RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIREMINISTÈRE DE L'ENSEIGNEMENT SUPERIEUR ET DE LARECHERCHE SCIENTIFIQUEUNIVERSITE FERHAT ABBES SETIF 1

Théorie des langages

Support de cours, exercices de TD

Licence 2ème année

Faculté des Sciences

Département d'Informatique

Dr. Ahlem Drif

Dernière mise à jour : 25/05/2019

Site web : https://fsciences.univ-setif.dz/page_professionnelles/280

Table des matières

IV. Langages algébriques

V. Automates à pile

VI. Machine de Turing

Chapitre I: Introductions aux langages

I.1 Introductions aux langages

La structure de base de la théorie des langages sont les mots, on peut en donner une définition mathématique:

Définition 1 (Monoïde) Un monoïde est une structure algébrique consistant en un ensemble muni d'une loi de composition interne associative (noté ".") et un élément neutre noté ε.

Définition 2 Le monoïde est dit libre s'il possède une base (un sous ensemble) dont les éléments sont indépendants. On a donc existence et unicité d'une factorisation sur un monoïde libre.

Définition 3

En *TL*, la base est appelée alphabet, les éléments de cette base sont appelés lettres, la loi du monoïde est appelée concaténation. Une concaténation de lettre forme un mot, l'élément neutre ε est ainsi logiquement dénommé le mot vide. Un ensemble de mots est appelé langage.

I.2 Alphabets et mots

Définition 1 Un alphabet, noté *X*, est un ensemble non vide de symboles (ou lettres).

Exemple: alphabet du langage C: A….Z, 0…9, ==, <,=,(,), ….

Définition 2 Un mot défini sur un alphabet *X*, est une suite finie d'élément de *X*.

Exemple

abb, cba, aaab, bcaaa, ab, sont des mots construits sur l'alphabet *X={a, b, c}*

On définit:

- *x.y* (ou simplement *xy*): la concaténation des deux mots *x* et *y*, autrement dit, le mot formé en faisant suivre les lettres de *x* par les lettres de *y*:
  si $x = a_1a_2............a_m$ $y = b_1b_2..........b_n$ alors: $x.y = xy = a_1a_2............a_mb_1b_2..........b_n$
- L'opération concaténation n'est pas commutative: $xy \neq yx$,
- La concaténation est une loi de composition interne.
- La concaténation est associative: *(xy)z = x(yz)*
- L'élément neutre est le mot vide: $\forall x \in X^*$ $\varepsilon x = x\varepsilon = x$,
- x^n: le mot *x* concaténé *n* fois ($x^0 = \epsilon$, $x^1 = x$, $x^2 = xx, x^3 = xxx$,…),
- $|x|$: la longueur du mot *x*, tel que: $|x|$ = nombre de lettres qui composent *x*,

- $|x|_a$: l'occurrence d'une lettre a dans le mot *x*, c à d son nombre d'apparition, *exemple*: $|011|_1 = 2,\ |00|_1 = 0$
- x^R : le mot obtenu en inversant les lettres de *x*: si $x = a_1 a_2 \ldots\ldots\ldots\ldots a_m$ *alors* $x^R = a_m \ldots\ldots\ldots\ldots\ldots\ldots a_1$
- X^+: l'ensemble des mots de longueur supérieure ou égale à 1 que l'on peut construire à partir de l'alphabet *X*,
- X^*: l'ensemble des mots que l'on peut construire à partir de *X*, y compris le mot vide: $X^* = X^0 + X^1 + X^2 + \ldots\ldots\ldots\ldots$ avec $X^0 = \{\varepsilon\}$ ou bien: $X^* = \{\varepsilon\} \cup X^+$,

Lemme de lévi

Soit *X* un alphabet et soient *a, b, c et d* quatre mots quelconques de X^* tel que *ab=cd*

Il existe trois cas exprimant la relation entre *a, b, c et d* :

1) Si $|a| < |c|$ *Alors* $c = af$ *et* $b = fd$
2) Si $|a| = |c|$ *Alors* $a = c$ *et* $b = d$
3) Si $|a| > |c|$ *Alors* $a = cf$ *et* $d = fb$

Démonstration du lemme de lévi: voir TD1.

I.3 Opérations définies sur les langages:

Un langage, défini sur un alphabet *X*, est un ensemble de mots définis sur *X*. Autrement dit, un langage est un sous-ensemble de X^*.

Deux langages particuliers sont indépendants de l'alphabet *X*: le langage vide ($L = \Phi$) et le langage contenant le seul mot vide ($L = \{\varepsilon\}$).

Exemple:

1- Soit *X= {0, 1},*

a- Soit *L* le langage sur *X* formé des mots 0, 00, 1, 11. *L* s'écrit: *L= {0, 00, 1, 11}*

b- Soit le langage *L* sur *X* tel que *L* soit formé de tous les mots qui commencent par 0. L peut donc s'écrire:

$L = \{w \in X^* \,/\, w = 0w' \; avec \;\; w' \in X^*\}$.

2- Soit *X= {a, +, *, (,)}*

Soit *L* le langage sur *X* formé des expressions arithmétiques bien parenthésées sur a. Le mot ((a+a)*a) appartient à ce langage.

Remarque:

Le monoïde libre engendré par X est un ensemble infini.

L peut être fini ou infini.

Exemple:

Les langages des exemples 1-b et 2- sont des ensembles infinis.

Le langage de l'exemple1-a est fini.

Opérations définies sur les langages:

Soient deux langages L_1 et L_2 respectivement définis sur les alphabets X_1 et X_2 :

- L'union de L_1 et L_2 est le langage défini sur $X_1 \cup X_2$ contenant tous les mots qui sont soit contenus dans L_1 , soit contenus dans L_2 :

$$L_1 \cup L_2 = \{x \,/\, x \in L_1 \; ou \; x \in L_2\}$$

- L'intersection de L_1 et L_2 est le langage défini sur $X_1 \cap X_2$ contenant tous les mots qui sont contenus à la fois dans L_1 et L_2 :

$$L_1 \cap L_2 = \{x \,/\, x \in L_1 \; et \; x \in L_2\}$$

- Le complément de L_1 est le langage défini sur X_1 contenant tous les mots qui ne sont pas dans L_1 :

$$C(L_1) = \{x \,/\, x \notin L_1\}$$

- La différence de L_1 et L_2 est le langage défini sur X_1 contenant tous les mots de L_1 qui ne sont pas dans L_2 :

$$L_1 - L_2 = \{x \,/\, x \in L_1 \; et \; x \notin L_2\}$$

- Le produit ou concaténation de L_1 et L_2 est le langage défini sur $X_1 \cup X_2$ contenant tous les mots formés d'un mot de L_1 suivi d'un mot de L_2 :

$$L_1 . L_2 = \{xy \,/\, x \in L_1 \; et \; y \in L_2\}$$

- La fermeture itérative de L_1 (ou fermeture de Kleene ou itéré de L_1) est l'ensemble des mots formés par une concaténation finie de mots de L_1 :

$$L_1^* = \{x / \exists k \geq 0 \, et \; x_1,...,x_k \in L_1 \, tels \; que \, x = x_1x_2...x_k\}$$

$L_1^* = \varepsilon \cup L_1 \cup L_1^2 \cup L_1^3 \cup ... \cup L_1^n \cup ...$

$L^* = L^0 + L^+$

$L^+ = LL^* = L^*L$

Propriétés sur le produit de langage

X alphabet, L, L' , L'' trois langages sur X :

1) $L(L'.L'') = (LL')L''$

2) $L(L'+L'') = LL' + LL''$

3) $L(L' \cap L'') \neq LL' \cap LL''$

Propriétés sur l'étoile de langage

Soient L, R deux langages sur X:

1) $L^* = (L^*)^*$

2) $L^* = L^*.L^*$

3) $L(RL)^* = (LR)^* L$

4) $(L+R)^* = (L^* R^*)^*$

5) $(L+R)^* = (R^* L)^* R^*$

I.4 Description d'un langage

- Un langage fini peut être décrit par l'énumération des mots qui le composent.
- Certains langages infinis peuvent être décrits par l'application d'opérations à des langages plus simples.
- Certains langages infinis peuvent être décrits par un ensemble de règles appelé grammaire.
- Enfin, certains langages infinis ne peuvent pas être décrits, ni par l'application d'opérations, ni par un ensemble de règles. On parle alors de langage indécidable.

I.5 Grammaires

Un langage peut être défini comme l'ensemble des mots satisfaisant un certain nombre de règles. Cette vue du concept de langage a son origine dans des essais de formalisation du langage naturel.

I.5.1 Définition 1

Une grammaire est un quadruplet $G = (T, N, S, P)$

T: est le vocabulaire terminal, ç à dire l'alphabet sur lequel est défini le langage.

N: est le vocabulaire non terminal, ç à dire l'ensemble des symboles qui n'apparaissent pas dans les mots générés, mais qui sont utilisés au cours de la génération. Un symbole non terminal désigne une "catégorie syntaxique".

$S \in N$: est le symbole de départ ou l'axiome.

P: est un ensemble de règles dites de réécriture ou de production de la forme:

$u_1 \to u_2$ avec $u_1 \in (T \cup N)^+$ et $u_2 \in (T \cup N)^*$

La signification intuitive de ses règles est que: u_1 peut être remplacer par u_2.

Exemple:

Axiome = *S*

$N = \{S, E, E', T, T', F\}$

$T = \{i, (,), +, *\}$

$P = \{$

$$S \to E$$
$$E \to T\,E'$$
$$E' \to +T\,E'$$
$$E' \to v$$
$$T \to F\,T'$$
$$T' \to *F\,T'$$
$$T' \to v$$
$$F \to i$$
$$F \to (E)$$

$\}$

I.5.2 Terminologie

Le langage défini, ou généré, par une grammaire est l'ensemble des mots qui peuvent être obtenus à partir du symbole de départ par application des règles de la grammaire. Plus formellement, on introduit les notions de dérivation entre mots, d'abord en une étape, ensuite en plusieurs étapes:

Définition 2

Soit une grammaire $G=(T,N,S,P)$, une forme non vide $u \in (T \cup N)^+$ et une forme éventuellement vide $v \in (T \cup N)^*$, la grammaire G permet de dériver *v* de *u* en une étape (noté $u \rightarrow v$) si et seulement si:

- $u = xu'y$
- $v = xv'y$
- $u' \rightarrow v'$ est une règle de *P*.

Exemple:

Quels sont les mots générés par la grammaire de l'exemple précédent?

Définition 3

Une forme *v* peut être dérivée d'une forme *u* en plusieurs étapes:

- $u \Rightarrow^* v$: si *v* peut être obtenue de *u* par une succession de *0*, *1* ou plusieurs dérivations en une étape.
- $u \Rightarrow^+ v$: si *v* peut être obtenue de *u* par une succession de *1* ou plusieurs dérivations en une étape.

Définition 4

Le langage généré par une grammaire $G=(T,N,S,P)$ est l'ensemble des mots sur *T* qui peuvent être dérivés à partir de *S*:

$$L(G) = v \in T^* \,/\, S \Longrightarrow^* v$$

Remarque:

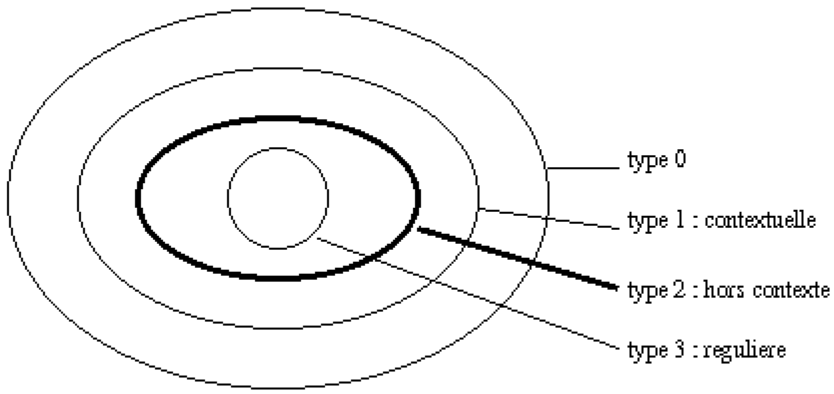
Une grammaire définit un seul langage. Par contre, un même langage peut être engendré par plusieurs grammaires différentes.

I.5.3. Hiérarchie de Chomsky

En introduisant des critères plus ou moins restrictifs sur les règles de production, on obtient des classes de grammaires hiérarchisées, ordonnées par inclusion. La classification des grammaires, définie en 1957 par *Noam CHOMSKY*, distingue quatre classes illustrées dans le tableau suivant :

	Langages	Grammaires	Procédure effective
3	Rationnels ou réguliers	Régulières à droite (régulières à gauche)	Automates finis
2	Algébriques ou non-contextuels	Algébriques, non-contextuelles	Automates à pile
1	Contextuels	Contextuelles, monotones $\alpha \to \beta$ ou $A \to \epsilon$ $\alpha, \beta \in (N \cup T)^*$, A axiome	Machine de Turing à l'espace linéairement borné
0	Récursivement énumérables	Contextuelles avec effacement aucune contrainte	Machine de Turing

Propriété Les grammaires de type 0 englobent les grammaires de type 1 qui englobent les grammaires de type 2 qui englobent les grammaires de type 3.



A chaque type de grammaire est associé un type de langage:
les grammaires de type 3 génèrent les langages réguliers,
les grammaires de type 2, les langages hors-contexte et
les grammaires de type 1, les langages contextuels.
Les grammaires de type 0 permettent de générer tous les langages "décidables", autrement dit, tous les langages qui peuvent être reconnus en un temps fini par une machine.

