Université d'Aix-Marseille III - Licence Math-Info, 2ème année

I5 : Langages formels, automates et grammaires - Notes de cours

Dans ce cours, nous étudierons les langages formels. Nous verrons deux manières de les modéliser : les machines abstraites (automates et machines de Turing) et les grammaires formelles.

1 Intérêt des langages formels en informatique théorique

Une catégorie fondamentale de problèmes que l'on se pose en informatique regroupe les *problèmes de décision*. Ils correspondent à une question dont la réponse doit être "oui" ou "non" du type "est-ce que ceci est une solution à tel problème?". Ex : est-ce que cette suite de chiffres représente un nombre premier ? Est ce que cette suite de caractères correspond à un programme C correctement formé ? Plus formellement, de façon générale, la question à laquelle répond un problème de décision est "Est-ce que cette suite de symboles fait partie de l'ensemble des suites de symboles qui correspondent à une solution de mon problème?".

Un langage formel se définit simplement comme un ensemble de suites de symboles. On peut alors considérer qu'un langage formel contient l'ensemble des suites de symboles pour lesquelles la réponse d'un problème de décision est "oui". Si on considère l'ensemble de tous les langages possibles, nous avons affaire à l'ensemble des ensembles de solutions de tous les problèmes possibles. La question est alors de savoir si, quel que soit le langage formel, il est toujours possible de construire une machine qui permette de répondre "oui" si et seulement la suite de symboles fournie en entrée appartient au langage et donc si c'est une solution du problème que le langage représente.

Il existe différents types de machines abstraites permettant de représenter plus ou moins de langages formels et donc capables de résoudre plus ou moins de problèmes informatiques. La machine de Turing, définie dans les années 30, est un modèle abstrait de machine capable de faire exactement les mêmes calculs que les ordinateurs passés et actuels. Les automates finis, des modèles plus simples et moins puissants de machines, ont été étudiés dans les années 40 et 50, avec pour objectif de modéliser certaines fonctionnalités du cerveau. A la fin des années 50, Chomsky commence indépendamment l'étude des grammaires formelles, destinées au départ à modéliser les langues naturelles. Depuis, on s'est rendu compte que machines abstraites et grammaires formelles étaient des façons équivalentes de modéliser des langages formels. Les grammaires formelles sont particulièrement utilisées en compilation (analyse lexicale et syntaxique des langages de programmation).

Type de langage formel	Type de machine abstraite	Type de grammaire formelle
récursivement énumérables	Machines de Turing	non restreintes
contextuels	Machines de Turing bornées linéairement	contextuelles
hors contexte	Automates à pile	hors contexte
rationnels	Automates	régulières

FIG. 1 – *Equivalences entre machines abstraites et grammaires formelles pour la hiérarchie des 4 types de langages formels (du plus général au moins général).*

2 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).

Définition 2.1 *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.

Définition 2.2 *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.

Définition 2.3 *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.

Définition 2.4 *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.

Définition 2.5 *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 occurence d'un facteur est l'apparition de ce facteur a une position précise dans le mot.

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

Définition 2.6 *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 $\Sigma.\Sigma' = \{ac, ad, bc, bd\}$.

Définition 2.7 *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 $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup ....$ 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 : $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$.

Définition 2.8 *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.

En général, on définira un langage en utilisant une notation ensembliste de la forme {w | propriété sur w}, qui signifie "l'ensemble des mots w (de Σ^*) tels qu'ils respectent une certaine propriété". Ex : {w | w 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.

Définition 2.9 *Produit de langages*
Le produit (de concaténation) de 2 langages L et L' est défini par : $L.L' = \{w.w' \mid w \in L,\ w' \in L'\}$.

Définition 2.10 *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 $L^* = L^0 \cup L^1 \cup L^2 \cup ....$ 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 : $L^* = L^+ \cup \{\epsilon\}$.
Remarque : l'étoile d'un langage fini est un langage infini dénombrable.

3 Automates d'états finis

Les automates sont des machines abstraites qui modélisent des systèmes discrets dont l'état va varier au cours du temps à cause d'influences externes. On peut représenter un automate par un graphe (cf cours de I6) dont chaque sommet est un état et chaque arc est une transition possible d'un état vers un autre. Chaque arc est étiqueté par le nom de l'action qui permet de passer d'un état vers un nouvel état. Un automate possède un état initial et un ou plusieurs états finaux, qui correspondent à des états désirés pour le système. Lorsqu'on va présenter une suite d'actions en entrée de l'automate, celui-ci va passer par une succession d'états. Si le dernier état atteint est un état final alors on en concluera que cette suite d'actions permet au système d'atteindre un état désiré.

