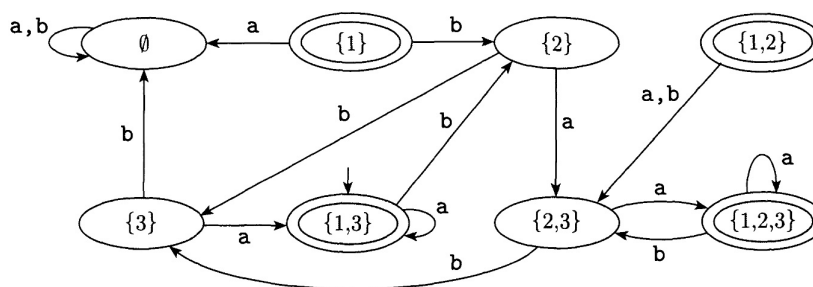
$\delta' =$	a	b	$\delta'' =$	a	b
\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
$\{1\}$	\emptyset	$\{2\}$	$\{1\}$	\emptyset	$\{2\}$
$\{2\}$	$\{2, 3\}$	$\{3\}$	$\{2\}$	$\{2, 3\}$	$\{3\}$
$\{3\}$	$\{1\}$	\emptyset	$\{3\}$	$\{1, 3\}$	\emptyset
$\{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$	$\{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{1, 3\}$	$\{1\}$	$\{2\}$	$\{1, 3\}$	$\{1, 3\}$	$\{2\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$	$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$	$\{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$

$$q'_0 = \{1\}$$

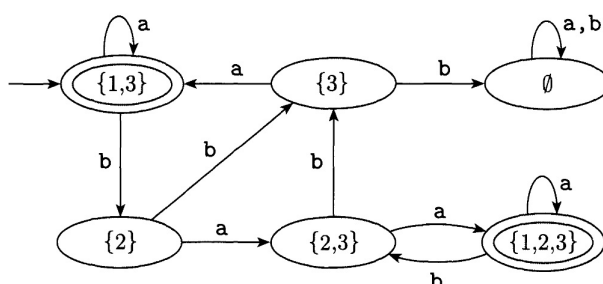
$$q''_0 = \{1, 3\}$$

Etat final : $F' = \{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}$

ce qui conduit à l'automate suivant :



On peut supprimer un état qui n'a pas de transition entrante et qui n'est pas l'état de départ.



Démonstration étape 3 : \Leftarrow (équivalence des automates - idée)

Par construction, l'ADF obtenu à l'étape 2 simule dans ses transitions toutes les branches d'exécutions en même temps à chaque lecture de symbole sur le mot d'entrée.

En conséquence, tout mot reconnu par l'ANF est également reconnu par l'ADF équivalent. \square

Conséquences de l'équivalence :

- tout langage reconnu par un ANF est aussi reconnu par un ADF, et vice-versa.
- un ANF n'est pas plus puissant qu'un ADF (il ne permet pas de reconnaître plus de langages).
- toute propriété démontrée sur un ANF sera également valide sur un ADF.

On se pose maintenant la question : quel type de langages peut être reconnu par un automate fini ?

On note L_{AF} l'ensemble des langages reconnus par un automate fini, à savoir :

$$L_{AF} = \{\mathcal{L}(M) \mid M \text{ est un ADF ou un ANF}\}$$

2.4 Fermeture

2.4.1 Fermeture par union

Afin de répondre à cette question, intéressons-nous tout d'abord au problème de fermeture par les opérations régulières de L_{AF} .

Comment vérifier que L_{AF} est fermé par union ?

Il faut vérifier que $\forall L_1, L_2 \in L_{AF}, L_1 \cup L_2 \in L_{AF}$.

Si $L_1 \in L_{AF}$ (resp. $L_2 \in L_{AF}$), alors il existe un automate fini M_1 (resp. M_2) tel que $\mathcal{L}(M_1) = L_1$ (resp. $\mathcal{L}(M_2) = L_2$).

Si $L_1 \cup L_2 \in L_{AF}$, il existe un automate fini $M_{1 \cup 2}$ tel que $\mathcal{L}(M_{1 \cup 2}) = L_1 \cup L_2$.

Première idée naïve (et fausse) de preuve :

Soit l'ADF $M_{1 \cup 2}$ à simuler et w la chaîne d'entrée :

- simuler d'abord M_1
- si M_1 n'accepte pas w , alors simuler M_2 .

Qu'est ce qui ne va pas ?

la chaîne w n'est mise en entrée de $M_{1 \cup 2}$ qu'une seule fois.

obligerait à «rembobiner» $w \Rightarrow$ impossible.

solution : il faut simuler les deux machines **en même temps**.