Les langages qui ne peuvent pas être générés par une grammaire de type 0 sont dits "indécidables".

Enfin, à chaque type de grammaire est associé un type d'automate qui permet de reconnaître les langages de sa classe: les langages réguliers sont reconnus par des automates finis, les langages hors-contexte sont reconnus par des automates à pile, et les autres langages, décrits par des grammaires de type 1 ou 0, sont reconnus par des machines de Turing. Ainsi, la machine de Turing peut être considérée comme le modèle de machine le plus puissant qu'il soit, dans la mesure où tout langage (ou plus généralement, tout problème) qui ne peut pas être traité par une machine de Turing, ne pourra pas être traité par une autre machine.

Exercices de TD

Exercice 1: ***Lemme de lévi***

Soit X un alphabet et soient *a, b, c et d* quatre mots quelconques de X^* tel que $ab=cd$

Il existe trois cas exprimant la relation entre *a, b, c et d* :

1) Si $|a| < |c|$ $Alors\ \ c = af\ \ et\ \ b = fd$
2) Si $|a| = |c|$ $Alors\ \ a = c\ et\ \ b = d$
3) Si $|a| > |c|$ $Alors\ \ a = cf\ \ et\ \ d = fb$

- Démontrez le lemme de lévi.

Exercice 2: ***Application du Lemme de lévi***

Soient *u, v et w* $\in X^*$.

- Démontrez que: Si $u^2v^2 = w^2$ alors $uv = vu$.

Exercice 3:

Soit X un alphabet et x, y deux mots quelconques de X^*.

- Démontrez par récurrence que : $(x.y)^R = y^R.x^R$

Exercice 4: ***Propriétés sur l'étoile de langage***

Soient L, R deux langages sur X, démontrez que:

1) $L^* = (L^*)^*$

2) $L^* = L^*.L^*$

3) $L^+ = LL^* = L^*L$

4) $L(RL)^* = (LR)^*L$

5) $(L+R)^* = (L^*R^*)^*$

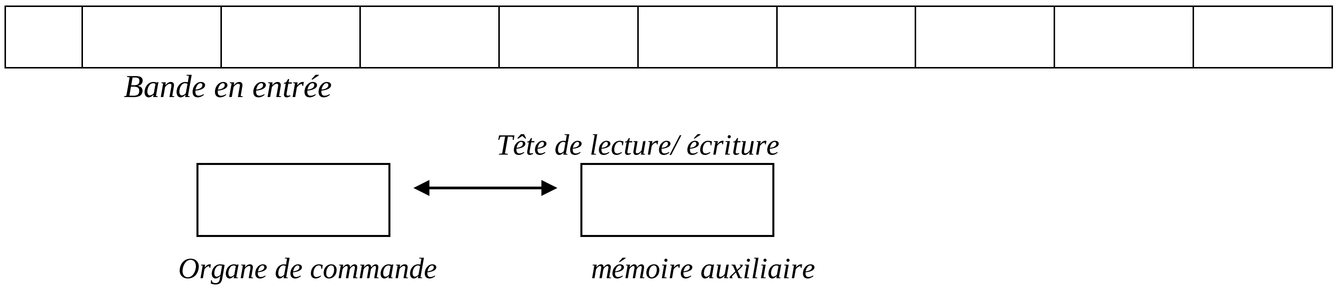
6) $(L+R)^* = (R^*L)^*R^*$

Chapitre II: Les automates à états finis

II.1 Introduction

Un automate est composé de 3 parties:

1. une bande en entrée finie ou infinie sur laquelle va s'inscrire le mot à lire.
2. un organe de commande qui permet de gérer un ensemble d'état.
3. éventuellement une mémoire auxiliaire de stockage.



L'automate qui reconnaît les langages de type:

- Type 3: c'est l'automate d'états finis (AEF)
- Type 2: automates à pile.
- Type 1: automate à borne linéaire.
- Type 0: machine de Turing.

Remarque: On distingue les automates d'états finis déterministes et indéterministes. Intuitivement, la différence réside dans le fait d'aboutir à un seul état (déterministe) ou à plusieurs états (indéterministe) à partir d'un état donné en lisant une lettre.

II.2 Automates d'états finis déterministes

Définition 1: Un automate fini déterministe est un quintuplet:

$$A = (X, Q, q_0, \delta, F)$$

Où:

X: l'alphabet (d'entrée)

Q: l'ensemble des états

q_0: l'état initial

δ : la fonction de transition

$\delta : Q \, x \, X \rightarrow Q$

F: l'ensemble des états finaux ($F \subseteq Q$)

II.3 Représentation d'un automate d'états fini

Il existe plusieurs manières de représenter un *AEF*:

a- à travers la définition, ç à d en expliquant les cinq paramètres:

Exemple: $A = (\{a,b\}, \{q_0, q_1, q_2\}, q_0, \delta, \{q_1, q_2\})$

Avec $\delta(q_0, a) = q_0$, $\delta(q_0, b) = q_1$, $\delta(q_1, b) = \delta(q_2, a) = \delta(q_2, b) = q_2$

b- à travers une représentation matricielle:

$q_j = \delta(q_i, x_j)$

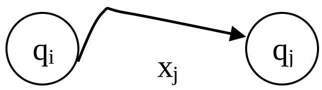
Q \ **X**	x_1 ………	x_j ………………	x_m
état initial: $\mathbf{q_0}$			
état final: **.**			
.			
.			
$\mathbf{q_i}$		$q_j = \delta(q_i, x_j)$	
.			
état final: **.**			
$\mathbf{q_n}$			

Exemple: $A = (\{a,b\}, \{q_0, q_1, q_2\}, q_0, \delta, \{q_1, q_2\})$

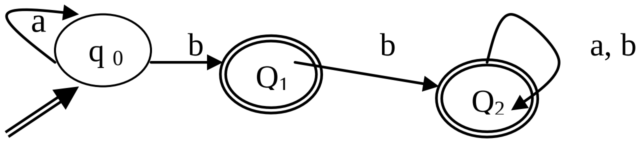
Q \ **X**	a	b
état initial: $\mathbf{q_0}$	$\mathbf{q_0}$	$\mathbf{q_1}$
état final: $\mathbf{q_1}$		$\mathbf{q_2}$
état final: $\mathbf{q_2}$	$\mathbf{q_2}$	$\mathbf{q_2}$

c- à travers une représentation graphique

$q_j = \delta(q_i, x_j)$



Exemple: $A = (\{a,b\}, \{q_0, q_1, q_2\}, q_0, \delta, \{q_1, q_2\})$



II.4 Fonctionnement d'un automate d'états fini

Afin de pouvoir lire des mots, on étend δ *à* δ^* de manière unique par:

$$\delta^* : Q \, x \, X^* \to Q$$
$$(q, x) \mapsto q'$$

Nous distinguons trois cas sur la longueur de x:

Cas 1: $|x| = 0 \qquad \delta^*(q,x) = q$

Cas 2: $|x| = 1 \qquad \delta^*(q,x) = \delta(q,x)$

Cas 3: $|x| > 1 \qquad \delta^*(q,x) = \delta[\delta(q,a), x']$ avec $x = ax'$, a$\in$ X, $x' \in X^*$

Le fonctionnement de l'automate se fait à travers une succession de configuration *(q, w), q*$\in$*Q, w* est le mot sur la bande en entrée.

II.5 Langage reconnu par un AEF

$L(A) = \{ w \in X^* / (q_0, w) \xrightarrow{*} q_F \}$ $q_f \in F$

II.6 Mots reconnus par un AEF

$$A = (X, Q, q_0, \delta, F)$$

f$\in$? L(A) , f$\in$ x^*

1ier cas:

$(q_0, f) \xrightarrow{*} q \; et \; q \in F \Rightarrow f \in L(A)$

2ème cas:

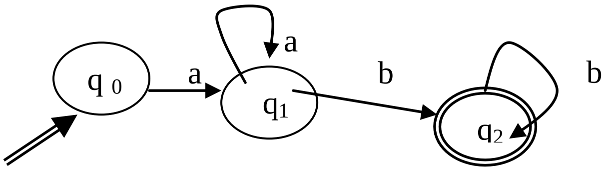
$(q_0, f) \xrightarrow{*} q \; et \; q \notin F \Rightarrow f \notin L(A)$

3ème cas:

$(q_0, f) \xrightarrow{*} (q, w)\, et\; q \in Q,\; w \in X^*\; avec\; \delta^*(q, w)\, n'existe\; pas\, blocage\;\; f \in L(A)$

Exemple:

Soit *A* l'automate EF défini par le graphe suivant:



Faisons fonctionner l'automate sur quelques mots:

1/aab 2/aa 3/aaba 4/ε

$(q_0, aab) \mapsto (q_1, ab) \mapsto (q_1, b) \mapsto q_2 \;\; q_2 \in F\;\; donc\;\; aab \in L(A)$

$(q_0, aa) \mapsto (q_1, a) \mapsto q_1 \qquad q_1 \notin F\;\; donc\;\; aa \notin L(A)$

$(q_0, aaba) \mapsto (q_1, aba) \mapsto (q_1, ba) \mapsto (q_2, a) \mapsto$
$\delta(q_2, a)\, n'existe\;\; pas\;\; il\;\; y\, a\, donc\;\; blocage\, et\;\; ainsi\;\; aaba \notin L(A)$

(q_0, ε) n'existe pas il y a blocage et $\varepsilon \notin L(A)$

II.7 Minimisation d'un AEF *(*Automate minimal)

Soit $A = (X, Q, q_0, \delta, F)$ un AEF.

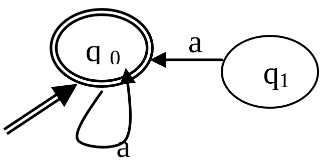
Réduire *A* implique la construction **de l'automate minimal** équivalent.

But : obtenir un automate ayant le minimum d'états possible. En effet, certains états peuvent être équivalents.

Définitions:

a/ On dit que l'état q (q$\in$ Q) est **accessible** si $\exists\, f \in X^* / \delta^*(q_0, f) = q$

Exemple: q_1 est inaccessible.(aucun arc n'arrive sur lui)



b/ On dit que l'état q et p (q,p$\in$ Q) sont $\beta -$*équivalent t* si et seulement si:

$$p\ \beta\ q \Leftrightarrow ((\forall\ f\ \in X^*)\ (\delta^*(p,f)\in F \Leftrightarrow \delta^*(q,f)\in F)$$

c/ Soit une relation *R* sur *Q*. On définit une congruence d'automate sur R par translation de la relation sur les nouveaux états obtenus à travers δ sur les lettres de l'alphabet *X*.

Formellement:

$$p\ R\ q \Rightarrow (\ \forall x\in X,\ \ \delta(p,x)\ \ R\ \ \delta(q,x)\ \ )$$

Théorème:

La relation β est une congruence d'automates.

Démonstration:

Soit $f\in X^*\ ,\ f = xw\ \ avec\ x\in X\ \ et\ \ w\in X^*$

$$p\ \beta\ q \Leftrightarrow (\delta^*(p,f)\in F)\Leftrightarrow\delta^*(q,f)\in F) \Leftrightarrow (\delta^*(p,xw)\in F)\Leftrightarrow\delta^*(q,xw)\in F)$$

$$\Leftrightarrow (\delta^*(p,x),w)\in F)\Leftrightarrow\delta^*(q,x),w)\in F) \Rightarrow (\delta(p,x)\ \ \beta\ \ \delta(q,x))$$

Algorithme de minimisation d'un automate:

Soit *A* automate d'état fini déterministe.

1/ Eliminer tous les états inaccessible dans *A*.

2/ Regrouper les états congruents suivant des classes d'états, à travers la relation de congruence d'automates β.

$\boldsymbol{A = (X, Q, q_0, \delta, F)}$ sans états inaccessibles on obtient :

$$\boldsymbol{A' = (X', Q', q_0', \delta', F')}$$

a/ $p\ \beta_0\ q \Leftrightarrow \begin{cases} P\in F\ et\ \ q\in F \\ P\notin F\ \ et\ q\notin F \end{cases}\ ou$

b/ Si $(p\ \beta\ q)et\ \ (\forall x\in X,\ \delta(P,x)\ \beta_k\ \ \delta(q,x))\, alors : \beta_{k+1} = \beta_k$

c/ Arrêt quand $\beta_{k+1} = \beta_k$

$X' = X$

$q_0' = la\ classe\ qui\ \ contient\ q_0$

$F' = toutes\ \ les\, classes\, qui\ \ contient\ \ des\, états\ \ finaux\ \ de\ \ départ$

$Q' = nouveaux\ noms\, d'\, états\ \ attribués\ \ à\, chaque\ classe\ obtenue$

$\delta' = sera\, obtenu\ \ en\, définissant\ \ chaque\ \ nouvel\ \ état\ sur\ \ X$

Théorème:
A tout AEF déterministe correspond un AEF déterministe minimal.

Exemple: soit l'AEF A({x, y}, {1,2,3,4,5,6,7},1,δ, F) avec F={1, 2} et δ défini par

Q \ X	x	y
état initial: **1**	2	5
2	2	4
3	3	2
4	5	3
5	4	6
6	6	1
7	5	7

1/ Faite le graphe de A

2/ Construire l'automate minimal équivalent à A.

Solution : **1/** dessiner le graphe selon la définition de A (tâche à faire pendant le cours)

2/ Etape 1: l'état 7 est inaccessible, donc on le supprime.

