

I5

Langages formels, automates et grammaires formelles

Nicolas Prcovic

04.91.28.89.51

`nicolas.prcovic@univ-cezanne.mrs`

Contenu général du cours

Etude des langages formels selon deux manières de les modéliser :

- les machines abstraites (automates et machines de Turing)
- les grammaires formelles.

Qu'est-ce qu'un langage formel ?

- Un langage formel se définit comme un ensemble de suites de symboles.

Qu'est-ce qu'un langage formel ?

- Un langage formel se définit comme un ensemble de suites de symboles.
- Exemples:
 - {cœur, pique, carreau, trèfle}

Qu'est-ce qu'un langage formel ?

- Un langage formel se définit comme un ensemble de suites de symboles.
- Exemples:
 - {cœur, pique, carreau, trèfle}
 - {0, 1, 10, 11, 100, 101, ...}

Qu'est-ce qu'un langage formel ?

- Un langage formel se définit comme un ensemble de suites de symboles.
- Exemples:
 - {cœur, pique, carreau, trèfle}
 - {0, 1, 10, 11, 100, 101, ...}
 - {0, 2, 4, 6, 8, 10, ...}

Qu'est-ce qu'un langage formel ?

- Un langage formel se définit comme un ensemble de suites de symboles.
- Exemples:
 - {cœur, pique, carreau, trèfle}
 - {0, 1, 10, 11, 100, 101, ...}
 - {0, 2, 4, 6, 8, 10, ...}
 - {2, 3, 5, 7, 11, 13, 17, ...}

Qu'est-ce qu'un langage formel ?

- Un langage formel se définit comme un ensemble de suites de symboles.
- Exemples:
 - {cœur, pique, carreau, trèfle}
 - {0, 1, 10, 11, 100, 101, ...}
 - {0, 2, 4, 6, 8, 10, ...}
 - {2, 3, 5, 7, 11, 13, 17, ...}
 - Ensemble des textes de programme C syntaxiquement correct

Lien entre pb informatique et lang. formel

- Chaque suite de symbole appartenant au langage représente un "objet".
- Lien : solution du problème $\iff$ élément du langage.

Lien entre pb informatique et lang. formel

- Chaque suite de symbole appartenant au langage représente un "objet".
- Lien : solution du problème $\iff$ élément du langage.
- Exemples:
 - Quelles sont les couleurs existant dans un jeu de cartes ?

Lien entre pb informatique et lang. formel

- Chaque suite de symbole appartenant au langage représente un "objet".
- Lien : solution du problème $\iff$ élément du langage.
- Exemples:
 - Quelles sont les couleurs existant dans un jeu de cartes ?
 - Est-ce que *cette suite de chiffres* représente *un nombre pair* ?

Lien entre pb informatique et lang. formel

- Chaque suite de symbole appartenant au langage représente un "objet".
- Lien : solution du problème $\iff$ élément du langage.
- Exemples:
 - Quelles sont les couleurs existant dans un jeu de cartes ?
 - Est-ce que *cette suite de chiffres* représente *un nombre pair* ?
 - Est-ce que *cette suite de chiffres* représente *un nombre premier* ?

Lien entre pb informatique et lang. formel

- Chaque suite de symbole appartenant au langage représente un "objet".
- Lien : solution du problème $\iff$ élément du langage.
- Exemples:
 - Quelles sont les couleurs existant dans un jeu de cartes ?
 - Est-ce que *cette suite de chiffres* représente *un nombre pair* ?
 - Est-ce que *cette suite de chiffres* représente *un nombre premier* ?
 - Est ce que *cette suite de caractères* correspond à *un programme C syntaxiquement correct* ?

Historique

- Machine de Turing (années 30) pour modéliser une machine qui résout le plus de problèmes possibles.

Historique

- Machine de Turing (années 30) pour modéliser une machine qui résout le plus de problèmes possibles.
- Automates (années 40 et 50) pour modéliser certaines fonctionnalités du cerveau.

Historique

- Machine de Turing (années 30) pour modéliser une machine qui résout le plus de problèmes possibles.
- Automates (années 40 et 50) pour modéliser certaines fonctionnalités du cerveau.
- Grammaires formelles (fin des années 50, Chomsky) pour modéliser les langues naturelles. Actuellement : pour la compilation.

Equivalences entre machines abstraites et grammaires formelles

Langage formel	Machine abstraite	Grammaire
récurziv. énum.	Machines de Turing	non restreintes
contextuels	MdT bornées linéair.	contextuelles
hors contexte	Automates à pile	hors contexte
rationnels	Automates	régulières

Les langages formels

Les langages formels représentent uniquement l'**aspect syntaxique** des langages et **non la sémantique** qu'on pourrait leur associer (comme on le ferait avec les langues naturelles).

Les langages formels

- **Définition Alphabet**

Un alphabet Σ est un ensemble fini, non vide, de symboles.

Les langages formels

- **Définition Alphabet**

Un alphabet Σ est un ensemble fini, non vide, de symboles.

- **Ex :**

- l'ensemble $\{0, 1\}$ des 2 chiffres binaires,
- l'ensemble des 10 chiffres arabes,
- l'ensemble des 26 lettres de l'alphabet français,
- l'ensemble des caractères ASCII, etc.

Les langages formels

- **Définition** Mot (ou chaîne)

Un mot w est une suite de symboles d'un alphabet donné. La longueur d'un mot w , noté $|w|$ est le nombre de symboles qu'il contient. Le *mot vide*, noté ϵ , est le mot de longueur nulle.

Les langages formels

- **Définition Mot (ou chaîne)**
Un mot w est une suite de symboles d'un alphabet donné. La longueur d'un mot w , noté $|w|$ est le nombre de symboles qu'il contient. Le *mot vide*, noté ϵ , est le mot de longueur nulle.
- Ex : 10110 est un mot créé à partir de l'alphabet $\{0, 1\}$ et a pour longueur 5, tous les fichiers ASCII sont des mots créés à partir des caractères ASCII.

Les langages formels

- **Définition Sous-mot**

Un sous-mot v est une sous-suite de la suite des symboles d'un mot w .

Les langages formels

- **Définition Sous-mot**

Un sous-mot v est une sous-suite de la suite des symboles d'un mot w .

- Ex : si $w = abcd$ alors abd , cd , ac , b , $abcd$ et ϵ sont des exemples de sous-mots de w .

Les langages formels

- **Définition** Concaténation de mots

La concaténation de 2 mots w et w' est le mot égal à la suite des symboles de w suivi de la suite des symboles de w' . Elle se note $w.w'$ ou ww' .

Les langages formels

- **Définition** Concaténation de mots

La concaténation de 2 mots w et w' est le mot égal à la suite des symboles de w suivi de la suite des symboles de w' . Elle se note $w.w'$ ou ww' .

- Ex : La concaténation de $w = abc$ et de $w' = fg$ est $w.w' = abcfg$.

Les langages formels

- **Définition** Facteur d'un mot

Le facteur v d'un mot w est un mot tel qu'il existe des mots u et u' tels que $w = u.v.u'$.

Un facteur gauche (ou préfixe) est un facteur tel que $u = \epsilon$.

Un facteur droit (ou suffixe) est un facteur tel que $u' = \epsilon$.

Une occurrence d'un facteur est l'apparition de ce facteur à une position précise dans le mot.

Les langages formels

- **Définition** Facteur d'un mot

Le facteur v d'un mot w est un mot tel qu'il existe des mots u et u' tels que $w = u.v.u'$.

Un facteur gauche (ou préfixe) est un facteur tel que $u = \epsilon$.

Un facteur droit (ou suffixe) est un facteur tel que $u' = \epsilon$.

Une occurrence d'un facteur est l'apparition de ce facteur à une position précise dans le mot.

- **Remarque** : un facteur est un sous-mot mais l'inverse n'est pas vrai en général.

Les langages formels

- **Définition** Produit de 2 alphabets

On note Σ^1 l'ensemble des mots de longueur 1 constitués à partir de chacun des symboles de Σ .
Le produit de 2 alphabets Σ et Σ' est l'ensemble de mots défini par $\Sigma.\Sigma' = \{ w.w' \mid w \in \Sigma^1, w' \in \Sigma'^1 \}$.

Les langages formels

- **Définition** Produit de 2 alphabets

On note Σ^1 l'ensemble des mots de longueur 1 constitués à partir de chacun des symboles de Σ .

Le produit de 2 alphabets Σ et Σ' est l'ensemble de mots défini par $\Sigma.\Sigma' = \{ w.w' \mid w \in \Sigma^1, w' \in \Sigma'^1 \}$.

- Ex : si $\Sigma = \{a, b\}$ et $\Sigma' = \{c, d\}$ alors $\Sigma.\Sigma' = \{ac, ad, bc, bd\}$.

Les langages formels

● Définition Puissance d'un alphabet

La k^{eme} puissance d'un alphabet Σ , notée Σ^k , se définit par : $\Sigma^0 = \{\epsilon\}$ et $\forall k > 0, \Sigma^k = \Sigma^{k-1}.\Sigma^1$. C'est l'ensemble de tous les mots de longueur k que l'on peut former à partir de Σ .

Les langages formels

- **Définition** Puissance d'un alphabet

La k^{eme} puissance d'un alphabet Σ , notée Σ^k , se définit par : $\Sigma^0 = \{\epsilon\}$ et $\forall k > 0, \Sigma^k = \Sigma^{k-1}.\Sigma$. C'est l'ensemble de tous les mots de longueur k que l'on peut former à partir de Σ .

- **Ex :** si $\Sigma = \{a, b\}$ alors $\Sigma^0 = \{\epsilon\}$, $\Sigma^1 = \{a, b\}$, $\Sigma^2 = \{aa, ab, ba, bb\}$, etc.

Les langages formels

- **Définition** Puissance d'un alphabet

La k^{eme} puissance d'un alphabet Σ , notée Σ^k , se définit par : $\Sigma^0 = \{\epsilon\}$ et $\forall k > 0, \Sigma^k = \Sigma^{k-1}.\Sigma^1$. C'est l'ensemble de tous les mots de longueur k que l'on peut former à partir de Σ .

- **Ex :** si $\Sigma = \{a, b\}$ alors $\Sigma^0 = \{\epsilon\}$, $\Sigma^1 = \{a, b\}$, $\Sigma^2 = \{aa, ab, ba, bb\}$, etc.

- On appelle étoile d'un alphabet $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$. C'est l'ensemble de tous les mots qu'on peut former à partir de Σ . On note Σ^+ le même ensemble mais privé du mot vide : $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$.

