République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Boudiaf de M'sila
Faculté de Mathématiques et d'Informatique
Département d'Informatique

Initiation à la théorie des langages

Cours, exercices et travaux pratiques

Par : Dr. HEMMAK Allaoua

- Janvier 2022 -

Cours de ThL / PRESENTATION

ENSEIGNANT : A. HEMMAK (depuis 2013)

MATIERE : ThL (Théorie des Langages)
Niveau : 2° LICENCE ; UEF 1.
Coefficient : 03 ;
Crédits : 05 ;
VH = 1 COURS + 1 TD + 1 TP / semaine = 1.30 h + 1.30 h + 1.30 h / semaine ;
Outils TP : EDI DEV C++ ; FLEX ;
Evaluation : examen 50 % ; contrôle continu 50 % ;

Objet
Etude des langages formels et leurs outils d'analyse, de conception et d'implémentation tels que les grammaires formelles et les automates.

Prérequis
Algèbre fondamentale et algorithmique.

Contenu : 7 chapitres

I. Les bases de la THL ;
II. Grammaires formelles ;
III. Grammaires algébriques
IV. Langages réguliers ;
V. Automates finis ;
VI. Automates à pile ;

Objectifs

- Initiation à la compilation ;
- Initiation au TAL (Traitement Automatique des Langages) ;
- Acquisition de compétences d'analyse et d'implémentation d'outils de langages naturels et de langages formels.

Préface

La théorie des langages est une discipline qui se situe entre les mathématiques, la linguistique et l'informatique et s'intéresse à l'étude des concepts, outils et techniques permettant de formaliser les langages afin que l'on puisse par la suite traduire ces formalismes mathématiques en programmes informatiques pour, au final, automatiser ces langages, on parle alors du Traitement Automatique des Langages (TAL). Ainsi, le TAL consiste essentiellement à fournir des outils logiciels permettant d'analyser, de générer et de reconnaitre les mots du langage à automatiser. Tous types de langages sont ainsi visés par cette approche, à l'égard :

- des langages naturels ou humains (tels que l'arabe, l'anglais, ..) comme MS Word ou OpenOffice ;
- des langages de programmation tels que les compilateurs et interpréteurs comme java, C, pascal, python, … ;
- 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, … ;

Formellement, comme en linguistique, tout langage possède son propre alphabet, c'est l'ensemble des symboles (ou lettres, briques de base) constituant les éléments du langage. Un langage est généralement défini par l'ensemble de ses éléments, ainsi, le C est l'ensemble de programmes C, le français est l'ensemble des textes français. Un langage est bien souvent un ensemble infini à l'opposé de son alphabet. C'est la raison triviale pour laquelle on a recours au TAL, puisque on ne peut, dans la plupart des cas, dresser une liste exhaustive de tous les éléments d'un langage.

Ce cours est destiné aux étudiants de 2ème année licence informatique afin de leurs permettre :

- de bien comprendre les langages déjà étudiés ;
- de faciliter l'apprentissage de nouveaux langages ;
- de s'adapter aux outils de TAL utilisés dans leurs quotidien ;
- de s'initier au TAL ;
- d'assimiler le fonctionnement des compilateurs et interpréteurs ;
- d'acquérir des compétences de formalisation, de conception et d'implémentation d'outils de TAL.
- de renforcer leurs compétences d'élaborer des algorithmes et implémenter des programmes.

Ce cours est composé de six chapitres :

- Le premier chapitre expose les notions fondamentales de la THL ainsi que sa terminologie.
- Le deuxième chapitre est consacré aux grammaires formelles.
- Vu leur importance dans l'étude des langages de programmation, les grammaires algébriques sont abordées en détail au chapitre 4.

- Le quatrième chapitre est dédié au type le plus restreint de langages, en l'occurrence les langages réguliers, vu leur importance dans plusieurs situations.
- Les chapitres 5 et 6 sont consacrés aux automates suivant la hiérarchie de complexité.

Pour bien digérer le cours, chaque chapitre est suivi d'une série d'exercices et une séance de travaux pratiques.

TABLE DES MATIERES

CHAPITRE I
LES BASES DE LA THL

1. Définitions & terminologie…….……..……..9
   1.1. Lettre…….……..……9
   1.2. Alphabet…….……..……9
   1.3. Mot……..……9
   1.4. Parties d'un mot……..……10
   1.5. Operations sur les mots……..…....10
      1.5.1. Miroir……..……10
      1.5.2. Longueur……..……....10
      1.5.3. Concaténation……..……11
      1.5.4. Puissance et Fermeture transitive d'un mot……..……11
   1.6. Puissance et Fermeture transitive d'un alphabet……..……11
   1.7. Langage formel…….12
2. Operations sur les langages…….12
   2.1. Somme de deux langages…….12
   2.2. Produit de deux langages…….12
   2.3. Puissance et fermeture transitive d'un langage……12
   2.4. Quotients de deux langages…….13
   2.5. Miroir d'un langage…….13
3. Outils de description de langages……13
4. Exercices ……15
5. Travaux pratiques ……..……18

CHAPITRE II
GRAMMAIRES FORMELLES

1. Exemples introductifs…….……..……21
2. Définition…….……22
3. Langage engendré par une grammaire…….……22
4. Types de grammaires, hiérarchie de Chomsky……24
5. Exercices……..……26
6. Travaux pratiques...……..……29

CHAPITRE III
GRAMMAIRES ALGEBRIQUES

1. Dérivation gauche et dérivation droite…….32
2. Arbre de dérivation……...32

3. Grammaire ambigüe…….33
4. Grammaire réduite……33
5. Grammaire propre……34
6. Récursivité gauche……35
7. Factorisation gauche…….36
8. Forme normale de Chomsky……36
9. Forme normale de Greibach…….36
10. BNF……..…36
11. Exercices……..38
12. Travaux pratiques...……..……39

CHAPITRE IV
LANGAGES REGULIERS

1. Langages réguliers…….……..…42
   1.1.Définition…….……..…42
   1.2.Propriétés…….……..……42
2. Expressions régulières…….……42
   2.1.Définition…….……..……42
   2.2.Propriétés…….……..……43
3. Lemme d'Arden…….…..……44
4. Théorème du gonflement…….……45
5. Expression régulière associée à une grammaire régulière……..……..……45
6. Exercices……..……47

CHAPITRE V
AUTOMATES FINIS

1. Introduction……51
2. Définition et terminologie……..51
   2.1. Représentation graphique……52
   2.2. Exécution et configuration d'un AF, langage reconnu par un AF……..53
   2.3. Etat puits, état poubelle…….……..54
   2.4. AFs équivalents……...……54
   2.5. AF complet……..56
3. Operations sur les AFs……57
   3.1. AF complémentaire……..……57
   3.2. Somme de deux AFs……59
   3.3. Produit de deux AFs……59
4. AFs et expressions régulières…….59
5. AFs et grammaires de type 3……..60
6. Automate minimal…… 60

7. Automate fini indéterministe……..….61
8. Exercices……..……...……61

CHAPITRE VI
AUTOMATES A PILE

1. Introduction……65
2. Rappels sur les piles…….…..66
3. Définition…….…..…….67
4. Configuration et exécution…….…...……68
5. Les critères d'acceptation……...……69
6. Automates à pile déterministes……...……71
7. Automates à pile et langages algébriques……...……72
8. Grammaire algébrique vers automate à pile…….……...……73
9. Automate à pile vers grammaire algébrique……..….……74
10. Clôture des langages algébriques……....……78
11. Exercices…….……..…..…….81

Références bibliographiques…….……..…..……..84

CHAPITRE I

LES BASES DE LA ThL

Plan

1. Définitions & terminologie
   2.1.Lettre
   2.2. Alphabet
   2.3. Mot
   2.4. Parties d'un mot
   2.5. Operations sur les mots
      2.1.1. Miroir
      2.1.2. Longueur
      2.1.3. Concaténation
      2.1.4. Puissance et Fermeture transitive d'un mot
   2.6.Puissance et Fermeture transitive d'un alphabet
   2.7. Langage formel
3. Operations sur les langages
   3.1. Somme de deux langages
   3.2. Produit de deux langages
   3.3. Puissance et fermeture transitive d'un langage
   3.4. Quotients de deux langages
   3.5. Miroir d'un langage
4. Outils de description de langages
5. Exercices
6. Travaux pratiques

1. Définitions & terminologie

1.1. Lettre ou symbole = entité élémentaire (atomique, brique de base) constituant les mots d'un langage.

✍ **Exemples : a , b , 0 , 1 , if , x1 , x23 , while , → , ⇒ , (, # , {, …**
Une lettre dans un jeu d'échecs est un coup comme (f1)-(e3).

☞ **Attention**
Une lettre ≠ un caractère au sens informatique ;
Une lettre ≠ une lettre au sens linguistique ;
Une lettre = un caractère ou une suite de caractères, selon le contexte.

1.2. Alphabet = ensemble de toutes les lettres d'un langage.

✍ **Exemples**
Alphabet des variables du langage C = {a,b,..,z, A,B, … , Z, 0, 1, …, 9, _ , $}.
Alphabet des expressions arithmétiques en C = {+,-,*,/,%} U alphabet des constantes arithmétiques U alphabet des variables arithmétiques. (U = union).
Alphabet de la langue française = {a,b,..,z, A,B, … , Z, é,è, ç, à, ù, 0, 1, …, 9, ., ; , : ,,, !, ?}.

☞ **Attention**
Alphabet de la langue française au sens linguistique = {a,..,z} qui est loin de l'alphabet du "*langage formel français*", qui est l'ensemble de tous les caractères dont on a besoin pour formaliser, concevoir puis implémenter des outils d'analyse, de correction et de génération des textes en français.

1.3. Mot = une suite de lettres de l'alphabet.

✍ **Exemples :** university ; aabbbab ; 01000001 ; 01001000 ; if(x1==0) x=x23+1 ; .

Mot vide = un mot qui ne contient aucune lettre, noté ε.

☞ **Remarques**
- Dans un langage de programmation, un mot peut désigner une constante, une variable, une instruction ou le programme tout entier.
- Dans un langage naturel : un mot peut désigner un mot au sens linguistique, une phrase, un paragraphe ou encore tout un texte.
- Dans un jeu d'échecs, un mot peut être un coup, une série de coups ou toute une partie.

1.4. Parties d'un mot (préfixe, suffixe, facteur, quotient)

Soient u,v, w, α des mots définis sur un alphabet A.

- Si u= αv alors α est dit préfixe de u, de plus, si α ≠ u et α≠ε alors α est dit préfixe propre de u.
- Si u= v α alors α est dit suffixe de u, de plus, si α ≠ u et α≠ε alors α est dit suffixe propre de u.
- Si u= vαw alors α est dit facteur de u.
- Si u= αv alors α est dit quotient gauche de u par v et noté uv-1.
- Si u= v α alors α est dit quotient droit de u par v et noté v-1u.

✍ **Exemples**

Si u=abbc alors

- Les préfixes de u sont ε,a,ab,abb et u. ses préfixes propres sont a,ab,abb.
- Les suffixes de u sont ε,c,bc,bbc et u. ses suffixes propres sont c,bc,bbc.
- En C, les trois formes de l'instruction if : if simple , if avec else et if avec else et elseif ont le même préfixe.
- dans le moteur de recherche Google, quand vous tapez une chaine, Google vous propose des phrases ayant votre chaine comme préfixe.
- Dans le shell Unix, la commande ls thl* vous fournit la liste des fichiers et dossiers dont le préfixe de leurs noms est thl.
- Avec votre smartphone, composer les chiffres 0772, Android vous proposera les contacts dont les numéros ayant 0772 comme préfixe.
- En recherchant un contact, si vous tapez abd, on vous proposera les contacts dont les noms ayant abd comme facteur.
- Dans le moteur de recherche Google, quand vous tapez le mot "*university*", ce dernier vous propose les requêtes dont le préfixe est "*university*" comme "*university of M'sila"*.

1.5. Operations sur les mots

1.5.1. Miroir (ou transposée) d'un mot u est le mot noté $\bar{u}$ obtenu par inversion de l'ordre des lettres de u.

✍ **Exemples**

- Si u=abbc alors $\bar{u}$ = cbba
- Si u= " x+=y " alors $\bar{u}$ = "y=+x "
- Si u= " radar " alors $\bar{u}$= u (mot palindrome).

1.5.2. Longueur d'un mot u, notée |u| est le nombre de lettres de u.

Convention : |ε|=0

✍ **Exemples**

| abbc |=4

|if(x1==0) x=x23+1 ;|= 17 si l'alphabet est {i,f,(,), ;,x,=,1,2,3,+}

|if(x1==0) x=x23+1 ;|= 12 si l'alphabet est {if,(,), ;,x, x23, =,1,+}

La longueur d'un mot u par rapport à la lettre a, notée $|u|_a$ est le nombre d'occurrences de a dans u.

✍ **Exemples**

| abbc $|_b$=2 : | abbc $|_a$=1 ; | abbc $|_d$=0 ; |if(x1==0) x=x23+1 ;$|_=$= 3.

1.5.3. Concaténation

La concaténation de deux mots u et v, dans cet ordre, est le mot noté u.v ou uv tout court, obtenu par juxtapositions des lettres de v après celles de u.
Remarque : uv est également appelé produit de u et y dans cet ordre.

✍ **Exemples**

- Si u= "University" et v= "of M'sila" alors uv = "University of M'sila"
- Si u= "x++" ; et v= "x=2" ; " alors uv = " x++ ; x=2 ; "

Propriétés de la concaténation

- La concaténation est associative : $\forall u ;v ;w\ (uv)w=u(vw)$
- En général, elle n'est pas commutative.
- ε en est l'élément neutre : $\forall u\ u\varepsilon=\varepsilon u=u$
- Morphisme : $|uv|=|u|+|v|$ et $|uv|a=|u|_a+|v|_a$

1.5.4. Puissance et fermeture transitive d'un mot

La puissance d'un mot u est un mot noté u^n défini comme suit : $u^2=uu$;
Si u est un mot et n un entier strictement positif alors $u^n=u^{n-1}u= u\ u^{n-1}$;
Et par convention : $u^0= \varepsilon$ et $u^1= u$
La fermeture transitive d'un mot u est un mot noté u* tel que : $u^*= u^0$ ou u^1 ou u^2 ou…

☞ **Remarque :** $|u^n|=n|u|$

1.6. Puissance et Fermeture transitive d'un alphabet

Soit A un alphabet.
L'ensemble des mots de longueur 2 sur A est noté A^2.
L'ensemble des mots de longueur 3 sur A est noté A^3.
L'ensemble des mots de longueur N sur A est noté A^n.
Et par convention : $A^n= \{\varepsilon\}$ et $A^1= A$
La fermeture transitive (ou itération de Kleene) de A est l'ensemble de mots sur A noté A* défini par : $A^*= A^0 \cup A^1 \cup A^2 \cup\ldots = \bigcup_0^\infty A^i$
Et particulier $A^+= A^1 \cup A^2 \cup\ldots = \bigcup_1^\infty A^i$

☞ **Remarques**

- A* est l'ensemble de tous les mots sur A.
- $A^*= A^+U\{\varepsilon\}$

1.7. Langage formel

Un langage défini sur un alphabet A est un ensemble de mots sur A.