Principe : de l'union de deux ADFs

Supposons que :

$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepte L_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepte L_2

Alors, l'ADF $M_{1 \cup 2}$ suivant accepte $L_1 \cup L_2$:

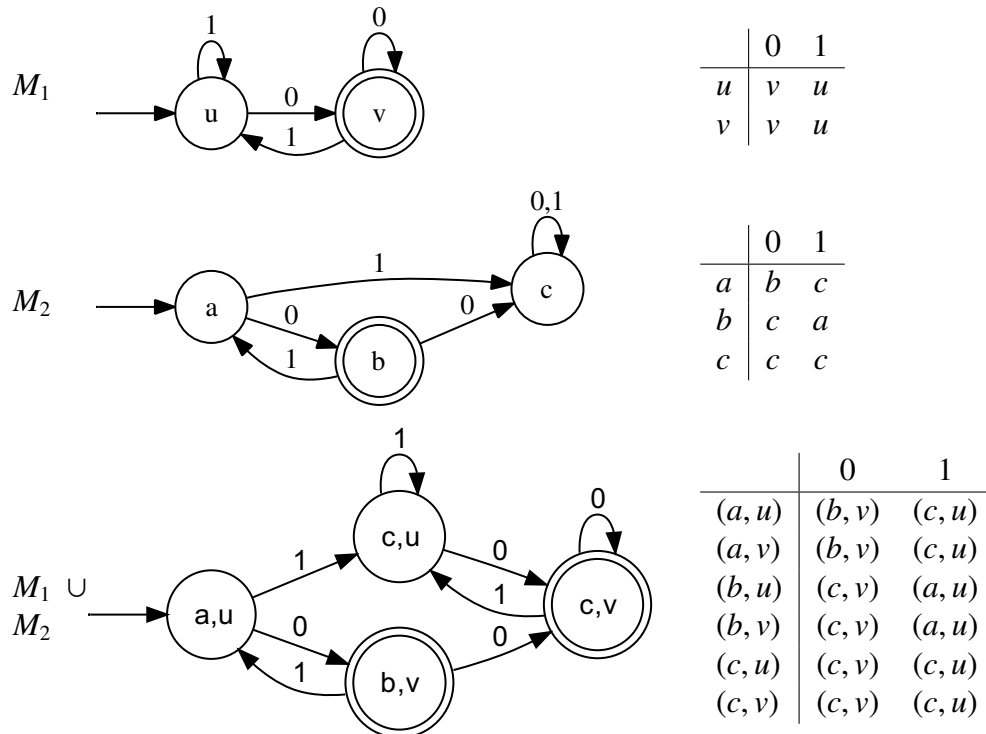
- $Q = Q_1 \times Q_2$
tous les couples possibles avec $r_1 \in Q_1$ et $r_2 \in Q_2$
- Σ ne change pas.
- $\forall (r_1, r_2) \in Q, a \in \Sigma, \delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
simule M_1 (resp. M_2) sur le premier (resp. second) élément du couple.
- $q_0 = (q_1, q_2)$
- $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ ou } r_2 \in F_2\} = (F_1 \times Q_2) \cup (Q_1 \times F_2)$
état acceptant si l'un des états du couple est acceptant.

Cet ADF simule de façon évidente $M_1 \cup M_2$.

Donc, les langages réguliers sont fermés par union.

Ce principe est explicité dans l'exemple ci-dessous.

Exemple : de l'union de deux ADFs



Note : les états (a, v) et (b, u) peuvent être supprimés car ce ne sont pas l'état de départ, et aucune transition n'y aboutit.

Remarques :

- si lors de la construction de l'ADF $M_{1 \cup 2}$, on avait défini F comme $F_1 \times F_2$, alors on obtient l'intersection.
 $F_1 \times F_2$ correspond à l'ensemble des couples pour lesquels les deux ADFs M_1 et M_2 acceptent en même temps.
- Soit un ADF $M = (Q, \Sigma, \delta, q_0, F)$. Que se passe-t-il si l'on construit l'ADF $\overline{M} = (Q, \Sigma, \delta, q_0, \overline{F})$ où $\overline{F} = Q \setminus F$ (= le complémentaire de F dans Q) ?
 Alors, les états acceptants devient rejettants et vice-versa.
 \overline{M} reconnaît donc le langage complémentaire de M .
 Donc, $\mathcal{L}(\overline{M}) = \Sigma^* \setminus \mathcal{L}(M)$.

Conséquences : L_{AF} est fermé par union, intersection et complémentation ?

Pour les deux autres opérations régulières, la construction semble beaucoup plus difficile :

- concaténation : comment faire pour trouver où découper la chaîne $w = w_1 w_2$ avec $w_1 \in L_1$ et $w_2 \in L_2$?
- étoile : comment savoir en combien de p morceaux découper la chaîne $w = w_1 \dots w_p$, où la découper et reconnaître chaque morceau $w_i \in L$?

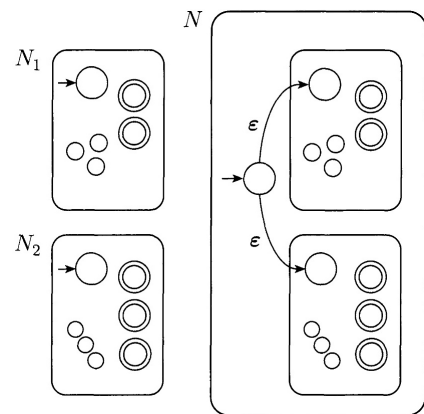
Les ANFs peuvent-ils nous aider dans cette tâche ?

