Cours de THL

Motivations

L'objectif de ce cours est une initiation à la théorie des langages formels. De manière générale, les langages sont les supports naturels de communication. Ils permettent aux hommes d'échanger entre eux des informations et des idées, ils leur permettent également de communiquer avec les machines.
Les langages utilisés dans la vie de tous les jours entre êtres humains sont dits naturels. Ils sont généralement informels et ambigus et demandent toute la subtilité d'un cerveau humain pour être interprétés correctement. Les langages créés par l'homme pour communiquer avec les ordinateurs sont des langages artificiels. Ils doivent être formalisés et non ambigus pour pouvoir être interprétés par une machine.
Au départ, un ordinateur ne comprend qu'un seul langage, pour lequel il a été conçu : son langage machine. Pour communiquer avec des langages plus évolués, il est nécessaire d'utiliser un interpréteur (qui traduit interactivement les instructions entrées au clavier), ou bien un compilateur (qui traduit tout un programme).
Plusieurs types de langages existent en informatique :

- Des langages de description tels que latex et html ;
- Des langages de modélisation tels qu'UML ;
- Des langages de commandes tels que le Shell Unix ;
- Des langages de requêtes tels que SQL ;
- Des protocoles de communication tels que HTTP, TCP, … ;
- Des langages de programmation tels que C, java, pascal,… ;

L'interprétation ou la compilation d'un texte se décomposent généralement en trois étapes.

1. Une première phase d'analyse lexicale permet de décomposer le texte en entités élémentaires appelées lexèmes (token en anglais).

2. Une deuxième phase d'analyse syntaxique permet de reconnaître des combinaisons de lexèmes formant des entités syntaxiques.

3. Une troisième phase d'analyse sémantique permet de générer le code objet directement compréhensible par la machine (ou bien un code intermédiaire qui devra être de nouveau traduit dans un code machine).
Considérons par exemple, le (morceau de) texte ***C*** suivant : ***cpt = i + 3.14;***

1. L'analyse lexicale permet d'identifier les lexèmes suivants : un IDENTIFICATEUR de valeur cpt, un OPERATEUR de valeur =, un IDENTIFICATEUR de valeur i, un OPERATEUR de valeur +,
un REEL de valeur 3.14 et un POINT VIRGULE.

2. L'analyse syntaxique permet de reconnaître que cette combinaison de lexèmes forme une instruction C syntaxiquement correcte, et qu'il s'agit d'une affectation entre la variable d'identificateur cpt et l'expression arithmétique résultant de l'addition de la variable d'identificateur i avec le réel 3.14.

3. Enfin, l'analyse sémantique vérifie le bon typage des variables cpt et i, puis génère le code objet correspondant à cette instruction.
Les phases d'analyse lexicale et syntaxique constituent en fait un même problème (à deux niveaux différents). Dans les deux cas, il s'agit de reconnaître une combinaison valide d'entités : une combinaison de caractères formant des lexèmes pour l'analyse lexicale, et une combinaison de lexèmes formant des programmes pour l'analyse syntaxique. La théorie des langages permet de résoudre ce type de problème.

Plan du cours

En théorie des langages, l'ensemble des entités élémentaires est appelé l'alphabet. Une combinaison d'entités élémentaires est appelé un mot.
Un ensemble de mots est appelé un langage, ce dernier est décrit par une grammaire. A partir d'une grammaire, on peut construire une procédure effective (appelée automate) permettant de décider si un mot fait partie du langage ou non.

Dans le chapitre1 de ce cours, nous définissons ces différentes notions, et nous décrivons certaines de leurs propriétés.

Il existe différentes classes de langages, correspondant à différentes classes de grammaires et d'automates.

Dans le chapitre2, nous étudions la classe des langages réguliers, correspondant aux grammaires régulières et aux automates finis. Cette classe de grammaire est typiquement utilisée pour décrire les entités lexicales d'un langage de programmation comme le *JAVA* ou le *langage C, ...*,

Dans la même partie du cours, nous allons voir les expressions régulières qui permettent de dénoter un langage régulier.

Au chapitre 3, nous étudions la classe des langages hors contexte et qui sont appelés aussi langage à contexte libre, correspondant aux grammaires hors contexte de type 2 et aux automates à pile.

Cette classe de grammaire, plus puissante que la classe des grammaires régulières, est typiquement utilisée pour décrire la syntaxe d'un langage de programmation.

CHAPITRE1 :

CONCEPTS UTILISES EN THL

Quelques Concepts

- Un *alphabet*, noté A, est un ensemble fini non vide de symboles.

Exemples d'alphabets :

- A_1 = {0, 1}→ Cet alphabet génère des mots binaires: 011, 1000, ….

- A_2 = {a, b, c, . . ., z}→ Cet alphabet génère tous les mots de la langue française, anglaise, …

- A_3 = {if, then, else, id, nb, and, or, not, =, +, -, *, /} → Avec cet alphabet, on produit des instructions du langage ***pascal***.

- Un *mot*, défini sur un alphabet A, est une suite finie d'éléments de A.

Exemples de mots :

- sur l'alphabet A1, les mots possibles sont: 0001, 101, 1100, ...
- sur l'alphabet A2, les mots possibles sont: informatique, university, licence, ….
- sur l'alphabet A3, les mots possible sont : if x=5, if y=3 then z=3,…..

- La *longueur d'un mot* u défini sur un alphabet A, notée lul est le nombre d'occurrences des symboles qui composent u.
- La *longueur du mot vide* est égale à 0. On écrit : **lεl=0**

Exemple :

l0001l=4

l101l=3

lif y=3 then z=z+1l=10

- on note $\mathbf{A}^+$ l'ensemble des mots de longueur supérieure ou égale à 1 que l'on peut construire à partir de l'alphabet A.

- on note $\mathbf{A}^*$ l'ensemble des mots que l'on peut construire à partir de A, y_compris le mot vide :

$$A^* = \{\epsilon\} \cup A^+$$

- **ε** est le mot vide, **Φ** est l'ensemble vide. Les deux symboles n'ont pas la même signification.

- *La concaténation*

Soient w_1 et w_2 deux mots définis sur l'alphabet A. La concaténation de w_1 et w_2 est un mot w défini sur le même alphabet. w est obtenu en écrivant w_1 suivi de w_2, en d'autres termes, on colle le mot w_2 à la fin du mot w_1 :

$w_1 = a_1...a_n$, $w_2 = b_1b_2...b_m$

$w = a_1...a_nb_1b_2...b_m$

La concaténation est notée par le point, mais il peut être omis s'il n'y a pas de d'ambiguïté.

On écrira alors : $w = w_1.w_2 = w_1w_2$.

Soient w_1 et w_2 deux mots définis sur l'alphabet A. La concaténation de w_1 et w_2 est un mot w défini sur le même alphabet. w est obtenu en écrivant w_1 suivi de w_2, en d'autres termes, on colle le mot w_2 à la fin du mot w_1 :

$w_1 = a_1...a_n$, $w_2 = b_1b_2...b_m$ $w = a_1...a_nb_1b_2...b_m$

La concaténation est notée par le point, mais il peut être omis s'il n'y a pas de d'ambiguïté.

On écrira alors : w = w1.w2 = w1w2.