Un langage formel sur A est une partie de A*.

✍ **Exemples**

- Sur l'alphabet A= {a,b}, on peut définir les langages suivants :

L1 : langage de mots commençant par a ;
L2 : langage de mots contenant autant de a que de b ;
L3 : langage de mots de longueur paire ;
L4 : langage des mots palindromes ;
…

- Langage des programmes C contenant if ;
- Langage des noms de variables en C ;
- Langage des expressions arithmétiques en C ;
- Langage des nombres réels en C ;
- Langage des adresses IP4 ;
- Langage des adresses http ;
- Langage des adresses mail ;

2. Operations sur les langages

Operateurs binaires

2.1. Somme de deux langages

Soient A un alphabet, L1 et L2 deux langages sur A.

$L1+L2 = L1 \cup L2 = \{u \in A^*/ u \in L1 \text{ ou } u \in L2\}$

2.2. Produit (ou concaténation) de deux langages

Soient A un alphabet, L1 et L2 deux langages sur A. $L1.L2 = L1\ L2 = \{uv \in A^*/ u \in L1 \text{ et } v \in L2\}$

2.3. Quotients de deux langages

Soient A un alphabet, L1 et L2 deux langages sur A.

Le quotient gauche de L1 par L2 : $L2^{-1}.\ L1 = \{v^{-1}u \in A^*/ u \in L1 \text{ et } v \in L2\}$

Le quotient droit de L1 par L2 : $.\ L1.\ L2^{-1} = \{ u\ v^{-1} \in A^*/ u \in L1 \text{ et } v \in L2\}$

Operateurs unaires

2.4. Puissance et fermeture transitive d'un langage

Soient A un alphabet et L un langage sur A. $L^2 = L.L = \{uv \in A^*/\ u \in L \text{ et } v \in L\}$
Si n est un entier strictement positif : $L^n = L^{n-1}.L = L.L^{n-1}$
Et par convention : $L^0 = \{\varepsilon\}$ et $L^1 = L$
La fermeture transitive (ou itération de Kleene) de L est L* défini par :
$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots = \bigcup_0^\infty L^i = \sum_0^\infty L^i$
Et particulier $L^+ = L^1 \cup L^2 \cup \ldots = \bigcup_1^\infty L^i = \sum_1^\infty L^i$

☞ **Remarque :** $L^* = L^+ \cup \{\varepsilon\}$
Attention : il ne faut pas confondre entre le langage vide ∅, qui ne contient aucun mot, et le langage {ε} qui contient un seul mot qui est le mot vide ε. $\emptyset \neq \{\varepsilon\}$

2.5.Miroir d'un langage

Soient A un alphabet, L un langage sur A.
Le miroir de L : $\overline{L} = \{\overline{u} \in A^*/\ u \in L\}$
Le quotient droit de L1 par L2 :. $L1.\ L2^{-1} = \{\ u\ v^{-1} \in A^*/\ u \in L1 \text{ et } v \in L2\}$

✍ **Exemples**
Soient deux langages L1 et L2 définis Sur l'alphabet A= {a,b,c} comme suit :
L1 = {a, ab, bb} ; L2 = {ab, ac}
L1+L2 = {a, ab, bb , ac}
L1.L2 = {aab, aac, abab , abac , bbab , bbac}
$L2^2$ = {abab, abac, acab , acac }
$L2^3 = L2.\ L2^2$ = {ababab, ababac, abacab , abacac , acabab, acabac, acacab , acacac }
L2* = {(ab+ac)*}

☞ **Remarques**
Sachant qu'un langage est un ensemble, donc, toutes les opérations ensemblistes, comme par exemple l'intersection, le complément, … ainsi que leurs propriétés, restent, bien évidemment, valables pour les langages.

3. Outils de description de langages

Un langage peut être décrit par :

- La liste exhaustive de ses mots, s'il est fini, comme :
  Langage des mots clés du langage C = {include , if , while , do , scanf,…} ;
- Une expression verbale, comme : "langage de mots qui commencent par a" ;
- Une expression formelle, comme : L = {u∈A*/ |u|a=|u|b} ;
- Une grammaire, un ensemble de règles permettant d'analyser et de générer les mots du langage (voir chapitre 2), de la forme : phrase → Gs Gv ou Affectation → Variable = Expression ; en y spécifiant le vocabulaire et le symbole initial.

- Une expression régulière si le langage est régulier. Une expression régulière est une expression définie par des lettres et les opérateurs . ,+ , *. (voir chapitre 3). Comme : a(b+c)*ad
- Un automate, une machine abstraite, permettant de reconnaitre (accepter ou rejeter) les mots du langage. Un automate consiste en un ruban d'entrée, qui contient le mot à reconnaitre, une tête de lecture pour lire le mot, une unité centrale pour traiter le mot, et éventuellement une mémoire. (voir chapitres 4,5,6) schématisé comme suit (fig. 1.1.).

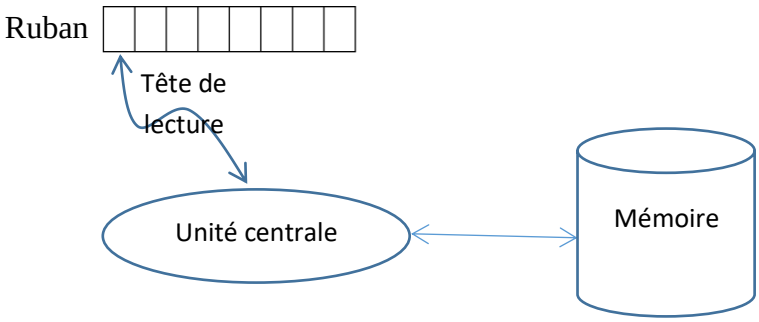


Fig. 1.1. Schéma logique d'un automate.

Le ruban contient le mot à reconnaitre, la tête de lecture sert à balayer et lire le mot de gauche à droite, la mémoire est utilisée pour stocker momentanément certains symboles et l'unité centrale consiste en une fonction de transition d'états. En partant d'un état initial, la fonction de transition change l'état de l'automate selon l'état actuel, le contenu de la mémoire et le symbole lu. L'automate devrait fournir une réponse si le mot est accepté ou rejeté.

Série d'exercices I

Exercice 1

Proposer un alphabet pour chacun des langages suivants :

1. Langage des nombres binaires ;
2. Langage des nombres romains ;
3. Langage des identificateurs en C ;
4. Langage des nombres réels en C ;
5. Langage d'une calculatrice simple ;
6. Langage des services d'un opérateur téléphonique mobile.
7. Langage des chaines d'ADN
8. Langage de notation de graphes dont les sommets sont des entiers.
9. Transcription phonétique des sons du français.

Exercice 2

On considère l'alphabet X = {a,b, c}. On rappelle que |w| représente la longueur du mot w.
Soit deux mots w = ababc et q = caba.
1. Calculez w^0, w^1 et w^2 , wq^2w
3. Calculez $|w|_{ab}$, $|(ab)^4|$ et $|(ab)^4|_{aba}$
4. Donnez les préfixes, les préfixes propres, les suffixes et les suffixes propres de q
5. Donnez le miroir du mot wq.

Exercice 3

Soit l'alphabet $\sum$ = {a; b}.
1. Etant donnés les mots u = aa et v = bba, écrire les mots uv, $(uv)^2$ et u^3v.
2. Donner la transposée de u et v. que peut on conclure?
3. Quel est le quotient gauche de v par b?
4. Enoncer tous les mots de longueur 2 définis sur $\sum$.
5. Soient les ensembles :

L1 = {uv / u $\in \sum+$, v $\in \sum+$ } ;
L2 = {uv / u $\in \sum*$, v $\in \sum+$ } ;
L3 = {uv / u $\in \sum*$, v $\in \sum*$ }
A quoi correspondent ces ensembles ?

Exercice 4

On considère les ensembles de mots E_1 et E_2 définis sur l'alphabet A = {0; 1; 2} comme suit :
- E_1 est l'ensemble des mots de longueur paire,
- E_2 est l'ensemble des mots comportant autant de 0 que de 1 et autant de 1 que de 2.

Définir de façon plus formelle ces deux ensembles et déterminer pour chacun d'eux si la concaténation est une loi interne et si le mot vide en est un élément.

Exercice 5

Sur l'alphabet A = {0, 1}, on considère les langages L_1 et L_2 définis par
$L_1 = \{01^n / n \in N\}$ $L_2 = \{0^n1 / n \in N\}$. Définir les langages $L_1 L_2$, $L_1 \cap L_2$ et L_1^2 .

Exercice 6

Sur l'alphabet A = {a, b}, on considère le langage L_1 des mots formés de n fois la lettre a suivi de n fois la lettre b, et le langage L_2 des mots comportant autant de a que de b.

- Définir formellement ces deux langages.
- Que sont les langages suivants : $L_1 \cup L_2$, $L_1 \cap L_2$, L_1^2 , L_2^2 ?
- Que peut-on dire de L_1^* et L_2^* par rapport à L_1 et L_2 ?

Exercice 7

Soient A un alphabet et L, L_1, L_2, L_3 et L_4 cinq langages sur A. Montrer que :

1. si $L_1 \subseteq L_2$ et $L_3 \subseteq L_4$ alors $L_1.L_3 \subseteq L_2.L_4$
2. si $L_1 \subseteq L_2$ alors $L_1^* \subseteq L_2^*$
3. $(L^*)^* = L^*$
4. $L^+ = LL^* = L^*L$
5. $L.L^* + \varepsilon = L^*$

Exercice 8

Soient A un alphabet et L, L_1, L_2, L_3 quatre langages sur A. Parmi les propriétés suivantes, Lesquelles sont vraies ?

1. $L + \varnothing = \varnothing + L = L$
2. $L_1 + L_2 = L_2 + L_1$
3. $(L_1 + L_2) + L_3 = L_1 + (L_2 + L_3)$
4. $L_1.L_2 = L_2.L_1$
5. $L + A^* = A^* + L = A^*$
6. $(L_1.L_2).L_3 = L_1.(L_2.L_3)$
7. $\{\varepsilon\}.L = L.\{\varepsilon\} = L$
8. $\varnothing.L = L.\varnothing = \varnothing$
9. $L_1.(L_2 + L_3) = L_1.L_2 + L_1.L$

Exercice 9

Soit A = {a, b} un alphabet. Ecrire les expressions rationnelles (régulières) qui dénotent les langages suivants :

- L_1 = l'ensemble des mots de A^* se terminant par a
- L_2 = l'ensemble des mots de A^* contenant au moins un a
- L_3 = l'ensemble des mots de A^* contenant au plus un b
- L_4 = l'ensemble des mots de A^* contenant un nombre pair de a
- L_5 = l'ensemble des mots de A^* contenant autant de a que de b

Avez-vous rencontré une difficulté pour décrire l'un de ces langages ? Qu'en conclure ?

Exercice 10

Montrer que les relations “être-préfixe-de”, “être-suffixe-de” et “être-facteur-de” sont des relations d’ordre partiel sur A*.

Exercice 11

Soient les langages L1 = {cab, ba} et L2 = {aa, ba, ε} sur l'alphabet Σ = {a, b, c},. Décrire chacun des langages suivants L1 ∩ L2, L1 ∪ L2, L1 \ L2, L2 \ L1, L1 - L2, L2 - L1, $L1^2$, $L2^2$, L2*, $L2^+$, Σ* .

Exercice 12

Soit l’alphabet Σ = {a, b} et soient les langages :

L1 = $\{a^n b^p : n, p \in N\}$

L2 = $\{a^n b^n : n \in N\}$

L3 = $\{a^n : n \in N\}$ = a*

L4 = $\{b^n : n \in N\}$ = b*

1 - Donner des mots de chacun des langages. 2 - Déterminer l’intersection L1 ∩ L2.

3 - Déterminer les mots de L1 qui ne sont pas dans L3 puis ceux de L2 qui ne sont pas dans L4 (en d’autres termes, déterminer L1 \ L3 et L2 \ L4).

4 - Pourquoi peut-on écrire que L1 = L3 · L4, c’est-`a-dire que $\{a^n b^p : n, p \in N\}$ = a*b* ? Peut-on aussi écrire que L2 est égal aussi à a*b* ?

Travaux pratiques I

Sous l'EDI DEV C++, créer un espace de travail que l'on appelle TPTHL et dans ce dossier créer un dossier que l'on appelle TP1. Créer un nouveau projet console et introduire le code ci-dessous, puis le compiler.

1) Considérons un langage défini sur l'alphabet A= {a,b}.
   a) Exécuter le programme ci-dessous en introduisant comme input chacun des mots suivants : aaaba ; bbba ; a

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main ()
{
  char mot[100];
  printf ("Entrer un mot : ");
  gets (mot);
  printf ("Le résultat  est %u \n",strlen(mot));
  system("PAUSE");
}
```

   b) Exécuter ce code avec des mots de votre choix.
   c) Que fait ce programme ? (modifier alors l'instruction printf affichant le résultat).
   d) Modifier ce code afin qu'il fournisse le nombre de b dans les mots précédents. Essayer d'autres mots.
   e) Soit un L1 le langage défini sur l'alphabet A des mots qui commencent et se terminent par a.
   Ecrire un programme C qui teste si un mot appartient à L1.
   f) Même question pour le langage L2 des mots contenant autant de a que de b.
   g) Même question pour le langage L3 des mots de la forme a*b*.
   h) Même question pour le langage L4 des mots de la forme $a^n b^n$.
   i) Tester chacun de ces programmes en l'exécutant avec des mots bien choisis.

2) Considérons un langage défini sur l'alphabet B={if , x11 , y1 , z , (,) , = , ;}.
   a) En s'aidant du programme de la question 1), écrire un programme C permettant de calculer la longueur du mot u="if(x11==y1) z=y1;"
   b) Essayer ce programme avec d'autres mots définis sur cet alphabet.
3) Taper, compiler puis exécuter le programme suivant en utilisant comme input des mots de votre choix.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main ()
{
  char mot[100];
  printf ("Entrer un mot: ");
  gets (mot);
  int n = strlen(mot);
  for(int i=0;i<n; i++)
      printf ( "%c", mot[n-i-1]);
      printf ("\n");
  system("PAUSE");
}
```

a) **Que fait ce programme ?**
b) **Modifier ce programme afin qu'il teste si un mot est un palindrome.**

4) **a) Ecrire un programme C qui lit deux mots puis les concatène et affiche le résultat.**

c) **Ecrire un programme C qui lit un mot puis affiche ses préfixes propres.**
d) **Ecrire un programme C qui lit un mot puis affiche ses suffixes propres.**
e) **Tester ces programmes à l'aide d'exemples appropriés.**

CHAPITRE II

GRAMMAIRES FORMELLES

Plan

1. Exemples introductifs
2. Définition
3. Langage engendré par une grammaire
4. Types de grammaires, hiérarchie de Chomsky
5. Exercices
6. Travaux pratiques

1. Exemples introductifs

Exemple 1

En linguistique, une grammaire est un outil de description et d'analyse des langages. Il s'agit d'un ensemble de règles par lesquelles sont construites les phrases valides du langage. Voici un exemple trivial de la grammaire anglaise :

sentence → <subject> <verb-phrase> <object>
subject → This | Computers | I
verb-phrase → <adverb> <verb> | <verb>
adverb → never
verb → is | run | am | est | tell
object → the <noun> | a <noun> | <noun>
noun → university | world | cheese | mouse | lies

En utilisant ces règles ou productions, on peut dériver des phrases simples comme :
This is the university.
Computers run the world.
the cheese eat the mouse.
I never tell lies.
Voici la dérivation de la première phrase :

<sentence> → <subject> <verb-phrase> <object>
→ This <verb-phrase> <object>
→ This <verb> <object>
→ This is <object>
→ This is the <noun>
→ This is the university

Exemple 2

Voici une portion de règles de la grammaire du langage C :
<chiffre> → 0 |1 |2 |3 |4 |5 |6 |7 |8 |9|
<chiffre non nul> → 1 |2 |3 |4 |5 |6 |7 |8 |9|
<entier> → 0 | <chiffre non nul> <chiffre>
<entier signé> → - <entier> | + <entier>
<affectation> → <variable> = <expression> ;
<if> → if (<cond>) <inst> ; | if (<cond>) <inst> ; else <inst> ;
Par exemple, Pour vérifier la syntaxe de l'instruction if(x1==10) x23=20 ; on doit rechercher s'il existe une suite de règles parmi celles citées ci-dessus qui permettent de générer cette instruction.