Principe : de l'union de deux ANFs

L'union de deux ANFs N_1 et N_2 en un nouvel ANF N reconnaissant $\mathcal{L}(N_1) \cup \mathcal{L}(N_2)$ s'effectue tout simplement en :

- ajoutant un nouvel état de départ,
- en reliant ce nouvel état de départ aux états de départs des ANFs N_1 et N_2 par une transition ϵ .

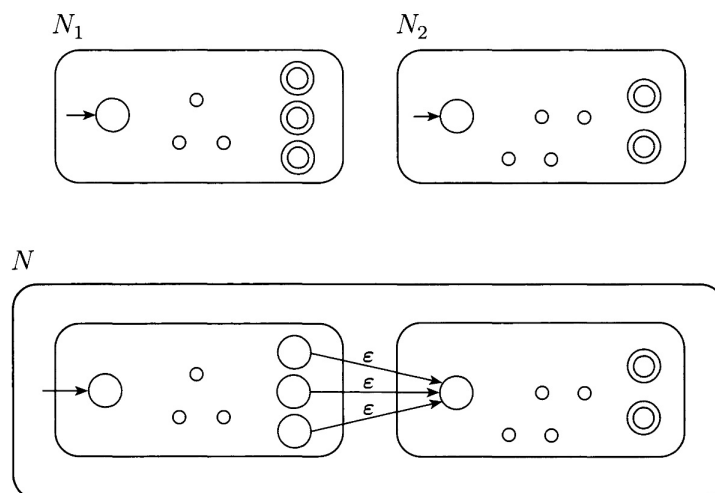
Au début de la lecture d'un mot, la transition ϵ produit deux branches d'exécution ; chacune s'occupant d'exécuter l'un des ANFs sur le mot.



La construction de l'union de deux ANFs est donc triviale.

2.4.2 Fermeture par concaténation

Principe : de la concaténation de deux ANFs

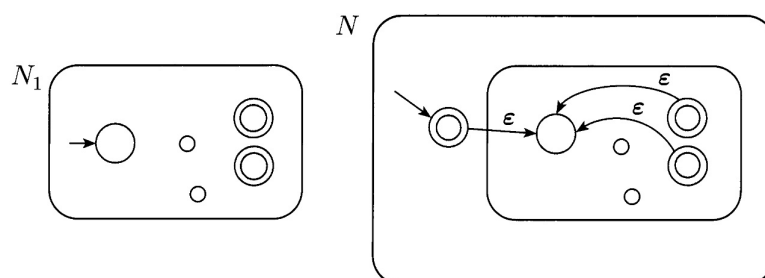


- L'état de départ du premier ANF N_1 reste celui de la concaténation N .
- On ajoute à tous les états finaux de N_1 une transition ϵ supplémentaire vers l'état initial de N_2 .
- L'ensemble des états finaux de N_2 devient celui de la concaténation N .

Ainsi, lorsqu'un mot $w = w_1w_2$ (avec $w_1 \in \mathcal{L}(N_1)$ et $w_2 \in \mathcal{L}(N_2)$), c'est le non-déterminisme qui permet à l'ANF N de décider quand il convient de passer du premier sous-ANF N_1 au second sous-ANF N_2 à travers l'ensemble de ses branches d'exécution.

2.4.3 Fermeture par opérateur étoile

Principe : de l'"étoilification" d'un ANF N_1 en un ANF N



- un nouvel état de départ est ajouté, et celui-ci est relié à l'état de départ de l'ANF N_1 par une transition ϵ .

Cet ajout est nécessaire car l'opérateur $*$ peut également signifier "aucune répétition".

- une transition ϵ relie chaque état de l'ensemble des états finaux de N_1 à l'état de départ de N_1 . L'ajout de cette transition permet à l'automate N_1 de pouvoir recommencer un nouveau mot de $\mathcal{L}(N_1)$ chaque fois qu'un de ses mots a été reconnu.

Encore une fois, c'est le non-déterminisme qui permet, à travers l'ensemble des branches d'exécution, de simuler le nombre de passage nécessaire à travers le sous-ANF N_1 .

2.4.4 Conséquences

Une conséquence de l'équivalence entre les ANFs et les ADFs est que si une propriété est démontrée pour l'un, elle est vraie pour l'autre.

Nous venons donc de voir que la classe L_{AF} est fermée par :

- union
- intersection
- complémentation
- concaténation
- opérateur étoile

La classe L_{AF} semble avoir donc plus de propriétés que celle des langages réguliers

\Rightarrow cet ensemble de classe est probablement plus petit.

Qu'en est-il vraiment ?

2.5 Equivalence entre AF et ER

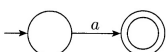
THÉORÈME 3 : ER \Rightarrow ANF

Toute expression régulière peut être reconnue par un ANF.