Étape 2: Appliquons l'algorithme de minimisation:

- β_0: {1, 2} ; {3, 4, 5, 6}
- β_1 : Considérons {1, 2}: $\begin{matrix}\delta(1,x)=2\\ \delta(2,x)=2\end{matrix}$ même état $\begin{matrix}\delta(1,y)=5\\ \delta(2,y)=4\end{matrix}$

  $\Rightarrow 1\beta_1 2$

  Considérons {3, 4, 5, 6}: $\begin{matrix}\delta(3,x)=3\\ \delta(4,x)=5\end{matrix}$:3 β_0 5 $\begin{matrix}\delta(3,y)=2\\ \delta(4,y)=3\end{matrix}$ 2 et 3 ne sont pas en relation suivant β_0 et ainsi 3 et 4 ne seront pas dans la même classe suivant β_1

Faire toutes les combinaison possible sur les états, les résultats sont les suivants:

- β_0: {1, 2} ; {3, 4, 5, 6}
- β_1 :{1, 2} ; {3, 6} ; { 4, 5}
- β_2 :{1, 2} ; {3, 6} ; { 4, 5}
- $\beta_1 = \beta_2$ donc Arrêt.

Paramètres de l'automate minimal:

$X' = \{x, y\}$; $Q' = \{S_0, S_1, S_2\}$ Avec

S_0 représentant la classe {1, 2}

S_1 représentant la classe {3, 6}

S_2 représentant la classe {4, 5}

S_0 est l'état initial. δ' défini ainsi:

Q \ X	x	y
état initial: S_0	S_0	S_2
S_1	S_1	S_0
S_2	S_2	S_1

II.8 Automates d'états finis indéterministes

II.8.1 Définition

Soit A un AEF indéterministe; $A = (X, Q, q_0, \delta, F)$

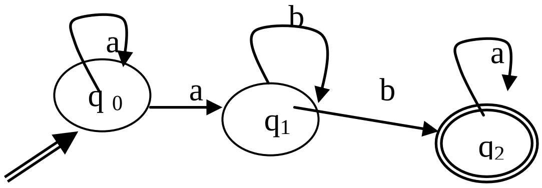
La fonction de transition est défini comme suit:

$$\delta : Q \times X \to P(Q)$$
$$(q, x) \mapsto \{q_i\}$$

Remarquons que lorsque $\delta(q, x) = q'$ nous retrouvons la définition d'un AEF déterministe.

II.8.2 Représentation d'un AEF indéterministe

Exemple:



$A(\{a,b\},\{q_0,q_1,q_2\},q_0,\delta,\{q_2\})$

$\delta(q_0,a)=\{q_0,q_1\},\quad \delta(q_1,b)=\{q_1,q_2\},\ \delta(q_2,a)=q_2$

Q \ X	a	b
état initial: q_0	$\{q_0, q_1\}$	
q_1		$\{q_1, q_2\}$
q_2	q_2	

II.8.3 Fonctionnement d'un AEF indéterministe:

$\delta^*(\sigma,x)=\cup_{q\in\sigma}\delta^*(q,x)\ \ \forall\sigma\in P(Q),x\in X^*$

Exemple:

Le fonctionnement de l'automate précédent pour le mot aab sur l'état q_0 donne l'ensemble $\{q_1, q_2\}$

On a :

$$\delta^*(q_0,aab)=\delta^*[\delta(q_0,a),ab]=\delta^*(\{q_0,q_1\},ab)=\delta^*(q_0,ab)\cup\delta^*(q_1,ab)=\delta^*[\delta(q_0,a),b]\cup\delta^*[\delta(q_1,a),b]$$
$$=\delta^*[\{q_0,q_1\},b]\cup\text{O}=\delta^*(q_0,b)\cup\delta^*(q_1,b)=\delta\ (q_0,b)\cup\delta\ (q_1,b)=q_2\cup\{q_1,q_2\}=\{q_1,q_2\}$$

II.8.4 Langage reconnu par un AEF indéterministe

$L(A)=\{w\in X^*/(q_0,w)\xrightarrow{*}(\sigma,\varepsilon)\}avec\ \sigma\cap F\neq\Phi$

Exemple:

1/ (q_0, aab)$\xrightarrow{*}\{q_1,q_2\}$ $\{q_1,q_2\}\cap F\neq\Phi$ *donc* $aab\in L(A)$
2/ (q_0, aa)$\mapsto(\{q_1,q_2\},a)\mapsto(q_0,a)\cup(q_1,a)\mapsto\{q_0,q_1\}\cup\Phi$ et $\{q_0,q_1\}\cap F=\Phi$ *donc* $aa\notin L(A)$

3/ (q_0, abb)

II.8.5 Relation entre les AEF déterministes et indéterministes

Pour tout automate fini non déterministe, il est possible de construire un automate fini déterministe équivalent (c'est-à-dire qui accepte le même langage).

- Entrée: $A = (X, Q, q_0, \delta, F)$
- Sortie: $A' = (X', Q', q_0', \delta', F')$

Déterminons les différents paramètres de A'

$X' = X$

$q_0' = \{q_0\}$

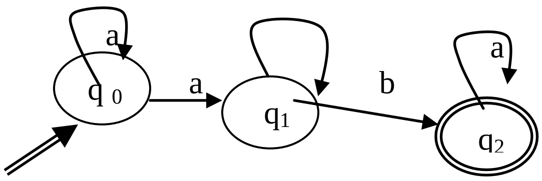
$F' = \{S \in P(Q) / S \cap F \neq \phi\}$

$Q' = P(Q)$

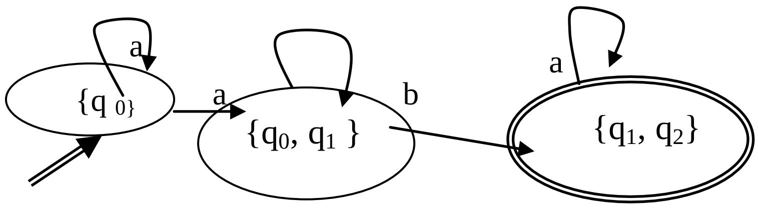
$\delta' = P(Q)\, x\, X \rightarrow P(Q)$

$(\sigma, x) \mapsto \{\delta(S,x) / \delta(S,x) \in Q\ \forall S \in \sigma\}$

Exemple: Reprenons l'exemple précédent:



L'automate déterministe correspond est :



II.8.6 Déterminisation d'un AFN sans ε-transitions

Principe : considérer des ensembles d'états plutôt que des états.

1. Partir de l'état initial
2. Rajouter dans la table de transition tout les nouveaux "états" produits, avec leur transition
3. Recommencer 2 jusqu'à ce qu'il n'y ait plus de nouvel "état"
4. Tous les "états" contenant au moins un état terminal deviennent terminaux
5. Renuméroter alors les états.

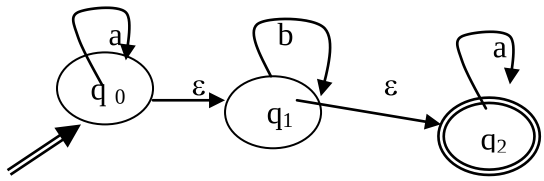
II.9 Automates d'états finis indéterministes avec ε-transitions

Définition: *ε-transition*

On appelle ε-transition, une transition par le symbole ε entre deux états.

Remarque: un automate d'états fini ne possède pas de ε-transition.

$A = (X, Q, q_0, \delta, F)$ $\delta' = Q_0(X \cup \{\varepsilon\}) \to P(Q)$



Q \ X	a	b	ε
état initial: q_0	$\{q_0\}$		$\{q_1\}$
q_1		$\{q_1\}$	$\{q_2\}$
q_2	$\{q_2\}$		

Définition: *ε-fermeture*

On appelle ε-fermeture de l'ensemble d'états Q l'ensemble des états accessibles depuis un état q_i de Q par des ε-transitions.

$\varepsilon - fermuture(P) = \bigcup_{q \in p} \varepsilon - fermuture(q)$

Calcul de l' ε-fermeture de

$T = \{e_1, \ldots, e_n\}$:

Mettre tous les états de T dans une pile P
Initialiser ε-fermeture(T) à T
Tant que P est non vide faire
Soit p l'état en sommet de P
dépiler P
Pour chaque état e tel qu'il y a une ε-transition entre p et e faire

Si *e* n'est pas déja dans ε-fermeture(*T*)
ajouter *e* à ε-fermeture(*T*)
empiler *e* dans *P*
finsi
finpour
fin tantque

Exemple:

Soit l'AFN

état	a	b	c	ε
0	2	-	0	1
1	3	4	-	-
2	-	-	1,4	0
3	-	1	-	-
4	-	-	3	2

e_0=0

On a $\varepsilon - fermuture(\{0\}) = \{0,1\}$, $\varepsilon - fermuture(\{1,2\}) = \{1,2,0\}$, $\varepsilon - fermuture(\{3,4\}) = \{3,4,0,1,2\}$,..

II.10 Déterminisation d'un AFN qui contient ε-transitions

1. Partir de l'ϵ-fermeture de l'état initial
2. Rajouter dans la table de transition toutes les ϵ-fermetures des nouveaux "états" produits, avec leurs
transitions
3. Recommencer 2 jusqu'à ce qu'il n'y ait plus de nouvel "état"
4. Tous les "états" contenant au moins un état terminal deviennent terminaux
5. Renuméroter alors les états.

Exercices de TD

Exercice 1: ***Automates à construire***

Construire, si possible, les automates déterministes qui reconnaissent les langages suivants sur l'alphabet {a, b}:

- Tous les mots sans b.
- Tous les mots qui se terminent par ab.
- Tous les mots dans lesquels chaque a et suivi d'un b.
- Tout les mots qui contiennent autant de a que de b.

Exercice 2: ***Automates à construire***

Construire les automates qui reconnaissent les langages suivants sur l'alphabet {0,1}:

- $(00 + 01)^*$
- $0(10 + 01)^*$

Exercice 3: ***Automates et Arithmétique Décimale***

Construire les automates sur l'alphabet *{*0, 1, 2, 3*, 4, 5, 6, 7, 8,* 9*}* qui acceptent tous les entiers naturels représentés en système décimal qui sont:

1. multiples de 5;

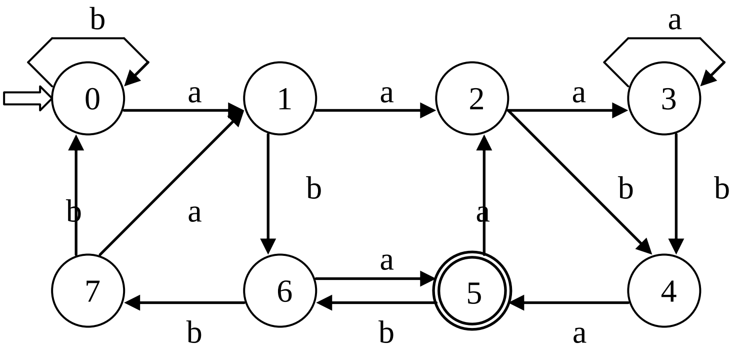
2. multiples de 3;

Indication: Un nombre décimal est multiple de 3 si et seulement si la somme de ses chiffres est multiple de 3.

Exercices de TD

Exercice 1: ***Automate minimal***

Soit l'automate suivant, définit sur *X={a, b}* :



1. Construire l'automate minimal.

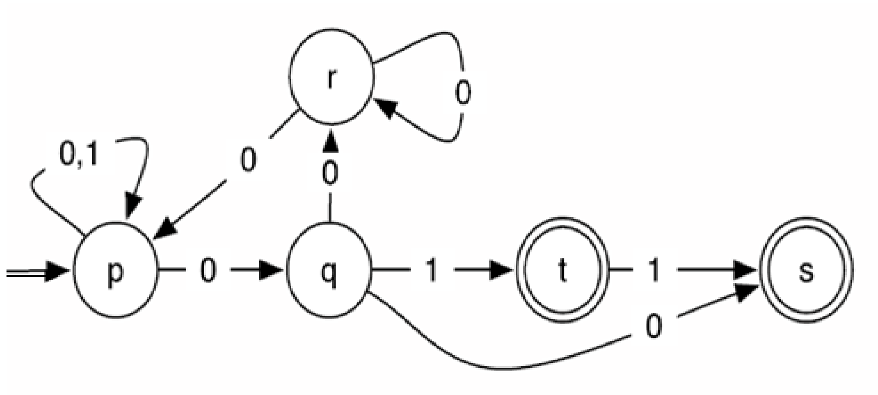
Exercice 2: ***Automates non déterministes & déterministes***

1. Construire un automate reconnaissant tous les mots qui finissent par aba.

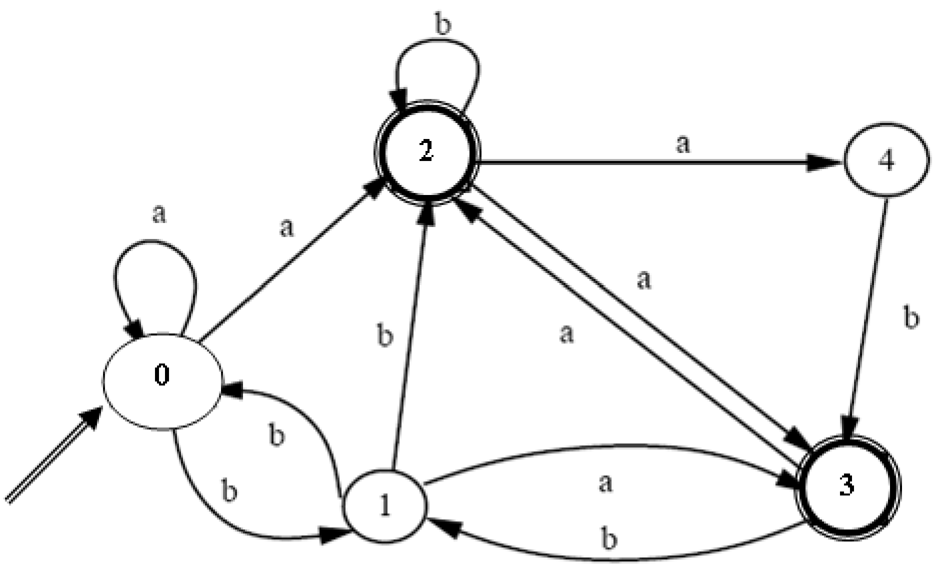
2. Déterminiser l'automate obtenu.