Ex : on veut modéliser un système simple composé d'un coffre-fort et d'un document top secret. On peut agir sur le système en **o**uvrant ou en **f**ermant le coffre ou en **p**laçant ou en **r**etirant 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. A partir de l'état 2, on peut passer à l'état 1 en fermant le coffre et on peut passer à l'état 3 en plaçant le document dans le coffre, etc. Si notre problème consiste à mettre en sûreté notre document, l'état 4 est le seul état final. Si la situation actuelle est que le coffre est fermé et vide alors l'état initial est l'état 1.

3.1 Automates d'états finis déterministes

Définition 3.1 *Automate d'états finis déterministe*
Un automate d'états finis déterministe (AEFD) est un quintuplet (Q, Σ, δ, 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.*

Définition 3.2 *Extension aux mots de la fonction de transition*
La fonction de transition peut être étendue aux mots. Son extension $\widehat{\delta}$ est définie sur $Q \times \Sigma^* \rightarrow Q$ avec :
- *$\widehat{\delta}(q_i, \epsilon) = q_i$.*
- *$\forall\ w \in \Sigma^*, \forall\ c \in \Sigma$: $\widehat{\delta}(q_i, w.c) = \delta(\widehat{\delta}(q_i, w), c)$.*

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

Définition 3.3 *Mot accepté par un automate*
Un mot $w \in \Sigma^*$ est dit accepté *par un automate (Q, Σ, δ, q_0, F) si $\widehat{\delta}(q_0, w) \in F$.*

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.

Définition 3.4 *Langage reconnu par un automate*
Un langage reconnu *par un automate A est défini par : $L(A) = \{w \in \Sigma^* \mid \widehat{\delta}(q_0, w) \in F\}$.*

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

Représentations d'un automate

On peut représenter un automate 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 (voir fig. 2).

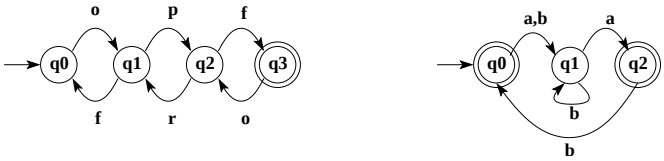

FIG. 2 – *Deux exemples d'AEFD. Le premier (à gauche) est celui du coffre-fort. Le deuxième (à droite) reconnaît notamment les mots ϵ, aa, ba, aab, aba, bab, bba, abba, etc*

On peut aussi représenter un automate par une *table de transition*. Cette table contient en ligne les états possibles de l'automate et en colonne les symboles de Σ. L'état initial est préfixé par une flèche et 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.
Exemples avec les AEFD de la figure 2 :

	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

Définition 3.5 *Langage reconnaissable*
Un langage est dit reconnaissable *s'il existe un AEFD le reconnaissant.*

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.

3.2 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 3.6 *Automate d'états finis non déterministes*
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 :
$\delta : Q \times \Sigma \rightarrow 2^Q$
$\delta(q_i, c) = Q_j$ *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.

Définition 3.7 *Extension aux mots de la fonction de transition*
La fonction de transition peut être étendue aux mots. Son extension $\widehat{\delta}$ est définie sur $Q \times \Sigma^* \rightarrow 2^Q$ avec :
- $\widehat{\delta}(q_i, \epsilon) = \{q_i\}$.
- $\forall\ w \in \Sigma^*, \forall\ c \in \Sigma$: $\widehat{\delta}(q_i, w.c) = \bigcup_{q_j \in \widehat{\delta}(q_i, w)} \delta(q_j, c)$.

$\widehat{\delta}(q_i, w) = Q_j$ 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.

Définition 3.8 *Mot accepté par un AEFND*
Un mot $w \in \Sigma^*$ est dit accepté *par un AEFND $(Q, \Sigma, \delta, q_0, F)$ si $\widehat{\delta}(q_0, w) \cap F \neq \emptyset$.*

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 3.9 *Langage reconnu par un AEFND*
Un langage reconnu *par un AEFND A est défini par : $L(A) = \{w \in \Sigma^* \mid \widehat{\delta}(q_0, w) \cap F \neq \emptyset\}$.*

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.

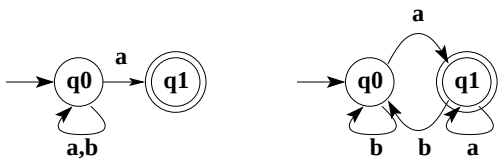

FIG. 3 – *Le langage $\{w.a \mid w \in \Sigma^*\}$ reconnu par un AEFND (à gauche) et un AEFD (à droite).*

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.

Théorème 3.1 *Tout langage reconnu par un AEFND peut aussi être reconnu par un AEFD.*

La preuve de ce théorème est 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 $A_D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_N)$ 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 $\forall c \in \Sigma$: $\delta_D(S,c) = \bigcup_{s \in S} \delta_N(s, c)$.

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

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

On a donc $L(A_D) = \{w \in \Sigma^* \mid \widehat{\delta}_D(\{q_0\}, w) \in F_D\} = \{w \in \Sigma^* \mid \widehat{\delta}_D(\{q_0\}, w) \cap F_N \neq \emptyset\} = \{w \in \Sigma^* \mid \widehat{\delta}_N(q_0, w) \cap F_N \neq \emptyset\} = L(A_N)$. □

Construction d'un AEFD à partir d'un AEFND

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.