Idée de la preuve :

on reprend la définition d'une ER :


- $R = a$ où $a \in \Sigma$, alors $\mathcal{L}(R) = \{a\}$

l'ANF associé est : 

- $R = \epsilon$, alors $\mathcal{L}(R) = \{\epsilon\}$

l'ANF associé est : 

- $R = \emptyset$, alors $\mathcal{L}(R) = \emptyset$

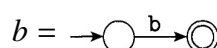
l'ANF associé est : 

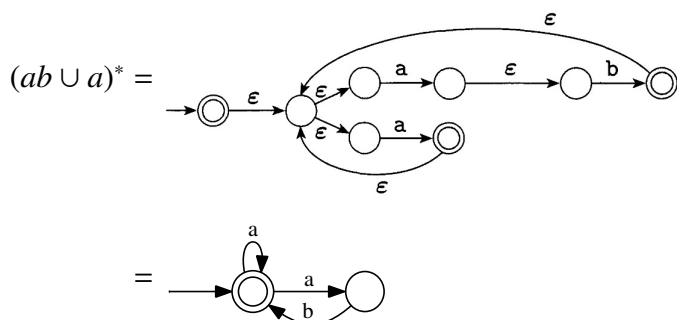
- $R = R_1 \cup R_2$: fermeture des ANFs par union.
- $R = R_1 \circ R_2$: fermeture des ANFs par concaténation.
- $R = R_1^*$: fermeture des ANFs par l'étoile.

□

Exemple : construction d'un ANF à partir d'une ER

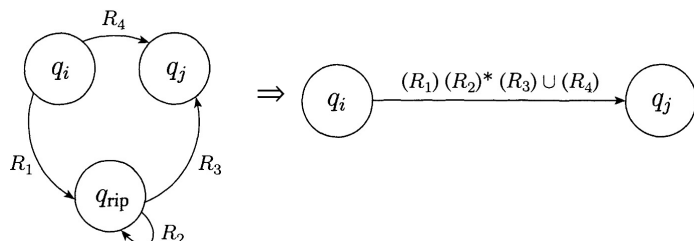
construire l'ANF associé à l'ER : $(ab \cup a)^*$



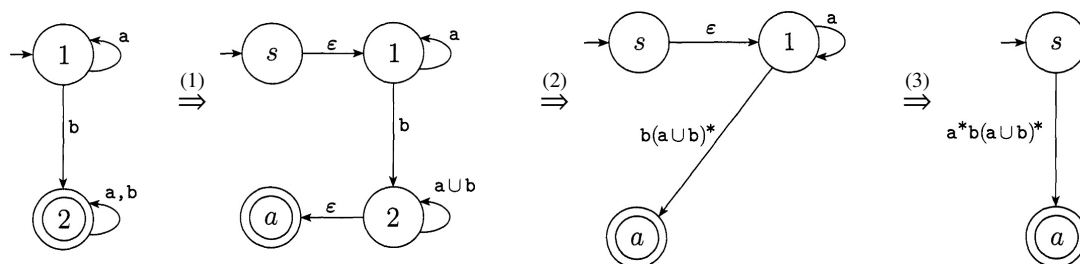


Idée de la preuve :

1. On définit un AGNF = un ANF avec une expression régulière sur ses transitions avec une seule sortie sur l'état d'entrée et une seule entrée sur l'état de sortie.
2. Il est possible de supprimer un état en remplaçant tout cycle passant par cet état par une expression régulière de la façon suivante :



3. L'AGNF est réduit état par état jusqu'à n'avoir plus qu'une seule transition :



- ADF, ANF et ER sont équivalents.
- les langages reconnus par un AF ou un AR sont des langages réguliers.
- inversement, si un langage est régulier, alors il reconnu par un AF ou un ER.

- donner son expression régulière,
- donner son ADF,
- donner son ANF

au choix !

Remarque : tout langage L dont le cardinal est fini est régulier (*i.e.* si $\#L$ fini, alors L est régulier).

Exercice : le démontrer.

3 Langages non réguliers

Existe-t-il des langages non réguliers ?

Ou de manière équivalente : existe-t-il des langages qui ne peuvent être reconnus par aucun AF ?

Il existe un AF possède peu ou pas de mémoire. Tout langage nécessitant de la mémoire n'est donc pas régulier.

Existe-t-il un AF qui accepte :

- $A = \{0^n 1^n \mid n \geq 0\}$
- $B = \{w \mid w \text{ a le même nombre de 0 et de 1}\}$
- $C = \{w \mid w \text{ a le même nombre d'occurrences de 01 et de 10}\}$

Nous allons présenter deux méthodes permettant de déterminer si un langage n'est pas régulier :

- Le **théorème de Myhill-Nerode** qui donne un critère permettant de conclure de manière définitive si un langage est régulier ou pas (condition nécessaire et suffisante).
- Le **lemme de l'étoile** qui donne une condition qui, si elle n'est pas vérifiée, permet de conclure que le langage n'est pas régulier (condition nécessaire de régularité).

3.1 Théorème de Myhill-Nerode

Soit un langage quelconque $L \subseteq \Sigma^*$ (régulier ou non).

DÉFINITION 16 : mots L -équivalents

Deux mots u et v sont L -équivalents si :

$$\forall z \in \Sigma^*, uz \in L \Leftrightarrow vz \in L$$

ou, exprimé autrement :

pour tout z de Σ^* tq $uz \in L$ alors $vz \in L$ et vice-versa.

Remarques :

- on note $u \equiv_L v$ si u et v sont L -équivalents.
- si $u \not\equiv_L v$, alors u et v sont dits L -disjoints.
- cette relation partitionne l'ensemble Σ^* en sous-espaces disjoints appelés L -classes.
- deux mots u et v sont dans la même L -classe ssi $u \equiv_L v$.