Les langages formels

- **Définition Langage**

Un langage L se définit à partir d'un alphabet Σ comme étant un sous-ensemble de Σ^* .

Les langages formels

- **Définition Langage**

Un langage L se définit à partir d'un alphabet Σ comme étant un sous-ensemble de Σ^* .

- Tout sous-ensemble de Σ^* est potentiellement un langage, y compris le langage vide $\emptyset$ (ne contenant aucun mot), le langage $\{\epsilon\}$ (contenant uniquement le mot vide) et le langage égal à Σ^* .

Les langages formels

- **Définition Langage**

Un langage L se définit à partir d'un alphabet Σ comme étant un sous-ensemble de Σ^* .

- Tout sous-ensemble de Σ^* est potentiellement un langage, y compris le langage vide $\emptyset$ (ne contenant aucun mot), le langage $\{\epsilon\}$ (contenant uniquement le mot vide) et le langage égal à Σ^* .
- Ex : le langage des chiffres binaires pairs (c-a-d se terminant par 0), le langage des nombres entiers décimaux inférieurs à 100, le langage des mots français du Larousse, etc.

Les langages formels

- En général, on définira un langage en utilisant une notation ensembliste de la forme $\{w \mid \text{propriété sur } w\}$, qui signifie "l'ensemble des mots w (de Σ^*) tels qu'ils respectent une certaine propriété".
- Ex : $\{w \mid w \text{ se termine par } a\}$, $\{w \mid 2 \leq |w| \leq 5\}$.

Les langages formels

- En général, on définira un langage en utilisant une notation ensembliste de la forme $\{w \mid \text{propriété sur } w\}$, qui signifie "l'ensemble des mots w (de Σ^*) tels qu'ils respectent une certaine propriété".
- Ex : $\{w \mid w \text{ se termine par } a\}$, $\{w \mid 2 \leq |w| \leq 5\}$.
- Remarque : l'ensemble des parties de Σ^* , noté $P(\Sigma^*)$ ou 2^{Σ^*} , est l'ensemble de tous les langages sur Σ .

Les langages formels

Définition Produit de langages

Le produit (de concaténation) de 2 langages L et L' est défini par : $L.L' = \{w.w' \mid w \in L, w' \in L'\}$.

Les langages formels

- **Définition** Puissance d'un langage

La k^{eme} puissance d'un langage L , notée L^k , se définit par : $L^0 = \{\epsilon\}$ et $\forall k > 0, L^k = L^{k-1}.L$.

Les langages formels

- **Définition** Puissance d'un langage

La k^{eme} puissance d'un langage L , notée L^k , se définit par : $L^0 = \{\epsilon\}$ et $\forall k > 0, L^k = L^{k-1}.L$.

- **Ex :** si $L = \{ab, cd\}$ alors $L^0 = \{\epsilon\}$, $L^1 = \{ab, cd\}$, $L^2 = \{abab, abcd, cdab, cdcd\}$, etc.

Les langages formels

- **Définition** Puissance d'un langage

La k^{eme} puissance d'un langage L , notée L^k , se définit par : $L^0 = \{\epsilon\}$ et $\forall k > 0, L^k = L^{k-1}.L$.

- Ex : si $L = \{ab, cd\}$ alors $L^0 = \{\epsilon\}$, $L^1 = \{ab, cd\}$, $L^2 = \{abab, abcd, cdab, cdcd\}$, etc.

- On appelle étoile d'un langage, le langage $L^* = L^0 \cup L^1 \cup L^2 \cup \dots$. Il contient tous les mots qu'on peut obtenir en concaténant un nombre quelconque de mots de L .

- On appelle étoile propre de L et on note L^+ le même langage mais privé du mot vide : $L^+ = L^* / \{\epsilon\}$.

Les langages formels

- **Définition** Puissance d'un langage

La k^{eme} puissance d'un langage L , notée L^k , se définit par : $L^0 = \{\epsilon\}$ et $\forall k > 0, L^k = L^{k-1}.L$.

- Ex : si $L = \{ab, cd\}$ alors $L^0 = \{\epsilon\}$, $L^1 = \{ab, cd\}$, $L^2 = \{abab, abcd, cdab, cdcd\}$, etc.

- On appelle étoile d'un langage, le langage $L^* = L^0 \cup L^1 \cup L^2 \cup \dots$. Il contient tous les mots qu'on peut obtenir en concaténant un nombre quelconque de mots de L .

- On appelle étoile propre de L et on note L^+ le même langage mais privé du mot vide : $L^+ = L^* / \{\epsilon\}$.

- Remarque : l'étoile d'un langage fini est un langage infini dénombrable.

Automates d'états finis

- Automate : machine abstraite qui modélise un système discret dont l'état va varier au cours du temps à cause d'influences externes.

Automates d'états finis

- Automate : machine abstraite qui modélise un système discret dont l'état va varier au cours du temps à cause d'influences externes.
- On peut représenter un automate par un *graphe* dont chaque sommet est un état et chaque arc, étiqueté par un nom d'action, est une transition possible d'un état vers un autre.

Automates d'états finis

- Automate : machine abstraite qui modélise un système discret dont l'état va varier au cours du temps à cause d'influences externes.
- On peut représenter un automate par un *graphe* dont chaque sommet est un état et chaque arc, étiqueté par un nom d'action, est une transition possible d'un état vers un autre.
- Un automate possède un état initial et un nombre quelconque d'états finaux (désirés pour le système).

Automates d'états finis

- Automate : machine abstraite qui modélise un système discret dont l'état va varier au cours du temps à cause d'influences externes.
- On peut représenter un automate par un *graphe* dont chaque sommet est un état et chaque arc, étiqueté par un nom d'action, est une transition possible d'un état vers un autre.
- Un automate possède un état initial et un nombre quelconque d'états finaux (désirés pour le système).
- L'automate vérifie si la suite d'actions permet de passer de l'état initial à un état final.

Exemple

- On veut modéliser un système simple composé d'un coffre-fort et d'un document top secret.
- On peut agir sur le système en ouvrant ou en fermant le coffre ou en plaçant ou en retirant le document du coffre.
- Le système peut avoir 4 états : (0) coffre fermé et vide, (1) coffre ouvert et vide, (2) coffre ouvert et plein et (3) coffre fermé et plein.
- Etat final (désiré) : mettre en sûreté notre document (état 4). Etat initial : état 0.
- Problème (de décision) : est-ce que [cette suite d'actions] permet de mettre le document en sécurité ?

Automates d'états finis déterministes

Définition Automate d'états finis déterministe

Un automate d'états finis déterministe (AEFD) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où :

Automates d'états finis déterministes

Définition Automate d'états finis déterministe

Un automate d'états finis déterministe (AEFD) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'états.

Automates d'états finis déterministes

Définition Automate d'états finis déterministe

Un automate d'états finis déterministe (AEFD) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'états.
- Σ est un alphabet contenant des symboles d'entrée.

Automates d'états finis déterministes

Définition Automate d'états finis déterministe

Un automate d'états finis déterministe (AEFD) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'états.
- Σ est un alphabet contenant des symboles d'entrée.
- $\delta : Q \times \Sigma \rightarrow Q$ est la fonction de transition telle que $\delta(q_i, s) = q_j$ ssi dans l'état q_i la lecture du symbole s fait passer dans l'état q_j .

Automates d'états finis déterministes

Définition Automate d'états finis déterministe

Un automate d'états finis déterministe (AEFD) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'états.
- Σ est un alphabet contenant des symboles d'entrée.
- $\delta : Q \times \Sigma \rightarrow Q$ est la fonction de transition telle que $\delta(q_i, s) = q_j$ ssi dans l'état q_i la lecture du symbole s fait passer dans l'état q_j .
- q_0 est l'état initial.

Automates d'états finis déterministes

Définition Automate d'états finis déterministe

Un automate d'états finis déterministe (AEFD) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'états.
- Σ est un alphabet contenant des symboles d'entrée.
- $\delta : Q \times \Sigma \rightarrow Q$ est la fonction de transition telle que $\delta(q_i, s) = q_j$ ssi dans l'état q_i la lecture du symbole s fait passer dans l'état q_j .
- q_0 est l'état initial.
- $F \subseteq Q$ est l'ensemble des états finaux.

Représentations d'un automate

On peut représenter un automate par une *table de transition*. Cette table contient :

- en ligne les états possibles de l'automate.
- en colonne les symboles de Σ .
- L'état initial est préfixé par une flèche.
- Les états finaux sont préfixés par une astérisque.
- Chaque case du tableau contient l'état résultant de la lecture du symbole en colonne à partir de l'état en ligne.

Représentations d'un automate

On peut représenter un automate par une *table de transition*. Cette table contient :

- en ligne les états possibles de l'automate.
- en colonne les symboles de Σ .
- L'état initial est préfixé par une flèche.
- Les états finaux sont préfixés par une astérisque.
- Chaque case du tableau contient l'état résultant de la lecture du symbole en colonne à partir de l'état en ligne.

	o	f	p	r
$\rightarrow q_0$	q_1			
q_1		q_0		q_2
q_2		q_3	q_1	
$*q_3$	q_2			

	a	b
$\rightarrow *q_0$	q_1	q_1
q_1	q_2	q_1
$*q_2$		q_0

Représentations d'un automate

On peut aussi représenter un AEFD par un *diagramme de transition*. Il s'agit d'un graphe dont :

- les sommets sont les états de Q ,
- les arcs étiquetés par des symboles de Σ sont les transitions d'un état à un autre suite à la lecture d'un symbole.
- l'état initial est signalé par une flèche entrante.
- les états finaux sont cerclés.

AEFD

Définition Extension aux mots de la fonction de transition
La fonction de transition peut être étendue aux mots. Son extension $\hat{\delta}$ est définie sur $Q \times \Sigma^* \rightarrow Q$ avec :

- $\hat{\delta}(q_i, \epsilon) = q_i$.
- $\forall w \in \Sigma^*, \forall c \in \Sigma: \hat{\delta}(q_i, w.c) = \delta(\hat{\delta}(q_i, w), c)$.

AEFD

Définition Extension aux mots de la fonction de transition
La fonction de transition peut être étendue aux mots. Son extension $\hat{\delta}$ est définie sur $Q \times \Sigma^* \rightarrow Q$ avec :

- $\hat{\delta}(q_i, \epsilon) = q_i.$

- $\forall w \in \Sigma^*, \forall c \in \Sigma: \hat{\delta}(q_i, w.c) = \delta(\hat{\delta}(q_i, w), c).$

$\hat{\delta}(q_i, w) = q_j$ signifie que la lecture du mot w à partir de l'état q_i mène à l'état q_j .