3.3 Automates d'états finis non déterministes 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.

Définition 3.10 *AEFND avec ϵ-transitions*
Un AEFND avec ϵ-transitions est un quintuplet (Q, Σ , q_0, δ, F) où Q, Σ, q_0, δ et F ont la même signification que pour un AEFND mais où δ est définie sur $Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$.

Définition 3.11 *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 : $\forall\ S \subseteq Q$, ϵ-Fermeture(S) = $\underset{s \in S}{\cup}$ ϵ-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.

Définition 3.12 *Extension aux mots de la fonction de transition*
L'extension aux mots $\widehat{\delta}$ de la fonction de tranSsition est définie sur $Q \times \Sigma^* \rightarrow 2^Q$ avec :
- *$\widehat{\delta}(q_i, \epsilon) = \epsilon$-fermeture($q_i$).*
- *$\forall\ w \in \Sigma^*$, $\forall\ c \in \Sigma$: $\widehat{\delta}(q_i, w.c) = \underset{q_j \in \widehat{\delta}(q_i,w)}{\cup} \epsilon$-fermeture($\delta(q_j, c)$).*

Equivalence entre un AEFND avec ϵ-transitions et un AEFD

Bien que les 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.

Théorème 3.2 *Tout langage reconnu par un AEFND avec ϵ-transitions peut aussi être reconnu par un AEFD.*

On va démontrer que si $A_E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ est un AEFND avec ϵ-transitions 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($\underset{s \in S}{\cup} \delta_E(s, c)$).

Démontrons d'abord que $\forall w$, $\widehat{\delta}_D(q_D, w) = \widehat{\delta}_N(q_0, w)$. On le démontre par récurrence sur $|w|$:
- Pour $|w| = 0$, c-a-d $w = \epsilon$, $\widehat{\delta}_D(q_D, \epsilon) = \widehat{\delta}_E(q_0, \epsilon) = \epsilon$-fermeture($q_0$).
- Posons que pour tout mot w de longueur n, on a $\widehat{\delta}_D(q_D, w) = \widehat{\delta}_E(q_0, w) = \{p_1, p_2, ...p_k\}$.
  $\forall c \in \Sigma$, $\widehat{\delta}_D(q_D, w.c) = \delta_D(\widehat{\delta}_D(q_D, w), c) = \delta_D(\{p_1, p_2, ...p_k\}, c) = \epsilon$-Fermeture($\underset{1 \leq i \leq k}{\cup} \delta_E(p_i, c)$) = $\underset{1 \leq i \leq k}{\cup} \epsilon$-fermeture($\delta_E(p_i, c)$) = $\widehat{\delta}_E(q_0, w.c)$. On a alors $\widehat{\delta}_D(q_D, w) = \widehat{\delta}_E(q_0, w)$ qui est aussi vérifiée pour tous les mots de longueur n+1.

On a donc $L(A_D) = \{w \in \Sigma^* \mid \widehat{\delta}_D(q_D, w) \in F_D\} = \{w \in \Sigma^* \mid \widehat{\delta}_D(q_D, w) \cap F_E \neq \emptyset\} = \{w \in \Sigma^* \mid \widehat{\delta}_E(q_0, w) \cap F_E \neq \emptyset\} = L(A_E)$. □

4 Les langages réguliers/rationnels

Le but de ce chapitre est de présenter les langages réguliers (ou rationnels), les expressions régulières, qui sont leur représentation, et de montrer qu'ils constituent l'ensemble des langages reconnaissables (par des AEFD)

4.1 Les langages réguliers et les expressions régulières

Définition 4.1 *Langages réguliers*
Etant donné un alphabet Σ, les langages réguliers (ou rationnels) sont des langages définis récursivement par :

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

Définition 4.2 *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 +.
Par ailleurs, on note r^+ l'expression $r.r^*$.
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), $\emptyset^* = \epsilon$, $(r^*)^* = r^*$, $(\epsilon+r)^* = r^*$, $(r^*.s^*)^* = (r+s)^*$.

Définition 4.3 *Langage d'une expression régulière*
Etant donné une expression régulière r, L(r) désigne le langage représenté par r.

4.2 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.

Théorème 4.1 *Tout langage défini par une expression régulière est reconnaissable par un AEFND avec ϵ-transition.*

La preuve est constructive : on montre comment créer l'automate à partir de l'expression régulière. Plus précisément, nous allons montrer que l'on peut construire un AEFND avec ϵ-transition tel qu'aucun arc n'arrive vers son état initial et qu'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 :

- Pour k=0 (voir la partie gauche de la figure 4) :
  - $\emptyset$ 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 $\delta(c) = q_f$.
- 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 :
  - $r = r_1 + r_2$. 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.
    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 :
    - $\delta(q_0, \epsilon) = \{q_1, q_2\}$.