- lu.vl= lul+ lvl avec u,v Є A^*

Exemples:

1) Si w_1=aabc, w_2=ca, w_3=Φ, w_4=ε

On a alors:

w_1w_2=aabcca, w_2w_1=caaabc w_1w_3=aabcΦ=Φ

w_3w_1=Φaabc=Φ w_1w_4=aabcε=aabc=w_1 w_4w_1=εaabc=aabc=w_1

2) Soit l'alphabet A={a,b}

1. Etant donnés les mots **u = aa** et **v = bab**, écrire les mots uv, $(uv)^2$ et u^3v.
2. Enoncer tous les mots de longueur 2 définis sur A.

- *Le mot miroir*

Soit $w = a_1a_2...a_n$ un mot sur A. On appelle mot miroir de w et on le note par w^R le mot obtenu en écrivant w à l'envers, c'est-à-dire que $w^R = a_n...a_2a_1$.

Exemples : - le mot miroir de test est tset.
- le mot miroir de a est a.
- le mot miroir de ε est ε.

Exercice :

1. Donner le mot miroir des expressions: elle, elu par cette crapule, esope reste ici et se repose, un emu a son os au menu.
2. Comment sont appelées les expressions précédentes ?

- *Mots conjugués*

Deux mots x et y sont dits conjugués s'il existe deux mots u et v tels que : x = uv et y = vu.

- *Notion du langage*

Un langage est un ensemble (fini ou infini dénombrable) de mots définis sur un alphabet donné.

Exemples de langages :

- $L_1 = \{ab, a, ba, bb\}$;
- $L_2 = \{\omega \in \{a,b\}^* / |\omega| \geqslant 3\}$;
- $L3 = \{\omega \in \{a,b\}^* / |\omega| \equiv 0\,[5]\}$;
- $L_4 = \{\omega \in \{a,b\}^* / |\omega|_a \equiv 0\,[3]\}$;
- $L_5 = \{a^n / n \geqslant 0\}$;
- $L_6 = \{a^i b^j / i \geqslant 0, j \geqslant 1\}$;

- *Opérations sur les langages*

Les langages sont des ensembles. On peut alors leur appliquer n'importe quelle opération ensembliste. Cependant, vu la particularité des éléments constituant les langages (des mots), certaines de leurs opérations ne peuvent être appliquées qu'à eux.

Soient L, L_1 et L_2 trois langages définis sur l'alphabet X, on définit les opérations suivantes :

- ***L'union*** : $L1 \cup L2 = \{w | w \in L_1$ **ou** $w \in L_2\}$;
- ***L'intersection*** : $L1 \cap L2 = \{w | w \in L1$ **et** $w \in L2\}$;
- ***La concaténation*** (opération non ensembliste) : $L_1.L_2 = \{w | \exists u \in L_1, \exists v \in L_2 : w = uv\}$;
- ***L'exposant*** (opération non ensembliste) : $L^n = L.L...L$ (n fois).

Nous avons alors :

$L^0 = \{\varepsilon\}$ ……*(a)*

$L^{n+1} = L^n . L$ ……*(b)*

On déduit alors que :

$L^+ = L . L^*$ ……*(1).*

$L^* . L^* = L^*$ ……*(2).*

Exercice : Montrer les deux relations *(1)* et *(2)*?

-***La complémentation*** : Le complément du langage L est noté par $\overline{\mathbf{L}} = \{w \notin L\}$

Exemple1 : Soient les 2 langages suivant :

$L_1=\{a^{2n+1}, n<=3\}$:

$L_2=\{a^{2n}, n<=3\}$:

Questions

1) Calculer $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1.L_2$

Nous avons :

- $L_1 \cup L_2 = \{a^7,a^5,a^3,a^1\} \cup \{a^6,a^4,a^2,a^0\} = \{a^7,a^6,a^5,a^4,a^3,a^2,a^1,a^0\} = \{a^n, 0<=n<=7\}$
- $L_1 \cap L_2 = \{a^7,a^5,a^3,a^1\} \cap \{a^6,a^4,a^2,a^0\} = \varphi$
- $L_1.L_2 = a^{2n+1}. a^{2n} = a^{2n+1+2n} = \{a^{4n+1}, n<=3\}$

Exemple2 : Soient L_1, L_2 et L_3 trois langages définis par :

L_1= {aa},
L_2= {$a^i b^j$ / i, j > 0}
L_3= {ab, b}.

Définir les langages suivants: L_1^*, $L_2 \cup L_3$, $L_1.L_3$, $L_1 \cap L_3$, $L_2 \cap L_3$.

- Une grammaire est un quadruplet $G = (\pi, N, S, P)$ où

  π: est le vocabulaire terminal, i.e, . Est un ensemble fini non vide de mots.

  N : est le vocabulaire non terminal, i.e, l'ensemble des symboles qui n'apparaissent pas dans les mots générés, mais qui sont utilisés au cours de la génération de ces mots. C'est un ensemble fini non vide.

  S : est un élément de N, est le symbole de départ ou *axiome*. C'est à partir de ce symbole que l'on commencera la génération de mots au moyen des règles de la grammaire.

  P: est un ensemble de règles dites de réécriture ou de production.

Exemple :

π={0,1}
N={A, B, S}
P={ S->A , S->B , A->0A, A->ε, B->1B, B->ε}

- **Langage engendré (généré) par une grammaire**
  Le langage défini, engendré ou généré, par une grammaire est l'ensemble des mots qui peuvent être obtenus à partir du symbole de départ (S) par application des règles de la grammaire. Plus formellement est l'ensemble des dérivations terminales de son axiome.

Exemples de langages :

$L_1 = \{ab, a, ba, bb\}$; Langage fini.
$L_2 = \{\omega \in \{a, b\}^* / |\omega| > 3\}$; Langage infini.
$L_3 = \{ 0^n 1^n / n \geq 0 \}$; Langage infini.

Exercices

1. Donner le langage formel généré par la grammaire suivante :
   π={0,1}
   N={A, B, S}
   { S->AB, A->0A/ε, B->1B/ε}

2. Soit la grammaire *G = ({a, b, c} , {S, A} , S , P) où P contient les règles suivantes :*

$S \rightarrow aS \mid bA \qquad A \rightarrow cA \mid \varepsilon$

- *Déterminer si les mots :*

w1 = abac, w2 = aabccc, w3 = cabbac et w4 = ab sont générés par G.

- *Trouver le langage généré par G (qu'on note L(G)).*

- **Types de grammaires**

En introduisant des critères plus au moins restrictifs sur les règles de production, on obtient des classes de grammaires hiérarchisées, ordonnées par inclusion. La classification des grammaires, définies en 1957 par Noam **CHOMSKY**, distingue les quatre classes suivantes:

-1-*Grammaires de type 0*

Ce sont des grammaires sans restriction sur les règles, donc toutes les grammaires sont de type 0. Les productions de ce type se présentent sous forme : u->v

u, v $\in(\pi \cup N)^*$

-2- *Grammaires de type 1*

Les grammaires de type 1 sont appelées aussi les grammaires sensibles au contexte ou contextuelles. Les règles de la grammaire sont de la forme:

uAv->uWv avec : A $\in$ N et W, u, v $\in(N \cup \pi)^*$

-3- *Grammaires de type 2*

Les grammaires de type 2 sont appelées aussi les grammaires hors contexte, contexte libre, algébrique ou de Chomsky. C'est la grammaire la plus utilisée en théorie des langages et en compilation.