Exercice 3: ***Automates non déterministes & déterministes***

1. Rendre les automates suivants déterministes:

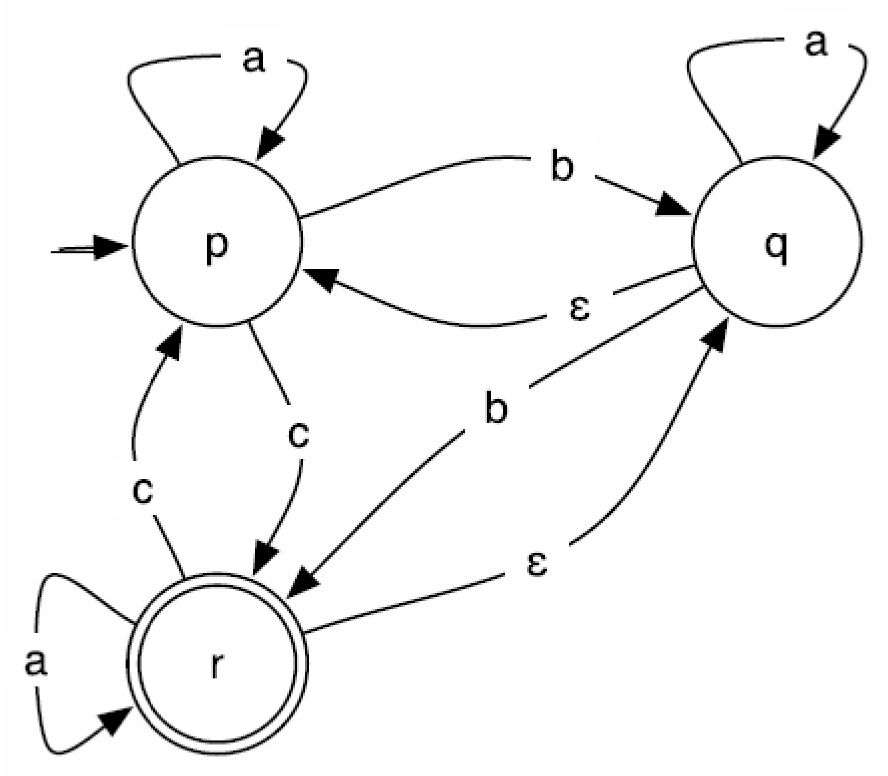


2.



Exercice 4: ***ε-transition***

1. Déterminiser l'automate suivant:



Exercice 5: ***Automates et Arithmétique***

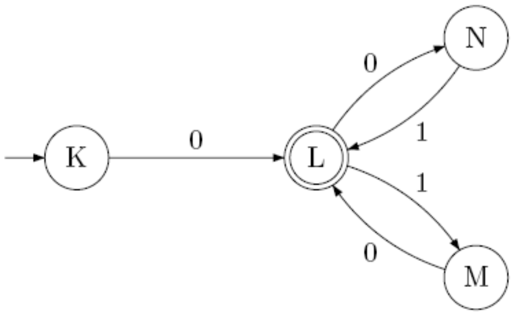
1. Pour le langage $M = \overline{(a^3)^*(a^4)^*}$ sur l'alphabet *{a}* construire un automate qui le reconnaisse.

2. Est-ce que *M* est vide, non-vide et fini, ou bien infini? Si le langage *M* est fini, donner la liste de tous ses mots.

3. Appliquer ce résultat pour trouver tous les entiers naturels non représentables sous la forme $3m + 4n$ avec $m, n \in \mathbb{N}$.

Exercice 6: ***Programmation d'un automate à'état fini***

1. Ecrire un programme (en C ou en Pascal) simulant l'AEF suivant:



2. Est ce que ce programme accepte le mot 000?

3. Ecrire un programme qui implémente l'AEF représentant les nombres entiers.

Chapitre III: Langages réguliers

III.1. Langage régulier

Définition: Un langage L est dit régulier s'il est accepté par un automate d'états fini A.

Définition formelle:

L régulier $\Leftrightarrow \exists\ A \,/\, L = L(A)$

III.2. Expression régulière (ER)

Soit X un alphabet quelconque ne contenant pas les symboles$\{*, +, |, ., (,)\}$.

Une expression régulière est un mot défini sur l'alphabet $X \cup \{*, +, |, ., (,)\}$ permettant de représenter un langage régulier de la façon suivante :

- L'expression régulière ε dénote le langage vide ($L = \{\varepsilon\}$);
- L'expression régulière a ($a \in X$) dénote le langage$L = \{a\}$;
- Si r est une expression régulière qui dénote L alors $(r)^*$ (resp. $(r)^+$) est l'expression régulière qui dénote L^* (resp. L^+);
- Si r est une expression régulière dénotant L et s une expression régulière dénotant L' alors $(r)|(s)$ est une expression régulière dénotant $L + L'$. L'expression régulière $(r).(s)$ (ou simplement $(r)(s)$) dénote le langage $L.L'$.

Les expressions régulières sont également appelées expressions rationnelles. L'utilisation des parenthèses n'est pas obligatoire si l'on est sûr qu'il n'y ait pas d'ambiguïté quant à l'application des opérateurs $*, +, |, .$. Par exemple, on peut écrire $(a)^*$ ou a^* puisque l'on est sûr que $*$ s'applique juste à a. Par ailleurs, on convient à utiliser les priorités suivantes pour les différents opérateurs : 1)$*, +$, 2). et 3)|.

Exemple :

1. a^* : dénote le langage régulier a^n $(n \geq 0)$;

2. $(a|b)^*$: dénote les mots dans lesquels le symbole a ou b se répètent un nombre quelconque de fois. Elle dénote donc le langage de tous les mots sur $\{a, b\}$;

3. $(a|b)^* ab(a|b)^*$: dénote tous les mots sur $\{a, b\}$ contenant le facteur ab.

III.3 Utilisation des expressions régulières

Les expressions régulières sont largement utilisées en informatique. On les retrouve plus particulièrement dans les *shell* des systèmes d'exploitation où ils servent à indiquer un ensemble de fichiers sur lesquels on est appliqué un certain traitement. L'utilisation des expressions régulières en DOS, reprise et étendue par WINDOWS, est très limitée et ne concerne que le caractère "*" qui indique zéro ou plusieurs symboles ou le caractère " ?" indiquant un symbole quelconque. Ainsi, l'expression régulière "f*" indique un mot commençant par f suivi par un nombre quelconque de symboles, "*f*" indique un mot contenant f et "*f*f*" indique un mot contenant deux f. L'expression "f ?" correspond à n'importe quel mot de deux symboles dont le premier et f. Le tableau suivant résume l'utlisation des expressions régulières.

Expression	Signification
$[abc]$	les symboles a,b ou c
$[\hat{}abc]$	aucun des symboles a, b et c
$[a - e]$	les symboles de a jusqu'à $e(a, b, c, d, e)$
.	n'importe quel symbole sauf le symbole fin de ligne
$a*$	a se répétant 0 ou plusieurs fois
$a+$	a se répétant 1 ou plusieurs fois
$a?$	a se répétant 0 ou une fois
$a\|bc$	le symbole a ou b suivi de c
$a\{2,\}$	a se répétant au moins deux fois
$a\{,5\}$	a se répétant au plus cinq fois
$a\{2,5\}$	a se répétant entre deux et cinq fois
$\backslash x$	La valeur réelle de x (un caractère spécial)

Exemples:

- $[\hat{}ab]*$: les mots qui ne comportent ni a ni b
- $[ab]*$: tous les mots sur $\{a, b\}$
- $([\hat{}a] * a[\hat{}a] * a[\hat{}a]*)*$ les mots comportant un nombre pair de a
- $(ab\{,4\})*$ les mots commençant par a où chaque a est suivi de quatre b au plus.

III.4 Expressions régulières ambiguës

Définition: Une expression régulière est dite ambiguë s'il existe au moins un mot pouvant être mis en correspondance avec l'expression régulière de plusieurs façons.

Cette définition fait appel à la correspondance entre un mot et une expression régulière. Il s'agit, en fait, de l'opération qui permet de dire si le mot appartient au langage décrit par l'expression régulière. Par exemple, prenons l'expression régulière a^*b^*. Soit à décider si le mot aab est décrit ou non par cette expression. On peut écrire:

$$\underbrace{aa}_{a^*}\,\underbrace{b}_{b^*}$$

Ainsi, le mot est décrit par cette E.R. Il n'y a qu'une seule façon qui permet de le faire correspondre. Ceci est valable pour tous les mots de ce langage. L'E.R n'est donc pas ambiguë. Considérons maintenant l'expression $(a|b)^*a(a|b)^*$ décrivant tous les mots sur $\{a, b\}$ contenantle facteur a. Soit à faire correspondre le mot aab, on a :

Il existe donc au moins deux façons pour faire correspondre aab à l'expression précédente, elle est donc ambiguë. L'ambiguïté pose un problème quant à l'interprétation d'un mot. Par exemple, supposons que, dans l'expression $(a|b)^*a(a|b)^*$, l'on veut comparer la partie à gauche du facteur a à la partie droite du mot. Selon la méthode de correspondance, le résultat est soit vrai ou faux ce qui est inacceptable dans un programme cohérent.

Comment lever l'ambiguïté d'une E.R?

Il n'existe pas une méthode précise pour lever l'ambiguïté d'une E.R. Cependant, on peutdire que cette opération dépend de ce que l'on veut faire avec l'E.R ou plutôt d'une *hypothèsede reconnaissance*. Par exemple, on peut décider que le facteur fixe soit le premier a du mot àreconnaître ce qui donne l'expression régulière : $b^*a(a|b)^*$. On peut également supposer quec'est le dernier a du mot à reconnaître ce qui donne l'expression régulière $(a|b)^*ab^*$.

III.5 Grammaires régulières et les automates à états finis

Le théorème suivant établit l'équivalence entre les AEF, les grammaires régulières et les expressions régulières.

Théorème: (Théorème de Kleene) Soient Λ_{reg} l'ensemble des langages réguliers (générés par des grammaires régulières), Λ_{rat} l'ensemble des langages décrits par toutes les expressions régulières et Λ_{AEF} l'ensemble de tous les langages reconnus par un AEF. Nous avons,alors, l'égalité suivante :

$\Lambda_{reg} = \Lambda_{rat} = \Lambda_{AEF}$

Le théorème annonce que l'on peut passer d'une représentation à une autre du fait del'équivalence entre les trois représentations. Les sections suivantes expliquent comment passer d'une représentation à une autre.

III.5.1 Arbre de dérivation et grammaires régulières

Soit une grammaire $G=(T,N,S,P)$ avec des productions avec un seul non-terminal par partie gauche.

T: est le vocabulaire terminal,

N: est le vocabulaire non terminal,

$S \in N$: est le symbole de départ ou l'axiome.

P: est un ensemble de règles de production de la forme:

- un arbre de dérivation pour un mot *w* engendré par *G* est un arbre dont :
- la racine est étiquetée par l'axiome S
- les feuilles sont étiquetées par des éléments de $T \cup \{\varepsilon\}$
- les noeuds internes le sont par des éléments de *N*
- un noeud interne étiqueté *B* a des fils étiquetés de gauche à droite

α_1, α_2, $\ldots\alpha_n$, s'il existe dans *P* une production :

$B \rightarrow \alpha_1\alpha_2...\alpha_n$

- *w* est formé de la concaténation des feuilles lues dans un parcours de l'arbre.

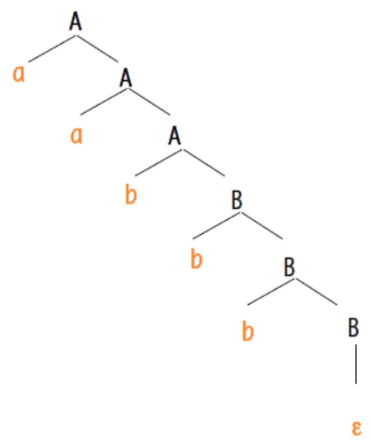
Exemple :

$G=(T,N,S,P)$ avec :

N = { A, B }

T = { a, b }

$$p=\begin{cases} A \rightarrow aA \\ A \rightarrow bB \\ B \rightarrow bB \\ B \rightarrow \varepsilon \end{cases}$$



III.5.2 Grammaire linéaire à droite

Une grammaire $G=(T,N,S,P)$ est régulière à droite si les éléments de P sont de la forme :

$$p=\begin{cases} A\rightarrow \alpha B \\ A\rightarrow \alpha \quad /\alpha \in T^* \end{cases} \quad B\in N, A\in N$$

III.5.3 Grammaire linéaire à gauche

Une grammaire $G=(T,N,S,P)$ est régulière à gauche si les éléments de P sont de la forme :

$$p=\begin{cases} A\rightarrow B\alpha \\ A\rightarrow \alpha \quad /\alpha \in T^* \end{cases} \quad B\in N, A\in N$$

Il existe un algorithme pour passer d'une grammaire régulière à gauche à une grammaire régulière à droite engendrant le même langage.