AEFD

Définition Mot accepté par un automate

Un mot $w \in \Sigma^*$ est dit *accepté* par un automate $(Q, \Sigma, \delta, q_0, F)$ si $\hat{\delta}(q_0, w) \in F$.

AEFD

Définition Mot accepté par un automate

Un mot $w \in \Sigma^*$ est dit *accepté* par un automate $(Q, \Sigma, \delta, q_0, F)$ si $\hat{\delta}(q_0, w) \in F$.

En d'autres termes, un mot est accepté par un automate si la lecture des symboles de w à partir de q_0 mène à un état final de l'automate.

AEFD

Définition Langage reconnu par un automate

Un langage *reconnu* par un automate A est défini par :

$$L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}.$$

AEFD

Définition Langage reconnu par un automate

Un langage *reconnu* par un automate A est défini par :

$$L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}.$$

En d'autres termes, le langage reconnu par un automate est l'ensemble des mots qu'il accepte.

AEFD

Définition Langage reconnaissable

Un langage est dit *reconnaissable* s'il existe un AEFD le reconnaissant.

AEFD

Définition Langage reconnaissable

Un langage est dit *reconnaissable* s'il existe un AEFD le reconnaissant.

Déterminisme :

Un automate d'états finis est dit déterministe si, étant donné un mot qu'on lui donne à lire, on est toujours sûr que l'automate transite toujours par la même séquence d'états.

Automates d'états finis non déterministes

- Contrairement aux AEFD, les automates d'états finis non déterministes (AEFND) peuvent se retrouver dans **plusieurs états en même temps** après avoir lu un symbole à partir d'un état donné.

Automates d'états finis non déterministes

- Contrairement aux AEFD, les automates d'états finis non déterministes (AEFND) peuvent se retrouver dans **plusieurs états en même temps** après avoir lu un symbole à partir d'un état donné.
- **Définition AEFND**
Un automate d'états finis non déterministe est un quintuplet $(Q, \Sigma, q_0, \delta, F)$ où Q , Σ , q_0 et F ont la même signification que pour un AEFD mais où δ est défini par :
$$\delta : Q \times \Sigma \rightarrow 2^Q$$

$$\delta(q_i, c) = Q_j \text{ avec } Q_j \subseteq Q.$$

Automates d'états finis non déterministes

- Contrairement aux AEFD, les automates d'états finis non déterministes (AEFND) peuvent se retrouver dans **plusieurs états en même temps** après avoir lu un symbole à partir d'un état donné.
- **Définition AEFND**
Un automate d'états finis non déterministe est un quintuplet $(Q, \Sigma, q_0, \delta, F)$ où Q , Σ , q_0 et F ont la même signification que pour un AEFD mais où δ est défini par :
$$\delta : Q \times \Sigma \rightarrow 2^Q$$
$$\delta(q_i, c) = Q_j \text{ avec } Q_j \subseteq Q.$$
- Ainsi, après avoir lu une **suite de symboles**, un AEFND peut se trouver au même moment dans **plusieurs états différents**.

AEFND

- **Définition** Extension aux mots de la fonction de transition
La fonction de transition peut être étendue aux mots.

Son extension $\hat{\delta}$ est définie sur $Q \times \Sigma^* \rightarrow 2^Q$ avec :

- $\hat{\delta}(q_i, \epsilon) = \{q_i\}$.
- $\forall w \in \Sigma^*, \forall c \in \Sigma: \hat{\delta}(q_i, w.c) = \bigcup_{q_j \in \hat{\delta}(q_i, w)} \delta(q_j, c).$

AEFND

- **Définition** Extension aux mots de la fonction de transition
La fonction de transition peut être étendue aux mots.
Son extension $\hat{\delta}$ est définie sur $Q \times \Sigma^* \rightarrow 2^Q$ avec :
 - $\hat{\delta}(q_i, \epsilon) = \{q_i\}$.
 - $\forall w \in \Sigma^*, \forall c \in \Sigma: \hat{\delta}(q_i, w.c) = \bigcup_{q_j \in \hat{\delta}(q_i, w)} \delta(q_j, c)$.
- $\hat{\delta}(q_i, w) = Q_j$ signifie que la lecture du mot w à partir de l'état q_i mène à tous les états de Q_j en même temps.

AEFND

- **Définition** Mot accepté par un AEFND
Un mot $w \in \Sigma^*$ est dit *accepté* par un AEFND $(Q, \Sigma, \delta, q_0, F)$ si $\hat{\delta}(q_0, w) \cap F \neq \emptyset$.

AEFND

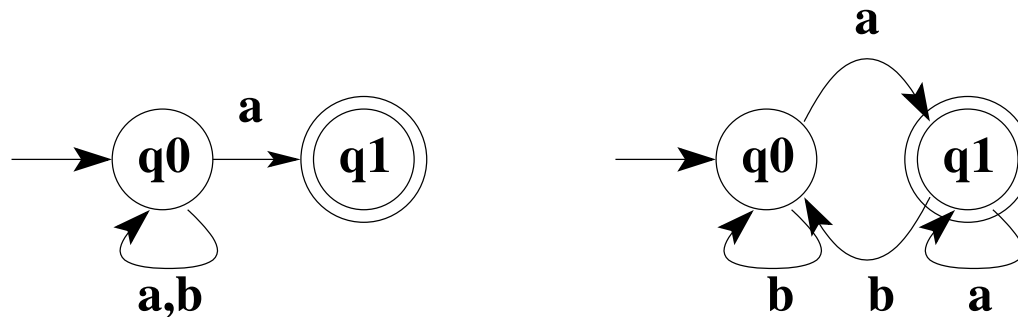
- **Définition** Mot accepté par un AEFND
Un mot $w \in \Sigma^*$ est dit *accepté* par un AEFND $(Q, \Sigma, \delta, q_0, F)$ si $\hat{\delta}(q_0, w) \cap F \neq \emptyset$.
- En d'autres termes, un mot est accepté par un automate si la lecture des symboles de w à partir de q_0 mène à un ensemble d'états dont **au moins un est final**.

AEFND

- **Définition** Mot accepté par un AEFND
Un mot $w \in \Sigma^*$ est dit *accepté* par un AEFND $(Q, \Sigma, \delta, q_0, F)$ si $\hat{\delta}(q_0, w) \cap F \neq \emptyset$.
- En d'autres termes, un mot est accepté par un automate si la lecture des symboles de w à partir de q_0 mène à un ensemble d'états dont **au moins un est final**.
- **Définition** Langage reconnu par un AEFND
Un langage *reconnu* par un AEFND A est défini par :
$$L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}.$$

Avantage d'un AEFND sur un AEFD

L'intérêt des AEFND est qu'ils permettent plus facilement de définir un automate reconnaissant un langage donné.



Le langage $\{w.a \mid w \in \Sigma^*\}$ reconnu par un AEFND (à gauche) et un AEFD (à droite).

Equivalence entre AEFD et AEFND

- Bien que les AEFND aient l'air plus puissant que les AEFD en terme de possibilité de reconnaître un langage, il s'avère que **leur puissance leur est égale.**

Equivalence entre AEFD et AEFND

- Bien que les AEFND aient l'air plus puissant que les AEFD en terme de possibilité de reconnaître un langage, il s'avère que **leur puissance leur est égale**.
- Il est évident que tout AEFD est aussi un AEFND.

Equivalence entre AEFD et AEFND

- Bien que les AEFND aient l'air plus puissant que les AEFD en terme de possibilité de reconnaître un langage, il s'avère que **leur puissance leur est égale**.
- Il est évident que tout AEFD est aussi un AEFND.
- **Théoreme**
Tout langage reconnu par un AEFND peut aussi être reconnu par un AEFD.

Preuve

Preuve constructive : on va démontrer que si $A_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ est un AEFND alors le langage qu'il reconnaît est aussi reconnu par l'AEFD

$A_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ où :

- $Q_D = 2^{Q_N}$ est l'ensemble des sous-ensembles de Q_N .
- F_D est l'ensemble des états de Q_D qui contiennent un état de F_N , c-a-d les états dont l'intersection avec F_N est non vide.
- $\forall S \subseteq Q_N$ et $\forall c \in \Sigma : \delta_D(S, c) = \bigcup_{s \in S} \delta_N(s, c)$.

Preuve

Démontrons d'abord que $\forall w, \hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$.
Preuve par récurrence sur $|w|$:

- Pour $|w| = 0$, c-a-d $w = \epsilon$, $\hat{\delta}_D(\{q_0\}, \epsilon) = \hat{\delta}_N(q_0, \epsilon) = \{q_0\}$.
- Posons que pour tout mot w de longueur n , on a
 $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w) = \{p_1, p_2, \dots, p_k\}$.
 $\forall c \in \Sigma, \hat{\delta}_D(\{q_0\}, w.c) = \delta_D(\hat{\delta}_D(\{q_0\}, w), c) = \delta_D(\{p_1, p_2, \dots, p_k\}, c) = \bigcup_{1 \leq i \leq k} \delta_N(p_i, c) = \hat{\delta}_N(q_0, w.c)$.

On a alors $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$ qui est aussi vérifiée pour tous les mots de longueur $n+1$.

Preuve

On a donc $L(A_D)$

$$= \{w \in \Sigma^* \mid \hat{\delta}_D(\{q_0\}, w) \in F_D\}$$

$$= \{w \in \Sigma^* \mid \hat{\delta}_D(\{q_0\}, w) \cap F_N \neq \emptyset\}$$

$$= \{w \in \Sigma^* \mid \hat{\delta}_N(q_0, w) \cap F_N \neq \emptyset\}$$

$$= L(A_N).$$



Construction d'un AEFD à partir d'un AEF

- En pratique, quand c'est possible, on ne générera pas 2^n états d'un AEFD à partir des n états d'un AEFND : on se contentera de générer tous les états accessibles depuis $\{q_0\}$.
- La procédure de génération de tous ces états consiste à maintenir l'ensemble Q des états qu'on a généré mais dont on a pas encore généré les successeurs.
 - Au début, $Q = \{q_0\}$.
 - Tant que Q n'est pas vide, on retire un état q de Q et on ajoute à Q tous les états directement accessibles depuis q et qui n'ont pas déjà été générés précédemment.

AEFND avec ϵ -transitions

- On peut étendre le modèle de l'AEFND en lui permettant des ϵ -transitions, c'est-à-dire des **transitions sans lecture de symbole**.
- L'intérêt est, une fois encore, de faciliter la construction d'automates reconnaissant un langage donné.