Les règles de la grammaire sont de la forme: A->W avec : A $\in$N, W $\in(N \cup \pi)^*$

-4- *Grammaires de type 3*

Les grammaires de type 3 sont appelées aussi les grammaires régulières à droite (respectivement à gauche)

Les règles de la grammaire sont de la forme:

A->w B (respectivement A->Bw)

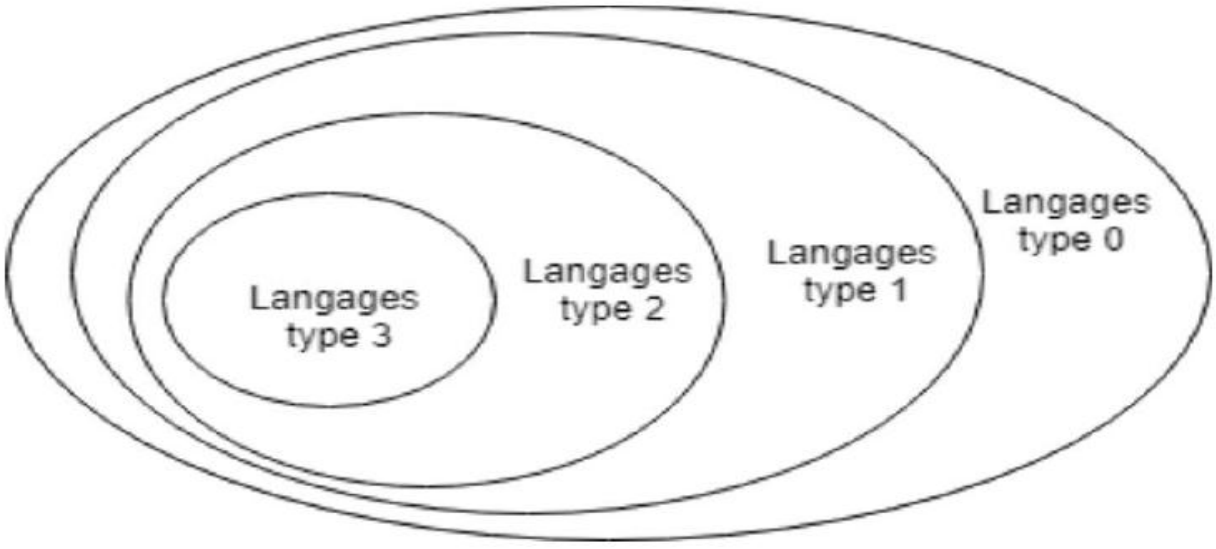
Ou bien A->w avec : A, B $\in$ **N** et w $\in \pi^*$.

- ***Type de langage***

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 génèrent les langages algébriques (à contexte libre) et les grammaires de type 1 génèrent les langages contextuels (à contexte lié) et les grammaires de type 0 permet 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 ".

En conclusion, nous arrivons à cette décision exprimée sous forme graphique :



Relation d'inclusion entre les différents types de langages.

Exercice

Pour chacune des grammaires Gi = ({a, b, c} , {S, A, R, T} , S , Pi), (i=1,..,5) ; donner le type de celle-ci, puis trouver le langage engendré par chacune d'elles :

1) P1 : S → bA A → aA / ε
2) P2 : S → aSc / ε
3) P3 : S → aSbS / bSaS / ε
4) P4 : S → 1S00 / 100
5) P5 : S → aAb / ε A → ab Ab → ε

- ***Conclusion***

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 à états fini** *(AEF),*

- Les langages algébriques (contexte libre) sont reconnus par des **automates à pile;**

- Les langages à contexte liés sont reconnus par des **automates à bornes linéaires** (*ABL*)

- Les langages de type 0 sont reconnus par les **machines de Turing.**

Exercice : Pour chacun des langages suivants, donner une grammaire qui l'engendre:

a) $L_1 = \{ 0^{2n} / n \geq 0 \}$, vérifier les mots : 00, 000, 0000.

b) $L_2 = \{ 0^n 1^n / n \geq 0 \}$, vérifier les mots : 001, 0011.

c) $L_3 = \{ a^n b^{2n} / n \geq 0 \}$, vérifier les mots : ab, aab, abb.

d) $L_4 = \{ a^n b^m c^{n-m} / n \geq m \geq 0 \}$, verifier les mots: abc, aabcc,

e) $L_5 = \{$ palindromes de $\{a, b\}^* \}$, vérifier les mots : bb, abab, baab.

f) $L_6 = \{ a^m b^n a^n b^m / n \geq 1, m \geq 0 \}$, verifier les mots: ba, bbab, aabb.

g) $L_7 = \{a^n b^n a^m b^m a^k b^k / n, m, k \geq 1\}$, verifier les mots: ab, aab, abba.

h) $L_8 = \{$tous nombre binaire$\}$, vérifier les mots : 001, 0101, 111.

i) $L_9 = \{ww^R / w \in \{0, 1\}^*\}$, vérifier les mots : 1001, 1100, 0110.

j) $L_{10} = \{a^n b^n c\, a^m b^m / n \geq 0, m \geq 0\}$, vérifier les mots : c, acb, aacb.

k) $L_{11} = \{ a^n b^m c^m d^n / n, m \geq 0 \}$, vérifier les mots : abcd, acb, ad.

L) $L_{12} = \{$Les nombres décimaux impairs, vérifier les mots : 230, 673.

CHAPITRE2 :

LANGAGES REGULIERS ET AUTOMATES A ETATS FINI

Définition d'une grammaire régulière
Les grammaires de type 3 sont appelées aussi les grammaires régulières à droite (respectivement à gauche).
Les règles de la grammaire sont de la forme:

A->wB (respectivement A->Bw)
ou bien
A->w

Avec A, B Є $\mathbf{N}$ et wЄ$\boldsymbol{\pi}^*$

Autrement dit, le membre de gauche de chaque règle est constitué d'un seul symbole non terminal, et le membre de droite est constitué d'un symbole terminal éventuellement suivi (respectivement précédé) d'un seul symbole non terminal.

Les langages réguliers sont des langages engendrés par des grammaires régulières.
Les grammaires régulières sont utilisées beaucoup dans la phase d'analyse lexicale.

Exemple: $\pi = (\pi, N, S, P) = (\{a, b\}, \{S,A\}, S, P)$
P= (S→a S
S→bA
A→a)

Donner une définition formelle de ce langage ?

Définition d'un automate
Les automates sont des machines fictives (programmes), qui prennent en entrée une chaîne de symboles et qui effectuent un algorithme de reconnaissance de la chaîne. Si l'algorithme se termine dans certaines conditions, l'automate accepte cette chaîne, sinon, la chaine n'est pas reconnue.
Le langage reconnu par un automate, est l'ensemble des chaînes (mots) qu'il accepte.

Nous avons deux types de systèmes:

-Les systèmes de *génération* qui sont les grammaires.

-Les systèmes de *reconnaissance* qui sont les automates.

Automate d'états fini(AEF)

Définition

Un AEF est composé d'un ensemble fini d'états (représentés graphiquement par des cercles), d'une fonction de transition décrivant l'action qui permet de passer d'un état à l'autre (ce sont les flèches), d'un état initial (l'état désigné par un cercle pointé par une flèche pleine) et d'un ou plusieurs états finaux (désignés par des cercles doublés).
Un AEF est donc un graphe orienté et étiqueté où les nœuds correspondent aux états et les valeurs des arcs aux symboles terminaux
Les AEF n'utilisent aucune mémoire pour reconnaitre une chaine, ils utilisent seulement leurs états.