III.6. Algorithme de passage de l'automate à la grammaire

Pour tout automate, il existe une expression régulière reconnaissant le même langage. L'automate permet d'établir un système d'équations aux langages de la manière suivante :

$A=(X,Q,q_0,\delta,F)$
$G=(T,N,S,P)$
$T=X$
$N=Q$, q_i ,$q_j \in Q$
$S=q_0$

$$p=\begin{cases} Si\ (\delta(q_i,a)=q_j)\ alors\ \ on\ écrit : \\ (q_i\rightarrow aq_j)\in P \\ Si\,(q_i\in F)\,alors\ on\ écrit : \\ (q_i\rightarrow \varepsilon \quad)\in P \end{cases}$$

III.7. Algorithme de passage de la grammaire à l'automate

Pour toute expression régulière, il existe un automate reconnaissant le même langage. Il existe deux méthodes permettant de réaliser cette tâche. La première fait appel à la notion de *dérivée* tandis que la deuxième construit un automate comportant des ∈-transitions en se basant sur les propriétés des langages réguliers.

Méthode de Thompson

La méthode de Thompson permet de construire un automate en procédant à la décomposition de l'expression régulière selon les opérations utilisées. Soit une grammaire régulière à droite *G*, alors l'algorithme à utiliser est le suivant :

$G = (T, N, S, P)$
$A = (X, Q, q_0, \delta, F)$
$T=X,\ F \cup N=Q,\ S=q_0$

$Si : A \rightarrow \alpha\ B$ *alors on écrit* $\delta(A, \alpha) = B$ *avec* $\alpha \in T^*$
$Si : A \rightarrow \alpha$ *alors on écrit* $\delta(A, \alpha) = R$ *avec* $R \in T$
$F = \{R \,/\, \delta(A, \alpha) = R\}$
$Si : \varepsilon \in L(G)$ *alors on écrit* $q_0 \in F$

Remarque : il y a des renommages implicites dans la construction.

III.8. Transformation d'une grammaire linéaire à droite à une grammaire de Kleene

Grammaire de Kleene :

$G = (T, N, S, P)$

$$p = \begin{cases} A \rightarrow aB & avec\ A, B \in N\ \ et\ a \in T \\ A \rightarrow a \end{cases}$$

$Si\ \varepsilon \in L(G) \Rightarrow (S \rightarrow \varepsilon) \in P$

Transformation :

$G = (T, N, S, P) \rightarrow G'(T', N', S', P')$

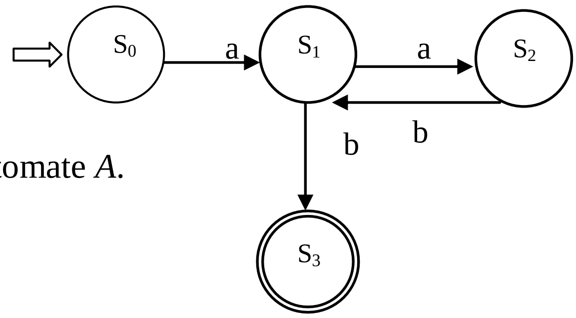
$T' = T, N' = N \cup \{A_i\}, S' = S$

$Si\ \ A \rightarrow \alpha_1 \alpha_2 ... \alpha_n B\ \ alors$

$$p \begin{cases} A \rightarrow \alpha_1 A_1 \\ A_1 \rightarrow \alpha_2 A_2 \\ . \\ . \\ . \\ A_{n-1} \rightarrow \alpha_n B \end{cases}$$

$Si\, \varepsilon \in L(G) \rightarrow A \rightarrow \varepsilon$

Exercices de TD



Exercice 1: ***Passage de l'automate à la grammaire***

Soit l'automate *A* suivant, définit sur $X=\{a, b\}$:

1. Donner la grammaire linéaire à droite qui correspond à l'automate *A*.

2. Donner le langage engendré par la grammaire obtenue.

Exercice 2: ***Passage de la grammaire à l'automate***

1. Donner l'automate d'états finis qui correspond à la grammaire suivante:

$$G = (\{a,b\},\{S,B\},S,P)$$

$$p:\begin{cases} S \to aB \\ B \to abB / b \end{cases}$$

2. Construire l'automate pour la grammaire suivante :

$$p:\begin{cases} S \to aB \\ S \to bA \\ B \to bB / \varepsilon \\ A \to aA / \varepsilon \end{cases}$$

Exercice 3: ***Grammaire de Kleene***

1. Construire l'automate pour la grammaire suivante :

$$p:\begin{cases} S \to abaaA \\ S \to aabb \\ A \to abS \\ A \to aaa \end{cases}$$

Exercice 4: Langages Réguliers

Répondre par: vrai ou faux ?

a. Il existe un nombre fini de langages réguliers.

b. Tout langage fini est régulier.

c. Si le complément de *L* est fini, alors *L* est régulier.

d. Les deux énoncés suivants sont logiquement équivalents :

• *L* est régulier ;

• *L* est reconnu par un automate d'état fini.

e. Le langage: $\{0^n1^{2n} \mid 0 < n < 1000$ et n est pair$\}$ est régulier.

f. La classe des langages réguliers est fermée pour l'union, l'intersection et le complément.

G. Si L2 est régulier, alors tout langage L1 tel que $L1 \subseteq L2$ est régulier.

Exercice 5: ***Expression régulière***

1. Donner une grammaire régulière reconnaissant l'expression régulière:

a. $aab(a/b)^* bb(ab/ba)^+$

b. $(abbc/baba)^+ aa (cc/bc)^*$

Chapitre IV: Langages algébriques

IV.1. Introduction

Les langages algébriques représentent la couche qui suit immédiatement celle des langages réguliers dans la hiérarchie de Chomsky. Remarquons, cependant, que le niveau de complexité est inversement proportionnel au type du langage et, par conséquent, le nombre d'algorithmes existants tend à diminuer en laissant la place à plus d'intuition.

IV.2. Définition des grammaires hors-contextes

Soit une grammaire $G = (T, N, S, R)$

- T : symboles terminaux
- N : symboles non-terminaux
- $S \in N$: axiome (symbole de départ)

La grammaire G est non-contextuelle (context-free in english), ou algébrique, si les productions sont de la forme $R \subset N \times (N \cup T)^*$: règles

Une règle s'écrit $A \to \alpha$ avec $A \in N$ et $\alpha \in (N \cup T)^*$

Des règles $A \to \alpha$ et $A \to \beta$ s'écrivent $A \to \alpha|\beta$

Exemple: Expressions mathématiques

- $N = \{S, E\}$ et $T = \{+,*,\div,\sqrt{\ }, (,), 1,2,3 \ldots\}$
- Règles :

$$S \to E$$
$$E \to E + E$$
$$E \to E * E$$
$$E \to E \div E$$
$$E \to (E)$$
$$E \to \sqrt{E}$$
$$E \to 1|2|3 \ldots$$

➢ Une dérivation possible :

$$S \to E \to E + E \to E \div E + E \ldots \to 1 \div 2 + 3 * \sqrt{9}$$

IV. 2 .1. Dérivations

La derivation est l'opérations qui génèrent le langage pour une grammaire.

- Un mot$\alpha \in (N \cup T)^*$se dérive en un mot $\beta \in (N \cup T)^*$si
  - α se décompose en $\alpha_1 A \alpha_2$avec $A \in N$
  - βse décompose en $\alpha_1 \gamma \alpha_2$ avec $\gamma \in (N \cup T)^*$
  - $A \to \gamma \in R$(c'est une règle)
- Exemple :$E + E \div E \to E + E * E \div E$
  - $\alpha_1 = E +$
  - $\alpha_2 = \div E$
  - $A = E$
  - $\gamma = E * E$
  - $E \to E * E \in R$

Suite de dérivations

- Par transitivité
  - Chaîne de dérivations $\alpha \to \beta \ldots \to \gamma = \alpha \overset{*}{\to} \gamma$
  - Fermeture transitive, clôture (étoile de Kleene)
  - Si$\gamma \in (N \cup T)^*$alors γ est une proto-phrase de G
- Ordre des dérivations
  - Possibilité d'analyses pour $1 + 2 + 3$
    - Dérivation gauche : réécrit le non-terminal le plus à gauche

      $E \to E + E \to 1 + E \to 1 + E + E \to 1 + 2 + E \to 1 + 2 + 3$
    - Dérivation droite : réécrit le non-terminal le plus à droite

      $E \to E + E \to E + 3 \to E + E + 3 \to E + 2 + 3 \to 1 + 2 + 3$

IV. 2.2. Langage généré

Un langage généré par une grammaire hors-contexte est dit langage hors-contexte. Notons que nous nous intéressons, en particulier, à ce type de langages du fait que la plupart des langages de programmation sont hors-contextes.

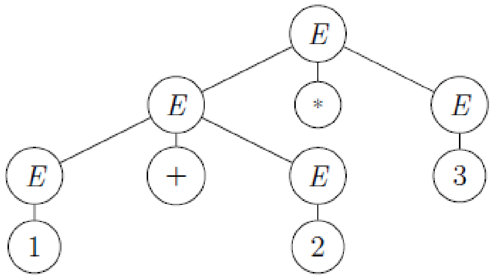
Soit G une grammaire, alors le langage généré par G est$L(G) = \{m \in T^* | S \overset{*}{\to} m\}$.

IV. 2.3. Arbre de dérivation

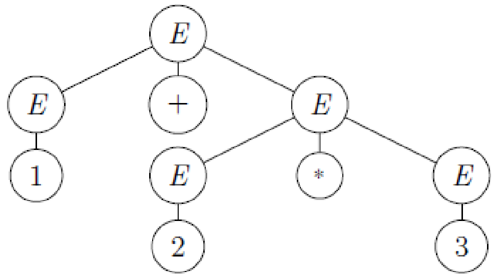
Une représentation graphique de la derivation est définit comme suit:

- Racine : symbole initial $= S$
- Nœud : symbole non-terminal $\in N$
- Feuille : symbole terminal$\in T$
- Relation parent-enfants : dérivation (règle)

➢ Les deux arbres suivants illustrent la dérivation à droite et à gauche de $1 + 2 * 3$



dérivation gauche



dérivation droite

IV. 2.4. Notion d'ambiguïté

Une grammaire est dite ambiguë si elle peut générer au moins un mot de plus d'une manière. En d'autres termes, si on peut trouver un mot généré par la grammaire et possédant au moins deux arbres de dérivation, alors on dit que la grammaire est ambiguë.

La grammaire de l'exemple précedent est ambiguë car le mot $1 + 2 * 3$possède deux arbres de dérivation. D'une manière générale, pour lever l'ambiguïté d'une grammaire, il n'y a pas de méthodes qui fonctionnent à tous les coups. Cependant, l'idée consiste généralement à introduire une hypothèse supplémentaire (ce qui va changer la grammaire) en espérant que le langage généré soit le même.

IV.3 Simplification des grammaires hors-contextes

Une grammaire est propre si elle est :

–ϵ-*libre*,

– dépourvue de symboles inutiles,

– sans cycle

ϵ-*libre* signifie qu'il n'y a pas de production donnant ϵ, par symbole inutile on entend à la fois ceux qui n'ont pas de contribution et ceux qui sont inaccessibles. Les cycles impliquent des productions singulières qui peuvent engendrer des boucles inutiles dans une dérivation.

- ***Symboles improductifs***
    - A est improductif s'il n'y a pas de $m \in T^*$ tel que $A \xrightarrow{*} m$
- ***Symboles inaccessibles***
    - A est inaccessible s'il n'y a pas de α et β tel que $S \xrightarrow{*} \alpha A \beta$
- ***ϵ-productions***
    - Une ϵ-production est une dérivation telle que $A \xrightarrow{*} \epsilon$

- ***Production simple***
    - $A \rightarrow B$ est une production simple si $A \in N$ et $B \in N$

Pour toute grammaire, il existe une grammaire équivalente sans symboles improductifs ni inaccessibles, sans ϵ-productions ni productions simples. En effet, on procède comme suit afin de nettoyer la grammaire:

1. ***Élimination des symboles improductifs***

➢ Calcul des symboles productifs

- Soit $P_0 = \emptyset$ et $i = 1$
- Soit $P_1 = \{A \in N, \exists \alpha \in T^*, A \rightarrow \alpha \in R\}$
- Tant que $P_i \neq P_{i-1}$
    - $P_{i+1} = P_i \cup \{A \in N, \exists \alpha \in (T \cup P_i)^*, A \rightarrow \alpha \in R\}$
    - $i \leftarrow i + 1$

Les symboles de $N \backslash P$ sont improductifs. On enlève, donc, ces symboles et les règles dans lesquels ils figurent.

2. ***Élimination des symboles inaccessibles***

➢ Calcul des symboles accessibles

- Soit $C_0 = \emptyset$, $C_1 = \{S\}$ et $i = 1$
- Tant que $C_i \neq C_{i-1}$
    - $C_{i+1} = C_i \cup \{A \in N, \exists \alpha, \beta \in (N \cup T)^*, X \in C_i, X \rightarrow \alpha A \beta \in R\}$

Les symboles de $N \backslash C$ sont inaccessible. On enlève, donc, ces symboles et les règles dans lesquels ils figurent. Une grammaire sans symboles improductifs et sans symbole inaccessible est dite grammaire réduite.