2. Définitions

Une grammaire "générale" est un ensemble de règles permettant d'analyser et de générer les mots d'un langage.

Une grammaire formelle G est un quadruplet (T, N, S, R) tel que :

- ✓ T est un ensemble fini et non vide de symboles dits *terminaux*, T est dit *vocabulaire terminal* de G.
- ✓ N est un ensemble fini et non vide de symboles dits *non-terminaux*, T est dit *vocabulaire non-terminal* de G. T et N sont disjoints et T∪N est dit vocabulaire de G, on le désigne par V.
- ✓ S est un élément spécifique de N dit *symbole initial* ou *axiome* de G.
- ✓ R est une relation finie et non vide de V*.N.V* vers V* (c.-à-d. R⊂ V*.N.V* × V*) dite *ensemble de règles de production* de G. Si (α,β)∈R on écrit : α→β et on lit : "*α produit β*" ou "*α donne β*". α→β s'appelle : "*règle de production*".
  (α→β)∈R veut dire α∈ V*.N.V* et β∈ V* c.-à-d. α et β sont des mots qui pourraient contenir des terminaux et/ou des non-terminaux mais α doit obligatoirement contenir au moins un non-terminal. β pourrait être, éventuellement, ε ; ainsi |α|≥1 bien évidemment.
  Si (α→β)∈R et (α→γ)∈R alors on écrit, pour abréviation : α→β|γ.

✍ **Exemples**

G1 = ({a,b}, {S,X} , S , {S→aXa , X→bX|ε}) ;

G2 = ({a,b}, {S} , S , {S→aSb|ε}) ;

G3 = ({a , + , * , - , / , %}, {E} , E , {E→E+E|E*E | E/E | E-E | E%E | (E) | a }) ;

☞ **Remarque**

La règle X→bX peut être utilisée plusieurs fois, elle est alors dite récursive.

3. Langage engendré par une grammaire

Soit une grammaire G= (T, N, S, R) et V =T∪N.

Si α→β∈R et u , v deux mots de V* tels que u=u'αu'' et v=v'βv'' alors on écrit : u⇒v et on lit : "*u se réécrit en v*". u⇒v s'appelle "*production*" ou une "*réécriture*".

C.-à-d. v peut être obtenu de u par substitution de α par β. (i.e. en appliquant la règle : α→β, on peut alors écrire $u \overset{k}{\Rightarrow} v$ tel que k est le numéro de la règle α→β dans R.)

Si u⇒u1⇒u2⇒…. ⇒v, alors on écrit : $u \overset{*}{\Rightarrow} v$ et on lit : "*u dérive v*". $u \overset{*}{\Rightarrow} v$ s'appelle une dérivation.

Une dérivation est alors une suite de productions.

Si v∈T alors $S \overset{*}{\Rightarrow} v$ est la dérivation de v par G.

Le langage engendré par la grammaire G est l'ensemble de mots de T* pouvant être dérivés de S par G. Autrement dit : $L(G)=\{x \in T^* / S \overset{*}{\Rightarrow} x\}$

✍ **Exemples**

Soit la grammaire G1 = ({a,b}, {S,X} , S , {S→aXa , X→bX|ε}) ;

- Donner la dérivation du mot v=abba par G1.
- Le mot u=ababa admet-il une dérivation par G1 ?
- Calculer L(G1).

✍ **Solution**

- Dérivation du mot v=abba par G1 :

Numérotons les 3 règles de G1 de 1 à 3.

on a : $S \overset{1}{\Rightarrow} aXa \overset{2}{\Rightarrow} abXa \overset{2}{\Rightarrow} abbXa \overset{3}{\Rightarrow} abba$, la dérivation du mot v=abba par G1 est alors 1,2,2,3 ; autrement dit v∈L(G1).

- Le mot u=ababa admet-il une dérivation par G1 ?

on a : $S \overset{1}{\Rightarrow} aXa \overset{2}{\Rightarrow} abXa$, pour dériver u de S par G1, on doit appliquer une regle qui substitue X par une chaine qui commence par a, or cette regle n'existe pas dans G1, donc le mot u n'a pas de dérivation par G1 ; autrement dit u∉L(G1).

- Calculons L(G1).

on a : $S \overset{1}{\Rightarrow} aXa \overset{2*}{\Rightarrow} ab^*Xa \overset{3}{\Rightarrow} ab^*a$; donc L(G1) ={ ab^*a }.

De la même manière, on trouve :

L(G2) ={ $a^n b^n$ }.

L(G3) = langage des expressions arithmétiques en C.

Et inversement, on peut calculer une grammaire G connaissant le langage associé L(G).

Par exemple pour L(G) = {u∈A*/ $|u|_a=|u|_b$} , on aura :

G= ({a,b}, {S} , S , {S→ aSb| bSa| abS| baS|ε}) ;

Grammaires équivalentes

Deux grammaires sont dites équivalentes si et seulement si elles génèrent le même langage.

G≡G' ⇔ L(G) = L(G') .

☞ **Remarques**

- Pour toute grammaire, il existe un seul langage engendré par cette grammaire.
- Cependant, pour tout langage, il pourrait exister plusieurs grammaires qui l'engendrent.

✍ **Exemples**

La grammaire G1 = ({a,b}, {S,X} , S , {S→aXa , X→bX|ε}) engendre l'unique langage L(G1) ={ ab^*a }.

En revanche, le langage L ={ ab^*a } peut être engendré par chacune des grammaires suivantes :

G1 = ({a,b}, {S,X} , S , {S→aXa , X→bX|ε}) ;

G1' = ({a,b}, {S,X ,Y} , S , {S→aY , Y→Xa , X→bX|ε}) ;

G1'' = ({a,b}, {S,X} , S , {S→aX , X→bX|a }) ;

Ces grammaires sont, bien évidemment, différentes, mais équivalentes, puisqu'elles engendrent le même langage.

4. Types de grammaires, hiérarchie de Chomsky

Le linguiste américain Noam Chomsky s'est intéressé aux langages formels et les repartit selon les types de grammaires formelles. En 1956, il proposa alors une classification dite *hiérarchie de Chomsky* qui distingue entre quatre types de langages (ou de grammaires) (Fig. 2.1)

Soit une grammaire G= (T, N, S, R) et V =T∪N.

Type3

Une grammaire est de type 3 si toutes ses règles sont soit linéaires à gauche soit linéaires à droite.

Une règle est dite linéaire à gauche si elle de la forme : $A \rightarrow Ba$ ou $A \rightarrow a$ tels que $a \in T$; $A,B \in N$;

Une règle est dite linéaire à droite si elle de la forme : $A \rightarrow aB$ ou $A \rightarrow a$ tels que $a \in T$; $A,B \in N$;

Une grammaire de type 3 est dite *rationnelle ou régulière*.

Type2

Une grammaire est de type 2 si toutes ses règles sont de la forme : $A \rightarrow x$ tel que $x \in V^*$;

(C.-à-d. de la forme $\alpha \rightarrow \beta$ avec $|\alpha|=1$.)

Une grammaire de type 2 est dite *algébrique ou hors contexte*.

Type1

Une grammaire est de type 1 si toutes ses règles sont de la forme : $uAv \rightarrow uxv$ tel que $x \in V^*$;

(C.-à-d. de la forme $\alpha \rightarrow \beta$ avec $|\alpha| \leq |\beta|$.)

u et v représentent le contexte de la règle $uAv \rightarrow uxv$, ceci veut dire : A ne peut être substitué par x que s'il est placé entre u et v.

Une grammaire de type 1 est dite *contextuelle*.

Type0

Aucune restriction sur les règles de la grammaire.

Une grammaire de type 0 est dite *générale*.

☞ **Remarques**

- Si R contient des règles linéaires à gauche et d'autres linéaires à droite, la grammaire n'est pas de type 3 mais plutôt de type 2, vu l'exclusion évoquée dans la définition du type 3. Voir grammaire G1' ci-dessus en contre-exemple.
- Il est tout à fait clair que : type3 ⊂ type2 ⊂ type1 ⊂ type0.

- La règle de la forme A→ε, dite *règle vide* ou *ε-règle*, est exceptée des définitions ci-dessus ; c.-à-d. tout type de grammaire pourrait contenir cette règle.[1]
- Le type d'un langage est celui de la grammaire réduite qui l'engendre. Par exemple, dans l'exemple précédent, le langage L est engendré par 3 grammaires : G1 (de type 2), G1' (de type 2), G1'' (de type 3), donc le langage L est de type 3 (linéaire).

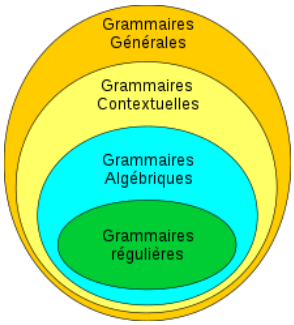


Fig. 2.1. Hiérarchie de Chomsky.

Recap.

type	Forme des règles	langage	Automate de reconnaissance
3	A→aB et A→a ou A→Ba et A→a	Réguliers ou rationnels	Automate fini
2	α→β avec |α|=1	Algébriques ou hors contexte	Automate à pile
1	α→β avec |α|≤|β|	Contextuels	Automate linéairement borné
0	/	Décidables	Machine de turing

[1] En fait, Chomsky se pencha uniquement sur les langages formels ne comprenant pas le mot vide.

Serie d'exercices II

Exercice 1

On considère la grammaire $G = (T, N, Ph, R)$ où

$T = \{$ un, une, le, la, enfant, garcon, fille, cerise, haricot, cueille, mange $\}$

$N = \{$ Ph, Gn, Gv, Df, Dm, Nf, Nm, V $\}$

$R = \{$ $Ph \rightarrow Gn\ Gv$

$Gn \rightarrow Df\ Nf \mid Dm\ Nm$

$Gv \rightarrow V\ Gn$

$Df \rightarrow$ une $\mid$ la

$Dm \rightarrow$ un $\mid$ le

$Nf \rightarrow$ fille $\mid$ cerise

$Nm \rightarrow$ enfant $\mid$ garçon $\mid$ haricot

$V \rightarrow$ cueille $\mid$ mange $\}$

Donner la dérivation de la phrase "une cerise cueille un enfant" .

Exercice 2

1. On considère la grammaire $G = (T, N, S, R)$ où T={b , c} ; N={S} ; R={$S \rightarrow bS \mid cc$}. Déterminer $L(G)$.

2. On considère la grammaire G=(T, N, S, R) où T={0,1} ; N={S} ; R={$S \rightarrow 0S \mid 1S \mid 0$}. Déterminer $L(G)$.

3. On considère la grammaire G = (T , N, S, R) où T = { a, b, 0} ; N={S, U}

R = { $S \rightarrow aSa \mid bSb \mid U$; $U \rightarrow 0U \mid \varepsilon$ }. Déterminer $L(G)$.

Exercice 3

1. Construire une grammaire pour le langage $L = \{ab^n a\ /n \in N\}$.
2. Construire une grammaire pour le langage $L = \{0^{2n} 1^n\ /n \geq 0\}$.

Exercice 4

On considère la grammaire G = (T,N,S,R) où

T = {a, b, c, d} ; N ={ S,U} ; R ={$S \rightarrow aU \mid c$; $U \rightarrow Sb \mid d$}

Donner le type de G et déterminer L(G).

Exercice 5

On considère le langage L des mots sur $\{a, b, c\}$ qui contiennent au moins une fois la chaine bac. Définir formellement L et construire une grammaire hors-contexte puis une grammaire régulière décrivant L.

Exercice 6

On considère le langage L des mots sur $\{0, 1\}$ qui représentent des entiers pairs non signés en base 2 (les mots de ce langage se terminent tous par 0 et ne commencent pas par 0, sauf pour l'entier nul). Définir formellement L et construire une grammaire régulière décrivant L.

Exercice 7

On considère la grammaire G = (T , N, S, R) où T = { a, b, c } ; N = { S, D, E }
R = { S → aSDE | c ; aD → ab ; bE → bc ; cD → DE ; bD → bb ; cE → cc }

1. Quel est le type de G ?
2. Ecrire la dérivation qui, partant de l'axiome, applique deux fois la première règle et une fois la seconde, et poursuivre la dérivation jusqu'à obtenir une chaine de terminaux.
3. En raisonnant par récurrence, déterminer L(G).

Exercice 8

Soit la grammaire G = (V_T, V_N, S, R) tels que : V_N = {S,Q,B} ; V_T = {a,b} ; axiome : S ;
Règles de réécritures : S → aQ ; Q → aQ ; Q → bB ; B → b

1. Quel est le type de cette grammaire ?
2. Dites si les mots suivants appartiennent au langage engendré par cette grammaire,
   Si oui donnez la structure arborescente associée à leur dérivation : baabab ; ab ; aaabb ; aaab.
3. Après la suite de symboles a consécutifs on désire pouvoir avoir une suite de plus de 2 symboles b consécutif. Ajoutez une règle qui permette cela.
4. Comment faire pour que la suite de symboles b puisse se réduire à un seul b ?