- $\forall q \in Q_1 \setminus \{f_1\}, \forall c \in \Sigma_1 \cup \{\epsilon\} : \delta(q, c) = \delta_1(q, c)$.
- $\forall q \in Q_2 \setminus \{f_2\}, \forall c \in \Sigma_2 \cup \{\epsilon\} : \delta(q, c) = \delta_2(q, c)$.
- $\delta(f_1, \epsilon) = \delta(f_1, \epsilon) = \{f_0\}$.

$\forall w \in (\Sigma_1 \cup \Sigma_2)^*$, $\{f_0\} \in \widehat{\delta}(q_0, w)$ si et seulement si $\{f_1\} \in \widehat{\delta}(q_1, w)$ ou $\{f_2\} \in \widehat{\delta}(q_2, w)$ donc $L(A) = L(A_1) \cup L(A_2)$.

- $r = r_1.r_2$. Soit $A = (Q_1 \cup Q_2 , \Sigma_1 \cup \Sigma_2, \delta, q_1, \{f_2\})$ où δ est défini par :
  - $\forall q \in Q_1 \setminus \{f_1\}, \forall c \in \Sigma_1 \cup \{\epsilon\} : \delta(q, c) = \delta_1(q, c)$.
  - $\forall q \in Q_2 \setminus \{f_2\}, \forall c \in \Sigma_2 \cup \{\epsilon\} : \delta(q, c) = \delta_2(q, c)$.
  - $\delta(f_1, \epsilon) = \{q_2\}$.

  $\forall w \in (\Sigma_1 \cup \Sigma_2)^*$, $\widehat{\delta}(q_1, w) = \{f_2\}$ si et seulement si $\exists w_1 \in \Sigma_1^*$, $\exists w_2 \in \Sigma_2^* : w = w_1.w_2$ et $\widehat{\delta}(q_1, w_1) = \{f_1\}$ et $\widehat{\delta}(q_2, w_2) = \{f_2\}$ donc $L(A) = L(A_1).L(A_2)$.
- $r = r_1^*$. Soit $A = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\})$ où *delta* est défini par :
  - $\delta(q_0, \epsilon) = \delta(f_1, \epsilon) = \{q_1, f_0\}$.
  - $\forall q \in Q_1 \setminus \{f_1\}, \forall c \in \Sigma_1 \cup \{\epsilon\} : \delta(q, c) = \delta_1(q, c)$.

  $\forall w \in \Sigma_1^*$, $\forall n \geq 0$, $\widehat{\delta}(q_0, w^n) = \{f_0\}$ si et seulement si $\widehat{\delta}(q_1, w) = \{f_1\}$ donc $L(A) = L(A_1)^*$.

On en déduit que le théorème est aussi vrai si r possède k opérateurs.□

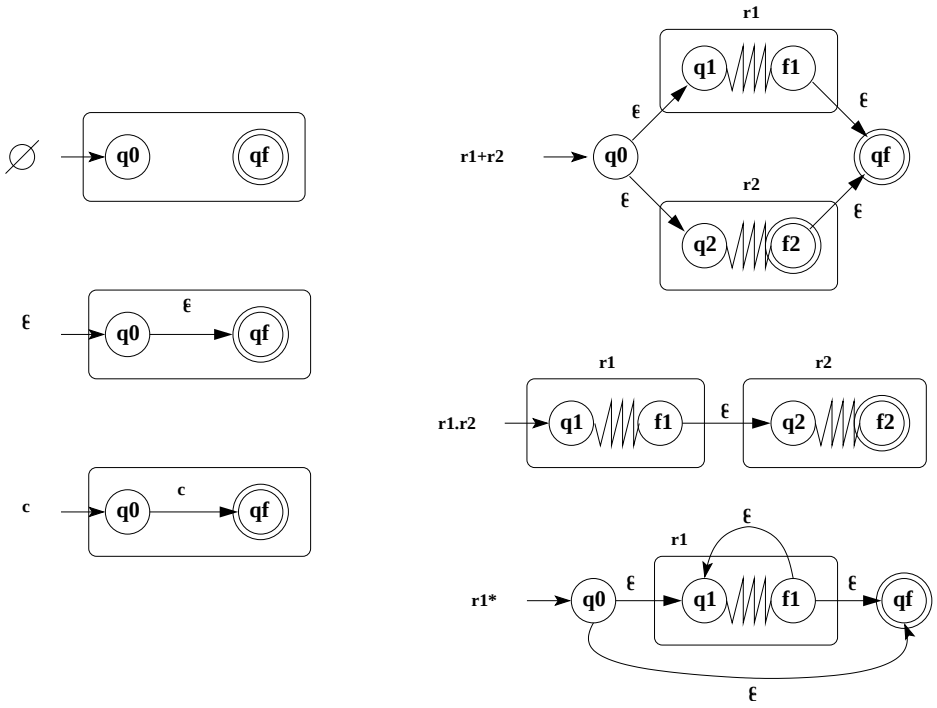

FIG. 4 – *Transformations d'une expression régulière en un AEFD avec ϵ-transition.*

Théorème 4.2 *Tout langage reconnaissable par un AEFD est régulier.*

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

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 ∀i,j,k, $L_{i,j}^{(k)}$ est régulier.