3. *Élimination des ϵ-productions*

- Calcul des symboles annulables
  - Soit $U_0 = \emptyset$ et $i = 1$
  - Soit $U_1 = \{A \in N, A \to \epsilon \in R\}$
  - Tant que $P_i \neq P_{i-1}$
    - $U_{i+1} = U_i \cup \{A \in N, \exists \alpha \in (U_i)^*, A \to \alpha \in R\}$
    - $i \leftarrow i + 1$

Les symboles de U sont annulables. On modifie les productions contenant des variables annulables .

- Modification de la grammaire
  - Remplacer les règles $A \to \alpha X \beta$ où $X \in U$ par $A \to \alpha X \beta | \alpha \beta$ (avec combinaisons possibles de X dans les règles)
  - Supprimer toutes les règles $A \to \epsilon$ (sauf pour S)
  - Supprimer toutes les règles $A \to A$

La grammaire ainsi obtenue est équivalente à la grammaire de depart (au mot vide près éventuellement).

4. *Équivalences et productions simples*

- Productions simples, dérivations et classes d'équivalences
  - Production simple : toute règle $A \to B$ avec $B \in N$
  - Soit la relation$\geq$ telle que $A \geq B$ si $A \overset{*}{\to} B$
  - Soit la relation$\approx$telle que $A \approx B$ si $A \geq B$ et $B \geq A$
  - Classes d'équivalences
    - Si $A \approx B$, tout ce qui est dérivé de A peut l'être de B
    - Relation réflexive, symétrique et transitive
    - L'ensemble des classes est une partition de N
- Modification de la grammaire
  - On conserve les productions non-simples
  - Pour chaque classe d'équivalence

    $\Rightarrow$Choisir un symbole qui remplace tous les autres

  $\Rightarrow$Pour chaque dérivation $A \overset{*}{\to} B$

- Pour chaque $B \to \beta$, ajouter $A \to \beta$

Exemple : Simplifier la grammaire suivante:

$G = (\{a, b, c\}, \{S, T, U, V, W, Z, X\}, S, P)$

1. $S \to T|U$
2. $U \to aYb|V$
3. $V \to W$
4. $X \to W|a$
5. $Y \to Z$
6. $Z \to c|\epsilon$

- Symboles productifs :$\{X, Z, Y, U, S\}$ ⇒retirer T, V et W

1. $S \to U$
2. $U \to aYb$
3.
4. $X \to a$
5. $Y \to Z$
6. $Z \to c|\epsilon$

- Symboles accessibles :$\{S, U, Y, Z\}$ ⇒ retirer X

1. $S \to U$
2. $U \to aYb$
3.
4.
5. $Y \to Z$
6. $Z \to c|\epsilon$

- ϵ-productions :$\{Z, Y\}$ ⇒ modifier 6, 2

1. $S \to U$
2. $U \to aYb|ab$
3.
4.
5. $Y \to Z$
6. $Z \to c$

- Productions simples :$S \to U$ et $Y \to Z$ ⇒ modifier 1, 2, 5, 6

1. $S \to aYb|ab$
2.
3.
4.
5. $Y \to c$

La grammaire obtenue est la suivante:

$G = (\{a, b, c\}, \{S, Y\}, S, P')$

1. $S \to aYb|ab$
2. $Y \to c$

IV.4 . Forme normale de Chomsky

Théorème: Pour tout langages hors-contexte il existe une grammaire en forme normale de Chomsky qui le génèrent.

$G = (T, N, S, R)$ est sous la forme normale de Chomesky (FNC) si toutes règles de la production sont de la forme:

$A \to BC$ avec $A, B, C \in N$

$A \to a$ avec $a \in T$

IV.4 .1 Mise sous forme normale de Chomsky

L'intérêt de la forme normale de Chomsky est que les arbres de dérivations sont des arbres binaires ce qui facilite l'application de pas mal d'algorithmes. Il est toujours possible de transformer n'importe quelle grammaire hors-contexte pour qu'elle soit sous la forme normale de Chomsky. Notons d'abord que si la grammaire est propre, alors cela facilitera énormément la procédure de transformation. Supposons que $G = (T, N, S, R)$ est une grammaire proper, on la tronsforme en une grammaire que $G' = (T', N', S', R')$ sous FNC comme suit:

1. Pour chaque terminal a, créer
   - Un symbole Z_a
   - Une règle $Z_a \to a$
2. Pour chaque règle $A \to \alpha$ où $|\alpha| > 1$
   - Tout terminal a de α est remplacé par Z_a
3. Pour chaque règle $A \to \alpha$ où $|\alpha| > 2$
   a. On décompose :$\alpha = A_1, A_2 \dots A_n$
   b. On crée les non-terminaux $Y_1, Y_2 \dots Y_{n-2}$
   c. On remplace $A \to \alpha$ par

   $A \to A_1Y_1$

   $Y_1 \to A_2Y_2$

   …

   $Y_{n-2} \to A_{n-1}A_n$

Exemple:

$S \rightarrow aB|bA \quad A \rightarrow a|aS|bAA \;\; B \rightarrow b|bS|aBB$

devient :

$S \rightarrow Z_aB|Z_bA$

$Z_b \rightarrow b$

$Z_a \rightarrow a$

$A \rightarrow a|Z_aS|Z_bX$

$X \rightarrow AA$

$B \rightarrow b|C_bS|Z_aY$

$Y \rightarrow BB$

IV.5 . La forme normale de Greibach

Soit *G = (T,N, S, R)* une grammaire hors-contexte. On dit que *G* est sous la *forme normale* de *Greibach* si toutes ses règles sont de l'une des formes suivantes :

$A \rightarrow aA_1A_2 \dots A_n, a \in T, A_i \in N - \{S\}$

$A \rightarrow a, a \in T$

L'intérêt pratique de la mise sous forme normale de *Greibach* est qu'à chaque dérivation, on détermine un préfixe de plus en plus long formé uniquement de symboles terminaux. Cela permet de construire plus aisément des analyseurs permettant de retrouver l'arbre d'analyse associé à un mot généré.

IV.5 .1 Récursivité

Théorème (Élimination de la récursivité directe à gauche) : Tout langage non contextuel sans le mot vide peut être engendré par une grammaire sans symbole inutile ni production vide ni production unitaire ni récursivité directe à gauche.

- Symbole récursif : $A \overset{*}{\rightarrow} \alpha A\beta$
    4. Si $\alpha = \epsilon$, A est récursif à gauche
    5. Si $\beta = \epsilon$, A est récursif à droite
    6. Si $\overset{*}{\rightarrow}$ne comporte qu'une dérivation : récursivité directe
    7. Si $\overset{*}{\rightarrow}$comporte plusieurs dérivations : récursivité indirecte

Une grammaire récursive comporte un symbole récursif.

Exemple : grammaire indirectement récursive à gauche

- $A \to B$
- $B \to CD$
- $C \to AE$

Suppression de la récursivité directe à gauche

- Remplacer toute règle $A \to Aa|b$
    - $A \to bA'$
    - $A' \to aA^{'}|\epsilon$

Exercices de TD

Exercice 1: ***Grammaire réduite : élimination des symboles improductifs & des symboles inaccessibles***

1. Construire une grammaire réduite équivalente à la grammaire suivante :

$G = (\{a,b\}, \{A, B, C, D, E, F, G, H\}, A, P)$

$$P : \begin{cases} A \to AB / B / C \\ B \to bB / aDF / \varepsilon \\ C \to bC / abE / \varepsilon \\ D \to bB / CD \end{cases}$$

Exercice 2: ***Grammaire propre : grammaire sans cycle & libre de*** ε

1. Donner une grammaire propre équivalente à la grammaire suivante :

$G = (\{a,b\}, \{S, A\}, S, P)$

$$P : \begin{cases} S \to Ab \\ A \to aA / \varepsilon \end{cases}$$

Exercice 3: ***Simplification de grammaires***

1. Construire une grammaire réduite équivalente à la grammaire suivante :

$S \to AB \mid a$

$A \to a$

2. Montrer que pour toute grammaire algébrique G n'engendrant pas le mot vide, il existe une grammaire algébrique propre et réduite G′ engendrant le même langage.

3. Donner une grammaire propre et réduite équivalente à la grammaire suivante :

$S \to Y\,Y \mid bWTY$

$T \to b \mid Wa$

$Y \to WW \mid Tb$

$W \to \varepsilon \mid aS$

Exercice 4: ***Exemples de langages algébriques***

1. Construire des grammaires pour les langages suivants :

$L_1 = \{a^n\, b^n \mid n \geq 0\}$

$L_2 = \{w \in \{a, b\}^* \,/\, |w|_a = |w|_b\}$

$L_3 = \{a^n\, b^p \mid n \neq p\}$

$L_4 = \{wcw' \,/\, w, w' \in \{a, b\}^* \text{ et } |w| = |w'|\}$

$L_5 = \{ww' \,/\, w, w' \in \{a, b\}^*, |w| = |w'| \text{ et } w' \neq w\}$

Exercice 5: ***Forme normale de Chomsky***

1. Transformez la grammaire hors contexte *G (T, N, S, P)* suivante en FNC :
N = {S, T}
T = {a, b}
$P = \{\ S \rightarrow SSS\,/\,T\,/\,\varepsilon,\ T \rightarrow a\,/\,aT\,/\,bbT\ \ \}$

2. Soit la grammaire hors contexte G = ({S, N, M}, {0, 1, a, b, *}, R, S) définie par les règles suivantes :

$S \rightarrow M * M$

$M \rightarrow a\,/\,b\,/\,N$

$N \rightarrow 0N\,/\,1N\,/\,\varepsilon$

Transformez *G* en *FNC*.

Exercice 6: ***Forme normale de Greibach***

1. Mettre la grammaire suivante sous forme normale de *Greibach* :
$A_1 \rightarrow A_2\ A_3$
$A_2 \rightarrow A3\ A_1 \mid b$
$A_3 \rightarrow A_1\ A_2 \mid a$

2. Mettre la grammaire suivante sous forme normale de *Greibach* :

$S \rightarrow (L)\,/\,a$

$L \rightarrow L, S\,/\,S$

Chapitre V: Automate à pile

V.1 . Introduction

Les langages algébriques sont spécifiés par des grammaires algébriques. Les automates à pile sont nécessaires pour reconnaître les langages algébriques.

langage	spécification	modèle exécutable
régulier	expression régulière	AFD
algébrique	grammaire algébrique	automate à pile

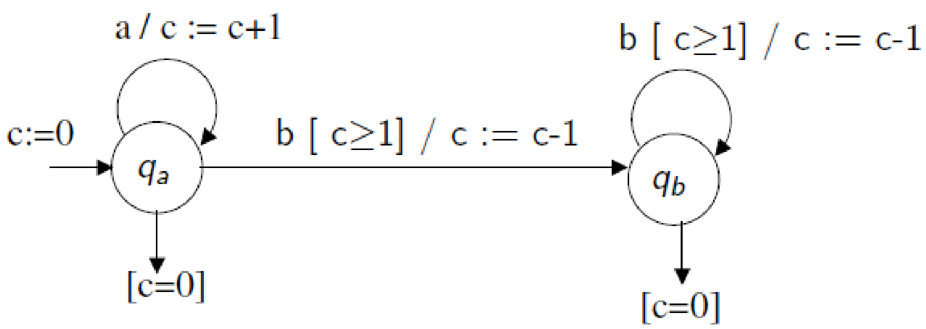
V.2. Automates à pile généraux

Avant de définir les automates à pile, nous présentons quelques exemples pour reconnaitre un langage algébrique.

Exemple 1 :

Pour reconnaitre $\{a^n b^n | n \geq 0\}$:

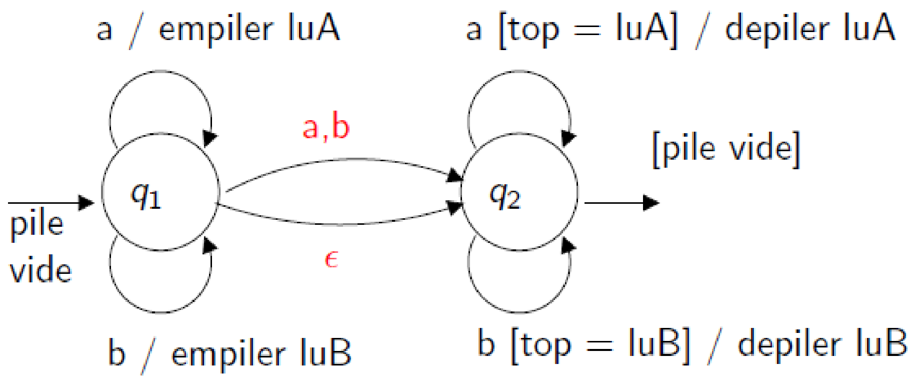
- Un automate à nombre fini d'états pour lire des *a* puis des *b*.
- Un compteur *c* pour compter les *a* et décompter les *b*.
- Arrêt quand le ruban est vide et état final et *c* vaut 0.