Exercice 9

Ecrivez une grammaire capable de générer des mots contenant autant de symboles a que de symboles b de la forme a^nb^n.

Exercice 10

La grammaire suivante est de type 1 car toutes ces règles sont non-raccourcissantes :
V_N = {S,B,C } ; V_T = {a,b,c} ; axiome : S ;
Règles de réécritures :

S → aSBC
S → abC
CB → BC
bB → bb
bC →
bc
cC →
cc

1. Exercez-vous à faire des dérivations de cette grammaire.
2. Dites si les mots suivants appartiennent au langage engendré par cette grammaire :
   baabab ; abc ; aaabbbccc ; abbccc ; aabcc .

Exercice 11

Soit la grammaire $G = (V, \Sigma, P, S)$, avec $V = \{a, b, S\}$, $\Sigma = \{a, b\}$ et $P = \{S \rightarrow aSa; S \rightarrow bSb; S$

$\rightarrow$ }.

1. Soit $G' = (V, \Sigma, P', S)$, avec $P' = P \cup \{S \rightarrow SS\}$. Montrez que *aabaab* $\in L(G')$.
2. Montrez ensuite que G' est ambigüe.
3. Quel est le langage engendré par G ? Justifier.
4. Pourquoi G n'est pas ambigüe ?

Exercice 12

Proposez une grammaire qui permet d'engendrer les formules de la logique des propositions.

Exercice 13

Soit la grammaire $G = (V, \Sigma, P, S)$, avec $V = \{\text{if,then,else},a,b,S\}$, $\Sigma = \{\text{if,then,else},a,b\}$ et $P = \{S \rightarrow \text{if } b \text{ then } S \text{ else } S ; S \rightarrow \text{if } b \text{ then } S ; S \rightarrow a\}$.

1. Démontrez que cette grammaire est ambigüe
2. Proposez une solution pour lever l'ambiguïté

Travaux pratiques II

Sous l'EDI DEV C++, dans le dossier TPTHL créer un nouveau dossier que l'on appelle TP2. Créer un nouveau projet console et introduire le code ci-dessous, puis le compiler.

1) Considérons un langage défini sur l'alphabet A= {0,1}.

a) Exécuter le programme ci-contre en introduisant comme input chacun des couples de mots suivants :

01000001 ; 100
01001000 ; 101
10011101 ; 111
10011101 ; 00
10011101 ; 1111
10011101 ; 010

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main ()
{
  char mot[100];
  char part [100];
  printf ("Entrer le premier mot: ");
  gets (mot);
  printf ("Entrer le second mot: ");
  gets (part);
if (strstr(mot,part)== NULL)
     printf (" NO \n");
else
    printf (" YES \n");
 system("PAUSE");
}
```

b) Que fait ce programme ? (Modifier alors les instructions printf affichant le résultat).

c) Rajouter la portion de code qui teste que les mots sont définis sur l'alphabet A. (voir TP1).

- Modifier ce code afin qu'il teste si un mot est un préfixe d'un autre mot.
- Modifier ce code afin qu'il teste si un mot est un suffixe d'un autre mot.
- Exécuter ces programmes à l'aide d'exemples appropriés.

d) Taper, compiler puis exécuter le programme ci-contre en introduisant comme input chacun des mots suivants :

1100 ; 10011101 ; 1011000111 ;

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main ()
{
  char mot[100];
  char part[100];
  printf ("Entrer un mot: ");
  gets (mot);
for (int k=1 ;k < strlen(mot); k++)
 {
memset (part, 0, sizeof (mot));
     for (int i=0 ; i< k; i++)
         part[i]= mot[i];
       printf ( "\n ");
     printf ( "%s" , part);
   }
   printf ( "\n ");
   system("PAUSE");
}
```

- Que fait ce programme ? (Rajouter alors les instructions printf affichant le résultat).
- Rajouter la portion de code qui teste que les mots sont définis sur l'alphabet A. (voir TP1).
- Modifier ce code afin qu'il fournisse les suffixes propres d'un mot.
- Exécuter ce programme à l'aide d'exemples appropriés.

e) En utilisant l'opérateur de concaténation de mots, écrire un programme C qui conjugue un verbe du premier groupe (*infinitif er*) à l'imparfait de l'indicatif.
Exemple : ce programme devrait lire un mot comme *regarder* et fournisse en sortie :

Je regardais
Tu regardais
Il, elle regardait
Nous regardions
Vous regardiez
Ils, elles regardaient

Tester ce programme avec d'autres verbes.
Penser à tous les temps, tous les modes et tous les groupes !

CHAPITRE III

GRAMMAIRES ALGEBRIQUES

Plan

1. Dérivation gauche et dérivation droite
2. Arbre de dérivation
3. Grammaire ambigüe
4. Grammaire réduite
5. Grammaire propre
6. Récursivité gauche
7. Factorisation gauche
8. Forme normale de Chomsky
9. Forme normale de Greibach
10. BNF
11. Travaux pratiques

1. Dérivation gauche et dérivation droite

La dérivation gauche d'un mot u par G est une dérivation de u dans laquelle à chaque production, le non-terminal le plus à gauche est substitué.
La dérivation droite d'un mot u par G est une dérivation de u dans laquelle à chaque production, le non-terminal le plus à droite est substitué.

✍ **Exemple**

G = ({a , b, + , × , }, {E} , E , {E→ E+E | E×E | a | b }) .
Donner la dérivation gauche et droite du mot a+b×a par G.
On a : $E \overset{1}{\Rightarrow} E + E \overset{3}{\Rightarrow} a + E \overset{2}{\Rightarrow} a + E \times E \overset{4}{\Rightarrow} a + b \times E \overset{3}{\Rightarrow} a + b \times a$;
La dérivation gauche du mot a+b×c est alors 1,3,2,4,3 .
On a : $E \overset{1}{\Rightarrow} E + E \overset{2}{\Rightarrow} E + E \times E \overset{3}{\Rightarrow} E + E \times a \overset{4}{\Rightarrow} E + b \times a \overset{3}{\Rightarrow} a + b \times a$;
La dérivation gauche du mot a+b×c est alors 1,2,3,4,3 .

2. Arbre de dérivation d'un mot par une grammaire algébrique

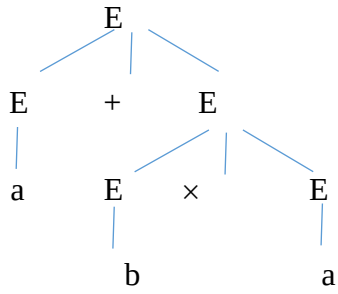
L'arbre de dérivation d'un mot u est une représentation graphique de de la dérivation de u sous forme d'un arbre où :

- La racine de l'arbre est l'axiome S ;
- Les nœuds internes de l'arbre sont les non-terminaux utilisés dans la dérivation de u ;
- Les feuilles de l'arbre sont les terminaux de u.

On distingue alors l'arbre de dérivation gauche et arbre de dérivation droite d'un mot par G.

✍ **Exemple**

Arbre de dérivation du mot a+b×a de l'exemple précèdent :



Dans l'exemple ci-dessus, les arbres de dérivation gauche et droite du mot a+b×a sont identiques.

3. Grammaire ambiguë

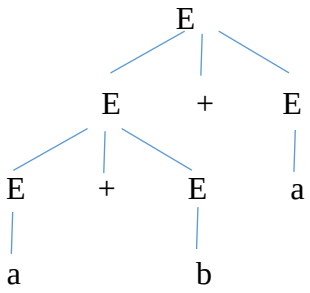
Une grammaire algébrique est dite ambiguë s'il existe au moins un mot possédant plus d'une dérivation gauche par cette grammaire.

✍ **Exemple**

G = ({a , b, + , × , }, {E} , E , {E→ E+E | a | b }) .
Donner deux dérivations gauches du mot a+b+a par G.
Que peut-on déduire ?

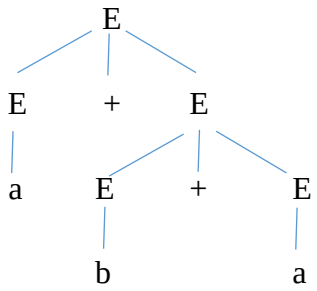
✍ **Solution**

On a : $E \overset{1}{\Rightarrow} E + E \overset{1}{\Rightarrow} E + E + E \overset{2}{\Rightarrow} a + E + E \overset{3}{\Rightarrow} a + b + E \overset{2}{\Rightarrow} a + b + a$; 1 ,1 ,2, 3, 2



Premier arbre de dérivation gauche du mot a+b+a.

Et on a : $E \overset{1}{\Rightarrow} E + E \overset{2}{\Rightarrow} a + E \overset{1}{\Rightarrow} a + E + E \overset{3}{\Rightarrow} a + b + E \overset{2}{\Rightarrow} a + b + a$; 1 ,2 ,1, 3, 2



Second arbre de dérivation gauche du mot a+b+a.

Donc le mot a+b+a admet deux dérivations gauches par G, par conséquent, G est ambigüe.

4. Symbole inutile, symbole utile, grammaire réduite

Soit G = (T,N,S,R) une grammaire algébrique.
Un symbole improductif de G est un non-terminal, qui apparait à droite d'au moins une règle et n'apparait à gauche d'aucune règle.
Un symbole inaccessible ou inatteignable de G est un non-terminal, autre que S, qui apparait à gauche d'au moins une règle et n'apparait à droite d'aucune règle.
Un symbole inutile est un non terminal inaccessible ou improductif.
La règle contenant un symbole intitule est également dite inutile.
Un symbole utile est un non terminal qui figure dans au moins une dérivation d'un mot de L(G).

✍ **Exemple**

G = ({a , b}, {S,X,Y} , S , {S→ abS | aX |b , Y→ abS | a }) .
X est un symbole improductif, Y est un symbole inaccessible, X et Y sont inutiles.
S est un symbole utile, en effet, il figure au moins dans la dérivation du mot abb.

Une grammaire réduite est une grammaire sans symboles inutiles.

Théorème

Pour toute grammaire algébrique, il existe une grammaire réduite équivalente.

Idée de l'algorithme

Il suffit de supprimer les symboles et règles inutiles.

✍ **Exemple**

La grammaire réduite équivalente à G = ({a, b},{S,X,Y}, S , {S→ abS | aX |b, Y→ abS|a })
est G' = ({a, b},{S}, S , {S→ abS | b}).

5. règle vide, règle unitaire, grammaire propre

Soit G = (T,N,S,R) une grammaire algébrique.
Une règle vide ou ε-règle est une règle de la forme A→ε où A∈N et A≠S.
Une règle unitaire est une règle de la forme A→B où A,B∈N .
Une grammaire est dite propre si elle ne contient ni règles vides ni règles unitaires.

✍ **Exemple**

La grammaire G = ({a , b}, {S,X} , S , {S→ abS | aX |ε , X→ bX | a }) est propre.

Théorème

Pour toute grammaire algébrique, il existe une grammaire propre équivalente.

Idée de l'algorithme

Il suffit de supprimer les règles vides et les règles unitaires en substituant le non terminal de gauche par le membre droit.

✍ **Exemple**

La grammaire propre équivalente à :
G = ({a, b},{S,X,Y}, S , {S→ abSY | Xa |Ybb , X→ b, Y→ε })
est G' = ({a, b},{S}, S , {S→ abS | ba | bb}).

6. Récursivité gauche

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

Une règle est dite récursive gauche (ou à gauche) si elle est de la forme $A \rightarrow A\alpha$ où $\alpha \in (T \cup N)^*$.
Le symbole A est dit également dans ce cas récursif gauche.

☞ **Remarque**

Dans certaines situations, la récursivité gauche pose de sérieux problèmes, c'est la raison pour laquelle on doit l'éliminer de la grammaire.

Algorithme d'Elimination la récursivité gauche :

Remplacer les règles : $A \rightarrow A\alpha$ et $A \rightarrow \beta$ par les règles : $A \rightarrow \beta B$ et $B \rightarrow \alpha B | \varepsilon$ et rajouter le non terminal B à N.

✍ **Exemple**

Eliminer la récursivité gauche de la grammaire :
G = ({a, b,c,d},{S,A}, S , {S→ Aa | b , A→Sd | Ac | ε })
On fixe un ordre sur les variables : S < A.
Pour S, la première variable : rien à faire, S n'a pas de récursivité gauche immédiate.
Pour A, la deuxième variable, on doit éliminer la production A → Sd : comme S → Aa | b, on la remplace par A → Aad | bd on obtient : S→Aa | b et A → Aad | Ac | bd | ε.
On élimine les récursivités gauches immédiates de A ce qui donne : S→ Aa | b , A→bdB | B , B→adB | cB | ε

7. Factorisation gauche

La factorisation gauche consiste à éliminer les productions ayant le même membre gauche et dont les membres droits commencent avec le même préfixe.

Comme par exemple :
Stmt → if Cond then Stmt | if Cond then Stmt else Stmt
Dans certaines situations, ce type de règles pose de sérieux problèmes, c'est la raison pour laquelle on doit factoriser ce type de règles.

Algorithme de factorisation gauche :

Remplacer les règles : $A \rightarrow \alpha u$ et $A \rightarrow \alpha v$ par les règles : $A \rightarrow \alpha X$ et $X \rightarrow u|v$ et rajouter le non terminal X à N.

✍ **Exemple**

Factoriser à gauche la grammaire :
G = ({a, b,c,d,e},{S}, S , {S→ bcdS | bcdSeS | a })

✍ **Solution**

La grammaire G n'est pas factorisée à gauche : deux dérivations de la variable S commencent par la même chaine bAdS, On la remplace par une grammaire équivalente factorisée à gauche :
G = ({a, b,c,d,e},{S,X}, S , {S→ bcdSX | a , X→ε | eS })

8. Forme Normale de Chomsky (FNC)

Une grammaire algébrique G = (T,N,S,R) est sous FNC si toutes ses règles sont sous la forme A→BC ou A→a tels que : a∈T ; A,B∈N.

☞ **Remarque**

La règle S→ε est exceptée de cette définition.

Théorème

Pour toute grammaire algébrique, il existe une grammaire sous FNC équivalente.

Idée de l'algorithme

Remplacer toute règle de la forme A→Bα, tels que B∈N et α ∈ $(T\cup N)^+$, par les deux règles : A→BC et C→α ; rajouter C à N .
Remplacer toute règle de la forme A→aα, tels que a∈T et α ∈ $(T\cup N)^+$, par les trois règles : A→CD et C→a , D→α ; rajouter C et D à N .

9. Forme Normale de Greibach (FNG)

Une grammaire algébrique G = (T,N,S,R) est sous FNG si toutes ses règles sont sous la forme A→ aα tels que : a∈T ; α∈N*.

☞ **Remarque**

La règle S→ε est exceptée de cette définition.

Théorème

Pour toute grammaire algébrique, il existe une grammaire sous FNG équivalente.