Représentations d'un AEF

Il existe trois principales représentations pour les AEF:

-la représentation matricielle,

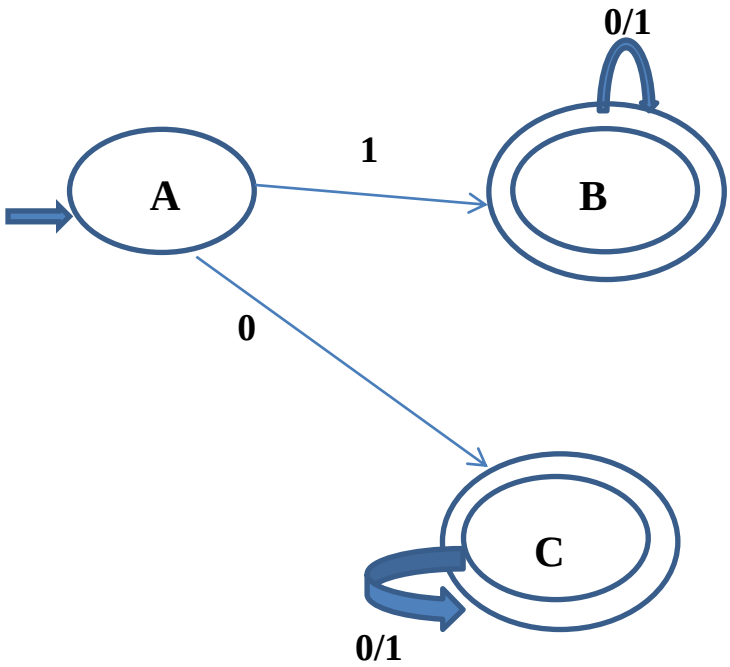
-sous forme de graphe orienté étiqueté, c'est la notation la plus représentée.

-ou sous forme de fonctions (relations)

Fonction de transition

f est la fonction de transition de l'automate A : f (q, a) = q'
Indique que si l'automate est dans l'état q et qu'il rencontre le symbole a, il passe à l'état q'.

Exemple1 : Trouver le langage accepté par l'automate suivant :



Exercice Trouver les AEF acceptant les langages suivants :

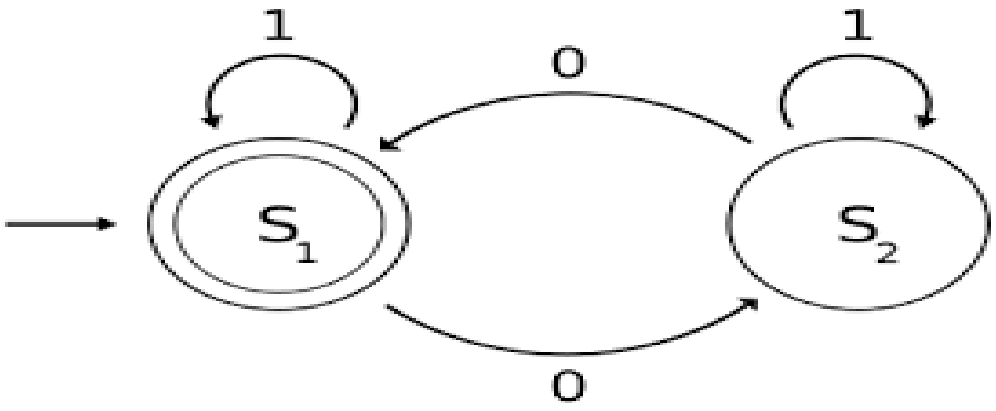
- $L_1=\{\varepsilon\}$
- $L_2=\{\varepsilon, a, ab\}$
- $L_3=\{a^n b^m c^k, n, m>=0, k>=2\}$
- $L_4=\{$Les entiers naturels pairs$\}$
- $L_5=\{a^n b^m c^{2*p}, n, m, p>=1\}$
- $L_6=\{$Les nombres décimaux impairs$\}$

Variantes d'automates d'états finis

a)Automates d'états fini déterministes (AEFD)

Un automate d'états fini **déterministe** est un automate tel qu'à partir d'un état donné et avec un symbole quelconque, l'automate ne doit pas s'offrir un choix de passer vers un autre état.

Exemple :

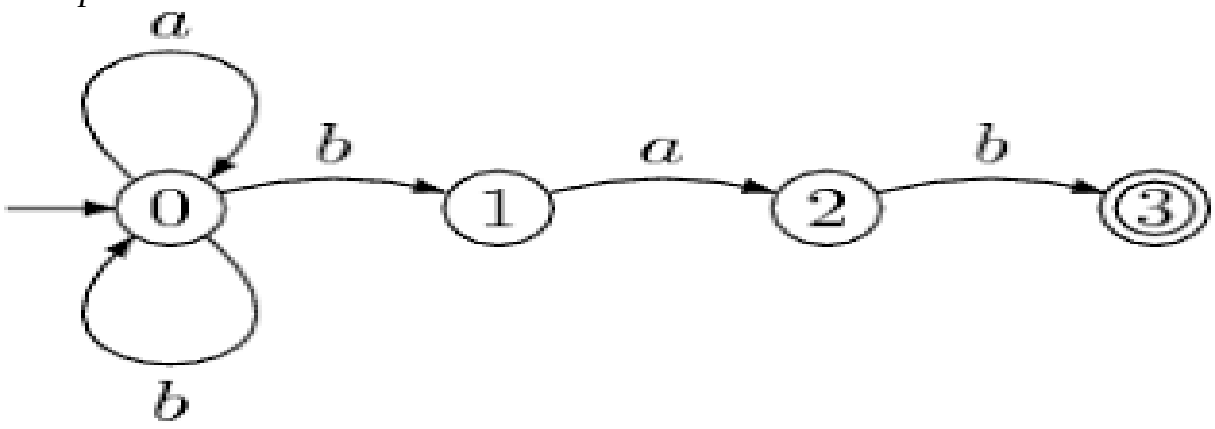


Nous remarquons que dans cet automate, de l'état S1 avec la lettre 0 on passe seulement à S2 (pas de choix) et avec la lettre 1 on passe seulement à S1 (ya pas d'autre choix). La même chose pour l'état S2.

b) Automates d'états finis non déterministes (AEFND)

Un automate d'états fini **non déterministe** est un automate tel qu'il existe au moins un couple formé d'un état et d'un symbole, qui admet plus d'une image par la fonction de transition. L'automate doit faire des choix pour progresser.

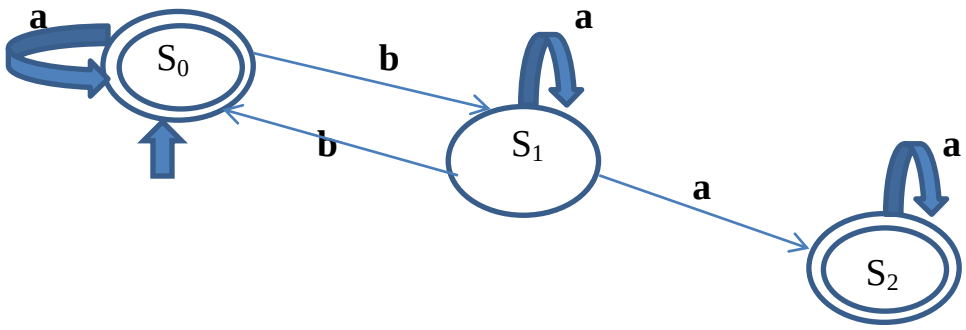
Exemple :



On remarque dans cet automate qu'au niveau de l'état 0, par application de la lettre b, l'automate passe par la lettre b vers deux états possibles : 0 ou 1 ce qui justifie que cet automate n'est pas déterministe.

- **Comment transformer un automate non déterministe en un automate déterministe :** Pour tout automate non déterministe, il existe un autre automate déterministe équivalent.