Exemple 2 :

Pour reconnaître $\{m \in \Sigma^* | m \; est \; un \; palindrome\}$:

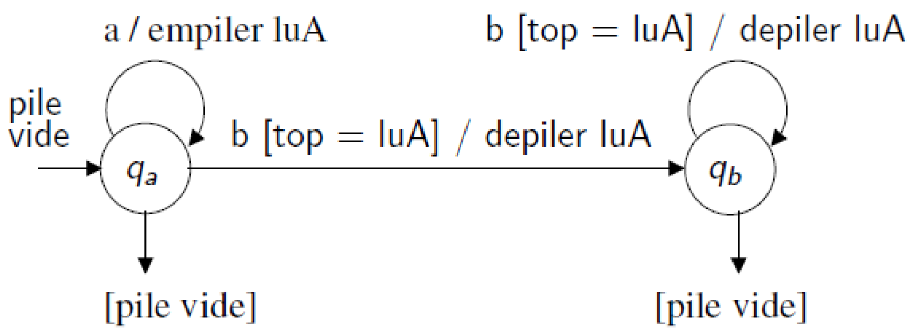
- Un compteur ne suffit pas !
- Il faut mémoriser les symboles lus puis les consulter.
- Mémorisation par empilement, vérification par dépilement.



Exemple 3 :

Soit le langage $\{a^n b^n | n \geq 0\}$:

- On empile *luA* quand on lit un *a*.
- On dépile *luA* quand on lit un *b*.
- Arrêt quand le ruban est vide et état final et la pile est vide.



V.3. Définition d'un automate *à* pile

Automate à nombre fini d'états	**exemple des palindromes :**
ensemble d'états Q	ex : $\{q_1, q_2\}$
états initial q_0	ex : q_1
ensemble d'états finaux $F \subseteq Q$	ex : $\{q_2\}$
alphabet d'entrée Σ	ex : $\{a, b\}$
Automate à Pile	
contient des éléments de l'alphabet de pile Z ex	: $\{luA, luB\}$

Relation de transition :

Pour un AF, une transition c'est:

- Quand on est dans l'état $q \in Q$;
- et que l'on a $a \in \sum$ sous la tête de lecture ;
- ou qu'on transite sur ϵ ;
- alors on passe dans l'état $q' \in Q$.

$$q, a \rightarrow q'$$
$$q, \epsilon \longrightarrow q'$$

Pour un automate à pile, une transition c'est :

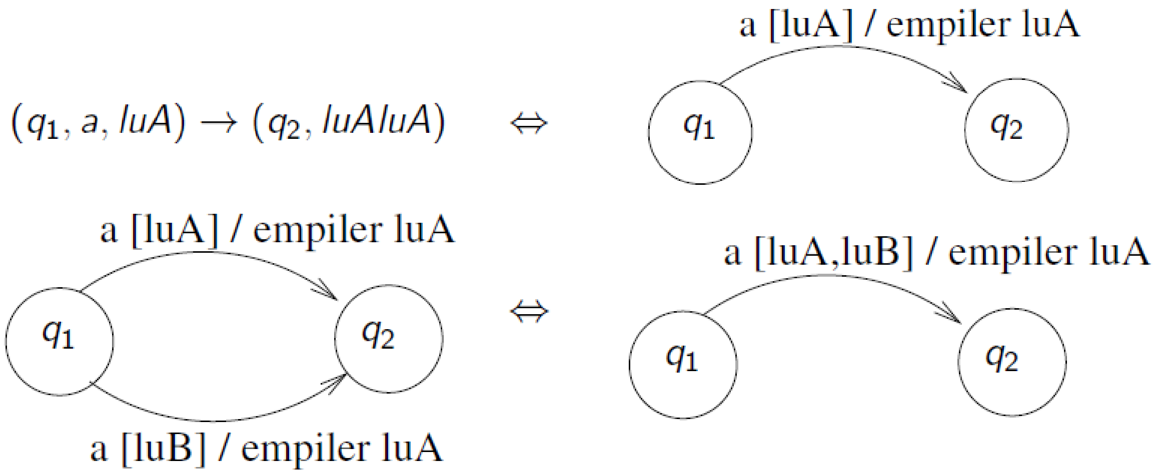
- Quand on est dans l'état $q \in Q$;
- et que l'on a $a \in \Sigma$ sous la tête de lecture ;
- ou qu'on effectue une ϵ-transition ;
- et que le sommet de pile est $z \in Z$;
- On passe dans l'état $q' \in Q$;
- et on modifie le sommet de pile en le remplaçant par des éléments de Z ou ϵ.

$$q, a, z \rightarrow q', z_1 z_2$$
$$q, \epsilon, z \rightarrow q', z$$
$$q, a, z \rightarrow q', \epsilon$$

Modification de la pile:

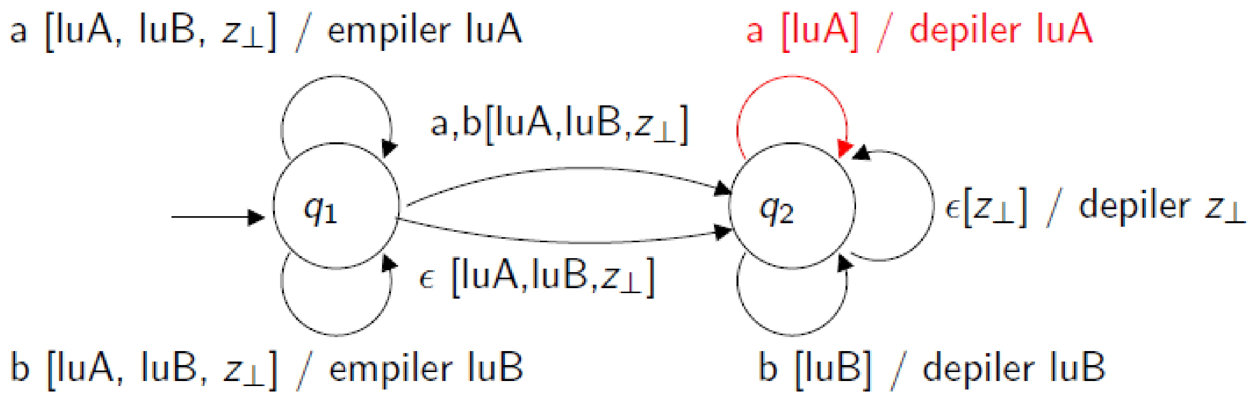
$q, a, z \rightarrow q', z_1 z_2$	Empiler z_2
$q, \epsilon, z \rightarrow q', z$	Ne pas toucher à la pile
$q, a, z \rightarrow q', \epsilon$	Dépiler z

V.3.1 Notation graphique



V.3.2 Exemple des palindromes

L'automate à pile des palindromes est représenté graphiquement par le graphe :



Dans ce qui suit nous expliquons les différentes transitions sur la pile :

- Dans l'état $q \in Q$; avec $a \in \Sigma$ sous la tête de lecture (Σ-transition) ; et avec $luA \in Z$ en sommet de pile ; alors on reste dans l'état $q_2 \in Q$; et on dépile : on remplace luA par ϵ.

$q_2, a, luA \rightarrow q_2, \epsilon$

- Dans l'état $q_1 \in Q$ et avec a sous la tête de lecture ; et quel que soit le sommet de pile: si on viens de lire un a (resp. b) : luA (resp. luB) ; si on n'a encore rien lu : pile initiale (vide). Pas de transition sur pile vide : symbole initial de pile $z_\perp \in Z$

- Dans l'état $q_1 \in Q$, avec a sous la tête de lecture ; et avec luA, luB ou $z_\perp$ en sommet de pile ; alors on reste dans $q_1 \in Q$; et on empile luA : on remplace le sommet x par $x\ luA$

$q_1, a, luA \to q_1, luA\ luA$

$q_1, a, luB \to q_1, luB\ luA$

$q_1, a, z_\perp \to q_1, z_\perp\ luA$

- Dans l'état $q_1 \in Q$; sans toucher la tête de lecture (ϵ-transition) ; et avec luA, luB ou $z_\perp$ en sommet de pile ; alors on passe dans $q_2 \in Q$ et on ne touche pas à la pile.

$q_1, \epsilon, luA \to q_2, luA$
$q_1, \epsilon, z_\perp \to q_2, z_\perp$
$q_1, \epsilon, luB \to q_2, luB$

Pour terminer on vide la pile (ϵ-transition)
$q_2, \epsilon, z_\perp \to q_2, \epsilon$

Récapitulatif:

$q_1, a, luA \to q_1, luA\ luA$	$q_1, a, z_\perp \to q_1, z_\perp\ luA$	$q_1, a, luB \to q_1, luB\ luA$
$q_1, b, luA \to q_1, luA\ luB$	$q_1, b, z_\perp \to q_1, z_\perp\ luB$	$q_1, b, luB \to q_1, luB\ luB$
$q_1, a, luA \to q_2, luA$	$q_1, a, z_\perp \to q_2, z_\perp$	$q_1, a, luB \to q_2, luB$
$q_1, b, luA \to q_2, luA$	$q_1, b, z_\perp \to q_2, z_\perp$	$q_1, b, luB \to q_2, luB$
$q_1, \epsilon, luA \to q_2, luA$	$q_1, \epsilon, z_\perp \to q_2, z_\perp$	$q_1, \epsilon, luB \to q_2, luB$
$q_2, a, luA \to q_2, \epsilon$	$q_2, b, luB \to q_2, \epsilon$	$q_2, \epsilon, z_\perp \to q_2, \epsilon$

V.4. Définition formelle (Automate à pile (AP))

Un automate à pile A est un tuple$(\Sigma, Z, z_\perp, Q, q_0, F, \Delta)$ où :

- Σest un alphabet d'entrée fini (les terminaux);
- Zest un alphabet de pile fini;
- $z_\perp \in Z$est le symbole initial de pile,
- Qest un ensemble fini d'états,
- $q_0 \in Q$est l'état initial;

- $F \subseteq Q$est l'ensemble des états finaux;
- $\Delta \subseteq Q \times (\sum \cup \{\epsilon\}) \times Z \times Q \times Z^*$est la relation de transition.

On pourrait choisir $\Delta \subseteq Q \times (\sum \cup \{\epsilon\}) \times Z^* \times Q \times Z^*$.

V.5. Exécution et configurations

Une exécution est une suite de configurations.

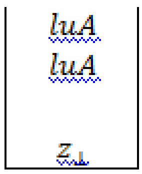
Pour un AF, une configuration est :

- mot restant à lire $m \in \Sigma^*$;
- état courant $q \in Q$.

Pour un AP, configuration définie par :

- le mot restant à lire $m \in \Sigma^*$;
- l'état courant $q \in Q$;
- le contenu de la pile de Z^*, lu du bas vers le haut de la pile.

Exemple: $(abbb, q_1, z_\perp\ luA\ luA)$



V.5.1 Définition (configuration)

Une configuration c d'un $AP(\Sigma, Z, z_\perp, Q, q_0, F, \Delta)$ est un élément de $\Sigma^* \times Q \times Z^*$.

Le passage dans A d'une configuration c_1 à une configuration c_2 s'écrit :

$$c_1 \vdash_A c_2$$

On note $\vdash_A^*$ la clôture réflexive et transitive de $\vdash_A$.

Deux modes de transition pour changer de configuration :

- sur une $\sum$-transition;
- sur une ϵ-transition.

V.5.2 Changement de configuration sur Σ-transition

Exemple :

Transition $q_1, b, luA \rightarrow q_1, luA\ luB$

Configuration $(bba, q_1, z_\perp\ luA)$

On aura alors :

$(bba, q_1, z_\perp\ luA) \vdash_A (ba, q_1, z_\perp\ luA\ luB)$

Définition ($c_1 \vdash_A c_2$ sur Σ-transition)

A passe d'une configuration $c_1 = (m_1, q_1, \alpha_1)$ à $c_2 = (m_2, q_2, \alpha_2)$ si :

- il existe une transition $(q_1, x, z) \rightarrow (q_2, \beta_2) \in \Delta$;
- m_1 est de la forme xm_2 ;
- α_1 est de la forme $\beta_1 z$;
- α_2 est de la forme $\beta_1\beta_2$;

$$\left(xm_2, q_1, \begin{array}{c}\alpha_1\\ \boxed{z}\\ \boxed{\beta_1}\end{array}\right) \vdash_A \left(m_2, q_2, \begin{array}{c}\alpha_2\\ \boxed{\beta_2}\\ \boxed{\beta_1}\end{array}\right)$$

Transition $q_1, b, luA \rightarrow q_1, luA\ luB$

Configuration $(bba, q_1, z_\perp\ luA)$

$$(\overbrace{b\,\underbrace{ba}_{m_2}}^{m_1}, q1, \underbrace{\overbrace{z_\perp}^{\beta_1}\ luA}_{\alpha_1}) \vdash (\underbrace{ba}_{m_2}, q1, \underbrace{\overbrace{z_\perp}^{\beta_1}\ \overbrace{luA\ luB}^{\beta_2}}_{\alpha_2})$$

$$\left(xm_2, q_1, \begin{array}{c}\alpha_1\\ \boxed{z}\\ \boxed{\beta_1}\end{array}\right) \vdash_A \left(m_2, q_2, \begin{array}{c}\alpha_2\\ \boxed{\beta_2}\\ \boxed{\beta_1}\end{array}\right)$$

V.5.3 Définition ($c_1 \vdash_A c_2$ sur ϵ-transition)

A passe d'une config $c_1 = (m, q_1, \alpha_1)$ à $c_2 = (m, q_2, \alpha_2)$ si :

- il existe une transition $(q_1, \epsilon, z) \rightarrow (q_2, \beta_2) \in \Delta$;
- α_1 est de la forme $\beta_1 z$ (z sommet de pile) ;
- α_2 est de la forme $\beta_1\beta_2$.

$$\left(m, q_1, \begin{matrix}\alpha_1\\ \boxed{z}\\ \boxed{\beta_1}\end{matrix}\right) \vdash_A \left(m, q_2, \begin{matrix}\alpha_2\\ \boxed{\beta_2}\\ \boxed{\beta_1}\end{matrix}\right)$$

Exemple :

Transition $q_1, \epsilon, luB \rightarrow q_2, luB$

Configuration $(ba, q_1, z_\perp\ luA\ luB)$

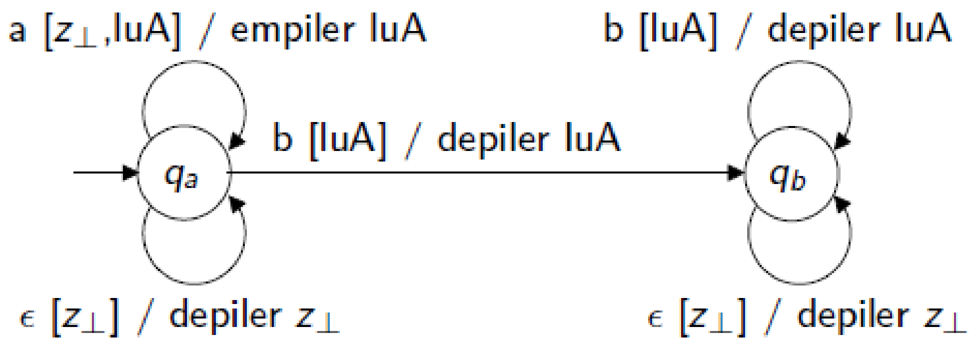
On aura alors :

$(ba, q_1, z_\perp\ luA) \vdash (ba, q_2, z_\perp\ luA\ luB)$

On ne touche pas à la tête de lecture.