AEFND avec ϵ -transitions

Définition AEFND avec ϵ -transitions

Un AEFND avec ϵ -transitions est un quintuplet $(Q, \Sigma, q_0, \delta, F)$ où Q, Σ, q_0, δ et F ont la même signification que pour un AEFND mais où δ est définie sur $Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$.

ϵ -fermeture

- **Définition** ϵ -fermeture

$\forall q \in Q$, ϵ -fermeture(q) est un ensemble d'états tel que :

- $q \in \epsilon$ -fermeture(q).
- si $r \in \epsilon$ -fermeture(q) alors $\{r' \in Q \mid r' = \delta(r, \epsilon)\} \subseteq \epsilon$ -fermeture(q).

ϵ -fermeture

- **Définition** ϵ -fermeture

$\forall q \in Q$, ϵ -fermeture(q) est un ensemble d'états tel que :

- $q \in \epsilon$ -fermeture(q).
- si $r \in \epsilon$ -fermeture(q) alors $\{r' \in Q \mid r' = \delta(r, \epsilon)\} \subseteq \epsilon$ -fermeture(q).

- L' ϵ -fermeture se définit aussi sur un ensemble d'états :

$$\forall S \subseteq Q, \epsilon\text{-Fermeture}(S) = \bigcup_{s \in S} \epsilon\text{-fermeture}(s)$$

ϵ -fermeture

- **Définition** ϵ -fermeture

$\forall q \in Q$, ϵ -fermeture(q) est un ensemble d'états tel que :

- $q \in \epsilon$ -fermeture(q).
- si $r \in \epsilon$ -fermeture(q) alors $\{r' \in Q \mid r' = \delta(r, \epsilon)\} \subseteq \epsilon$ -fermeture(q).

- L' ϵ -fermeture se définit aussi sur un ensemble d'états :

$$\forall S \subseteq Q, \epsilon\text{-Fermeture}(S) = \bigcup_{s \in S} \epsilon\text{-fermeture}(s)$$

- En d'autres termes, ϵ -fermeture(q) est l'ensemble de tous états accessibles depuis q en suivant une suite d' ϵ -transitions.

AEFND avec ϵ -transitions

Définition Extension aux mots de la fonction de transition
L'extension aux mots $\hat{\delta}$ de la fonction de transition est définie sur $Q \times \Sigma^* \rightarrow 2^Q$ avec :

- $\hat{\delta}(q_i, \epsilon) = \epsilon\text{-fermeture}(q_i)$.
- $\forall w \in \Sigma^*, \forall c \in \Sigma: \hat{\delta}(q_i, w.c) = \bigcup_{q_j \in \hat{\delta}(q_i, w)} \epsilon\text{-fermeture}(\delta(q_j, c))$.

Equivalence entre AEFD et ϵ -AEFND

- Bien que les ϵ -AEFND aient l'air plus puissants que les AEFD en terme de possibilité de reconnaître un langage, il s'avère que leur puissance leur est égale.

Equivalence entre AEFD et ϵ -AEFND

- Bien que les ϵ -AEFND aient l'air plus puissants que les AEFD en terme de possibilité de reconnaître un langage, il s'avère que leur puissance leur est égale.
- Tout AEFD est aussi un ϵ -AEFND.

Equivalence entre AEFD et ϵ -AEFND

- Bien que les ϵ -AEFND aient l'air plus puissants que les AEFD en terme de possibilité de reconnaître un langage, il s'avère que leur puissance leur est égale.
- Tout AEFD est aussi un ϵ -AEFND.
- **Théorème**
Tout langage reconnu par un ϵ -AEFND peut aussi être reconnu par un AEFD.

Preuve

Preuve constructive : on va démontrer que si $A_E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ est un ϵ -AEFND alors le langage qu'il reconnaît est aussi reconnu par l'AEFD $A_D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ où :

- $Q_D = 2^{Q_E}$.
- $q_D = \epsilon$ -fermeture(q_0).
- F_D est l'ensemble des états de Q_D qui contiennent un état de F_E , c-a-d les états dont l'intersection avec F_N est non vide.
- $\forall S \subseteq Q_N$ et $\forall c \in \Sigma : \delta_D(S, c) = \epsilon$ -Fermeture($\bigcup_{s \in S} \delta_E(s, c)$).

Preuve

Démontrons d'abord que $\forall w, \hat{\delta}_D(q_D, w) = \hat{\delta}_E(q_0, w)$.
Preuve par récurrence sur $|w|$:

- Pour $|w| = 0$, c-a-d $w = \epsilon$, $\hat{\delta}_D(q_D, \epsilon) = \hat{\delta}_E(q_0, \epsilon) = \epsilon\text{-fermeture}(q_0)$.
- Posons que pour tout mot w de longueur n , on a $\hat{\delta}_D(q_D, w) = \hat{\delta}_E(q_0, w) = \{p_1, p_2, \dots, p_k\}$.
 $\forall c \in \Sigma, \hat{\delta}_D(q_D, w.c) = \delta_D(\hat{\delta}_D(q_D, w), c)$
 $= \delta_D(\{p_1, p_2, \dots, p_k\}, c)$
 $= \epsilon\text{-Fermeture}(\bigcup_{1 \leq i \leq k} \delta_E(p_i, c))$
 $= \bigcup_{1 \leq i \leq k} \epsilon\text{-fermeture}(\delta_E(p_i, c)) = \hat{\delta}_E(q_0, w.c).$
- On a alors $\hat{\delta}_D(q_D, w) = \hat{\delta}_E(q_0, w)$ qui est aussi vérifiée pour tous les mots de longueur $n+1$.

Preuve

On a donc $L(A_D)$

$= \{w \in \Sigma^* \mid \hat{\delta}_D(q_D, w) \in F_D\}$ (définition d'un AEFD)

$= \{w \in \Sigma^* \mid \hat{\delta}_D(q_D, w) \cap F_E \neq \emptyset\}$ (définition de A_E)

$= \{w \in \Sigma^* \mid \hat{\delta}_E(q_0, w) \cap F_E \neq \emptyset\}$ (démontré juste avant)

$= L(A_E)$ (définition d'un ϵ -AEFND)

□

Les langages réguliers/rationnels

Le but de ce chapitre est de présenter

- les *langages réguliers (ou rationnels)*,
- les *expressions régulières*, qui sont leur représentation,
- de montrer qu'ils constituent l'ensemble des langages reconnaissables (par des AEFD)

Les langages rationnels

Définition Langages rationnels

Etant donné un alphabet Σ , les langages rationnels sont des langages définis récursivement par :

- $\emptyset$, $\{\epsilon\}$ et $\forall c \in \Sigma^1$, $\{c\}$ sont des langages rationnels.
- Si L et L' sont des langages rationnels alors $L \cup L'$, $L.L'$ et L^* sont des langages rationnels.

Expressions régulières

● Définition Expressions régulières

Etant donné un alphabet Σ , les expressions régulières sur Σ sont définies récursivement par :

- $\emptyset$ est une expression régulière représentant l'ensemble vide.
- ϵ et $\forall c \in \Sigma^1$, c sont des expressions régulières représentant respectivement $\{\epsilon\}$ et $\{c\}$.
- Si r et s sont des expressions régulières représentant les langages R et S alors $(r+s)$, $(r.s)$ ou (rs) et r^* représentent respectivement les langages $R \cup S$, $R.S$ et R^* .

Expressions régulières

● Définition Expressions régulières

Etant donné un alphabet Σ , les expressions régulières sur Σ sont définies récursivement par :

- $\emptyset$ est une expression régulière représentant l'ensemble vide.
 - ϵ et $\forall c \in \Sigma^1$, c sont des expressions régulières représentant respectivement $\{\epsilon\}$ et $\{c\}$.
 - Si r et s sont des expressions régulières représentant les langages R et S alors $(r+s)$, $(r.s)$ ou (rs) et r^* représentent respectivement les langages $R \cup S$, $R.S$ et R^* .
- Priorité : $*$ est prioritaire sur $.$ qui est prioritaire sur $+$.

Expressions régulières

● On note r^+ l'expression $r.r^*$.

Expressions régulières

- On note r^+ l'expression $r.r^*$.
- **Définition** Langage d'une expression régulière
Etant donné une expression régulière r , $L(r)$ désigne le langage représenté par r .

Propriétés

$\forall r, s, t :$

- $r+s = s+r$ (commutativité de l'union)
- $(r+s)+t = r+(s+t)$ (associativité de l'union)
- $(r.s).t = r.(s.t)$ (associativité du produit)
- $r.(s+t) = r.s+r.t$ et $(r+s).t = r.t+s.t$ (distributivité du produit par rapport à l'union)
- $\emptyset^* = \epsilon$, $(r^*)^* = r^*$, $(\epsilon+r)^* = r^*$, $(r^*.s^*)^* = (r+s)^*$.

Equivalence entre AEFD et langages réguliers

- Nous allons montrer que tout langage régulier est reconnaissable par un AEFD et qu'un AEFD ne peut reconnaître qu'un langage régulier.
- Pour ceci, il nous suffira de montrer que tout langage régulier est reconnaissable par un AEFND avec ϵ -transitions (car on a déjà vu l'équivalence avec un AEFD) puis que tout langage reconnaissable par un AEFD est régulier.

Exp. reg. -> AEFND

- **Théorème** Tout langage défini par une expression régulière est reconnaissable par un AEFND avec ϵ -transition.
- Preuve constructive : on montre comment créer l'automate à partir de l'expression régulière.
- Plus précisément, on montre que l'on peut construire un AEFND avec ϵ -transition tel que :
 - aucun arc n'arrive vers son état initial
 - aucun arc ne part de son **unique** état final.
- Cette preuve se fait par induction sur le nombre k d'opérateurs de l'expression régulière.

Induction (preuve) : cas de base

Pour $k=0$:

Induction (preuve) : cas de base

Pour $k=0$:

- $\emptyset$ est reconnu par $(\{q_0, q_f\}, \Sigma, \delta, q_0, \{q_f\})$
avec δ définie nulle part.

Induction (preuve) : cas de base

Pour $k=0$:

- $\emptyset$ est reconnu par $(\{q_0, q_f\}, \Sigma, \delta, q_0, \{q_f\})$ avec δ définie nulle part.
- $\{\epsilon\}$ est reconnu par $(\{q_0, q_f\}, \Sigma, \delta, q_0, \{q_f\})$ avec $\delta(\epsilon) = q_f$.