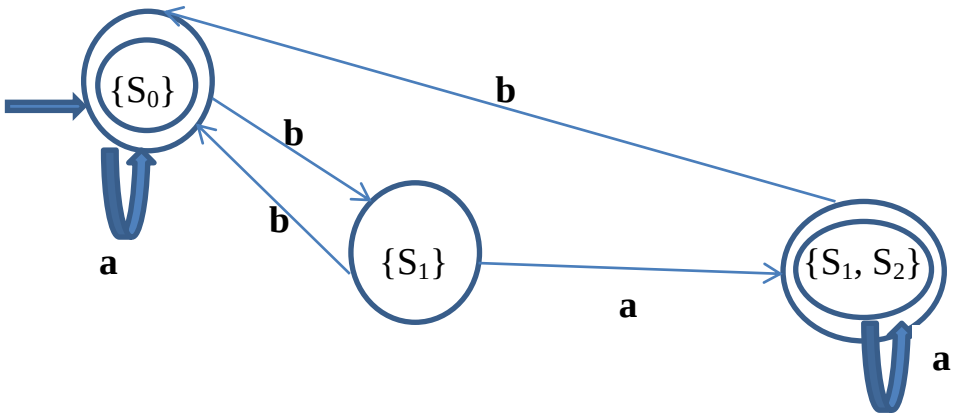
Exemple : Soit l'automate à états fini suivant :



Cet automate n'est pas déterministe car de l'état S1 par la lettre a, l'automate passe vers deux états possibles : S_1 ou S_2. Essayant de rendre cet automate déterministe en construisant une table de transition suivante :

états \ lettre	a	b
$\{S_0\}$	$\{S_0\}$	$\{S_1\}$
$\{S_1\}$	$\{S_1, S_2\}$	$\{S_0\}$
$\{S_1, S_2\}$	$\{S_1, S_2\}$	$\{S_0\}$

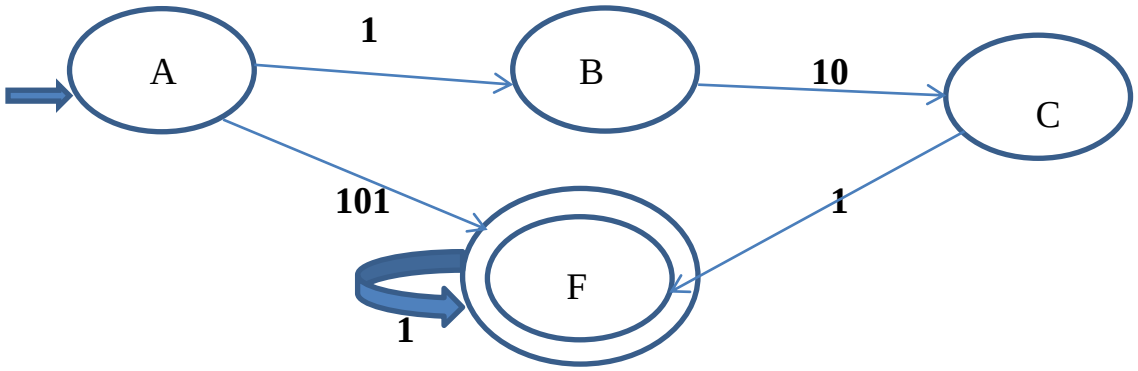
A partir de cette table, on obtient l'automate équivalent suivant :



Cet automate est déterministe car à partir de chaque état, l'automate n'offre pas un choix de passer d'un état à un autre état avec une lettre donnée.

Automate d'états finis généralisé (AEFG)

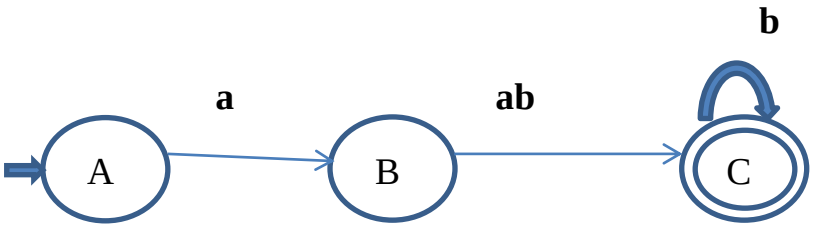
-Les transitions peuvent êtres engendrées par des mots au lieu de symboles. (voir suivante)
-Les transitions causées par le mot vide (ε) sont appelées transitions spontanées ou vide (ε-transition). C'est un changement d'état sans lecture.



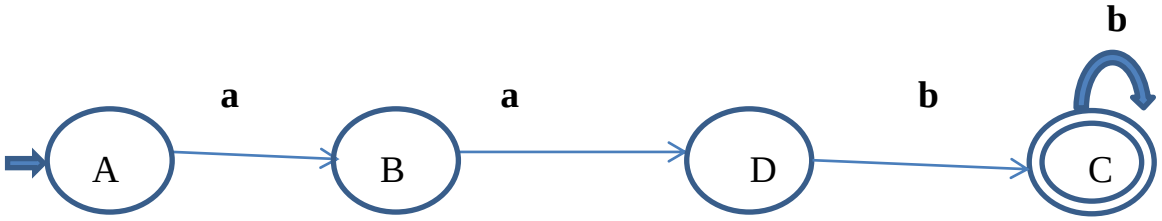
AEF généralisé

Transformation d'un AEF généralisé en un automate simple

Tout automate d'états finis généralisés admet un automate simple équivalent en ajoutant des transitions supplémentaires.



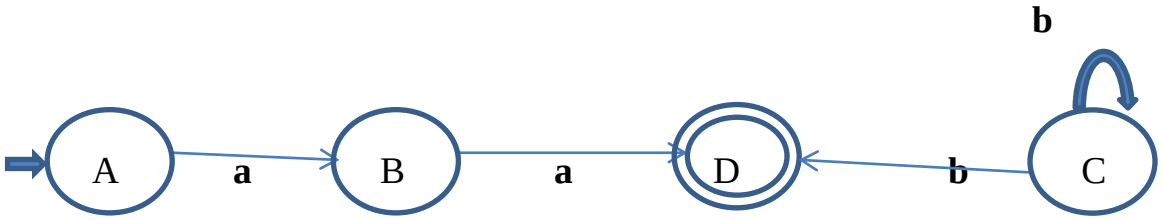
On a bien ici un automate generalisé car cet automate passe de l'état B à l'état C par le mot **ab**, pour rendre cet automate simple c'est facile puisque il suffit de rajeuter un nouvel état D comme suit :



Etat inaccessible

Un état d'un automate donné est dit inaccessible, si ce dernier n'est pas atteignable à partir de l'état initial.

Exemple

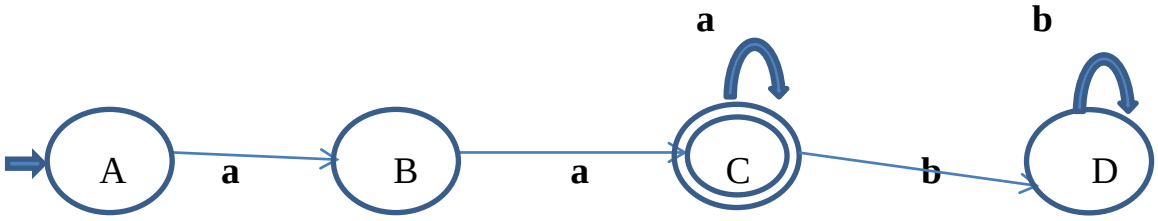


On remarque que l'état D est inaccessible.