Un ensemble E est dit :

- **fini** si $\#E \in \mathbb{N}$.
 \Rightarrow son cardinal est un entier.
- **dénombrable** s'il existe une bijection de E sur \mathbb{N} (\neq fini).
 \Rightarrow même nombre d'éléments que \mathbb{N} .
 ensemble infini indexable par des entiers.
- **infini** s'il est non dénombrable.
 \Rightarrow infini que l'on ne peut compter.

Exemples :

- $A = \{w \mid w \text{ est un mot de longueur } p\}$ est fini car $\#A = (\#\Sigma)^p$.
- $B = \{0^n 1 \mid n \geq 0\}$ est dénombrable.
- $C = 0\Sigma^*$ est infini car Σ^* n'est pas dénombrable.

Le théorème suivant donne une condition **nécessaire** et **suffisante** de régularité d'un langage.

THÉORÈME 5 : Myhill-Nerode

Un langage L est régulier ssi il existe un nombre fini de L -classes distinctes.

Idée intuitive de la démonstration :

\Rightarrow Soit $q_0 \dots q_n$ la chaîne des états par lesquels on passe pour accepter w .

Trivialement, toutes les sous-chaînes z qui partent de l'état q_i et qui conduisent à un état acceptant permettent de vérifier la L -équivalence, et ceci indépendamment du passé avant q_i et du q_i choisi.

Donc, cela ne dépend que de q_i . Et il ne peut donc y avoir plus de L -classes que d'états dans Q .

Or, par définition, le nombre d'états dans un ADF est fini.

\Leftarrow Inversement, il est possible pour tout langage L dont le nombre de L -classes est fini de construire son ADF associé.

D'abord quelques exemples d'utilisation de ce théorème.

Exemples :

— $L = \{0^n 1^n \mid n \geq 0\}$

Les mots de $E = \{0^n \mid n \geq 0\}$ sont L -distincts : $i \neq j$, $0^i 1^i \in L$ et $0^j 1^i \notin L$

$\Rightarrow 0^i$ et 0^j sont dans 2 L -classes différentes.

Or E n'est pas fini \Rightarrow infinité de L -classes $\Rightarrow L$ non régulier

— $L = \{w \in \{0, 1\}^* \mid w \text{ est un palindrome}\}$

Les mots de $E = \{0^n 1 \mid n \geq 0\}$ sont L -distincts : $i \neq j$, $0^i 1 0^i \in L$ et $0^j 1 0^i \notin L$

$\Rightarrow 0^i 1$ et $0^j 1$ sont dans 2 L -classes différentes.

Or E n'est pas fini \Rightarrow infinité de L -classes $\Rightarrow L$ non régulier

— $L = \{0^{2^n} \mid n \geq 0\}$

Les mots de L sont tous L -distincts.

En effet, $i \neq j$, $0^{2^i} 0^{2^i} = 0^{2^{i+1}} \in L$ et $0^{2^j} 0^{2^i} = 0^{2^i+2^j} \notin L$.

Or L n'est pas fini \Rightarrow infinité de L -classes $\Rightarrow L$ non régulier

Démonstration : (préambule)

soit $M = (Q, \Sigma, \delta, q_0, F)$ un ADF et $L = \mathcal{L}(M)$.

Notons δ^* la fonction de transition cumulée définie comme $\delta^*(q_0, w) = q$ l'état que l'ADF atteint lorsque l'on part de l'état q_0 après avoir traité la chaîne w .

A chaque état $q \in Q$ correspond un sous-ensemble $S_q \subseteq \Sigma^*$ défini par :

$$\begin{aligned} S_q &= \{w \in \Sigma^* \mid \delta^*(q_0, w) = q\} \\ &= \text{ensemble des mots allant de l'état } q_0 \text{ à l'état } q. \end{aligned}$$

Lemme 1 : pour tout $q \in Q$, tous les mots de S_q sont L -équivalents.

Démonstration : si $\delta^*(q_0, u) = q = \delta^*(q_0, v)$ alors $\delta^*(q_0, ux) = \delta^*(q, x) = \delta^*(q_0, vx)$ pour tout $x \in \Sigma^*$.
Donc, $ux \in L$ ssi $vx \in L$, et $u \equiv_L v$. \square

Lemme 2 : si $u \equiv_L v$ alors $ux \equiv_L vx$ pour tout mot $x \in \Sigma^*$.

Démonstration : si $u \equiv_L v$, alors $uz \in L \Leftrightarrow vz \in L$ pour tout $z \in \Sigma^*$, ce qui est aussi vrai lorsque z s'écrit sous la forme $z = xy$. Donc, $uxy \in L \Leftrightarrow vxy \in L$, et $ux \equiv_L vx$. \square

\Rightarrow soit $M = (Q, \Sigma, \delta, q_0, F)$ un ADF et $L = \mathcal{L}(M)$.

Montrons que M a un nombre fini de L -classes : par le lemme 1, on sait que pour chaque état q , tous les mots de S_q appartiennent à la même classe. Donc, il n'y a pas plus de L -classes que d'états dans Q . Or, le nombre d'état est fini. Donc, le nombre de L -classes aussi.