- Pour k = -1, $A_{i,j}^{(-1)}$ ne possède que les états q_i et q_j. Si i ≠ 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. 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. Dans tous les cas, $L_{i,j}^{(-1)}$ est bien un langage régulier.
- 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. On a alors $L_{i,j}^{(k)} = L_{i,j}^{(k-1)}$.
  - 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 $L_{i,j}^{(k)} = L_{i,k}^{(k-1)}.L_{k,j}^{(k-1)} \cup L_{i,j}^{(k-1)}$.
  - 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 $L_{i,j}^{(k)} = L_{i,k}^{(k-1)}.(L_{k,k}^{(k-1)})^*.L_{k,j}^{(k-1)} \cup L_{i,j}^{(k-1)}$.

  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. □

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.

Déduction d'une expression régulière à partir d'un AEFD par élimination d'états

L'idée de cette méthode est d'éliminer un par un les états internes (c-a-d des états qui ne sont 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. On commence par 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, ...q_n de s (les états dont il existe une transition d'eux vers s) et les successeurs p_1, p_2, ...p_m de s (les états dont il existe une transition de s vers eux). ∀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 $X.Y^*.Z+W$.

4.3 Langages non réguliers

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 4.1 *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 ≠ ϵ, (3) |x.y| ≤ n et (4) ∀k≥ 0 : $x.y^k.z \in L$.

Soit L un langage régulier et AEFD A = (Q, Σ, δ, q_0, F) un automate qui reconnaît L. Soit n le nombre d'états de A. Soit w=$c_1c_2...c_m$ un mot de L tel que ∀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 $\widehat{\delta}(q_0, c_1c_2...c_i) = \widehat{\delta}(q_0, c_1c_2...c_j)$ car il y a moins d'états dans l'automate que de facteurs gauches dans w. Soit $x = c_1...c_i$, $y = c_{i+1}...c_j$ et $z = c_{j+1}...c_m$. On a bien w = x.y.z, y ≠ ϵ (car i < j), |x.y| ≤ n (car j ≤ n) et $x.y^k.z \in L$ (car ∀k, $\widehat{\delta}(q_i, y) = \widehat{\delta}(q_i, y^k)$). □

Ce lemme signifie que, dans tout langage régulier, 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 régulier : 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| ≤ n et y ≠ ϵ mais qu'il existe une valeur k telle que x.y^k.z ∉ L.

Ex : Démontrons que L = {$a^m b^m$ | m≥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| ≤ n et y ≠ ϵ, 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 ∀ k≥0, on a x.y^k.z ∈ 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.

4.4 Minimisation des AEFD

Plusieurs AEFD différents peuvent reconnaître le même langage. Nous allons voir comment, à partir d'un AEFD, déterminer un AEFD équivalent (qui reconnaît le même langage) qui minimise le nombre d'états de l'automate.

Définition 4.4 *Etats équivalents, états distingables*
Deux états q et q' d'un AEFD (Q, Σ, δ, Q_0, F) sont dit équivalents si : ∀w ∈ Σ, $\widehat{\delta}$(q, w) ∈ F ⇔ $\widehat{\delta}$(q', w) ∈ F. Deux états seront dits distingables s'ils ne sont pas équivalents.

Dans un AEFD, il est possible de déterminer l'ensemble des états équivalents en déterminant récursivement l'ensemble des états distingables :

- Si q ∈ F et q' ∉ F alors q et q' sont distingables.
- S'il existe un symbole c tel que δ(q,c) et δ(q',c) sont distingables alors q et q' sont distingables.

Pour déterminer l'ensemble des états équivalents, il suffit de considérer l'ensemble des couples (q,q') d'états de Q × Q et de les marquer comme étant équivalents ou distingables selon le procédé suivant :

- Vérifier d'abord si q et q' ont déjà été marqués comme étant équivalents ou distingables.
- Si q = q' alors "ils" sont équivalents.
- Si q ∈ F et q' ∉ F (ou l'inverse) alors q et q' sont distingables. Les marquer comme tel.
- S'il existe un arc issu de q et étiqueté par un symbole mais pas d'arc issu de q' étiqueté par ce même symbole (ou la réciproque) alors q et q' sont distingables. Les marquer comme tel.
- Sinon, pour chaque symbole lisible à partir de q et q', déterminer récursivement si les successeurs sont équivalents ou distingables. Si les successeurs sont équivalents pour tous les symboles alors q et q' sont équivalents sinon q et q' sont distingables. Les marquer en conséquence.