Etat puits

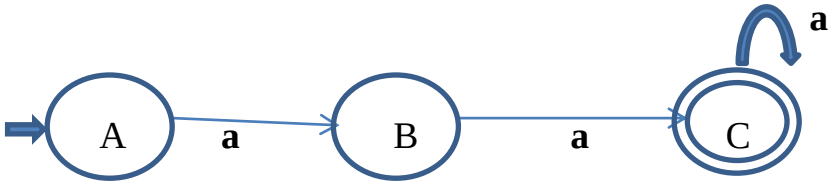
Un état d'un automate donné est dit état puits, si il'n y a pas un chemin de cet état vers n'importe quel état de l'automate, en plus ce dernier n'est pas final.

Exemple



L'état D dans cet automate est un état puis car, il est atteignable à partir de l'état initial A, mais il ne produit rien.

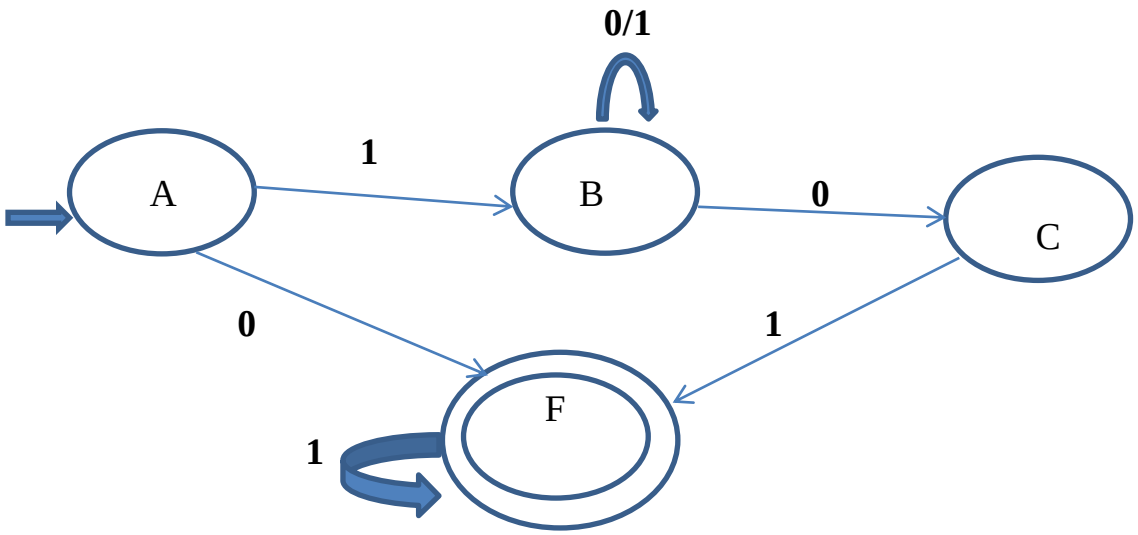
L'automate précèdent sera équivalent à cet automate ci-après:



Automate d'états fini complet (AEFC)
Un automate d'états fini défini sur un alphabet donné est dit complet, si on peut passer de tout état de l'automate avec toutes les lettres de cet alphabet vers un autre état.

Exemple

Dites est-ce que cet automate est complet, sinon, donner un automate équivalent complet.



Passage d'une grammaire régulière à un AEF

Pour toute grammaire régulière $G = (\pi, N, S, P)$, il existe un AEF équivalent tel que L(G)=L(A).

Exemple

Trouvons l'AEF équivalent à la grammaire suivante :
G=({a,b,c},{S,A,B}, S, P) P = ($S \rightarrow aS \mid bA$,

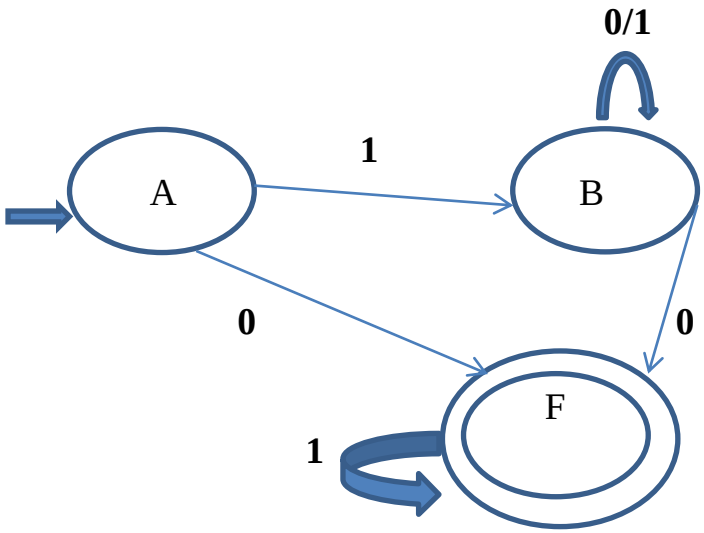
$A \rightarrow bA \mid cB \mid a$

$B \rightarrow cB \mid c$)

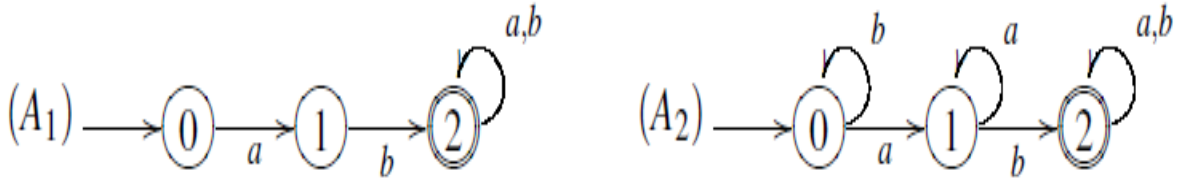
Passage d'un AEF à une grammaire régulière

Pour tout automate à états fini, il existe une grammaire régulière équivalente.

Exemple
Donner la grammaire équivalente à cet automate :



Exercice On considère deux automates A_1 et A_2 sur l'alphabet {a, b}



(a) Dans quel état se trouve l'automate A1 après lecture des mots a, ab, abb, abba ?
(b) Lesquels de ces mots sont reconnus par l'automate A_1 ?
(c) Que se passe-t-il quand on donne le mot aab à lire à l'automate A_1 ?
(d) Les mots aba^2b, a^2ba^2b, ab^4 et b^3a^2 sont-ils reconnus par l'automate A_1 ?
(e) Décrire les mots reconnus par l'automate A_1.
(f) Après lecture du mot b^3a^2, dans quel état se trouve l'automate A_2 ?
(g) Y a-t-il des mots que l'automate A_2 ne peut pas lire jusqu'au bout ?
(h) S'il n'a lu aucun a, dans quel état se trouve l'automate A_2 ?
(i) Dans quels cas l'automate A_2 se trouve-t-il dans l'état 1 ?

(j) Dans quels cas arrive-t-il à l'état final 2 ? Quels mots reconnaît-il ?

Les expressions régulières

Définition :
Les expressions régulières (ER) fournissent une autre méthode de définition des langages réguliers. Elles sont plus pratiques que les deux autres systèmes (grammaires régulières et automates).
Chaque expression régulière décrit un ensemble de chaines terminales. Le formalisme des expressions régulières utilise 03 opérations:

-La concaténation, -La fermeture notée * (puissance), -L'alternative notée + ou / (choix entre deux expressions)

Définition formelle

— ϕ est une expression régulière qui dénote (représente) le l'ensemble vide
— ε est une expression régulière qui dénote le mot vide $\{\varepsilon\}$
— a (où $a \in \pi$) est une expression régulière qui dénote le langage $\{a\}$.