Idée de l'algorithme

Remplacer toute règle de la forme A→Cα, tels que C∈N et α ∈ $(T\cup N)^+$, C→β par la règle: A→βα.
Remplacer toute règle de la forme A→axbα, tels que a,b∈T, x∈N* , α∈ $(T\cup N)^+$, par les deux règles : A→axBα et B→b ; rajouter B à N .

10. Forme de Backus-Naur (BNF, de l'anglais Backus-Naur Form)

BNF est une notation permettant de décrire les règles syntaxiques des langages de Programmation qui sont souvent algébriques sous la forme :
Au lieu de la flèche on utilise ::= ;

Les non-terminaux sont mis entre < et > ;
Le choix est dénoté par | ;
L'élément optionnel est mis entre crochets [et] ;
L'élément itératif est mis entre accolades { et } ;
Les caractères spéciaux du langage décrit sont mis entre guillemets " et " afin de distinguer des symboles de BNF.

✍ **Exemples**

<chiffre> ::= 0 |1 |2 |3 |4 |5 |6 |7 |8 |9|
<chiffre non nul> ::= 1 |2 |3 |4 |5 |6 |7 |8 |9|
<entier> ::= 0 |[+|-] <chiffre non nul> { <chiffre>}
<affectation> ::= <variable> "=" <expression> ";"
<if> ::= if "(" <cond> ")" (<inst> ";" | "{" { <inst> ";" } "}")

Travaux pratiques III

Dans ce TP, on se propose d'implémenter une grammaire génératrice de type 2. C.-à-d., étant donnée une grammaire hors-contexte G ; on demande d'écrire un programme C qui imprime un mot arbitraire généré par G. Bien évidemment, G pourrait être alors de type 3.

Principe de l'algorithme de génération :

- **Chaque terminal correspond à une constante de type caractère comme a,b,+,=,(,$,… ou de type chaine de caractères comme x1,a23, while, if, table, université,… (selon la grammaire donnée) ;**
- **Chaque non-terminal correspond à une procédure dont le nom est celui du non-terminal ;**
- **Chaque règle A→aB consiste à appeler la procédure A() et dans le corps de A(), imprimer 'a' puis appeler la procédure B() et ainsi de suite;**
- **Le programme principal doit appeler la procédure correspondant à l'axiome par exemple S() ;**
- **Si on trouve plusieurs règles ayant le même membre gauche, on en choisit un seul membre droit au hasard ;**
- **Si on trouve une règle récursive, on l'utilise un nombre de fois aléatoire.**

1. **Taper, compiler puis exécuter plusieurs fois le programme suivant. (noter bien les résultats).**