Induction (preuve) : cas de base

Pour $k=0$:

- $\emptyset$ est reconnu par $(\{q_0, q_f\}, \Sigma, \delta, q_0, \{q_f\})$ avec δ définie nulle part.
- $\{\epsilon\}$ est reconnu par $(\{q_0, q_f\}, \Sigma, \delta, q_0, \{q_f\})$ avec $\delta(\epsilon) = q_f$.
- $\{c\}$ est reconnu par $(\{q_0, q_f\}, \Sigma, \delta, q_0, \{q_f\})$ avec $\delta(c) = q_f$.

Induction (preuve) : cas général

- Posons que le théorème est vrai pour les expressions avec au plus k opérateurs. Soit r une expression ayant k opérateurs. Cette expression peut avoir 3 formes :
 - $r = r_1 + r_2$.
 - $r = r_1.r_2$.
 - $r = r_1^*$.
- Soient $A_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ et $A_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ tels que $L(A_1) = L(r_1)$ et $L(A_2) = L(r_2)$ et Q_1 et Q_2 sont disjoints.

$$\mathbf{r} = \mathbf{r}_1 + \mathbf{r}_2$$

- Soit $A = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{f_0\})$ où δ est défini par :
 - $\delta(q_0, \epsilon) = \{q_1, q_2\}$.
 - $\forall q \in Q_1 \setminus \{f_1\}, \forall c \in \Sigma_1 \cup \{\epsilon\} : \delta(q, c) = \delta_1(q, c)$.
 - $\forall q \in Q_2 \setminus \{f_2\}, \forall c \in \Sigma_2 \cup \{\epsilon\} : \delta(q, c) = \delta_2(q, c)$.
 - $\delta(f_1, \epsilon) = \delta(f_2, \epsilon) = \{f_0\}$.
- Soient $A_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ et $A_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ et $\forall w \in (\Sigma_1 \cup \Sigma_2)^*, \{f_0\} \in \widehat{\delta}(q_0, w)$ si et seulement si $\{f_1\} \in \widehat{\delta}(q_1, w)$ ou $\{f_2\} \in \widehat{\delta}(q_2, w)$ donc $L(A) = L(A_1) \cup L(A_2)$.

$$\mathbf{r} = \mathbf{r}_1.\mathbf{r}_2$$

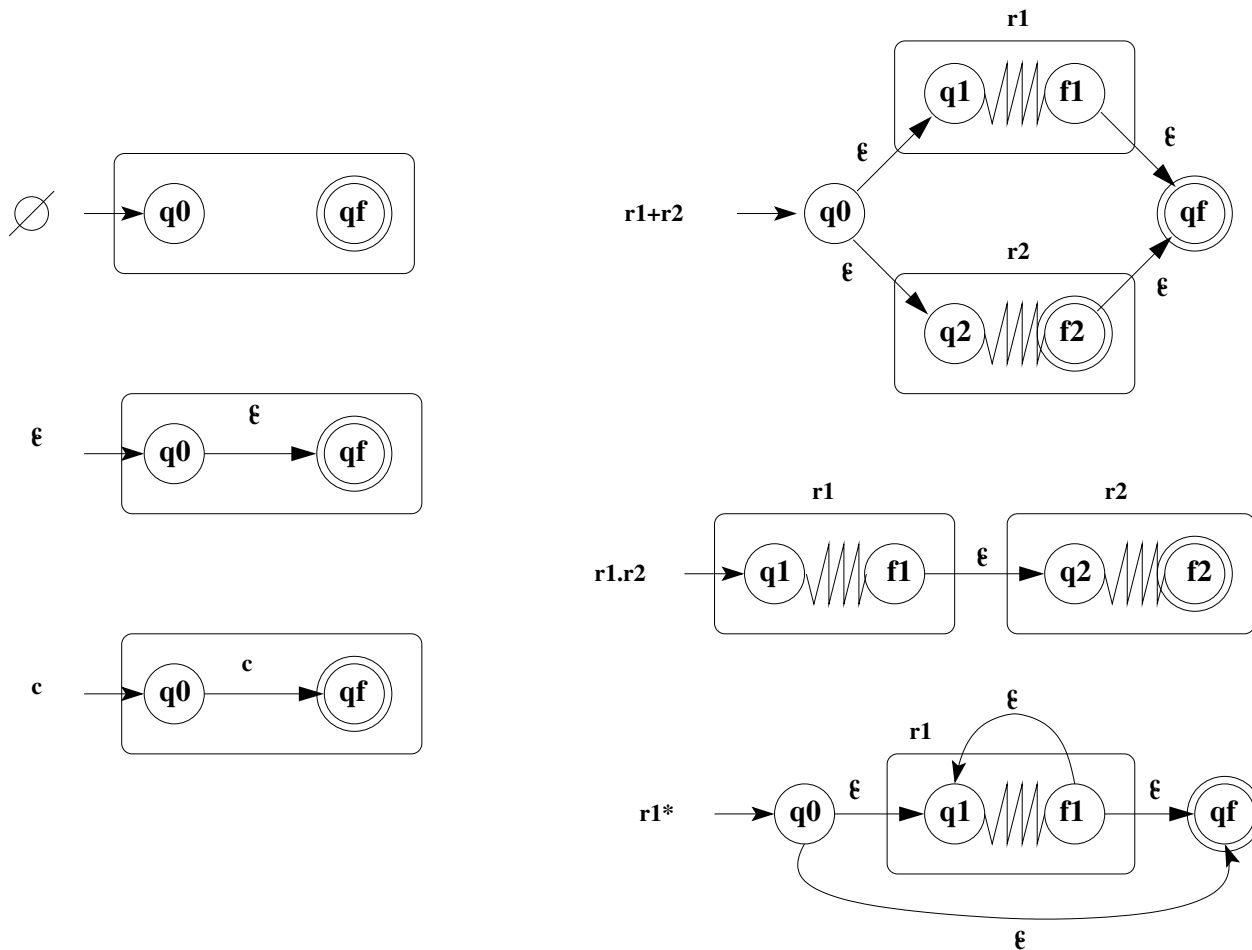
- Soit $A = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, q_1, \{f_2\})$ où δ est défini par :
 - $\forall q \in Q_1 \setminus \{f_1\}, \forall c \in \Sigma_1 \cup \{\epsilon\} : \delta(q, c) = \delta_1(q, c).$
 - $\forall q \in Q_2 \setminus \{f_2\}, \forall c \in \Sigma_2 \cup \{\epsilon\} : \delta(q, c) = \delta_2(q, c).$
 - $\delta(f_1, \epsilon) = \{q_2\}.$
- $\forall w \in (\Sigma_1 \cup \Sigma_2)^*, \hat{\delta}(q_1, w) = \{f_2\}$ si et seulement si $\exists w_1 \in \Sigma_1^*, \exists w_2 \in \Sigma_2^* : w = w_1.w_2$ et $\hat{\delta}(q_1, w_1) = \{f_1\}$ et $\hat{\delta}(q_2, w_2) = \{f_2\}$ donc $L(A) = L(A_1).L(A_2).$

$$\mathbf{r} = \mathbf{r}_1^*$$

- Soit $A = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\})$ où δ est défini par :
 - $\delta(q_0, \epsilon) = \delta(f_1, \epsilon) = \{q_1, f_0\}$.
 - $\forall q \in Q_1 \setminus \{f_1\}, \forall c \in \Sigma_1 \cup \{\epsilon\} : \delta(q, c) = \delta_1(q, c)$.
- $\forall w \in \Sigma_1^*, \forall n \geq 0, \hat{\delta}(q_0, w^n) = \{f_0\}$ si et seulement si $\hat{\delta}(q_1, w) = \{f_1\}$ donc $L(A) = L(A_1)^*$.

Conclusion (preuve)

On en déduit que le théorème est aussi vrai si r possède k opérateurs. $\square$



AEFD $\rightarrow$ Exp. reg.

- **Théorème**

Tout langage reconnaissable par un AEFD est régulier.

- **Preuve constructive, récursive**

Preuve

- Soit un AEFD $A = (\{q_0, q_1, \dots, q_n\}, \Sigma, \delta, q_0, F)$.
- On considère les AEFD suivants :
$$\forall i, j, k, A_{i,j}^{(k)} = (\{q_0, q_1, \dots, q_k\} \cup \{q_i, q_j\}, \Sigma, \delta_{i,j}^{(k)}, q_i, \{q_j\})$$

où $\delta_{i,j}^{(k)}$ est définie sur $\{q_0, q_1, \dots, q_k\} \cup \{q_i, q_j\} \times \Sigma \rightarrow \{q_0, q_1, \dots, q_k\} \cup \{q_i, q_j\}$ et $\forall c \in \Sigma, \forall q \in \{q_0, q_1, \dots, q_k\} \cup \{q_i, q_j\}, \delta_{i,j}^{(k)}(q, c) = \delta(q, c)$.

Preuve

- Soit un AEFD $A = (\{q_0, q_1, \dots, q_n\}, \Sigma, \delta, q_0, F)$.
- On considère les AEFD suivants :
$$\forall i, j, k, A_{i,j}^{(k)} = (\{q_0, q_1, \dots, q_k\} \cup \{q_i, q_j\}, \Sigma, \delta_{i,j}^{(k)}, q_i, \{q_j\})$$

où $\delta_{i,j}^{(k)}$ est définie sur $\{q_0, q_1, \dots, q_k\} \cup \{q_i, q_j\} \times \Sigma \rightarrow \{q_0, q_1, \dots, q_k\} \cup \{q_i, q_j\}$ et $\forall c \in \Sigma, \forall q \in \{q_0, q_1, \dots, q_k\} \cup \{q_i, q_j\}, \delta_{i,j}^{(k)}(q, c) = \delta(q, c)$.
- $A_{i,j}^{(k)}$ est l'automate A
 - qui ne conserve que les états $\{q_0, q_1, \dots, q_k\} \cup \{q_i, q_j\}$,
 - qui a pour état initial q_i
 - et pour état final q_j .

Preuve

- Appelons $L_{i,j}^{(k)}$ le langage $\{w \in \Sigma^* \mid \widehat{\delta}_{i,j}^{(k)}(q_i, w) = q_j\}$.
- On va démontrer par récurrence sur k que $\forall i,j,k, L_{i,j}^{(k)}$ est régulier.
(que c'est vrai pour $k = -1$, puis que si c'est vrai pour $k \geq -1$ alors c'est vrai pour $k+1$).

Preuve : cas où $k = -1$

- Pour $k = -1$, $A_{i,j}^{(-1)}$ ne possède que les états q_i et q_j .