Une fois qu'on a déterminé les classes d'équivalences des automates, il suffit de fusionner chaque classe pour qu'elle constitue un seul état. Un arc étiqueté relie deux états fusionnés ssi il reliait avec la même étiquette deux états appartenant respectivement aux classes d'équivalence correspondantes.

5 Automates à pile

Afin d'élargir l'ensemble des langages reconnus par un automate, on peut lui adjoindre une pile qui sert à mémoriser des symboles. Un automate à pile se caractérise par :

- la transition vers un nouvel état, qui est choisi non seulement en fonction de l'état courant et du symbole lu mais aussi en fonction du symbole se trouvant au sommet de la pile.
- le fait que le sommet de la pile est remplacé par un mot lors de la transition.

Définition 5.1 *Automate à pile non déterministe*
Un automate à pile non déterministe est un septuplet M = (Q, Σ, Γ, δ, q_0, □, 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 de la pile.*
- δ *est la fonction de transition définie sur* $Q \times \Sigma \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$ *telle que* δ*(q, c, s) est un ensemble de couples (q', w) avec q', le nouvel état de la machine,* $w \in \Gamma^*$ *le mot qui remplace le symbole s au sommet de la pile.*
- q_0 *est l'état initial.*
- $\square \in \Gamma$ *est le symbole qui se trouve initialement dans la pile.*
- *F est l'ensemble des états finaux.*

Remarque : lors d'une transition, si on a w = ϵ alors le sommet de la pile est retiré tandis que si w commence par s alors le reste de w est empilé.

Un mot est accepté par un automate à pile si, une fois ce mot lu complètement, on se trouve dans un état de F.
Remarque : les automates à pile non déterministes sont plus puissants que les automates à pile déterministes (non démontré en cours), eux-mêmes plus puissants que les AEFD. Par exemple, il existe un automate à pile qui reconnaît $\{a^n b^n \mid n \geq 0\}$.

6 Machine de Turing

Une machine de Turing (MT) est une machine abstraite qui permet de modéliser un ordinateur : les calculs qu'il peut effectuer sont les mêmes que ceux d'un ordinateur. 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.

Une machine de Turing 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 et dans quel sens (une à case gauche ou à droite) déplacer la tête de lecture. Initialement, la tête de lecture pointe sur le premier symbole d'une suite finie de symboles encadrée à gauche et à droite par deux suites infinies de cases vides.

Définition 6.1 *Machine de Turing*
Une machine de Turing est un septuplet M = (Q, Σ*,* Γ*,* δ*,* 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 la fonction de transition définie sur* $Q \times \Gamma \rightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ *telle que* δ*(q,c) = (q',c', D) avec q', le nouvel état de la machine, c' le symbole qui remplace c sur le ruban et* $D = \leftarrow$ *ou* $\rightarrow$*, le sens de déplacement de la tête de lecture (la case de gauche ou la case de 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 $c_1 c_2 ... c_{i-1} q c_i ... c_k$ que le ruban contient le mot $c_1 ... c_k$ 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 (trace d'un programme)

$\alpha q\beta \vdash \alpha' q' \beta'$ 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 $\overset{*}{\vdash}$ la fermeture transitive et réflexive de $\vdash$.

Définition 6.2 *Langage reconnu par une machine de Turing*

Le langage reconnu par une machine de Turing M est L(M) = $\{w \in \Sigma^* \mid q_0 w \overset{*}{\vdash} \alpha q \beta$ *avec* $q \in F$ *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	-	-	-	$(q_3, B, \rightarrow)$	$(q_4, \square, \rightarrow)$
$*q_4$	-	-	-	-	-

Démontrons (dans le détail) que M reconnaît bien L.

On va d'abord démontrer que $\forall n \geq 1, \forall i, 0 \leq i < n$, $A^i q_0 a^{n-i} B^i b^{n-i} \overset{*}{\vdash} A^{i+1} q_0 a^{n-i-1} B^{i+1} b^{n-i-1}$.

$\forall i, 0 \leq i < n$:
$$\begin{aligned} A^i q_0 a^{n-i} B^i b^{n-i} &\vdash A^{i+1} q_1 a^{n-i-1} B^i b^{n-i} \\ &\overset{*}{\vdash} A^{i+1} a^{n-i-1} B^i q_1 b^{n-i} \\ &\vdash A^{i+1} a^{n-i-1} B^{i-1} q_2 B B b^{n-i-1} \\ &\overset{*}{\vdash} A^i q_2 A a^{n-i-1} B^{i+1} b^{n-i-1} \\ &\vdash A^{i+1} q_0 a^{n-i-1} B^{i+1} b^{n-i-1} \end{aligned}$$

Si l'état global initial de M est $q_0 a^n b^n$ $(= A^0 q_0 a^{n-0} B^0 b^{n-0})$ alors $q_0 a^n b^n \overset{*}{\vdash} A q_0 a^{n-1} B b^{n-1} \overset{*}{\vdash} A^i q_0 a^{n-i} B^i b^{n-i}$ $\overset{*}{\vdash} A^n q_0 B^n \vdash A^n B q_3 B^{n-1} \overset{*}{\vdash} A^n B^n q_3 \vdash A^n B^n \square q_4$. Comme $q_4 \in F$, on a démontré que tous les mots de L sont acceptés par M.