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
using namespace std;
/* Notre grammaire: ({a,b,c};{S,A,B}; S ; { S → cA ; A →baA|Ba ; B → c|bc|cc  }) */
// LES NON TERMINAUX
void S();
void A();
void B();
 //  REGLE S → cA
void S() {
   printf("c");
   A(); } ;
// REGLE A → baA |Ba
void A() {
    int i=(int)(rand()*2/RAND_MAX);
    if (i==0)
      {    printf("ba");
           A();
      }
    else
      {    B();
           printf("a");
      }
 }
// B → c | bc | cc
void B() {
    int j=(int)(rand()*3/RAND_MAX);
    switch (j)
   {
   case 0:
      printf("c") ;
      break ;
   case 1:
      printf("bc") ;
      break ;
   case 2:
      printf("cc") ;
      break ;
   }
  } ;
// PROGRAMME PRINCIPAL
int main(int argc, char *argv[])
{
   printf ("%s\n","LE MOT GENERE PAR LA GRAMMAIRE G EST : ");
   srand(time(NULL));  // initialiser le générateur de nombres aléatoires
   S();
   printf ( "\n ");
   system("PAUSE"); }
```

2. **En déduire le langage L(G) engendré par la grammaire G.**
3. **Confirmer votre réponse à l'aide de la méthode vue au cours.**
4. **Rajouter à ce code les instructions permettant d'afficher le mot généré et sa dérivation (numéroter d'abord les règles de production)**
5. **Modifier le programme précèdent afin qu'il corresponde à la grammaire G' suivante :**
   S→AB|DaD ; A→Aa|aB ; B→bB|aA ; C→AB|aS ; ; D→dD|ε
   Trouver L(G')
6. **Etant donnée une grammaire Gfr d'un sous-ensemble du français.**

   Gfr = (T,N, <phrase>,R) tel que :

   T ={le, un, chat, chien, aime, poursuit, malicieusement, joyeusement, gentil, noir, blanc, beau}

   N =

   {<phrase>, <GN>, <GV>, <Art>, <Nom>, <Adj>, <Adv>, <verbe> }

   Axiome : <phrase>

   Règles R:

   1 :< phrase >⟶< GN >< GV >< GN >.

   2 :< GN >⟶<Art >< Adj >< Nom >

   3 :< GN >⟶< Art >< Nom >< Adj >

   4 :< GV >⟶< verbe > | < verbe >< Adv >

   5 :< Art >⟶ le | un

   6 :< Nom >⟶ chien | chat

   7 :< verbe >⟶ aime | poursuit

   8 :< Adj >⟶ blanc | noir | gentil | beau

   9 :< Adv >⟶ malicieusement | joyeusement

   En utilisant le programme ci-dessus, écrire un programme C qui génère 10 phrases par la grammaire Gfr

   Mini-projet : Comment peut-on générer de petits scripts java aléatoires ?

CHAPITRE IV

LANGAGES REGULIERS

Plan

1. Langages réguliers
    a. Définition
    b. Propriétés
2. Expressions régulières
    a. Définition
    b. Propriétés
3. Lemme d'Arden
4. Théorème du gonflement
5. Expression régulière associée à une grammaire régulière

---

1. Langages réguliers

1.1. Définition

Un langage régulier (ou rationnel) est défini récursivement comme suit :

- ✓ ∅ et {ε} sont des langages réguliers ;
- ✓ Si a est une lettre alors {a} est un langage régulier ;
- ✓ Si L1 et L2 sont deux langages réguliers, alors L1+L2 est un langage régulier ;
- ✓ Si L1 et L2 sont deux langages réguliers, alors L1.L2 est un langage régulier ;
- ✓ Si L est un langage régulier alors L* est un langage régulier.

Autrement dit :

Un langage régulier est tout langage obtenu par combinaison des 3 langages de base ∅ , {ε} , {a} avec les 3 operateurs rationnels + , . , * ; d'où le nom de *langages rationnels*.

1.2. Propriétés des langages réguliers

- La définition ci-dessus implique explicitement que l'ensemble des langages réguliers est stable par chacun des opérateurs + , . , *.
- L'ensemble des langages réguliers est également stable par chacun des opérateurs d'intersection, de complémentation et de miroir.
- Si L est régulier alors l'ensemble des préfixes des mots de L est régulier.
- Si L est régulier alors l'ensemble des suffixes des mots de L est régulier.
- Si L est régulier alors l'ensemble des facteurs des mots de L est régulier.
- Tout langage fini est régulier.

✍ **Exemples**

Les langages suivants sont réguliers :

- Langage des mots-clés du C ;
- Lainage des entiers signés du C ;
- Langage de mots sur l'alphabet {a,b,c} commençant par a ;

Les langages suivants ne sont pas réguliers :

- Langage des expressions bien parenthésées.
- Langage de mots sur l'alphabet {a,b,c} comportant autant de a que de b
- Langage de mots palindromes sur l'alphabet {a,b,c}.

2. Expressions régulières

Par abus, la définition des langages réguliers est restreinte aux mots de ceux-ci, on définit ainsi la notion d'expression régulière.

2.1. Définition

Une expression régulière est alors définie récursivement comme suit :

- ∅ et ε sont des expressions régulières ;
- Si a est une lettre alors a est une expression régulière ;

- Si u et v sont deux expressions régulières, alors u+v est une expression régulière ;
- Si u et v sont deux expressions régulières, alors u.v est une expression régulière ;
- Si u est une expression régulière alors u* est une expression régulière.

Autrement dit :
Une expression régulière est toute expression obtenue par combinaison des 3 expressions de base ∅ , ε , a avec les 3 operateurs rationnels + , . , * ;

✍ **Exemples**
Les expressions suivantes sont régulières :
- a(a2+b)*
- (a3b)+
- anbm , n,m∈N.

Les expressions suivantes ne sont pas régulières :
- $\bar{a}$(a+b)*
- anbn , n∈N.
- anbm , n,m∈N et n<m.

☞ **Remarque**
- Tout langage régulier est décrit (ou dénoté) par une expression régulière.
- Au fait, les expressions régulières représentent la restriction de la définition des langages réguliers aux mots.

✍ **Exemples**
a+bc dénote le langage {a,bc} ;
(ab)* dénote le langage {ε,ab, abab, ababab,…} ;
ab+c* dénote le langage {ε,ab, c, cc,ccc, …} ;

2.2. Propriétés des expressions régulières

En utilisant la définition récursive des expressions régulières et les propriétés ensemblistes, on peut aisément prouver la validité des propriétés suivantes :
Pour toutes expressions régulières x,y,z on a :
- $x + y = y + x$;
- $(x + y) + z = x + (y + z)$;
- $(x.y).z = x.(y.z)$;
- $x.\emptyset = \emptyset.x = \emptyset$;
- $x.\varepsilon = \varepsilon.x = x$;
- $x + \emptyset = \emptyset + x = x$;
- $x^*.x = x.x^*$;
- $(x + \varepsilon)^*.y = x^*.y$;
- $x^*.x^* = x^* = x^+$;

- $x^* = \varepsilon + x.x^*$;
- $\emptyset^* = \varepsilon$;
- $(x^*)^* = x^*$;
- $(x^* + y^*)^* = (x + y)^* = (x^*.y^*)^*$.

3. Lemme d'Arden

Si A et B sont deux langages réguliers alors le langage A*B est solution de l'équation d'inconnue L : L=AL+B et si A ne contient pas ε alors cette solution est unique.

Démonstration

Vérifions que A*B est solution de l'équation en remplaçant L par A*B dans l'équation L=AL+B.

Le premier membre : L= A*B.

Le second membre : AL+B= A(A*B)+B= A^+B+B=(A^++ε)B= A*B.

Il est donc clair que A*B est solution de cette équation.

Si ε∈A alors :

L=AL+B ⇔ L= (ε+A') L+B (en posant A= ε+A')

L=AL+B ⇔ L=L+ (A'L+B) (A'L+B est un langage incluant B)

L=AL+B ⇔ L=B+X (X langage quelconque).

Tout langage incluant B est solution de l'équation (l'équation admet une infinité de solutions).

Résolvons maintenant l'équation quand ε∉A:

L=AL+B ⇔ L=A(AL+B)+B

L=AL+B ⇔ L=A^2L+AB+B

L=AL+B ⇔ L=A^2(AL+B)+AB+B

L=AL+B ⇔ L= A^3L+ A^2L+AB+B

L=AL+B ⇔ L= …+A^nB+ A^{n-1}B+…+AB+B

L=AL+B ⇔ L= (…+A^n+ A^{n-1}+…+A+ε)B

L=AL+B ⇔ L= A*B

☞ **Remarque**

Par analogie, on peut vérifier aisément que Pour l'équation L=LA+B, la solution devient BA*.

✍ **Exemples**

Résoudre chacune des solutions suivantes :

L=abL+c

L=bL

L=bL +ε

L=(a+b)L+bb

L=L+ac

L=La+c

Résoudre le système d'équations suivant d'inconnues L1, L2, L3 :

$$\begin{cases} L1 = aaL2 + baL3 + \varepsilon \\ L2 = aL2 + bL3 + ba \\ L3 = bL3 + a \end{cases}$$

Solutions

Résolution des équations :

L=abL+c ⇔ L=(ab)*c

L=bL ⇔ L=b*∅=∅

L=bL +ε ⇔ L=b*ε=b*

L=(a+b)L+bb ⇔ L=(a+b)*bb

L=L+a*c ⇔ L=ε*a*c=a*c

L=La+c ⇔ L=ca*

Résolution du système d'équations :

$$\begin{cases} L1 = aaL2 + baL3 + \varepsilon & (1) \\ L2 = aL2 + bL3 + ba & (2) \\ L3 = bL3 + a & (3) \end{cases}$$

(3) ⇔ L3=b*a

(2) ⇔ L2=aL2+bb*a+a

(2) ⇔ L2=a*(b$^+$a+a)

(2) ⇔ L2=a*b$^+$a + a$^+$

(1) ⇔ L1=aaa*b$^+$a + aaa$^+$ + bab*a + ε

(1) ⇔ L1=a^{n}b^{m}a + a^k + bab*a + ε ; n≥2, m≥1 ; k≥3.

4. Théorème du gonflement ou de pompage (ou lemme de l'étoile).

Si L est un langage régulier alors il existe un entier n tel que tout mot u de L tel que |u|≥n peut être factorisé sous la forme xyz avec 1≤|y|≤n et ∀i∈N : xy^{i}z∈L.

Autrement dit :

Tout mot suffisamment long d'un langage régulier peut être "pompé", au sens qu'une partie centrale du mot peut être répétée un nombre quelconque de fois, et que chacun des mots ainsi produits est encore dans le langage.

✍ **Exemples**

L ={ε+b+ab*c}={ε, b, ac, abc, abbc, abbbc, abbbbc,…}

Il est clair que L est régulier puisqu'il est dénoté par une expression régulière.

Tous les mots de L de longueur supérieure ou égale à 3 (n existe bien et est égale à 3) peuvent être mis sous la forme xyz avec $1 \leq |y| \leq n$ et les mots obtenus par itération de y restent dans L.
Pour le mot abc, on peut prendre y=b et alors $\forall i \in N : ab^ic \in L$;
Pour le mot abbc, on peut prendre y=b ou y=bb et alors $\forall i \in N : a(bb)^ic \in L$;
Pour le mot abbbc, on peut prendre y=b, y=bb ou y=bbb et alors $\forall i \in N : a(bbb)^ic \in L$; et ainsi de suite, les mots de L peuvent être "pompés" tout en restant dans L.

☞ **Remarque**

Le lemme de l'étoile est couramment utilisé pour montrer qu'un langage donné n'est pas rationnel (en raisonnant par l'absurde ou preuve par contraposée). En revanche, il ne peut être employé pour démontrer qu'un langage est rationnel. En effet, il énonce une condition, nécessaire certes, mais non suffisante, de rationalité.
Ainsi, de l'implication :
L est régulier $\Rightarrow \exists n \in \aleph : \forall u \in L$ et $|u| \geq n : \exists x,y,z : u=xyz$ et $1 \leq |y| \leq n$ et $\forall i \in N : xy^iz \in L$.
On en déduit par contraposition l'implication suivante :
S'il n'existe aucun n vérifiant ces conditions $\Rightarrow$ L n'est pas régulier.

✍ **Exemples**

L3= $\{0^m1^m / m \in \aleph\}$, démontrons par l'absurde que L3 n'est pas régulier.
Supposons que L3 est régulier, il existe alors, selon le lemme du pompage, un entier n tel que tout mot u de L3 de longueur $\geq$ n, s'écrit u= xyz $(1 \leq |y| \leq n)$.
Si y ne contient que des 0 ($y=0^k$), alors xy^iz contiendrait plus de 0 que de 1 donc $xy^jz \notin L3$;
Si y ne contient que des 1 ($y=1^k$), alors xy^iz contiendrait plus de 1 que de 0 donc $xy^iz \notin L3$;
Si y contient des 0 et des 1 ($y=0^k1^l$),alors xy^iz ne serait pas de la forme 0^m1^m et donc $xy^iz \notin L3$;
Ainsi il n'existe aucun y vérifiant ce lemme, et par conséquent il n'existe aucun entier n associé, et ceci contredit l'hypothèse. Ainsi selon le lemme du pompage, L3 n'est pas régulier.

Série d'exercices IV

Exercice 1

Déterminer tous les mots de longueur maximale 4 qui appartiennent au langage dénoté par chacune des expressions régulières suivantes :

(i) $(b+ba)^*$ (ii) ab^*+b (iii) $(a+b)^*abb$

(iv) $(x+\varepsilon)^*dd^*$ (v) $(xd+\varepsilon)^*d^*$ (vi) $a^*(b+c)d^*$

Exercice 2

Donner une description en français des langages sur l'alphabet $\{a, b\}$ décrits par les expressions régulières suivantes :

(i) $(a+b)^*$ (ii) $a(a+b)^*$ (iii) $(a+b)^*a$ (iv) $(b+ab)^*(a+\varepsilon)$

(v) a^*+b^* (vi) $(aa+b)^*$ (vii) $(ab^*a+b)^*$

Exercice 3

Décrire sous la forme d'une expression régulière les langages suivants

1. le langage des mots de longueur 2.
2. le langage des mots de longueur au plus 2.
4. le langage des mots ayant au moins une occurrence de a.
5. le langage des mots ayant au moins une occurrence du facteur ab.
6. le langage des mots ayant au moins une occurrence de a puis ensuite une occurrence d'un b.
7. le langage des mots commençant par a, finissant par b et n'ayant pas deux a ou deux b consécutifs.

Exercice 4

Pour chacun des langages suivants, donner une expression régulière représentant son complément:

(i) $(a+b)^*b$; (ii) $((a+b)(a+b))^*$.

Exercice 5

En vous servant des équivalences suivantes vues en cours :

$\alpha+(\beta+\gamma)\equiv(\alpha+\beta)+\gamma$

$\alpha+\beta\equiv\beta+\alpha$

$\alpha(\beta+\gamma)\equiv\alpha\beta+\alpha\gamma$

$(\alpha+\beta)\gamma\equiv\alpha\gamma+\beta\gamma$

$\alpha+\emptyset\equiv\alpha$

$\alpha+\alpha\equiv\alpha$

$\emptyset\alpha\equiv\alpha\emptyset\equiv\emptyset$

$\alpha(\beta\gamma)\equiv(\alpha\beta)\gamma$

$\varepsilon\alpha\equiv\alpha\varepsilon\equiv\alpha$

Prouver les équivalences suivantes :

$\varepsilon+\alpha\alpha^*\equiv\alpha^*$

$\alpha^*(\beta\alpha^*)^*\equiv(\alpha+\beta)^*$

$(\varepsilon + \alpha)^* \equiv \alpha^*$

$(\alpha\beta)^*\alpha \equiv \alpha(\beta\alpha)^*$

$(\alpha^*\beta)^*\alpha^* \equiv (\alpha + \beta)^*$

$\varepsilon + \alpha^*\alpha \equiv \alpha^*$

$\emptyset^* \equiv \varepsilon$

$\alpha\alpha^* \equiv \alpha^*\alpha$