Preuve : cas où $k = -1$

- Pour $k = -1$, $A_{i,j}^{(-1)}$ ne possède que les états q_i et q_j .
- Si $i = j$, il y a un seul état et $L_{i,i}^{(-1)} = \{\epsilon\} \cup X^*$ avec X l'ensemble (éventuellement vide) de symboles permettant les transitions de q_i à q_i .

Preuve : cas où $k = -1$

- Pour $k = -1$, $A_{i,j}^{(-1)}$ ne possède que les états q_i et q_j .
- Si $i = j$, il y a un seul état et $L_{i,i}^{(-1)} = \{\epsilon\} \cup X^*$ avec X l'ensemble (éventuellement vide) de symboles permettant les transitions de q_i à q_i .
- Si $i \neq j$ alors $L_{i,j}^{(-1)} = (W \cup X.Y^*.Z)^*.X.Y^*$ avec W, X, Y, Z les ensembles (éventuellement vides) de symboles permettant les transitions respectivement de q_i à q_i , de q_i à q_j , de q_j à q_j et de q_j à q_i .

Preuve : cas où $k = -1$

- Pour $k = -1$, $A_{i,j}^{(-1)}$ ne possède que les états q_i et q_j .
- Si $i = j$, il y a un seul état et $L_{i,i}^{(-1)} = \{\epsilon\} \cup X^*$ avec X l'ensemble (éventuellement vide) de symboles permettant les transitions de q_i à q_i .
- Si $i \neq j$ alors $L_{i,j}^{(-1)} = (W \cup X.Y^*.Z)^*.X.Y^*$ avec W, X, Y, Z les ensembles (éventuellement vides) de symboles permettant les transitions respectivement de q_i à q_i , de q_i à q_j , de q_j à q_j et de q_j à q_i .
- Dans tous les cas, $L_{i,j}^{(-1)}$ est bien un langage régulier.

Preuve : cas général

- Posons que $L_{i,j}^{(k-1)}$ est régulier.
- L'automate $A_{i,j}^{(k)}$ possède l'état supplémentaire q_k par rapport à $A_{i,j}^{(k-1)}$. Pour déduire le langage $L_{i,j}^{(k)}$ à partir de $L_{i,j}^{(k-1)}$, il nous faut rajouter les mots qui sont reconnus si on passe par l'état q_k . On a alors 3 cas :
 - Il n'existe pas de chemin partant de q_i , menant à q_j et passant par q_k .
 - Il existe au moins un chemin partant de q_i , menant à q_j et passant par q_k et il n'existe pas de circuit dans l'automate $A_{i,j}^{(k)}$ contenant q_k .
 - Il existe au moins un chemin partant de q_i , menant à q_j et passant par q_k et il existe au moins un circuit dans l'automate $A_{i,j}^{(k)}$ contenant q_k .

Cas 1

- Il n'existe pas de chemin partant de q_i , menant à q_j et passant par q_k .

Cas 1

- Il n'existe pas de chemin partant de q_i , menant à q_j et passant par q_k .
- On a alors $L_{i,j}^{(k)} = L_{i,j}^{(k-1)}$.

Cas 2

- Il existe au moins un chemin partant de q_i , menant à q_j et passant par q_k et il n'existe pas de circuit dans l'automate $A_{i,j}^{(k)}$ contenant q_k .

Cas 2

- Il existe au moins un chemin partant de q_i , menant à q_j et passant par q_k et il n'existe pas de circuit dans l'automate $A_{i,j}^{(k)}$ contenant q_k .
- On a alors $L_{i,j}^{(k)} = L_{i,k}^{(k-1)} \cdot L_{k,j}^{(k-1)} \cup L_{i,j}^{(k-1)}$.

Cas 3

- Il existe au moins un chemin partant de q_i , menant à q_j et passant par q_k et il existe au moins un circuit dans l'automate $A_{i,j}^{(k)}$ contenant q_k .

Cas 3

- Il existe au moins un chemin partant de q_i , menant à q_j et passant par q_k et il existe au moins un circuit dans l'automate $A_{i,j}^{(k)}$ contenant q_k .
- On a alors $L_{i,j}^{(k)} = L_{i,k}^{(k-1)} \cdot (L_{k,k}^{(k-1)})^* \cdot L_{k,j}^{(k-1)} \cup L_{i,j}^{(k-1)}$.

Preuve : conclusion

- Dans tous les cas $L_{i,j}^{(k)}$ est régulier.

Preuve : conclusion

- Dans tous les cas $L_{i,j}^{(k)}$ est régulier.
- Le langage de l'automate A est l'union des langages $L_{0,j}^{(n)}$ pour toutes les valeurs de j telles que $q_j \in F$. Ce langage est donc régulier. $\square$

Preuve : conclusion

- Dans tous les cas $L_{i,j}^{(k)}$ est régulier.
- Le langage de l'automate A est l'union des langages $L_{0,j}^{(n)}$ pour toutes les valeurs de j telles que $q_j \in F$. Ce langage est donc régulier. $\square$
- Cette preuve indique un moyen de déduire l'expression régulière du langage reconnu par un AEFD. Il existe une méthode plus pratique d'arriver au même résultat.

Méthode par élimination d'état

- Idée : éliminer un par un les états internes (ni initial ni finaux) d'un AEFD pour produire finalement un AEFD équivalent
 - qui ne contient plus d'état interne
 - et dont les arcs sont étiquetés par des expressions régulières.

Méthode par élimination d'état

- Idée : éliminer un par un les états internes (ni initial ni finaux) d'un AEFD pour produire finalement un AEFD équivalent
 - qui ne contient plus d'état interne
 - et dont les arcs sont étiquetés par des expressions régulières.
- Départ : remplacer les étiquettes "a,b,c,..." par des expressions régulières "a+b+c+...".

Méthode par élimination d'état

- Idée : éliminer un par un les états internes (ni initial ni finaux) d'un AEFD pour produire finalement un AEFD équivalent
 - qui ne contient plus d'état interne
 - et dont les arcs sont étiquetés par des expressions régulières.
- Départ : remplacer les étiquettes "a,b,c,..." par des expressions régulières "a+b+c+..."
- Pour éliminer un état s , il faut considérer les prédécesseurs $q_1, q_2, \dots, q_n$ de s et les successeurs $p_1, p_2, \dots, p_m$ de s .
- $\forall i, j$, on remplace les arcs (q_i, s) étiqueté X , (s, s) étiqueté Y , (s, p_j) étiqueté Z et (q_i, p_j) étiqueté W par un arc (q_i, p_j) étiqueté $X.Y^*.Z+W$.

Langages non rationnels

- Tous les langages ne sont pas réguliers. Nous allons voir un moyen de démontrer que certains langages ne sont pas réguliers grâce au lemme du facteur itérant.
- **Lemme** Lemme du facteur itérant
Pour tout langage régulier L , il existe une constante n (dépendant de L) telle que pour tout mot w de L de longueur supérieure ou égale à n , il existe les mots x , y et z tels que (1) $w = x.y.z$, (2) $y \neq \epsilon$, (3) $|x.y| \leq n$ et (4) $\forall k \geq 0 : x.y^k.z \in L$.

Preuve du lemme

- Soit L un langage rationnel et un AEFD $A = (Q, \Sigma, \delta, q_0, F)$ qui reconnaît L . Soit n le nombre d'états de A . Soit $w = c_1 c_2 \dots c_m$ un mot de L tel que $\forall i \ c_i \in \Sigma$ et $m > n$.
- Il existe forcément deux valeurs i et j telles que $0 \leq i < j \leq n$ et $\hat{\delta}(q_0, c_1 c_2 \dots c_i) = \hat{\delta}(q_0, c_1 c_2 \dots c_j)$ car il y a moins d'états dans l'automate que de facteurs gauches dans w .
- Soit $x = c_1 \dots c_i$, $y = c_{i+1} \dots c_j$ et $z = c_{j+1} \dots c_m$. On a bien $w = x.y.z$, $y \neq \epsilon$ (car $i < j$), $|x.y| \leq n$ (car $j \leq n$) et $x.y^k.z \in L$ (car $\forall k, \hat{\delta}(q_i, y) = \hat{\delta}(q_i, y^k)$). $\square$

Intérêt du lemme

- Ce lemme signifie que, dans **tout** langage rationnel, il **existe une taille n** de mot telle que **tous les mots** de taille supérieure possède un facteur non vide qui, s'il apparaissait plusieurs fois à cet endroit dans le mot (au lieu d'une seule fois), ferait que le mot appartiendrait encore au langage.
- On utilise en général le lemme du facteur itérant pour montrer qu'un certain langage L **n'est pas rationnel** : il faut montrer que **quel que soit n** , on peut trouver un mot w qui **dépend de n** et est de la forme $w = x.y.z$ où $|x.y| \leq n$ et $y \neq \epsilon$ mais qu'il existe une valeur k telle que $x.y^k.z \notin L$.

Exemple

- Démontrons que $L = \{a^m b^m \mid m \geq 0\}$ n'est pas régulier.
- Si L était régulier, il existerait une constante n tel que pour tout mot w de L , on aurait w qui répond aux critères donnés dans le lemme du facteur itérant.
- Or, si on choisit $w = a^n b^n$, qui est de longueur supérieure à n , sachant qu'il faut que $w = x.y.z$, que $|x.y| \leq n$ et $y \neq \epsilon$, on a forcément $x.y$ qui ne contient que des symboles a et z qui contient n symboles b (et peut-être des symboles a devant).
- **Donc**, il est faux que $\forall k \geq 0$, on a $x.y^k.z \in L$: par exemple, pour $k=0$, on a $x.y^0.z = x.z$ qui contient au maximum $n-1$ symboles a et exactement n symboles b donc il n'appartient pas à L .

Machine de Turing

- Une machine de Turing (MT) est une machine abstraite qui permet de modéliser un ordinateur : les calculs qu'ils peuvent effectuer sont les mêmes.

Machine de Turing

- Une machine de Turing (MT) est une machine abstraite qui permet de modéliser un ordinateur : les calculs qu'ils peuvent effectuer sont les mêmes.
- Une MT est en quelque sorte un automate d'états finis qui lit et écrit des symboles sur les cases d'un **ruban de longueur infinie** et peut potentiellement **changer le sens** de sa lecture (déplacement d'une case vers la gauche ou vers la droite) après chaque lecture de symbole.