\Leftarrow supposons que L est un langage quelconque tel que le nombre de L -classes est fini. Notons $[w]$ la L -classe contenant $w \in \Sigma^*$.

Soit l'ADF $M = (Q, \Sigma, \delta, q_0, F)$, défini par :

- Q l'ensemble des L -classes.
- $q_0 = [\epsilon]$ la L -classe contenant le mot vide.
- F = ensemble des L -classes contenant un mot de L .
- $\delta : Q \times \Sigma \rightarrow Q$ est défini par $\delta([w], a) = [wa]$

Montrons maintenant que cet ADF a bien les propriétés recherchées.

- ⇐ — Existence de δ :
- Par définition, si $u \equiv_L v$, alors $[u] = [v]$.
 Par le lemme 2, on a $[ua] = [va]$ pour tout $a \in \Sigma^*$.
 Ceci nous assure que la fonction δ est bien définie.
 En particulier, $\forall w \in \Sigma^*, \delta(q_0, w) = [w]$.
- $[w] \in F$ ssi $w \in L$:
- Par définition de F , si $w \in L$, alors $[w] \in F$.
 Inversement, si $[w] \in F$, alors $[w]$ contient un mot $u \in L$.
 Puisque u et w sont L -équivalents, on a donc $w \in L$.

On en conclut que pour tout $w \in \Sigma^*$, on a :

$$\delta^*(q_0, w) = [w] \text{ et que } [w] \in F \text{ ssi } w \in L.$$

Donc, $L = \mathcal{L}(M)$ est un langage régulier car il est généré par l'ADF M .

□

3.2 Lemme de l'étoile

Autre propriété importante : lemme de l'étoile

aussi connu sous le nom de "Pumping lemma".

On considère les propriétés du graphe que représente un ADF $M = (Q, \Sigma, \delta, q_0, F)$:

- un ADF peut être vu comme un graphe de transition :
chaque mot de Σ^* peut être interprété comme un chemin dans le graphe partant du nœud q_0 .
- langage $\mathcal{L}(M)$ = ensemble des mots correspondant aux chemins partant de l'état initial et arrivant à un état final et le dernier nœud $q \in F$.

Deux remarques peuvent être faites.

REMARQUE 2 : principe du casier (pigeonhole principle)

Si un graphe possède n sommets, alors tout chemin de longueur supérieure ou égale à n doit nécessairement passer au moins deux fois par un même sommet.

Prendre n boîtes et $p > n$ balles, et essayer de ne mettre qu'une seule balle par casier.

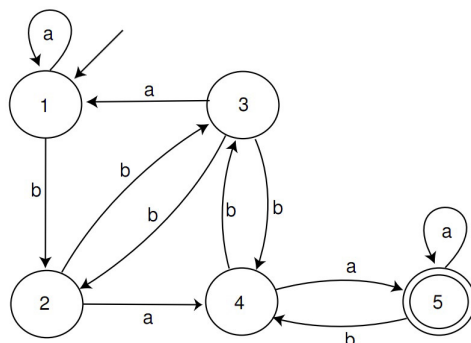
En terme de graphe, cela signifie l'existence d'un cycle :

Soit w un mot de L tel que $|w| > |Q|$. Alors il existe un état $p \in Q$ et une décomposition $w = xyz$ tels que :

1. $\delta^*(q_0, x) = p$ (x est le chemin de q_0 à p)
2. $\delta^*(p, y) = p$ (y est un cycle qui part de p et fait revenir à p)
3. $\delta^*(p, z) \in F$ (z est le chemin de p à un état acceptant)

Exemple :

Voir le graphe ci-contre avec les mots *aabaa*, *bbbbaa*, *baaba*, *babba*.



w	x	y	z	$\delta^*(1; x) = p$	$\delta^*(p; y) = p$	$\delta^*(p; z) = 5$
$aabaa$	ϵ	a	$abaa$	$\delta^*(1; \epsilon) = 1$	$\delta^*(1; a) = 1$	$\delta^*(1; abaa) = 5$
$bbbaa$	$bbba$	a	ϵ	$\delta^*(1; bbba) = 5$	$\delta^*(5; a) = 5$	$\delta^*(5; \epsilon) = 5$
$baaba$	ba	ab	a	$\delta^*(1; ba) = 4$	$\delta^*(4; ab) = 4$	$\delta^*(4; a) = 5$
$babba$	ba	bb	a	$\delta^*(1; ba) = 4$	$\delta^*(4; bb) = 4$	$\delta^*(4; a) = 5$

Conséquence du principe du casier :

- on peut toujours trouver un cycle dans les $|Q|$ premiers caractères de w .
- on note y ce **premier** cycle.
- la chaîne y a toujours au moins un caractère
c'est la partie qui se répète ; contrairement à x et z qui peuvent être ϵ .

Donc, on a :

1. $|y| > 0$
= il y a un cycle
2. $|xy| < |Q|$
= on effectue toujours au moins un cycle avant d'avoir lu Q caractères

REMARQUE 3 :

S'il existe un chemin dans le graphe qui passe deux fois par le même sommet, alors il existe un chemin qui passe autant de fois que l'on veut par ce sommet.