$(a + b) + (a + b)(a + b)^* + \varepsilon \equiv (a + b)^*$

$a(cc^* + \varepsilon) + (a + b)(c^* + (c^*)^*) \equiv (a + b)c^*$

$(x + y)\emptyset + (x + y)\varepsilon + ((x + y)^*\varepsilon)^* \equiv (x + y)^*$

$0(\varepsilon + 00)^*(1 + 01) + 1 \equiv 0^*1$

Exercice 6

Donnez une définition récursive à la syntaxe des expressions arithmétiques qui portent sur des entiers et qui utilisent les opérateurs +, −, /, *, et les parenthèses ().

Exercice 7

Donnez une expression régulière qui décrit les mots possibles pour une horloge numérique sur 24 heures où les heures, les minutes et les secondes sont séparées par « : ».

On admet qu'un système d'exploitation λ impose que le nom des fichiers réponde aux conditions suivantes :

- ils sont formés sur l'alphabet {a,b,c, . . . ,z,0,1,2, . . . ,9,.} ;

ils sont constitués par une chaîne de 1 à 8 caractères qui sont des lettres ou des chiffres, suivi du caractère « . » suivi de 3 caractères qui sont des lettres ou des chiffres ;

- ils ne peuvent commencer par un chiffre.

Représentez par une expression régulière la syntaxe de ces noms de fichiers.

Exercice 8

Dites si les langages suivants définis sur l'alphabet {0,1} sont réguliers :

L1= lge de mots commençant par 0.

L2= $\{0^n1^m / \ n, m \in N\}$

L3= $\{0^n1^n / \ n \in N\}$

L4= $\{0^n1^{2n} / \ n \in N\}$

L5= lge de mots contenant autant de 0 que de 1.

L6= lge de mots contenant au moins un 0.

L7= langage des mots de longueur paire.

Justifier votre réponse en utilisant le lemme de l'étoile.

Exercice 9

En utilisant le lemme du gonflement, Montrer que les langages suivants définis sur l'alphabet {0,1} sont irréguliers :

L1= {a^n / n premier}

L2= {$a^n b^m$ / n<m }

L2= {$a^n b a^n$ / n∈ N }

Exercice 10

Démontrer que tout langage fini est régulier.

Exercice 11

1) Résoudre chacune des équations suivantes d'inconnue le lge L:

L=0L+1

L= (a+b)L

L= 0L+0+1

L=L +ε

L=aL+L+b+Φ

2) Résoudre le système suivant d'inconnues L1 , L2 , L3 :

$$\begin{cases} L1 = aL2 + bL3 \\ L2 = bL3 + a \\ L3 = aL3 + b + \varepsilon \end{cases}$$

Exercice 12

Imaginez que vous, en tant que brillant hacker, souhaitiez connaître le nom de connexion d'une personne particulière. Vous avez trouvé, sur un système Unix, un répertoire avec plusieurs fichiers journaux que vous pensez contenir les informations que vous recherchez. Vous vous souvenez que vous pouvez utiliser l'utilitaire grep pour rechercher des modèles de texte. Par exemple, vous pouvez entrer la commande : grep -E "abc" * pour rechercher dans tous les fichiers des lignes contenant la séquence de caractères abc. (L'option -E est utilisé pour indiquer que vous utilisez la syntaxe étendue pour les expressions régulières.)

(a) Quelle commande entreriez-vous pour rechercher dans tous les fichiers des lignes commençant par un potentiel identifiant ? (N'oubliez pas qu'un nom de connexion se compose uniquement de lettres et de chiffres, et doit commencer par une lettre.).

(b) Vous réalisez que vous obtenez beaucoup trop de hits avec la commande ci-dessus. Laissez-nous ainsi supposons que le nom de connexion que vous recherchez comporte au moins 5 caractères. Comment voulez-vous affiner votre commande en conséquence ?

(c) Comment pourriez-vous affiner votre commande pour rechercher des potentiels les noms de connexion qui ne sont pas nécessairement au début de la ligne, mais qui sont soit au début ou à la fin de la ligne, soit précédé ou suivi d'un espace personnages ?

CHAPITRE V

AUTOMATES FINIS

Plan

1. Introduction
2. Définition et terminologie
   2.2. Représentation graphique
   2.3. Exécution et configuration d'un AF, langage reconnu par un AF
   2.4. Etat puits, état poubelle
   2.5. AFs équivalents
   2.6. AF complet
3. Operations sur les AFs
   3.1. AF complémentaire
   3.2. Somme de deux AFs
   3.3. Produit de deux AFs
   3.4. Itération d'un AF
4. AFs et expressions régulières
5. AFs et grammaires de type 3
6. Automate minimal
7. Automate fini indéterministe

1. Introduction

Supposons que l'on désire concevoir une machine abstraite dite automate permettant de reconnaitre les mots d'un langage. En d'autres termes, un programme qui prend en entrée un mot de ce langage et fournit en sortie une décision portant sur ce mot : accepté ou rejeté, c-à-d, ce mot fait partie de ce langage ou non respectivement (Fig 5.1).

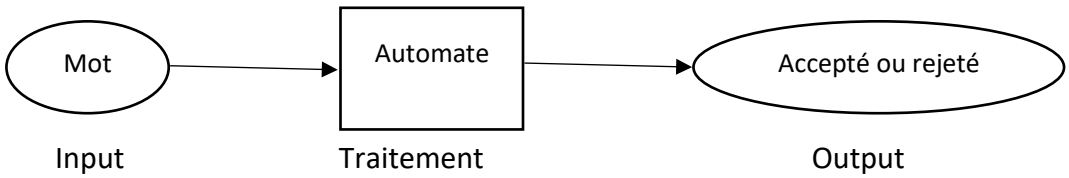


Fig 5.1. Fonctionnement d'un automate.

Ainsi, un automate doit être composé d'au moins trois éléments : une bande d'entrée pour contenir le mot à reconnaitre ; une tête de lecture pour balayer le mot et une unité centrale qui consiste en une mémoire dont le contenu définit l'état de l'automate, ainsi, le changement du contenu implique une transition de l'automate d'un état à un autre en fonction de son état courant et du symbole lu en partant d'un état initial déterminé, une fonction de transition est alors indispensable pour déterminer son nouvel état. Si le mot tout entier est ainsi traité en se retrouvant dans un état final, le mot est accepté, autrement, le mot est rejeté (Fig 5.2). Si sa mémoire est finie, le nombre d'état est forcément fini (par exemple, avec une mémoire de 3 bits, on peut définir un automate d'au plus 8 (2^3) états (selon sa fonction de transition).

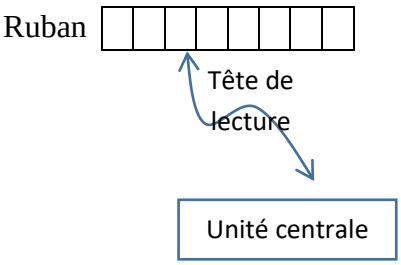


Fig. 5.2. Schéma logique d'un automate.

A tout instant, la configuration de l'automate est déterminée par son état courant et la chaine restante du mot à reconnaitre.

2. Définitions & terminologie

Définition.

Un Automate Fini déterministe (AF) est un quintuple $(Q, \Sigma, \delta, q_0, F)$ tel que :

- Q est un ensemble fini et non vide, dit ensemble des états de l'AF ;
- Σ est un ensemble fini et non vide, dit alphabet d'entrée de l'AF ;
- δ est une fonction de $Q \times \Sigma$ vers Q, dite fonction de transition de l'AF ;

- q_0 est un état de Q, dit état initial de l'AF ;
- F est une partie de Q, dite ensemble des états finaux de l'AF.

☞ **Remarque**

Le qualificatif *fini* désigne ici que le nombre d'états de l'AF est fini.

Cette définition est restreinte à un AF déterministe ; celle de l'AF indéterministe sera donnée plus loin dans ce chapitre.

✍ **Exemple**

Soit un AF A = ((Q, Σ, δ , q_0, F) tel que :

- Q = {1,2,3, 4} ;
- Σ = {a,b} ;
- δ est défini par δ(1,a)= 1 , δ(1,b)= 2 , δ(2,a)= 3 , δ(3,b)= 3 , δ(3,a)= 4 ;

  δ peut être également défini par une table comme suit :

δ	a	b
→1	1	2
2	3	
←3	4	3
←4		

  Les flèches entrantes et sortantes illustrent les états initiaux et finaux respectivement.

  Cette définition tabulaire est plus intuitive et plus appropriée ; elle est faisable puisque les ensembles Q et Σ sont toujours discrets et finis.

  La transition δ(1,b)= 2 veut dire : si l'AF se trouve dans l'état 1 et sa tête de lecture pointe sur le symbole a du mot à reconnaitre, alors l'AF passe à l'état 2 et sa tête de lecture se déplace au symbole immédiatement à gauche de a dans le mot.
- q_0 = 1 (état initial) ;
- F = {2,4} (ensemble des états finaux) ;

2.1. Représentation graphique

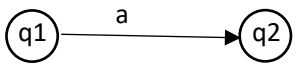
Afin de faciliter la notion d'automate et de comprendre son fonctionnement, une représentation graphique standard est adoptée de la manière suivante :

Un état q est représenté par un cercle :

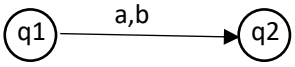
Un état initial q est représenté par un cercle et une flèche entrante :

Un état final q est représenté par un double cercle :

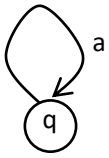
Une transition δ(q1,a)= q2 est représentée par :



Deux transitions de la forme δ(q1,a)= q2 et δ(q1,b)= q2 peuvent être représentées comme suit :

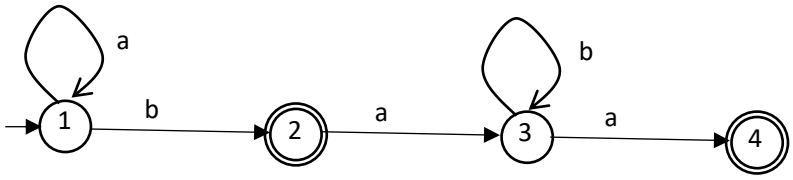


Une transition de la forme δ(q,a) = q est représentée par une boucle :



✍ **Exemple**

L'AF A de l'exemple précédent peut être représenté graphiquement comme suit :



2.2. Exécution et configuration d'un AF, langage reconnu par un AF

Soit un AF A = (Q, Σ, δ , q_0, F).

L'exécution d'un AF pour un mot u consiste à lire les symboles de u de gauche à droite en transitant d'un état à un autre selon sa fonction de transition.

Lors de cette exécution :

- Si l'AF se trouve dans l'état q et la tête de lecture pointe su le premier symbole a de la chaine restante α de u alors le couple (q, α) est appelé configuration (courante) de l'AF.
- Si on a δ(q,a) = q' et α = aα' alors la configuration suivante de l'AF A est (q', α') et on écrit : (q, α) $\vdash\!\!\!-^{a}$ (q', α') et on dit que l'AF a lu le symbole a (c'est la lecture du symbole a par l'AF A).
- Si la configuration (q'', α'') se résulte après plusieurs lectures de symboles successifs de u, on écrit : (q, α) $\vdash\!\!\!-^{*}$ (q'', α'').
- Sa configuration initiale est, bien entendu, (q_0, u) ; si sa configuration finale est (q_f, ε) avec $q_f \in F$, alors le mot u est accepté par l'AF A, c.-à-d., le mot u appartient au langage reconnu par A noté L(A), autrement, le mot u est rejeté par l'AF A ou u ∉ L(A).

Ainsi : le langage reconnu par l'AF A est : $L(A) = \{ u \in \Sigma^* / (q_0, u) \vdash\!\!\!-^{*} (q_f, \varepsilon) \text{ avec } q_f \in F\}$.

☞ **Remarque**

Le mot est rejeté dans deux cas :

- Si la configuration finale est (q_f, ε) avec $q_f \notin F$, c.-à-d., après avoir lu le mot tout entier , l'AF se retrouve dans un état non final ;
- Si l'AF se retrouve dans la configuration courante (q,au') et que δ n'est pas défini pour le couple (q,a) (δ(q,a) n'existe pas, comme le cas de δ(2,b) de l'exemple ci-dessus), on dit dans ce cas que l'AF se bloque en reconnaissant ce mot.

✍ **Exemple**

Exécuter l'AF A de l'exemple précédent pour chacun des mots : aababa ; ababb ; aabb.

Solution

(1, aababa) $\vdash\!\!\!-^{a}$ (1, ababa)
$\vdash\!\!\!-^{a}$ (1, baba)
$\vdash\!\!\!-^{b}$ (2, aba)
$\vdash\!\!\!-^{a}$ (3, ba)
$\vdash\!\!\!-^{b}$ (3, a)
$\vdash\!\!\!-^{a}$ (4, ε)

Le mot aababa est accepté par l'AF A (mot lu et 4 est un état final).

(1, ababb) $\vdash\!\!\!-^{a}$ (1, babb)
$\vdash\!\!\!-^{b}$ (2, abb)
$\vdash\!\!\!-^{a}$ (3, bb)
$\vdash\!\!\!-^{b}$ (3, b)
$\vdash\!\!\!-^{b}$ (3, ε)

Le mot ababb est rejeté par l'AF A (mot lu et 3 n'est pas final).

(1, aabb) $\vdash\!\!\!-^{a}$ (1, abb)
$\vdash\!\!\!-^{a}$ (1, bb)
$\vdash\!\!\!-^{b}$ (2, b)

Le mot aabb est rejeté par l'AF A (blocage en 2).

Langage reconnu par un AF à partir d'un état :

Soit un AF A = (Q, Σ, δ , q_0, F) et q un état de Q.

Le langage reconnu par l'AF A à partir de q est le langage noté $L_q(A)$ ou L_q tout court (s'il s'agit d'un seul AF) défini récursivement comme suit :

si on a δ(q,a) = q' et δ(q,b) = q'' alors $L_q = a.L_{q'} + b.L_{q''}$

et si q∈F alors on y ajoute ε, c.-à-d., $L_q = a.L_{q'} + b.L_{q''} + \varepsilon$.

En généralisant :

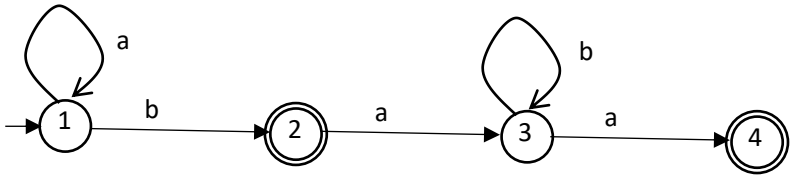
Le langage reconnu par l'AF A à partir de q est le langage noté L_q défini récursivement comme suit :

$L_q = \sum_{a\in\Sigma} a.L_{\delta(q,a)}$ si q ∉ F ;

$L_q = \sum_{a\in\Sigma} a.L_{\delta(q,a)} + \varepsilon$ si $q\in F$.

Résultat :
On peut ainsi déduire que : $L(A) = L_{q_0}$ ce qui permet de calculer le langage reconni par l'automate A.

✍ **Exemple**
Calculer le langage L(A) reconnu par l'AF A =



Solution
D'après la définition ci-dessus, on a :
$L(A) = L_1 = aL_1 + bL_2$
$L_2 = aL_3 + \varepsilon$
$L_3 = bL_3 + aL_4$
$L_4 = \varepsilon$
$\Rightarrow L_3 = bL_3 + a = b^*a$ (en appliquant le théorème d'Arden)
$\Rightarrow L_2 = ab^*a+\varepsilon$ (en remplaçant L_3)
$\Rightarrow L(A) = L_1 = aL_1 + b(ab^*a + \varepsilon)$ (en remplaçant L_2)
$= aL_1 + (bab^*a + b)$
$= a^*(bab^*a + b)$ (en appliquant le théorème d'Arden)

2.3. Etat puits, état poubelle

Définition.
Un état puits est un état q tel que : $\forall a \in \Sigma$: δ n'est pas défini en (q,a), c.-à-d., il n'y a aucune flèche qui part de q, il ne pourrait y avoir que des flèches entrantes.
Un état poubelle est un état puits non final.

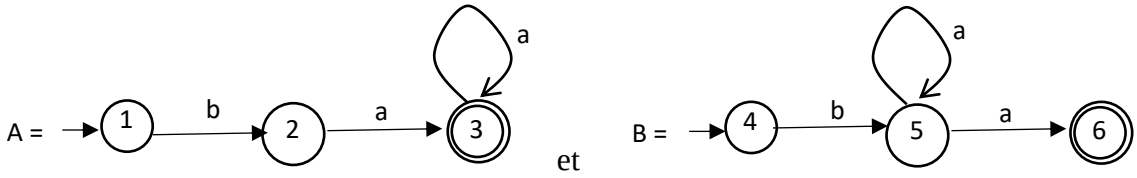
Exemple.
Dans l'exemple ci-dessus, l'état 4 est un état puits.

2.4. Automates équivalents

Définition.
Deux AFs A et A' sont dits équivalents si et seulement s'ils reconnaissent le même langage, soit : $A \equiv A' \Leftrightarrow L(A) = L(A')$

Exemple

Les deux AFs A et B suivants sont équivalents :



En effet, en utilisant la méthode précédente, on aura : L(A)=L(B)= ba^+.

Remarque

Un AF reconnait un seul langage, en revanche, un langage peut être reconnu par plusieurs AFs. Dans l'exemple ci-dessus, A et B deux automates différents reconnaissent le même langage L = ba^+.

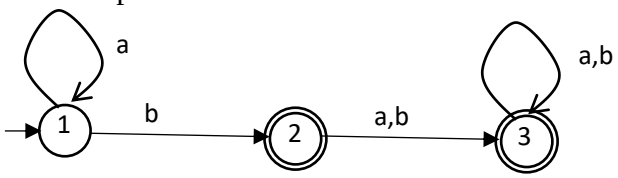
2.5. AF complet