MT : définition informelle

- Une MT peut se voir comme étant composé d'*un système de contrôle, d'une tête de lecture/écriture et d'un ruban de longueur infini.*

MT : définition informelle

- Une MT peut se voir comme étant composé d'*un système de contrôle, d'une tête de lecture/écriture et d'un ruban de longueur infini.*
- Le système de contrôle détermine
 - ce que sera le prochain état de la machine en fonction de l'état courant **et** du symbole du ruban pointé par la tête de lecture,
 - ce que sera le symbole qu'il faudra écrire à l'emplacement pointé par la tête de lecture
 - dans quel sens (une case à gauche ou à droite) déplacer la tête de lecture.

MT : définition informelle

- Une MT peut se voir comme étant composé d'*un système de contrôle*, d'*une tête de lecture/écriture* et d'*un ruban de longueur infini*.
- Le système de contrôle détermine
 - ce que sera le prochain état de la machine en fonction de l'état courant **et** du symbole du ruban pointé par la tête de lecture,
 - ce que sera le symbole qu'il faudra écrire à l'emplacement pointé par la tête de lecture
 - dans quel sens (une case à gauche ou à droite) déplacer la tête de lecture.
- Initialement, la tête de lecture pointe sur le 1^{er} symbole d'une suite finie de symboles encadrée à gauche et à droite par deux suites infinies de cases vides.

MT : définition formelle

Définition Une machine de Turing est un septuplet

$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ où :

- Q est l'ensemble des états de la machine.
- Σ est l'ensemble des symboles d'entrée de la machine.
- Γ est l'ensemble des symboles du ruban. $\Sigma \subset \Gamma$.
- δ est définie sur $Q \times \Gamma \rightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ telle que $\delta(q,c) = (q',c', D)$ avec
 - q' , le nouvel état de la machine,
 - c' le symbole qui remplace c sur le ruban
 - $D = \leftarrow$ ou $\rightarrow$, le sens de déplacement de la tête de lecture (gauche ou droite).
- q_0 est l'état initial.
- $\square \in \Gamma$ est le symbole de la case vide.
- F est l'ensemble des états finaux.

Représentation de l'état global d'une MT

On indique par $c_1c_2\dots c_{i-1}qc_i\dots c_k$

- que le ruban contient le mot $c_1\dots c_k$ encadré par des cases vides
- tandis que le système de contrôle se trouve dans l'état q
- et que la tête de lecture pointe sur c_i .

Succession d'états globaux d'une MT

- $\alpha q \beta \vdash \alpha' q' \beta'$ désigne une transition élémentaire de l'état global $\alpha q \beta$ à l'état global $\alpha' q' \beta'$
 - par la lecture du premier symbole de β ,
 - son remplacement par un nouveau symbole
 - et un déplacement de la tête de lecture à gauche ou à droite.
- On désignera par $\vdash^*$ la fermeture transitive et réflexive de $\vdash$.

Langage reconnu par une MT

Définition Le *langage reconnu par une machine de Turing* M est $L(M) = \{w \in \Sigma^* \mid q_0 w \stackrel{*}{\vdash} \alpha q \beta \text{ avec } q \in F \text{ et } \alpha, \beta \in \Gamma^*\}$.

Exemple

Le langage $L = \{a^n b^n \mid n \geq 1\}$ est reconnu par la machine de Turing $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, A, B, \square\}, \delta, q_0, \square, \{q_4\})$ avec δ définie par la table de transition suivante :

δ	a	b	A	B	$\square$
$\rightarrow q_0$	$(q_1, A, \rightarrow)$	-	-	$(q_3, B, \rightarrow)$	-
q_1	$(q_1, a, \rightarrow)$	$(q_2, B, \leftarrow)$	-	$(q_1, B, \rightarrow)$	-
q_2	$(q_2, a, \leftarrow)$	-	$(q_0, A, \rightarrow)$	$(q_2, B, \leftarrow)$	-
q_3	-	-	-	$(q_3, B, \rightarrow)$	$(q_4, \square, \rightarrow)$
$*q_4$	-	-	-	-	-

Les grammaires formelles

- Les grammaires formelles sont **un autre moyen** (que les machines abstraites) de représenter des langages formels.
- Elles indiquent comment on peut **générer** tous les mots du langage qu'elles modélisent.
- Historiquement, elles ont été définies par Chomsky pour représenter la structure des langues naturelles.

Relations

- On note $a \rightarrow b$ le fait qu'un couple (a,b) est élément d'une relation binaire.

Relations

- On note $a \rightarrow b$ le fait qu'un couple (a,b) est élément d'une relation binaire.
- **Définition** Fermeture d'une relation
Soit $\rightarrow$ une relation binaire sur $E \times E$. On définit $\xrightarrow{i}$ par :
$$\xrightarrow{0} = \{(x,x) \mid x \in E\}$$

et $\forall i > 0, \xrightarrow{i} = \xrightarrow{i-1} \circ \rightarrow$.

Relations

- On note $a \rightarrow b$ le fait qu'un couple (a,b) est élément d'une relation binaire.
- **Définition** Fermeture d'une relation
Soit $\rightarrow$ une relation binaire sur $E \times E$. On définit $\xrightarrow{i}$ par :
$$\xrightarrow{0} = \{(x,x) \mid x \in E\}$$

et $\forall i > 0, \xrightarrow{i} = \xrightarrow{i-1} \circ \rightarrow$.
- **Définition** Fermeture transitive $\xrightarrow{+}$ de la relation $\rightarrow$:
$$\xrightarrow{+} = \bigcup_{i>0} \xrightarrow{i}.$$
- **Définition** Fermeture transitive et réflexive $\xrightarrow{*}$ de la relation $\rightarrow$:
$$\xrightarrow{*} = \bigcup_{i \geq 0} \xrightarrow{i}.$$

Systemes de réécriture

- **Définition** Système de réécriture

Un *système de réécriture* défini sur un alphabet Σ est une relation finie R incluse dans $\Sigma^* \times \Sigma^*$.

- Ce système induit une relation $\xrightarrow{R}$ définie ainsi :

$\xrightarrow{R} = \{(w, w') \in \Sigma^* \times \Sigma^* \mid \exists u, v \in \Sigma^* : w = uxv \text{ et } w' = uyv \text{ et } (x, y) \in R\}$.

Systemes de réécriture

- **Définition** Système de réécriture

Un *système de réécriture* défini sur un alphabet Σ est une relation finie R incluse dans $\Sigma^* \times \Sigma^*$.

- Ce système induit une relation $\xrightarrow{R}$ définie ainsi :

$\xrightarrow{R} = \{(w, w') \in \Sigma^* \times \Sigma^* \mid \exists u, v \in \Sigma^* : w = uxv \text{ et } w' = uyv \text{ et } (x, y) \in R\}$.

- Les couples de $\xrightarrow{R}$ sont tous ceux qui s'obtiennent à partir de tout couple de R en **ajoutant le même préfixe et le même suffixe** au 1^{er} et au 2^e élément du couple de R .

- Ex : le système $R = \{(aa, bb)\}$ défini sur $\Sigma = \{a, b\}$ induit la relation $\xrightarrow{R} = \{(aa, bb), (aaa, abb), (aaa, bba), (baa, bbb), (aab, bbb), (aaaa, aabb), \dots\}$.

Syst. de réécriture pour générer un langage

- Un système de réécriture R permet de **générer un langage** L à partir d'un ensemble $A \subset \Sigma^*$ ainsi :

$$L = \{w' \in \Sigma^* \mid \exists w \in A : w \xrightarrow[R]{*} w'\}.$$

Syst. de réécriture pour générer un langage

- Un système de réécriture R permet de **générer un langage** L à partir d'un ensemble $A \subset \Sigma^*$ ainsi :

$$L = \{w' \in \Sigma^* \mid \exists w \in A : w \xrightarrow[R]{*} w'\}.$$

- Ex : $\Sigma = \{a, b\}$, $R = \{(ab, aabb)\}$ et $A = \{ab\}$ permet de générer $L = \{w \in \Sigma^* \mid ab \xrightarrow[R]{*} w\} = \{a^n b^n \mid n > 0\}$.

st. de réécriture pour reconnaître un langage

- Un système de réécriture R permet de reconnaître un langage L à partir d'un ensemble $F \subset \Sigma^*$ ainsi :

$$L = \{w \in \Sigma^* \mid \exists w' \in F : w \xrightarrow[R]{*} w'\}.$$

st. de réécriture pour reconnaître un langage

- Un système de réécriture R permet de reconnaître un langage L à partir d'un ensemble $F \subset \Sigma^*$ ainsi :
$$L = \{w \in \Sigma^* \mid \exists w' \in F : w \xrightarrow[R]{*} w'\}.$$
- Ex : $\Sigma = \{a, b, p, q\}$, $R = \{(pa, q), (qa, p), (q, b)\}$ et $F = \{b\}$ permet de reconnaître $L = \{w \in \Sigma^* \mid w \xrightarrow[R]{*} b\} = \{b\} \cup \{pa^{2n+1} \mid n \geq 0\} \cup \{qa^{2n} \mid n \geq 0\}.$

Grammaires formelles

Définition Grammaire

Une grammaire est un quadruplet (N, T, P, S) où :

- N est un ensemble fini de symboles dits *non terminaux*.
En pratique, on notera ces symboles en **majuscules**.

Grammaires formelles

Définition Grammaire

Une grammaire est un quadruplet (N, T, P, S) où :

- N est un ensemble fini de symboles dits *non terminaux*.
En pratique, on notera ces symboles en **majuscules**.
- T est un ensemble fini de symboles dits *terminaux*.
On a $T \cap N = \emptyset$.
En pratique, on notera ces symboles en **minuscules**.

Grammaires formelles

Définition Grammaire

Une grammaire est un quadruplet (N, T, P, S) où :

- N est un ensemble fini de symboles dits *non terminaux*.
En pratique, on notera ces symboles en **majuscules**.
- T est un ensemble fini de symboles dits *terminaux*.
On a $T \cap N = \emptyset$.
En pratique, on notera ces symboles en **minuscules**.
- P est un ensemble fini de *règles de production* de la forme : $\alpha \rightarrow \beta$ où $\alpha \in (N \cup T)^*.N.(N \cup T)^*$ et $\beta \in (N \cup T)^*$.

Grammaires formelles

Définition Grammaire

Une grammaire est un quadruplet (N, T, P, S) où :

- N est un ensemble fini de symboles dits *non terminaux*.
En pratique, on notera ces symboles en **majuscules**.
- T est un ensemble fini de symboles dits *terminaux*.
On a $T \cap N = \emptyset$.
En pratique, on notera ces symboles en **minuscules**.
- P est un ensemble fini de *règles de production* de la forme : $\alpha \rightarrow \beta$ où $\alpha \in (N \cup T)^*.N.(N \cup T)^*$ et $\beta \in (N \cup T)^*$.
- S est un symbole de N appelé *axiome*.
C'est le point de départ de la grammaire.