Evident : si on a trouvé un chemin permettant de tourner en rond, on peut le reprendre.

Autrement dit :

soit $y \in \Sigma^*$

soit $p \in Q$ tel que $\delta^*(p, y) = p$

alors pour tout entier $i \geq 0$, $\delta^*(p, y^i) = p$

Attention

Ceci n'est vrai **que** s'il existe un cycle dans le graphe. Il est facile de voir qu'alors le langage L engendré est infini (*i.e.* $\#L$ n'est pas fini).

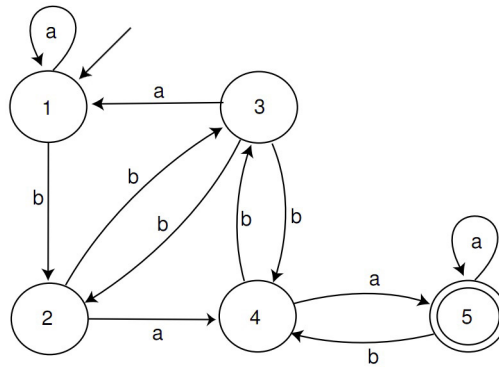
Mieux, si un langage régulier est infini, alors il y a un cycle dans son ADF.

En reprenant l'exemple précédent : $w = babba$

On a $y = bb$.

On peut vérifier que :

$$\begin{array}{ll}
 y^0 = \epsilon & \delta^*(4, \epsilon) = 4 \\
 y^1 = bb & \delta^*(4, bb) = 4 \\
 y^2 = bbbb & \delta^*(4, bbbb) = 4 \\
 y^3 = bbbbbb & \delta^*(4, bbbbbb) = 4
 \end{array}$$



Ces deux remarques conduisent au lemme de l'étoile.

Lemme de l'étoile (pumping lemma)

Soit un ADF $M = (Q, \Sigma, \delta, q_0, F)$. Si $\mathcal{L}(M)$ est infini, alors Pour tout mot w dans $\mathcal{L}(M)$ tel que $|w| \geq |Q|$. Alors, il existe une décomposition $w = xyz$ telle que :

1. $xy^iz \in \mathcal{L}(M)$ pour tout $i \geq 0$.
2. $|y| > 0$
3. $|xy| \leq |Q|$

Démonstration : voir les deux remarques précédentes. □

Notes :

- la longueur $|Q|$ est appelée la "pumping length".
- la condition est **nécessaire** mais pas **suffisante**.

Exemple :

En reprenant le mot $w = babba$. On peut le décomposer en 3 parties $x = ba$, $y = bb$, $z = a$ et vérifier que :

$$\{baa, babba, babbbba, babbbbbbba, \dots\} \subseteq \mathcal{L}(M)$$

Utilisation du lemme de l'étoile pour démontrer qu'un langage n'est pas régulier :

1. supposer que $M = (Q, \Sigma, \delta, q_0, F)$ tel que $L = \mathcal{L}(M)$.
2. démontrer qu'il existe un mot $w \in \mathcal{L}(M)$ tel que :
 - (a) $|w| \geq |Q|$
 - (b) **quelle que soit** la décomposition $w = xyz$, on peut toujours trouver un entier i tel que $xy^iz \notin L$.
3. **contradiction** : car L devrait vérifier le lemme de l'étoile.

On conclut que le langage n'est pas régulier.

Note : ce lemme signifie seulement que l'on commence à entrer dans un cycle qui peut se répéter (tout en restant dans le langage) AVANT d'avoir atteint la longueur de pompage. Il n'y pas de condition sur la régularité de z .

Observations préliminaires :

Si un langage L vérifie le lemme de l'étoile, et un mot quelconque $w \in L$ tq $|w| > |Q|$. w doit vérifier $w = xyz$ (avec $|xy| < |Q|$) et tout mot $w_n = xy^n z \in L$. La différence de longueur entre 2 mots consécutifs est :

$$|w_{n+1}| - |w_n| = |xy^{n+1}z| - |xy^n z| = |y| < |Q|$$

Cette longueur est donc indépendante de n .

Exemple 1 : Montrons que $L_1 = \{w \in 0^{n^2} | n \geq 1\}$ n'est pas un langage régulier.

Observons que $L_1 = \{0, 0^4, 0^9, 0^{16}, 0^{25}, 0^{36}, \dots\}$.

Notons $w_n = 0^{n^2}$, donc w_n est un mot de longueur n^2 .

La différence de longueur entre deux mots consécutifs est :

$$|w_{n+1}| - |w_n| = (n+1)^2 - n^2 = 2n + 1$$

Donc, si pour tout $m > n$, $|w_m| - |w_n| \geq 2n + 1$.

Trouver une décomposition xyz vérifiant $xy^n z \in L_1$ est donc impossible.

Ce langage ne peut donc pas être régulier.

Exemple 2 : Soit $L_2 = \{a^n | n \text{ est pair}\}$.

Montrons une mauvaise utilisation du lemme de l'étoile.

— essayons de montrer que L n'est pas régulier

soit $w = a^n$ et considérons la décomposition $w = xyz$ où $y = aaa$.