Induction
Si e, e' sont des expressions régulières alors e+e' , e.e', e*, e+ sont des expressions régulières.
Remarque
-Deux expressions régulières sont dites ε-équivalentes ssi elles dénotent le même langage.

Exemple1 Voici des exemples d'expressions régulières :

(a + b)* = { ε, a, b, aa, bb, ab, ba, aaa, bbb, aab, aba, …….. }
a*b + b*a = { b, ab, aab, aaab, aa….ab, a, ba, bba, bbba, bb…ba, …. }
ab*(c+ a) = ab*c + ab*a = { ac, abc, abbc, abbbc, ….., aa, aba, abba, abbba, ….}

Exemple2

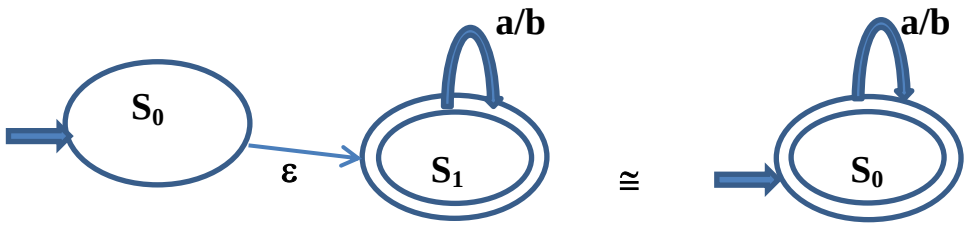
1. a^* : dénote le langage régulier a^n (n >=0) ;

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.

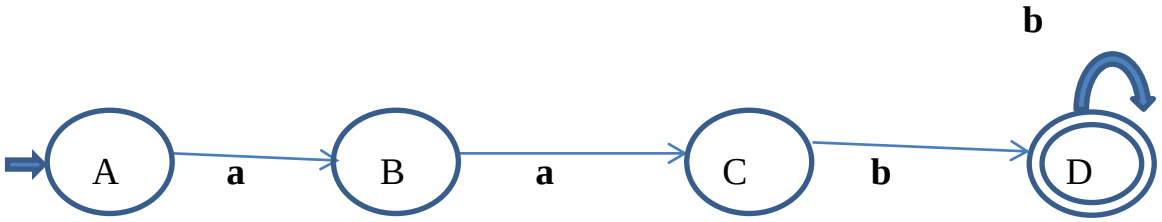
Proposition1 Pour toute expression régulière E, il existe un AEF qui reconnaît le langage dénoté par E.

Exemple pour l'expression régulière : $(a+b)^*$, il correspond l'AEF suivant :



Proposition2 Pour tout AEF A, il existe une expression régulière E qui dénote le langage reconnu par A.

Exemple Soit l'automate suivant



L'expression régulière qui le dénote est : **aabb*** = **aab**$^+$

Exercice Pour chacune des expressions régulières suivantes, dessiner un AEF reconnaissant les langages dénotés par celles-ci :

$$\alpha = aab\,; \qquad \beta = abba + bbab\,; \qquad \gamma = (aba)^* + (bab)^*.$$

Dérivées

Soit L un langage sur un alphabet X et $\omega \in X^*$, la dérivée de L par rapport à ω, notée $L \parallel \omega$, est définie par : $L \parallel \omega = \{z \in X^* / \omega.z \in L\}$.

Exemples : Soient les langages $L_1 = \{\varepsilon, a, ab, aa, ba\}$ et $L_2 = \{a^n / n \geqslant 0\}$.

$L_1 \parallel a = \{\varepsilon, b, a\}$;

$L_1 \parallel aa = \{\varepsilon\}$;

$L_2 \parallel a = L_2$.

Comment montrer qu'un langage est régulier ?

Plusieurs méthodes peuvent montrer la régularité d'un langage :

1. Tous les langages finis sont réguliers.
2. Si on trouve un AEF qui reconnait le langage L, alors L est régulier.
3. Si on trouve une grammaire régulière (à gauche ou à droite) générant L, alors L est régulier.
4. Si on trouve une expression régulière dénotant L, alors L est régulier.

Exemples

Montrer que les langages suivants sont réguliers :

1. {ali, mange, une, pomme}
2. {nombres binaires paires}
3. {nombres décimaux multiples de 5}
4. {$a^n b^m c^k$, n>=0, m>3, k=2t, t est un entier}

Exercices

1) Soit l'expression régulière : *ER*= (a*ab(a+b)*)* ?

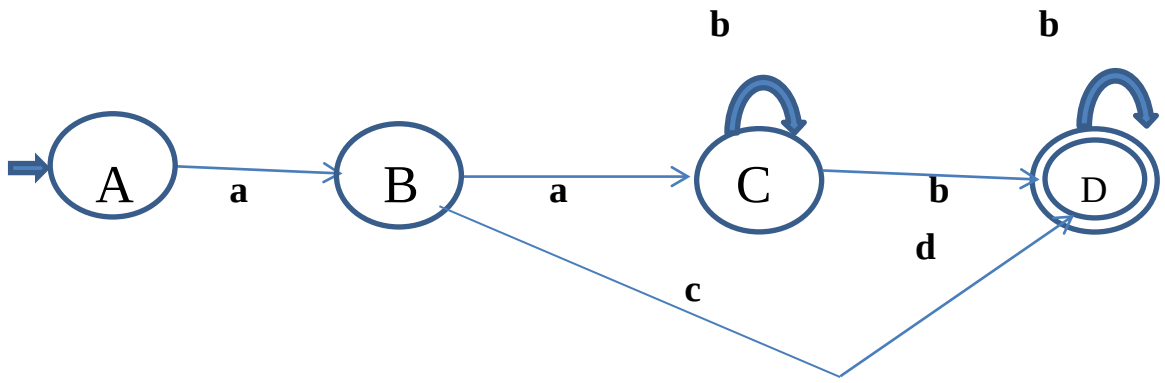
1. Est-ce-que les mots suivants sont dénotés par l'expression *ER* **:**
ab, abb, aa, bbb, baa, bbaa ??

2. Construire un automate d'états fini pour cette expression régulière.

2) Trouver les automates à états fini acceptant les langages dénotés par les expressions régulières suivantes :

- $L_{12} = a(ab + b^*)(aba)^*.$
- $L_{13} = (ab + a)^* + (aba^*b + ba)^+.$

3) Trouver l'expression régulière dénotant le langage accepté par l'automate suivant :



CHAPITRE3 :

LANGAGES ALGEBRIQUES ET AUTOMATES A PILES

Introduction

Les langages de type 2, appelés aussi les langages algébriques ou encore les langages à contexte libre, sont reconnus par des machines abstraites semblables aux AEF à une différence prêt qui est la mémoire, cette mémoire est une pile.
Le présent chapitre est consacré à ce type de langages ainsi qu'aux automates à piles.

Rappels sur les grammaires et langages algébriques

Définition d'une grammaire de type 2 :
Une grammaire G= (π, N, S, P) est dite à contexte libre (Algébrique ou de type 2) si et seulement si toutes ses règles de production sont sous la forme :

$A \rightarrow W$ **avec** $A \in N$ **et** $W \in (\pi \cup N)^*$

Exemple1 : Soit la grammaire :
$S \rightarrow AB \mid A \rightarrow aA \mid A \rightarrow aB \mid B \rightarrow bB \mid \varepsilon$

- Déterminer le langage engendré par cette grammaire algébrique.