Notation compacte

- On peut noter de manière compacte un ensemble de règles de la forme
 $\alpha \rightarrow \beta, \alpha \rightarrow \beta', \alpha \rightarrow \beta'', \dots$
ainsi :
 $\alpha \rightarrow \beta | \beta' | \beta'' | \dots$
- Ex : $G = (\{S, A, B\}, \{a, b\}, P, S)$ où $P = \{S \rightarrow A, S \rightarrow B, A \rightarrow aA, A \rightarrow a, B \rightarrow bB, B \rightarrow b\}$
 $= \{S \rightarrow A | B, A \rightarrow aA | a, B \rightarrow bB | b\}.$

Notation en informatique

- En informatique, la notation des grammaires change ainsi :
 - Un terminal est une suite de caractères.
 - Un non terminal est une suite de caractères entre $<$ et $>$.
 - $::=$ est utilisé à la place de $\rightarrow$.
 - $<>$ est utilisé à la place de ϵ .
- Ex : $\langle \text{Conditionnelle} \rangle ::= \text{if}(\langle \text{Test} \rangle) \langle \text{Instruction} \rangle \mid \text{if}(\langle \text{Test} \rangle) \langle \text{Instruction} \rangle \text{ else } \langle \text{Instruction} \rangle$

Dérivation

- **Définition Dérivation**

Une dérivation est l'application d'une règle de production à un mot.

- On note $\alpha \xRightarrow{(R)} \beta$, la dérivation qui permet de substituer un facteur de α égal à la partie gauche de la règle R par la partie droite de R pour former β .

Dérivation

- **Définition Dérivation**

Une dérivation est l'application d'une règle de production à un mot.

- On note $\alpha \xRightarrow{(R)} \beta$, la dérivation qui permet de substituer un facteur de α égal à la partie gauche de la règle R par la partie droite de R pour former β .

- On note $\xRightarrow{(G)}^*$ la fermeture transitive de $\Rightarrow$ sur P.

- Ex : $S \xRightarrow{(G)}^* aaa$ car $S \xRightarrow{(S \rightarrow A)} A \xRightarrow{(A \rightarrow aA)} aA \xRightarrow{(A \rightarrow aA)} aaA \xRightarrow{(A \rightarrow a)} aaa$.

Langage engendré par une grammaire

- **Définition** Langage engendré par une grammaire
Le langage $L(G)$ engendré par une grammaire $G = (N, T, P, S)$ est défini par
$$L(G) = \{w \in T^* \mid S \xRightarrow[(G)]{*} w\}.$$

Langage engendré par une grammaire

- **Définition** Langage engendré par une grammaire

Le langage $L(G)$ engendré par une grammaire

$G = (N, T, P, S)$ est défini par

$$L(G) = \{w \in T^* \mid S \xRightarrow[(G)]{*} w\}.$$

- **Ex :** Le langage engendré par la grammaire

$G = (\{S, A, B\}, \{a, b\}, P, S)$ où $P = \{S \rightarrow A \mid B, A \rightarrow aA \mid a, B \rightarrow bB \mid b\}$ est

$$L(G) = \{a^n \mid n \geq 1\} \cup \{b^n \mid n \geq 1\}.$$

Langage engendré par une grammaire

- **Définition** Langage engendré par une grammaire
Le langage $L(G)$ engendré par une grammaire $G = (N, T, P, S)$ est défini par
$$L(G) = \{w \in T^* \mid S \xRightarrow[(G)]{*} w\}.$$
- Ex : Le langage engendré par la grammaire
 $G = (\{S, A, B\}, \{a, b\}, P, S)$ où $P = \{S \rightarrow A \mid B, A \rightarrow aA \mid a, B \rightarrow bB \mid b\}$ est
$$L(G) = \{a^n \mid n \geq 1\} \cup \{b^n \mid n \geq 1\}.$$
- **Définition** Langage élargi engendré par une grammaire
Le langage $\hat{L}(G)$ engendré par une grammaire $G = (N, T, P, S)$ est défini par :
$$\hat{L}(G) = \{w \in (N \cup T)^* \mid S \xRightarrow[(G)]{*} w\}.$$

Typologie des grammaires

- En posant des restrictions sur les règles de production, on peut imposer aussi des restrictions sur les langages engendrés par les grammaires résultantes.
- Le but de ces restrictions est généralement d'obtenir des grammaires dont les algorithmes associés (décision, génération, ...) sont plus efficaces.
- Les quatre différents types de grammaire définies par Chomsky sont :
 - Les grammaires non restreintes (type 0).
 - Les grammaires contextuelles [context-sensitive] (type 1).
 - Les grammaires hors contexte [context-free] (type 2).
 - Les grammaires régulières (type 3).

Grammaires non restreintes

- Il n'y a aucune restriction sur P .
Règles de la forme : $\alpha \rightarrow \beta$ où $\alpha \in (N \cup T)^*.N.(N \cup T)^*$
et $\beta \in (N \cup T)^*$.
- Ex : $G = (\{S, T, R, A, C\}, \{a, b, c\}, P, S)$ où $p = \{S \rightarrow TA, T \rightarrow aTbR \mid \epsilon, Rb \rightarrow bR, RA \rightarrow Ac, cC \rightarrow c, A \rightarrow \epsilon\}$.

Grammaires contextuelles

- Les règles $\alpha \rightarrow \beta$ sont telles que $\alpha = wXw'$ et $\beta = wvw'$ avec $X \in N$, α et $\beta \in (N \cup T)^*$ et $v \in (N \cup T)^+$.
- Par ailleurs, on peut ajouter la règle $S \rightarrow \epsilon$ à condition qu'il n'existe aucune autre règle où S apparaît à droite.
- Ex : $G = (\{S, X, Y\}, \{a, b, c\}, P, S)$ où $p = \{S \rightarrow aSXY \mid aXY, aX \rightarrow ab, bX \rightarrow bb, bY \rightarrow bc, cY \rightarrow cc\}$.
- On ne peut appliquer une règle de production pour remplacer un non terminal que si son environnement/contexte correspond à la partie gauche de la règle.

Grammaires hors contexte

- Les règles $\alpha \rightarrow \beta$ sont telles que $\alpha \in N$ et $\beta \in (N \cup T)^*$.
- Ex : $G = (\{S, A\}, \{a, b, c\}, P, S)$ où $P = \{S \rightarrow aSb \mid abA, A \rightarrow cB, B \rightarrow b\}$.
- Contrairement aux grammaires contextuelles, on n'a pas à se soucier de l'environnement d'un non terminal pour lui appliquer une règle.
- Ces grammaires permettent d'engendrer la plupart des langages de programmation existants.

Grammaires régulières

- Les règles sont de la forme $A \rightarrow w \mid wB$ (linéaire à droite) ou $A \rightarrow w \mid Bw$ (linéaire à gauche) où A et $B \in N$ et $w \in T$.
- Ex: $G = (\{S, A\}, \{a, b\}, P, S)$ où $P = \{S \rightarrow aS \mid b\}$.
- Ces grammaires sont utilisées pour l'analyse lexicale.

Type d'un langage

- On démontre (pas dans le cadre de ce cours) que l'ensemble des langages générés par les grammaires de type i inclut strictement l'ensemble des langages générés par les grammaires de type $i - 1$.
- **Définition** Type d'un langage
Un langage est de type i s'il existe une grammaire de type i qui l'engendre mais qu'il n'existe pas de grammaire de type $i + 1$ qui puisse l'engendrer.

Equivalence entre AEFD et grammaire régulière

Langage formel	Machine abstraite	Grammaire
récurziv. énum.	Machines de Turing	non restreintes
contextuels	MdT bornées linéair.	contextuelles
hors contexte	Automates à pile	hors contexte
rationnels	Automates	régulières

Grammaire régulière $\rightarrow$ AEFD

● Théoreme

Tout langage engendré par une grammaire régulière est reconnaissable.

● Idée de la preuve: il faut montrer que le langage engendré par $G = (N, T, P, S)$ **linéaire à droite** est reconnu par $A = (Q, T, \delta, S, \{\epsilon\})$ avec :

- Q contenant S et tous les suffixes des parties droites de règles de P
- δ définie par :
 - si $A \in N$, $\delta(A, \epsilon) = \{\alpha \mid (A \rightarrow \alpha) \in P\}$
 - si $c \in T$ et $\alpha \in T^* \cup T^*.N$ alors $\delta(c.\alpha, c) = \{\alpha\}$.

Cela se fait en montrant, par induction sur la longueur d'un chemin dans l'automate, que $\forall w \in T^* : \alpha \in \delta(S, w)$

$\Leftrightarrow S \xRightarrow{*} w\alpha$.

Cas d'une grammaire régulière linéaire à gauche

Pour une grammaire linéaire à gauche, il suffit :

- d'inverser l'ordre des symboles dans les parties droites des règles afin d'obtenir une grammaire régulière linéaire à droite qui engendre le même langage mais avec tous les mots inversés,

Cas d'une grammaire régulière linéaire à gauche

Pour une grammaire linéaire à gauche, il suffit :

- d'inverser l'ordre des symboles dans les parties droites des règles afin d'obtenir une grammaire régulière linéaire à droite qui engendre le même langage mais avec tous les mots inversés,
- de construire l'automate correspondant selon la méthode précédente,

Cas d'une grammaire régulière linéaire à gauche

Pour une grammaire linéaire à gauche, il suffit :

- d'inverser l'ordre des symboles dans les parties droites des règles afin d'obtenir une grammaire régulière linéaire à droite qui engendre le même langage mais avec tous les mots inversés,
- de construire l'automate correspondant selon la méthode précédente,
- d'inverser le sens des arcs de l'automate et d'échanger les rôles de l'état initial et de l'état final, afin d'obtenir un automate qui reconnaît le langage originel.

Automate $\rightarrow$ grammaire régulière

● Théoreme

Tout langage reconnaissable est engendré par une grammaire régulière.

- Idée de la preuve : à partir de $A = (Q, \Sigma, \delta, q_0, F)$,
 - si q_0 n'est pas un état final, on définit la grammaire $G = (Q, \Sigma, P, q_0)$ où P est constitué des règles
 - $q \rightarrow c.q'$ quand $\delta(q, c) = q'$
 - (en plus) $q \rightarrow c$ quand $\delta(q, c) \in F$.
 - Si $q_0 \in F$ alors on rajoute la règle $S \rightarrow q_0 \mid \epsilon$ et S devient l'axiome de la grammaire (à la place de q_0).