Exécution:

Pour le langage $\{a^n b^n | n \geq 0\}$:



$(q_a, aabb, z_\perp) \vdash_A^* (q_b, \epsilon, \ )$
$(q_a, \epsilon, z_\perp) \vdash_A^* (q_a, \epsilon, \ )$

V.6. Les critères d'acceptation

Dans nos exemples, on accepte un mot si ruban vide et pile vide. Ce sont des cas particuliers.

Il y a deux critères d'acceptation possible:

- acceptation par état final (pour toute pile quand on s'arrête) ;
- acceptation par pile vide (pour tout état quand on s'arrête).

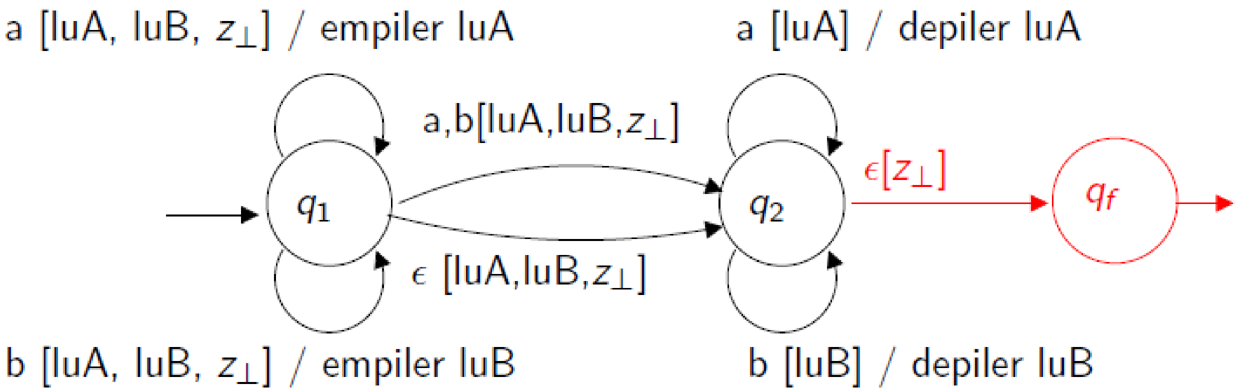
Ces deux critères sont équivalents.

V.6.1 Acceptation par état final

Un mot$m \in \Sigma^*$ est accepté par état final par un AP$A = (\Sigma, Z, z_\perp, Q, q_0, F, \Delta)$si pour la configuration$(m, q_0, z_\perp)$, il existe un état $q_f \in F$et un mot$z \in Z^*$tel que$(m, q_0, z_\perp) \vdash_A^* (\epsilon, q_f, z)$

Exemple :

L'exemple des palindromes sans vider la pile en q_2 :



On remplace$q_2, \epsilon, z_\perp \to q_2, \epsilon$ par $q_2, \epsilon, z_\perp \to q_f, z_\perp$
$(q_1, abba, z_\perp) \vdash^* (q_f, \epsilon, z_\perp)$: acceptation.

V.6 .2 Langage accepté

Le langage accepté par état final par un AP est l'ensemble des mots acceptés par cet automate

$$L^F(A) = \{m \in \Sigma^* | (m, q_0, z_\perp) \vdash_A^* (\epsilon, q_f, z)\}$$

Exercices de TD

Exercice 1:

Construire un automate à pile reconnaissant par pile vide le langage:

$L = \{a^n b^p \mid 0 < n <= p <= 2n\}$.

Exercice 2:

Les langages suivants sont-ils algébriques ? Si oui, donner un automate à pile reconnaissant le langage:

1. $L_1 = \{w \in \{a, b\}^* \ / \ |w|_a = |w|_b\}$

2. $L_2 = \{w \in \{a, b\}^* \ / \ |w|_a = 2\ |w|_b\}$

3. $L_3 = \{a^p, p \text{ premier}\}$

4. $L_4 = \{a^i b^j, j = i^2\}$

5. L_5 = {bin (i) bin (i + 1); où bin (i) est l'écriture en base 2 de i}

Exercice 3:

Soit le langage $L = \{ a^n b^m c^k \text{ avec } n+m = k\}$ ou $n+k=m$

1. Construire un automate à pile déterministe qui reconnait L.

2. Expliquez son principe de fonctionnement.

3. Vérifiez que l'automate prend en compte les cas où *n=0, m=0 et k=0.*

4. Donnez la suite de configurations pour le mot *aabbbccccc* (*abbbbccc*).

Chapitre VI: Machine de Turing

VI.1. Introduction

- Automate :

Modèle abstrayant la notion de calcul sans écriture *L* est décidable par automate si pour tout mot *w* de *L*, on peut répondre à la question « *w* appartient-il à *L* ? » en lisant le mot et en utilisant la mémoire finie.

- Machine de Turing :

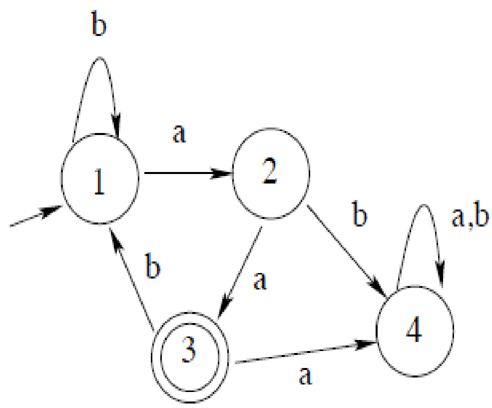
Modèle analogue avec une notion plus élaborée de calcul *L* est décidable par automate si pour tout mot w de L, on peut répondre à la question « *w* appartient-il à ? » en lisant le mot et en utilisant la mémoire finie mais aussi en écrivant des informations sur un support illimité.

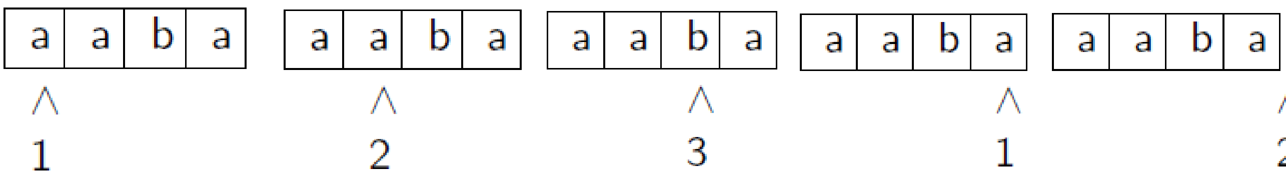
VI.2. Machine de Turing

Caractéristiques d'un automate fini:

- Etats : mémoire finie,
- Lecture des symboles,
- Programme : fonction de transition d'états

états	a	b
→ 1	2	1
2	3	4
$\boxed{3}$	4	1
4	4	4





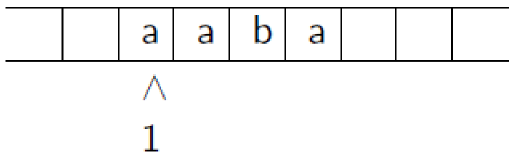
Caractéristiques d'une machine de Turing:

- Etats : mémoire finie,
- Lecture des symboles du ruban,
- Ecriture sur le ruban

- Programme :

  fonction de transition d'états et de déplacement et d'écriture

Support illimité de l'information : Ruban



VI.3. Définition formelle d'une Machine de Turing (MT)

Une machine de Turing à un ruban infini est septuplet $(Q, \Gamma, \Sigma, \delta, q_0, B, F)$ où
- Q ensemble fini d'état,
- Γalphabet fini des symboles du ruban,
- $\Sigma \subset \Gamma$alphabet fini des symboles d'entrée,
- $B \in \Gamma \backslash \Sigma$symbole particulier dit « blanc »
- $q_0$état initial
- Fensemble des états acceptants
- δrelation de transition

La MT est déterministe si pour chaque configuration, elle a au plus une possibilité d'évolution.

VI.3.1Relation de transition

$$\boldsymbol{\delta} \subset \boldsymbol{Q} \times \boldsymbol{\Gamma} \times \boldsymbol{Q} \times \boldsymbol{\Gamma} \times \{\leftarrow, \rightarrow\}$$

Notation d'une règle :

$$q, \sigma \rightarrow q', \sigma', m$$

Prédécesseur :
- q : état courant de la machine
- σsymbole lu sur le ruban

Successeur :
- q' : nouvel état de la machine
- σ'symbole à écrire sur le ruban
- m déplacement de la tête de lecture

Relation de transition : sous forme de table ou de diagramme.

VI.3.2 Notion de configuration

La configuration d'une MT décrit l'« état général » de la machine : état du ruban, état courant de la machine et position de la tête de lecture.

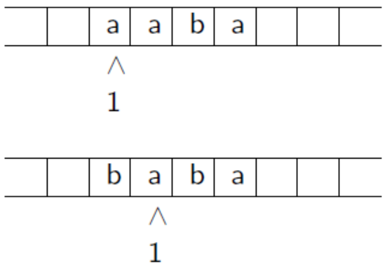
(f,q,p)

- $f : \mathbb{N} \rightarrow \Gamma$ le ruban
- $q \in Q$ l'état de la machine
- $p \in \mathbb{N}$ la position sur le ruban

La relation de transition permet alors de calculer chaque élément de la nouvelle configuration.

Exemple : Le tableau suivant illustre la fonction de transition :

Ancien état	Symbole lu	Symbole écrit	Mouv.	Nouvel état
1	□	□	→	arrêt
	a	b	→	1
	b	a	→	1



VI.3.3 Langage reconnu

Le langage accepté par $M = (Q, \Gamma, \Sigma, \delta, q_0, B, F)$ est défini par :

$L(M) = \{w \in \Sigma^*$tels que:

- l'état initial de M est q_0
- le mot w est écrit sur le ruban
- la tête de lecture est positionnée sur la première lettre de w
- M atteint un état acceptant de F en un nombre fini d'étape

IV.4. Classe de langages

Une MT s'arrête lorsque

- elle atteint un état final
- elle ne peut plus effectuer de transition

VI.4.1 Langage récursif

Un langage reconnu par une MT qui s'arrête sur tous les mots en entrée est dit langage récursif.

VI.4.2 Langage récursivement énumérable

Un langage reconnu par une MT qui s'arrête sur tous les mots du langage (et peut ne pas s'arrêter sur les autres) est dit langage récursivement enumerable, engendré par une grammaire de type 0.

VI.5. Fonction calculée par une machine de Turing

VI.5.1 Fonction calculée

La sortie d'une MT est le mot inscrit sur le ruban lorsque la MTs'arrête.

La fonction calculéef par une MT M est définie par :

A toute entréex sur laquelle M s'arrête, on associe la sortie:$f\ (x)\ =\ y$

Aucune image n'est associée au mot x sur lequel M ne s'arrête pas.

VI.5.2 Machine de Turing équivalente

On peut imaginer beaucoup de variantes de MT :

- sur un « demi » ruban.
- sur deux ou plusieurs rubans.
- la tête de lecture peut être stationnaire.
- non-déterminisme.
- écrire ou non de symbole blanc.

La machine de Turing semble bien représenter une notion de « calcul » par une « procédure effective ».

VI.5.3 Machine de Turing universelle

Une machine de Turing universelle est capable de simuler le comportement de n'importe quelle autre machine de Turing.

Exercice:

1. L'ensemble des machines de Turing est-il dénombrable ?

2. Existe-il un ensemble de fonctions non-dénombrables ?

3. Existe-t-il des fonctions non calculables ?

VI.5.4 Fonctions calculable

- Modélisation de la notion de calcul et procédure effective
- Ce n'est pas un résultat que l'on peut démontrer
- Fonctions calculables par MT = fonctions définies par λ-calculde Chruch
- Base de la théorie de la calculabilité
- Alonzo Church (1903 -1995), mathématicien, logicien américain.

Références:

1- Noureddine, Myriam. Théorie des langages. 1991.

2- Hopcroft, John E. Introduction to Automata Theory, Languages and Computation: *For VTU, 3/e*. Pearson Education India, 2008.

3- Séébold, Patrice. Théorie des automates. 1999.