Exemple2 : Soit la grammaire G dont les règles de production sont :
S → AB | ε
A → aAb | ε
B → bBa/ ε

1) Déterminer L(G).

Définition des langages de type2 (algébrique)
Ce sont les langages engendrés par des grammaires de type 2.

Remarque :
L'ensemble des langages réguliers est inclus dans l'ensemble des langages algébriques.

Arbre syntaxique
Vue l'utilisation d'un seul symbole non terminal à gauche des règles de production dans les grammaires à contexte libre, il est toujours possible de construire un arbre de dérivation pour n'importe quel mot généré.

Soit la grammaire G=(π, N, S, P) et soit $\omega \in L(G)$. Un arbre syntaxique associé à ω est construit comme suit:

• La racine de l'arbre est étiquetée par l'axiome

• Les nœuds intermédiaires (internes) contiennent des non terminaux

• Les feuilles sont des terminaux

La lecture de gauche à droite des feuilles de l'arbre reconstitue le mot auquel l'arbre est associé.

Exemple : soit la grammaire algébrique suivante :

G=({a,b,c},{S,A,B}, S, P) P = ($S \rightarrow aS \mid bA$

$A \rightarrow bA \mid cB \mid a$

$B \rightarrow cB \mid c$)

Donner l'arbre de dérivation des mots: **aaabba, bbbba.**

Rappels sur les grammaires et langages algébriques

Définition d'un mot ambigu :
Un mot ω est dit ambigu si et seulement s'il existe au moins deux arbres de dérivation différents qui lui sont associés.

Définition d'une grammaire ambiguë :
Une grammaire G est dite ambiguë si et seulement s'il existe au moins un mot ambigu appartenant à L(G).

Remarques :

1-Certains langages peuvent être générés, à la fois, par des grammaires ambiguës et des grammaires non ambiguës.

2-Il n'existe aucun algorithme qui permet de décider qu'une grammaire est non ambiguë, c'est-à-dire qu'on peut montrer qu'une grammaire est ambiguë, mais on ne peut pas prouver sur une grammaire donnée qu'elle est non ambiguë.

Exemple :

1. Montrer que la grammaire ci-dessous est ambiguë.

P= (S→ S∧S (1)
S→ S∨S (2)
S→ S⇒S (3)
S→ S⇔S (4)
S→ ⁄S (5)
S→ const (6))

2. Proposer une grammaire équivalente non ambiguë.

Forme normale de Chomsky (FNC)

Une grammaire G= (π, N, S, P) est dite sous forme normale de Chomsky (FNC) si et seulement si toutes ses règles de dérivation sont sous la forme:
A → BC ou A → a avec A, B, C ∈N et a ∈π

Théorème Pour toute grammaire algébrique (contexte libre) , il existe une grammaire équivalente sous forme normale de Chomsky.

Exercice Montrer ce théorème.

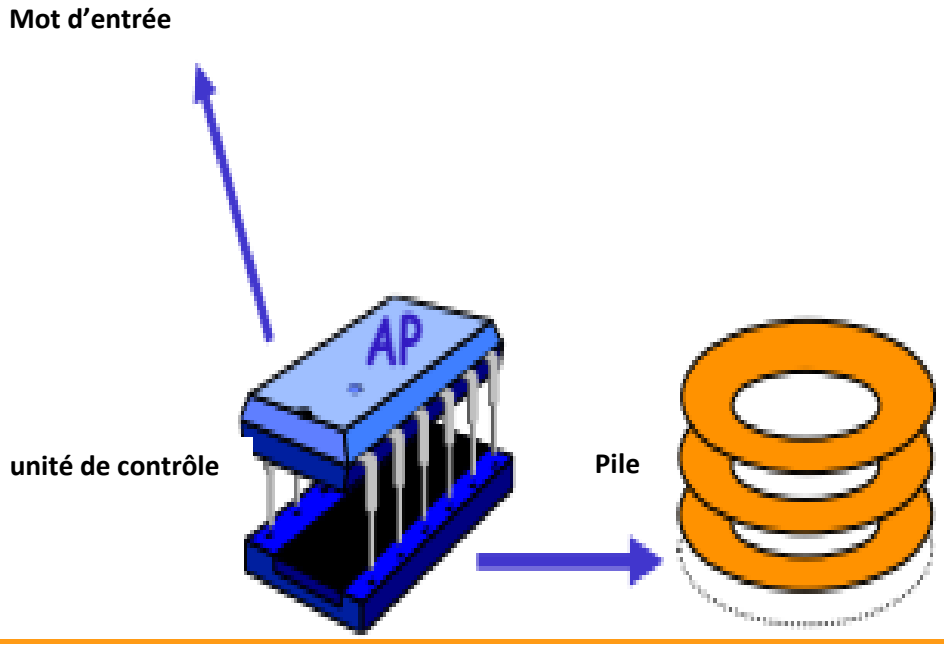
Récursivité à gauche

Une grammaire est récursive à gauche dès qu'elle contient une production de la forme : **B → B w avec B ∈N et w ∈(π∪N)***

AUTOMATE A PILE

Pourquoi une pile ?

•Les automates finis (AEF) n'ont pas d'autre mémoire que leurs états.
•Ils ne savent donc pas « compter » au-delà de leur nombre d'états.
•Une pile utilise une mémoire additionnelle non bornée.
•On accède à la pile uniquement par son sommet.
•Le nombre de symboles utilisés dans la pile est fini.
•La pile vide peut être un critère d'acceptation des mots**.**



Essayons maintenant de reconnaitre les mots du langage **L** de type 2 suivant :

L= $\{a^n b^n, n \geq 1\}$

L'automate de la figure précédente permet de lire les mots de ce langage en procédant comme suit :

1. Lire les a, les stocker dans la pile et ne pas changer d'état
2. A la rencontre du premier b, dépiler un a et ce b et puis changer d'état
3. Dépiler un a pour chaque b rencontré.

4. Si les a de la pile se terminent au même moment que les b lus, alors le mot appartient au langage.

Voici l'exécution complète de l'automate qui va lire les mots du langage L :

$\#S_0a \rightarrow \#aS_0$ (Empiler le a).

$aS_0a \rightarrow aaS_0$ (Empiler tous les a rencontrés).

$aS_0b \rightarrow S_1$ (Changer d'état une fois rencontré le premier b).

$aS_1b \rightarrow S_1$(Dépiler le dernier a et le premier b tout en restant dans le même état).

$\#S_1 \rightarrow \#$ acceptation (La pile est vide).

Exercice *:* Soit les langages suivants :

- **$L_1 = \{abc^md^m, m>=0\}$**
- **$L_2 = \{a^nb^mc^md^n, n>=1, m>=0\}$**
- **$L_3 = \{a^nb^ma^n, n>=1, m>=0\}$**
- **$L_4 = \{a^nb^mc^pd^m, n,m>=0, p>=2\}$**
- **$L_5 = \{a^nb^mc^{n-m}, n>=m>=1\}$**
- **$L_6 = \{wcw^R, w\in(0, 1)^*, c\in\pi\}$**
- **$L_7 = \{0^nwcw^R1^n, w\in(a,b)^*, c\in\pi\}$**
- **L_8= {nombres binaires pairs}**
- **L_9= {nombres décimaux quelconques}**
- **$L_{10} = \{a^nb^m, n< >m\}$**

1. Donner une grammaire de type2 qui génère chaque langage.
2. Donner les automates à pile acceptant à pile vide ces langages.