Il nous reste à montrer que si un mot n'appartient pas à L alors il n'est pas accepté par M. Pour cela, il suffit de démontrer que si un mot contient un a situé après un b ou s'il ne contient pas autant de a que de b alors il n'est pas accepté par M.

Remarquons d'abord que M ne peut que réécrire des a en A et des b en B. Aucune autre réécriture de symbole n'est possible. Pour que M atteigne l'état q_4, le seul état final de M, il faut qu'elle se trouve avant dans l'état q_0 et que tous les symboles situés à droite de la tête de lecture soient des B. Quand M est en q_0, si le symbole pointé par la tête de lecture n'est pas un B alors, pour que M s'arrête en q_4, il faut qu'elle quitte l'état q_0 et qu'elle y revienne : il faut qu'elle lise un a (qu'elle transforme en A) pour passer en q_1, puis une suite (éventuellement vide) de a et de B (qui la font rester en q_1), puis un b (qu'elle transforme en B) et ensuite M déplace sa tête de lecture vers la gauche jusqu'à ce qu'elle pointe à droite du a qu'elle a réécrit en A.

Pour que M puisse effectuer un aller-retour depuis q_0, il faut qu'elle trouve à chaque fois un a et un b qu'elle transforme en A et B. En conséquence, s'il n'y a pas autant de a que de b en entrée de M alors M ne pourra pas revenir en q_0 et le mot ne sera pas accepté.

Par ailleurs, après chaque aller-retour depuis q_0, M a commencé par lire un a et le réécrire en A puis a fini par placer sa tête de lecture juste après pour se retrouver dans l'état q_0. Pour que M puisse continuer ses allers-retours, il faut donc que le mot en entrée contienne une suite non vide de a. Dès que (et si) ils sont tous transformés en A, M va finir par lire un B (anciennement b) et passer à l'état q_3. Or, s'il y a un

a après le B (qui correspond au premier b présent dans le mot d'entrée) alors M ne peut pas passer à l'état q_4. Donc, si un a se trouve après un b dans le mot d'entrée, ce mot n'est pas accepté par M.

Nous pouvons donc finalement en conclure que le langage reconnu par M est L.

7 Les grammaires formelles

Dans le cadre des grammaires formelles, qui sont basées sur des systèmes de réécriture, nous utiliserons la notation $a \rightarrow b$ pour désigner un couple (a,b) élément d'une relation binaire.

Définition 7.1 *Fermeture d'une relation*
Soit $\rightarrow$ une relation binaire sur E×E. On définit $\xrightarrow{i}$ par : $\xrightarrow{0} = \{(x,x) \mid x \in E\}$ et $\forall i > 0$, $\xrightarrow{i} = \xrightarrow{i-1} \text{ o } \rightarrow$. On définit la fermeture transitive de la relation $\rightarrow$ par : $\xrightarrow{+} = \bigcup_{i>0} \xrightarrow{i}$. On définit la fermeture transitive et réflexive de la relation $\rightarrow$ par : $\xrightarrow{*} = \bigcup_{i\geq 0} \xrightarrow{i}$.

Définition 7.2 *Système de réécriture*
Un système de réécriture défini sur un alphabet Σ est une relation finie incluse dans $\Sigma^* \times \Sigma^*$. Ce système induit une relation $\underset{R}{\rightarrow}$ définie ainsi : $\underset{R}{\rightarrow} = \{(w, w') \in \Sigma^* \times \Sigma^* \mid \exists u,v \in \Sigma^* : w = uxv$ et $w' = uyv$ et $(x,y) \in R\}$.

En d'autres termes, les couples de $\underset{R}{\rightarrow}$ sont tous ceux qui s'obtiennent à partir de tout couple de R en ajoutant le même préfixe et le même suffixe au premier et au second élément du couple de R.
Ex : le système R = {(aa, bb)} défini sur Σ = {a, b} induit la relation $\underset{R}{\rightarrow}$ = {(aa, bb), (**a**aa, **a**bb), (aa**a**, bb**a**), (**b**aa, **b**bb), (aa**b**, bb**b**), (**aa**aa, **aa**bb), ...}.

Utilisation des systèmes de réécriture pour spécifier un langage

Un système de réécriture R permet de généner un langage L à partir d'un ensemble A $\subset \Sigma^*$ ainsi :
L = {w' $\in \Sigma^*$ | $\exists$ w $\in$ A : w $\xrightarrow[R]{*}$ w'}.
Ex : Σ = {a, b}, R = {(ab, aabb)} et A = {ab} permet de générer L = {w $\in \Sigma^*$ | ab $\xrightarrow[R]{*}$ w} = {$a^n b^n$ | n>0}.