Alors $xy^2z = a^{n+3} \notin L_2$ car $n+3$ n'est pas pair si n est pair.

Donc, L_2 n'est pas régulier.

— **FAUX** : ce langage est régulier

Comme nous le démontrerons ci-après

— **Qu'est ce qui ne va pas ?**

Montrons donc que ce langage est régulier.

— **Preuve 1** : avec le lemme de l'étoile

Prendre $y = aa$, $x = z = \epsilon$, alors $xy^i w = a^{2i} \in L_2$.

En réalité, en prenant toutes les valeurs de i , on génère L_2 en entier, et on démontre ainsi que le lemme s'applique pour tous les mots de L_2 .

Donc, le langage est régulier.

STOP : le lemme de l'étoile est une condition **nécessaire**, mais **pas suffisante** de régularité.

\Rightarrow On ne peut pas utiliser le lemme de l'étoile pour démontrer qu'un langage est régulier.

— **Preuve 2** :

Rappel : tout langage qui peut s'écrire sous la forme d'une expression régulière est régulier.

Il peut s'écrire $L_2 = (aa)^*$, donc il est régulier.

Exemple 3 : $L_3 = \{0^n 1^n | n > 0\}$

Montrons par l'absurde que L_3 n'est pas un langage régulier.

Supposons que L_3 est régulier. Soit n la "pumping length".

Considérons la chaîne $s = 0^n 1^n$. Par le lemme de l'étoile, s peut s'écrire sous la forme $s = xyz$, et $xy^k z \in L_3$ pour tout $k > 0$.

— si y ne contient que des 0, alors $xy^k z$ contient trop de 0.

— si y ne contient que des 1, alors $xy^k z$ contient trop de 1.

— si y contient des 0 et des 1, alors $xy^k z$ n'appartient pas à L_3 .

exemple : si $y = 01$, $y^3 = 010101$ et non 000111

Donc, pour **toutes** les écritures de s comme xyz , **aucune** ne permet que $xy^k z \in L_3$.

$\Rightarrow L_3$ n'est pas régulier.

4 Algorithmique

Un automate fini est un modèle simple pour lequel il est possible de répondre à de nombreuses questions algorithmiques.

Cela en fait un modèle à privilégier, avant de passer à des modèles plus forts pour lesquels ces réponses seront potentiellement ni décidables, ni calculables.

Question soit un ANF N et une chaîne w .

Est-ce que $w \in \mathcal{L}(N)$?

Réponse construire l'ADF équivalent à N , et l'exécuter sur w (w étant fini, N s'arrête en un temps fini).

Question Est-ce que $\mathcal{L}(N) = \emptyset$?

Réponse Question d'atteignabilité dans un graphe : y-a-t-il un chemin allant de l'état de départ à un état acceptant ?

il existe des algorithmes simples et efficaces pour cela (propagation itérative).

Question Comment construire $\overline{\mathcal{L}(N)}$?

Réponse Dans N , remplacer l'ensemble des états acceptants F par \overline{F} .

\Rightarrow Fermeture des ADFs par l'opérateur complément.

Question Comment construire $\mathcal{L}(N_1) \cap \mathcal{L}(N_2)$?

Réponse Rappel d'algèbre : $A \cap B = \overline{\overline{A} \cup \overline{B}}$

Donc $\mathcal{L}(N_1) \cap \mathcal{L}(N_2) = \overline{\overline{\mathcal{L}(N_1)} \cup \overline{\mathcal{L}(N_2)}}$.

Note : construction est équivalente à celle vu pour modifier l'union d'ADF en intersection.

Question Est-ce que $\mathcal{L}(N) = \Sigma^*$?

Réponse Vérifier que $\overline{\mathcal{L}(N)} = \emptyset$.

Question Est-ce que $\mathcal{L}(N_1) \subseteq \mathcal{L}(N_2)$?

Réponse Vérifier que $\mathcal{L}(N_1) \cap \overline{\mathcal{L}(N_2)} = \emptyset$.

Question Est-ce que $\mathcal{L}(N_1) = \mathcal{L}(N_2)$?

Réponse Vérifier que $\mathcal{L}(N_1) \subseteq \mathcal{L}(N_2)$ et $\mathcal{L}(N_2) \subseteq \mathcal{L}(N_1)$.

Ces constructions sont importantes car elles peuvent servir de base pour répondre aux mêmes types de question sur des modèles plus forts.

5 Résumé

- Tout langage régulier est reconnu par une expression régulière ou un automate fini.
- Les expressions régulières, les automates finis (déterministes ou non-déterministes) reconnaissent tout langage régulier (équivalence des modèles).
- La classe des langages réguliers est fermée par les opérations régulières (*i.e.* la composition régulière de langages réguliers est un langage régulier).
- Le théorème de Myhill-Nérode permet de déterminer si un langage est régulier ou pas.
- Le lemme de l'étoile est une condition nécessaire (mais pas suffisante) permettant de vérifier qu'un langage n'est pas régulier.
- Différentes propriétés sur les langages réguliers peuvent être testés (vide ? égalité ? ...).
- Il existe des langages non réguliers (= reconnu par aucun ADF/ANF/ER).