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

Définition 7.3 *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.*

On peut noter de manière compacte un ensemble de règles de la forme $\alpha \rightarrow \beta$, $\alpha \rightarrow \beta'$, $\alpha \rightarrow \beta''$, ... ainsi : $\alpha \rightarrow \beta|\beta'|\beta''|...$.

Ex : G = ({S, A, B}, {a, b}, P, S) où P = {S → A, S → B, A → aA, A → a, B → bB, B → b} = {S → A | B, A → aA | a, B → bB | b}.
En informatique, la notation des grammaires change un peu. On écrit <un_nom> à la place d'un non terminal et ::= à la place de →.

Définition 7.4 *Dérivation*
Une dérivation est l'application d'une règle de production à un mot. On note $\alpha \underset{(R)}{\Rightarrow} \beta$*, 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 $\overset{*}{\underset{(G)}{\Rightarrow}}$ la fermeture transitive de ⇒ sur P.
Ex : S $\overset{*}{\underset{(G)}{\Rightarrow}}$ aaa car S $\underset{(S\to A)}{\Rightarrow}$ A $\underset{(A\to aA)}{\Rightarrow}$ aA $\underset{(A\to aA)}{\Rightarrow}$ aaA $\underset{(A\to a)}{\Rightarrow}$ aaa.

Définition 7.5 *Langage engendré par une grammaire*
Le langage L(G) engendré par une grammaire G = (N, T, P, S) est défini par $L(G) = \{w \in T^* | \ S \overset{*}{\underset{(G)}{\Rightarrow}} w\}$.

Ex : Le langage engendré par la grammaire G = ({S, A, B}, {a, b}, P, S) où P = {S → A | B, A → aA | a, B → bB | b} est L(G) = $\{a^n | \ n \geq 1\} \cup \{b^n | \ n \geq 1\}$.

Définition 7.6 *Langage élargi engendré par une grammaire*
Le langage $\hat{L}(G)$ *engendré par une grammaire G = (N, T, P, S) est défini par :*
$\hat{L}(G) = \{w \in (N \cup T)^* | \ S \overset{*}{\underset{(G)}{\Rightarrow}} w\}$.

Typologie des grammaires - Hiérarchie de Chomsky

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 de type 0 : il n'y a aucune restriction sur P.
- Les grammaires contextuelles [context-sensitive] (type 1) : les règles $\alpha \to \beta$ sont telles que $\alpha = wXw'$ et $\beta = wvw'$ avec X ∈ N, α et $\beta \in (N \cup T)^*$ et $v \in (N \cup T)^+$. Par ailleurs, on peut ajouter la règle S → ϵ 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 → aSXY | aXY, aX → ab, bX → bb, bY → bc, cY → 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.
- Les grammaires hors contexte [context-free] (type 2) : les règles $\alpha \to \beta$ sont telles que $\alpha \in N$ et $\beta \in (N \cup T)^*$.
  Ex : G = ({S, A}, {a, b, c}, P, S) où P = {S → aSb | abA, A → cB, B → 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.
- Les grammaires régulières (type 3). Les règles sont de la forme A → w | wB (linéaire à droite) ou A → w | Bw (linéaire à gauche) où A et B ∈ N et w ∈ T.
  Ces grammaires sont utilisées pour l'analyse lexicale.

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 7.7 *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

Théorème 7.1 *Tout langage engendré par une grammaire régulière est reconnaissable.*

Idée de la preuve (constructive) : il faut montrer que, étant donné une grammaire G = (N, T, P, S) linéaire à droite, l'automate avec ϵ-transition A = (Q, T, δ, S, $\{\epsilon\}$) avec :

- Q contenant S et tous les suffixes des parties droites de règles de P
- δ définie par : (1) si A $\in$ N, δ(A, ϵ) = $\{\alpha \mid (A \rightarrow \alpha) \in P\}$ et (2) si c $\in$ T et $\alpha \in T^* \cup T^*.N$ alors δ(c.α, c) = $\{\alpha\}$.

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

[...]

Pour une grammaire linéaire à gauche, il suffit : (1) d'inverser l'ordre des symboles dans les parties droites des règles afin d'obtenir une grammaire régulière qui engendre le même langage mais avec tous les mots inversés, (2) de construire l'automate correspondant selon la méthode décrite ci-dessus, (3) 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.□

Théorème 7.2 *Tout langage reconnaissable est engendrable par une grammaire régulière.*

Idée de la preuve (constructive) : à partir d'un AEFD A = (Q, Σ, δ, q_0, F), si q_0 n'est pas un état final, on définit la grammaire G = (Q, Σ, P, q_0) où P est constitué des règles q $\rightarrow$ c.q' quand δ(q,c) = q' et, de plus, q $\rightarrow$ c quand δ(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). [...] □