Définition. Un AF A = (Q, Σ, δ , q_0, F) est dit complet si et seulement si : ∀(q,a) ∈ Q×Σ , ∃q'∈Q : δ(q,a)=q', c.-à-d., f est une application (toutes les cases de la table de δ sont remplies. Graphiquement, de chaque état q, et pour chaque symbole a, il part une flèche étiquetée par a).

Exemple

L'automate de l'exemple précédent n'est pas complet.

L'automate ci-dessous est complet :



Théorème

Pour tout AF non complet, il existe un AF complet lui est équivalent.

Preuve.

Il suffit de rajouter un état poubelle.

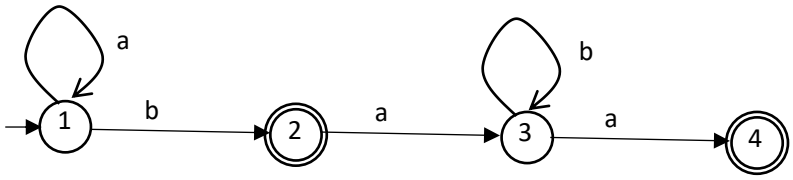
Exemple.

Compléter l'AF A= ({1,2,3,4},{a,b}, δ , 1 , {2,4})

=

δ	a	b
→1	1	2
←2	3	
3	4	3
←4		

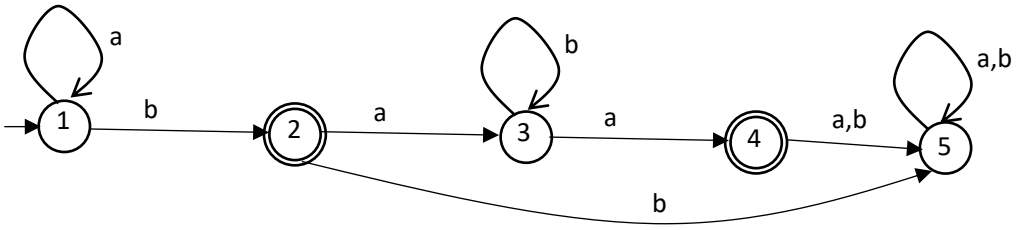
=



Solution

A complété = A' = ({1,2,3,4,5},{a,b}, δ' , 1 , {2,4}) ; (5 est un état poubelle.)

δ'	a	b
→1	1	2
←2	3	5
3	4	3
←4	5	5
5	5	5



Remarque

Le fait de compléter un AF est une tâche très simple mais aussi très importante, ceci évite le cas de blocage de l'AF et facilite ainsi son implémentation. En effet, pour un automate complet, on arrive toujours à lire tout le mot, c.-à-d. à une configuration de la forme (q,ε), ainsi, si q∈F le mot est accepté ; sinon, il est rejeté.

3. Operations sur les AFs

3.1. AF complémentaire

L'AF complémentaire d'un AF A est l'AF noté $\overline{A}$ tel que :

$L(\overline{A}) = \overline{L(A)}$

= Complément de L(A)

= langage de mots n'appartenant pas à L(A)

= langage de mots non reconnus par A.

Remarque

$\overline{A}$ peut être calculé en complétant A puis en y convertissant les états finaux en non finaux et vice versa.

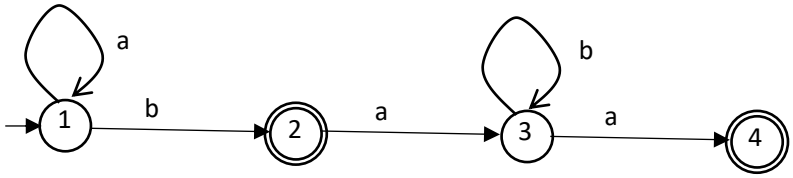
Exemple.

Calculer l'AF complémentaire $\overline{A}$ de l'AF A= ({1,2,3,4},{a,b}, δ , 1 , {2,4})

=

δ	a	b
→1	1	2
2	3	
3	4	3
←4		

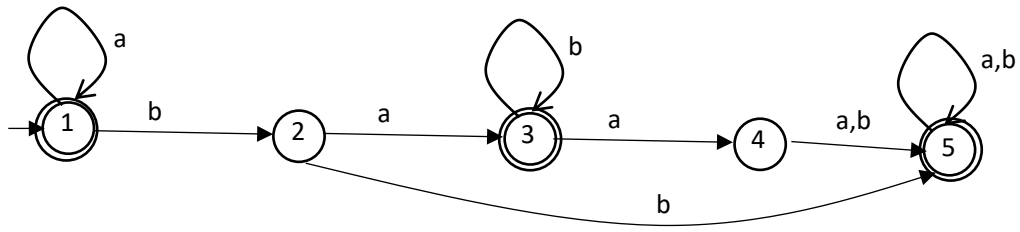
=



Solution

L'AF complémentaire de A = $\overline{A}$ = ({1,2,3,4,5},{a,b}, δ' , 1 , {1,3,5})

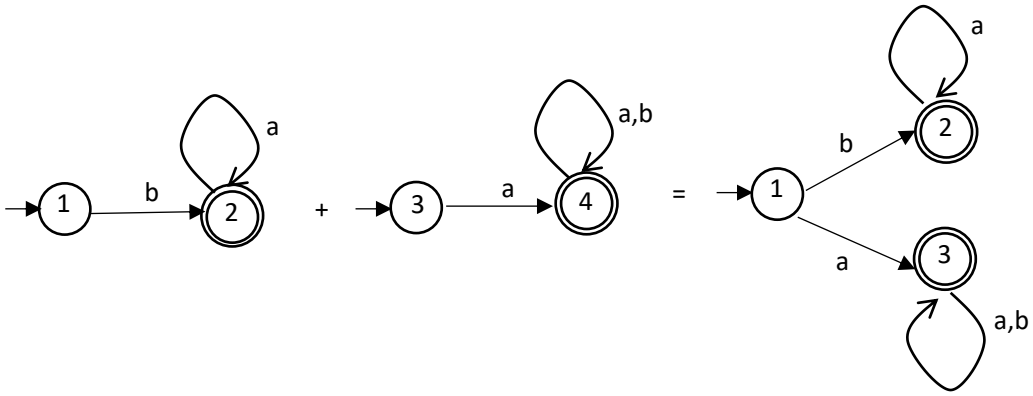
=



3.2. Somme de deux AFs

La somme (l'union) de deux AFs A et B est l'AF noté A+B tel que : L(A+B) = L(A) + L(B).

Exemple



Construction de A+B :

Deux méthodes sont possibles, une méthode directe, sans calculer les langages respectifs des automates A, B et A+B et l'autre, indirecte, en passant par ces langages.

Méthode directe :

Si A = (Q, Σ, δ , q_0, F) et B = (Q', Σ', δ' , q'_0, F') alors A+B = (Q∪Q', Σ∪Σ', δ'' , q''_0, F∪F') tels que : $q''_0 = q_0$ ou $q''_0 = q'_0$; si δ(q,a) = q' ou δ'(q,a) = q' alors δ''(q,a) = q'.

Méthode indirecte :

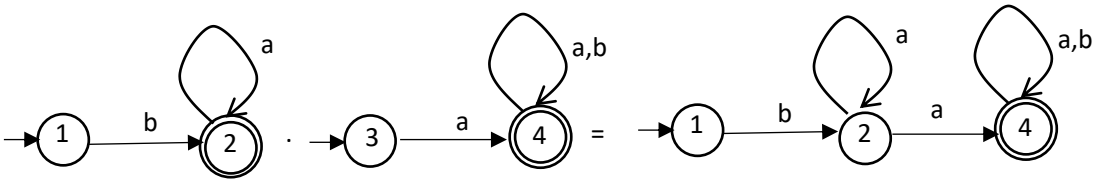
Pour calculer A+B :

- Calculer L(A) à partir de A ;
- Calculer L(B) à partir de B ;
- Calculer L(A) + L(B) à partir de L(A) et L(B) ;
- Calculer A+B à partir de L(A) + L(B).

3.3. Produit de deux AFs

Le produit (la concaténation) de deux AFs A et B est l'AF noté A.B tel que :
L(A.B) = L(A) .L(B).

Exemple



Construction de A.B :

Deux méthodes sont possibles, une méthode directe, sans calculer les langages respectifs des automates A, B et A.B et l'autre, indirecte, en passant par ces langages.

Méthode directe :

Si A = (Q, Σ, δ , q_0, F) et B = (Q', Σ', δ' , q'_0, F') alors A.B = (Q∪Q', Σ∪Σ', δ'' , q_0, F') tels que : si δ(q,a) = q' ou δ'(q,a) = q' alors δ''(q,a) = q'.

Méthode indirecte :

Pour calculer A.B :

- Calculer L(A) à partir de A ;
- Calculer L(B) à partir de B ;
- Calculer L(A) . L(B) à partir de L(A) et L(B) ;
- Calculer A.B à partir de L(A) . L(B).

4. Automate fini indéterministe

L'AF étudié jusqu'ici est dit automate fini déterministe, ce concept peut etre généralisé au concept d'automate fini indéterministe satisfaisant d'autres propriétés en plus.

Définition.

Un Automate Fini indéterministe (AFI) est un quintuple (Q, Σ, Δ , I , F) tel que :

- Q est un ensemble fini et non vide, dit ensemble des états de l'AFI ;
- Σ est un ensemble fini et non vide, dit alphabet d'entrée de l'AF ;
- Δ est une relation de Q×Σ* vers Q, dite relation de transition de l'AFI ;
- I est une partie de Q, dite ensemble des états initiaux de l'AF.
- F est une partie de Q, dite ensemble des états finaux de l'AF.

Remarque

Différences entre AF et AFI :

i. L'AF possède une fonction de transition, alors qu'un AFI possède une relation de transition, c.-à-d., Δ(q,a) est une partie de Q (dans la table on peut avoir plus d'un état, graphiquement, d'un état q, il peut y avoir plus d'une flèche étiquetée par a qui pointent vers des états différents.).

ii. La relation de transition s'étend aux mots y compris ε (on peut avoir par exemple : Δ(q,aba) = q' ou bien Δ(q,ε) = q').

iii. L'AF possède un seul état initial, alors qu'un AFI pourrait en avoir plus.

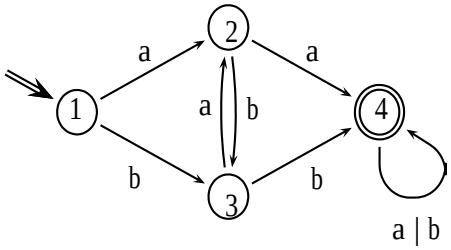
Série d'exercices V

Exercice 1

Construire un automate fini dont le langage soit Σ*aaΣ*, ou` Σ = {a, b}.

Exercice 2

1 - Soit A l'automate fini ci-dessous :



Démontrer que : L(A) = {a, b}* {aa, bb} {a, b}*.

2 - Déterminer un AF dont le langage soit {a, b}* {aa, bb} {a, b}*. L'automate fini sera déterministe complet, bien sûr, mais rien n'interdit de passer d'abord par un automate fini non déterministe, de le rendre déterministe, et enfin de simplifier l'AFdc obtenu.

Exercice 3

Constructions d'automates sur Σ = {a, b, c}. Proposer des automates (AFdc, puis, si c'est plus simple, AFd) reconnaissant chacun des langages suivants :

Σ∗ , {ε}, Σ, Σ2, langage des mots de longueur au plus 2, langage des mots de longueur 2 ou plus, ∅, langage

des mots commençant par a, langage des mots sans voyelle, langage des mots ne contenant pas deux lettres différentes, langage des mots ne contenant pas deux fois la même lettre.

Pour chacun de ces langages, proposer une expression régulière.

Exercice 4

Soit Σ = {a, b, c}, on désire construire un AF, le plus simple possible, qui reconnaisse le langage des mots sur

Σ qui contiennent le mot bac et se terminent par a.

1 - Donner une expression régulière de ce langage.

2 - Dessiner un AF, le plus simple possible, qui reconnaisse ce langage.

3 - Indiquer comment votre AF lit chacun des mots suivants (plus pr´ecis´ement, on demande d'indiquer toutes

les lectures de chaque mot) : abaca, abacb, abacbaca, bcaa, bcab, ε.

4 - Donner la définition mathématique compète de votre AF.

5 - Transformer votre AF en un AFD qui reconnaît le même langage.

6 - Indiquer comment votre AFD lit chacun des mots propos´es `a la question 3.

Exercice 5

Soit Σ = {a, b} un alphabet et soient les langages suivants : L1={a^n : n ≥ 0}, L2={b^n : n ≥ 0} , L3=L1L2.

1 - D´déterminer un automate fini dont le langage soit L1.

2 - Le langage L3 est-il le langage d'un automate fini ? Si la réponse est oui, construire un

tel automate.

3 - Construire une grammaire dont le langage soit L3.

Exercice 6

Reprendre les questions de l'exercice 11 pour le langage sur $\Sigma = \{a, b, c\}$ des mots qui contiennent le mot bac ou se terminent par a.

Exercice 7

Soit $\Sigma = \{a, b, c\}$ un alphabet. Déterminer un automate fini associé aux langages suivants construits sur Σ :

1 - le langage L1 des mots contenant au moins la suite de 3 lettres abc,

2 - le langage L2 des mots contenant exactement 4 b,

3 - le langage L3 des mots commençant par (a ou c), et se terminant par b.

Exercice 8

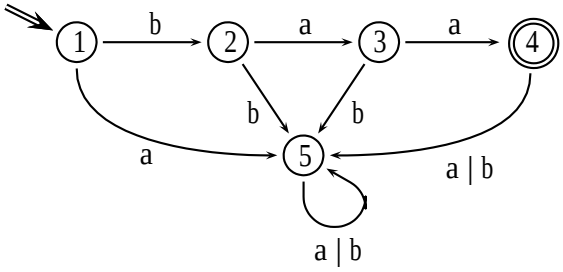
Soit l'automate $(\{0, 1, 2\}, \{a, b\}, \delta, \{0\}, \{2\})$ où δ est donnée par sa table de transition :

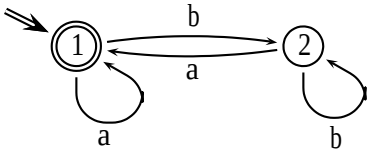
δ	a	b
0	0|1	1
1		2
2	0|1|2	1

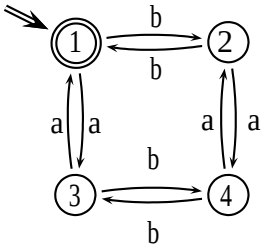
1. Pourquoi cet AF n'est-il pas déterministe complet ? Construire un AFdc qui reconnaisse le même langage.
2. Mêmes questions pour l'automate $(\{0, 1, 2\}, \{a, b\}, \delta, \{0, 1, 2\}, \{2\})$.
3. Mêmes questions pour l'automate $(\{0, 1, 2\}, \{a, b\}, \delta, \emptyset, \{2\})$.
4. Mêmes questions pour l'automate $(\{0, 1, 2\}, \{a, b\}, \delta, \{0\}, \{0, 1, 2\})$.

EXERCICE 9

Déterminer le langage de chacun des automates suivants :

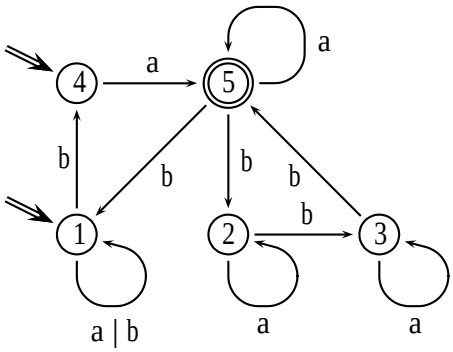


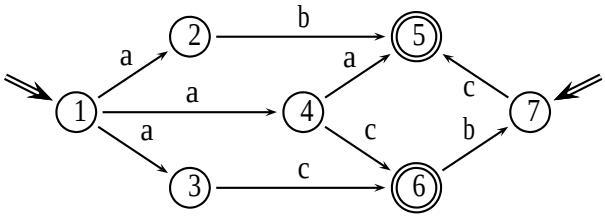




EXERCICE 10

Pour chacun des deux automates représentés ci-dessous, construire un AFDC qui reconnaisse le même langage :





CHAPITRE VI

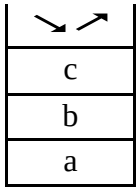
AUTOMATES A PILE

Plan

1. Introduction
2. Rappels sur les piles
3. Définition
4. Configuration et execution
5. Les critères d'acceptation
6. Automates à pile déterministes
7. Automates à pile et langages algébriques
8. Grammaire algébrique vers automate à pile
9. Automate à pile vers grammaire algébrique
10. Clôture des langages algébriques
11. Exercices

1. Introduction

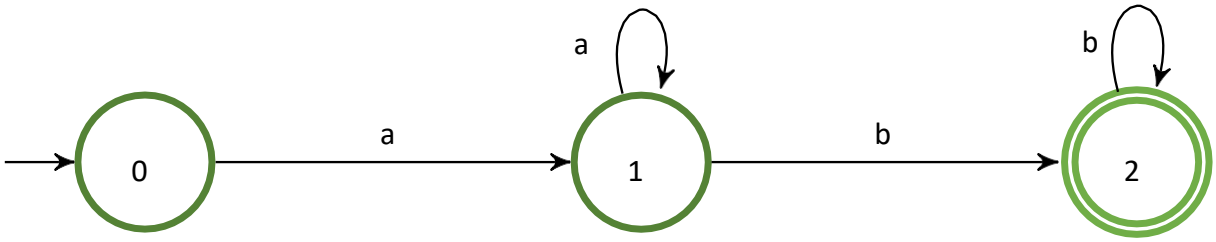
Grammaires hors contexte : génèrent des langages algébriques ;
Les automates finis acceptent (exactement) les langages réguliers ;
Langages réguliers : sous-ensemble strict des langages algébriques ;
Comment construire des automates qui acceptent les langages algébriques non réguliers ?
Un automate fini dispose par définition d'une mémoire finie ;
⇒ L'ajout d'une pile permet d'étendre les possibilités de mémorisation ;
→ Garder en mémoire les étapes de calculs passées
→ Conditionner les étapes de calculs à venir



Automate fini : défini principalement à partir de sa fonction de transition Automate à pile : enrichi la fonction de transition par un nouvel alphabet fini qui contient les symboles qui peuvent être empilés et dépilés des transitions conditionnées par le symbole en haut de la pile lors d'une transition dans l'automate, il est possible d'empiler ou de dépiler un symbole dans la pile.

Un exemple introductif

Soit l'automate suivant qui reconnaît le langage $\{a^n b^m \mid n, m > 0\}$
Cet automate ne peut pas reconnaitre le langage $\{a^n b^m \mid n = m > 0\}$: impossible de compter le nombre de a lus.



⇒ solution : Automate à pile :

- Empiler un symbole (T) à chaque passage dans la boucle de l'état 1 Dépiler un symbole à chaque passage dans la boucle de l'état 2
- Dépiler un symbole (T) à chaque passage dans la boucle de l'état 2 Dépiler un symbole à chaque passage dans la boucle de l'état 2

- Calcul réussi : pile vide

Trace du calcul pour w = aaabbb

Entrée	Etat	Pile
aaabbb	0	Pile vide
aabbb	1	Pile vide
abbb	1	T
bbb	1	TT
bb	2	TT
b	2	T
ε	2	Pile vide

De manière imagée, un automate à pile est composé de trois unités :

Une unité centrale, dont la configuration est symbolisée par un état ○

Un canal de lecture qui contient un mot à analyser

Une unité centrale dont la configuration est représentée par un état ;
Un canal de lecture qui contient le mot à analyser ;
Un canal de lecture-écriture qui est organisé en pile, et qui sert à contenir de l'information auxiliaire, en quantité non bornée.

2. Piles. Rappels

Pile : Type P
strategie: LIFO (Last In First Out)
constructeurs:

- constante pilevide ∈ P
- empiler: $E \times P \rightarrow P$
- depiler: $P \setminus \{pilevide\} \rightarrow P$
- sommet: $P \setminus \{pilevide\} \rightarrow E$
- est_vide: $P \rightarrow B$

On introduit un alphabet de pile Γ ;
Une pile p est un mot $p \in \Gamma^*$
Opérations sur les piles :
Tester si la pile est vide : déterminer si $p = \varepsilon$
Empiler un élément $x \in \Gamma$ dans une pile $p \in \Gamma^*$: $p \rightarrow xp$
Si la pile est non vide, elle est de la forme xp, où $x \in \Gamma$ et $p \in \Gamma^*$.
Dépiler l'élément x : $xp \rightarrow p$
On peut étendre ces notions à des mots. Ainsi, empiler un mot
$u = u1u2 \ldots ul$ revient à empiler successivement les lettres $u1, \ldots, ul \in \Gamma$.
Partant de la pile $p \in \Gamma*$, on obtient
$p \rightarrow u1p \rightarrow u2u1p \rightarrow \ldots \rightarrow ul \ldots u2ulp = uR\ p$

Remarque : On obtient le miroir du mot u dans la pile.

3. Définition

Un automate à pile non déterministe (en anglais pushdown automaton) est un septuplet M = (Σ, Γ, Z_0, Q, q_0, F , δ), où :

- Σ est l'alphabet d'entrée
- Γ est l'alphabet de pile
- Z0 ∈ Γ est le symbole initial de la pile Q est un ensemble fini d'états
- q0 ∈ Q est l'état initial de l'automate
- F ⊆ Q est l'ensemble des état finaux (on peut avoir F = ∅)
- δ est une fonction de Q×(Σ ∪ {ε})×(Γ ∪ {ε}) vers l'ensemble des parties de Q×(Γ ∪ {ε}).

Automate à pile : automate fini non-déterministe, à la différence près que la fonction de transition δ comporte trois arguments :

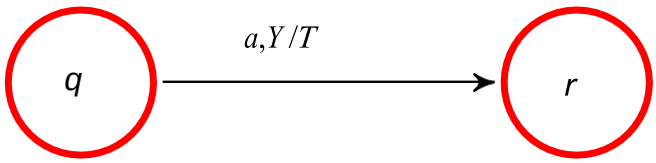
- l'état courant ;
- le symbole d'entrée courant ;
- le symbole courant en haut de la pile, appartenant à l'alphabet de pile.

Si (r , T)∈ δ(q, a, Y) (on peut écrire aussi (q, a, Y , r , T) ∈ δ ou (q, a, Y) → (r , T)), alors l'utilisation de cette transition conduira à :

- dépiler Y, Si Y = ε la transition a lieu indépendamment du symbole en haut de pile, qui reste inchangée ;
- empiler T, Si T = ε, aucun symbole n'est empilé ;
- lire le symbole a ;
- transiter dans l'état r ;

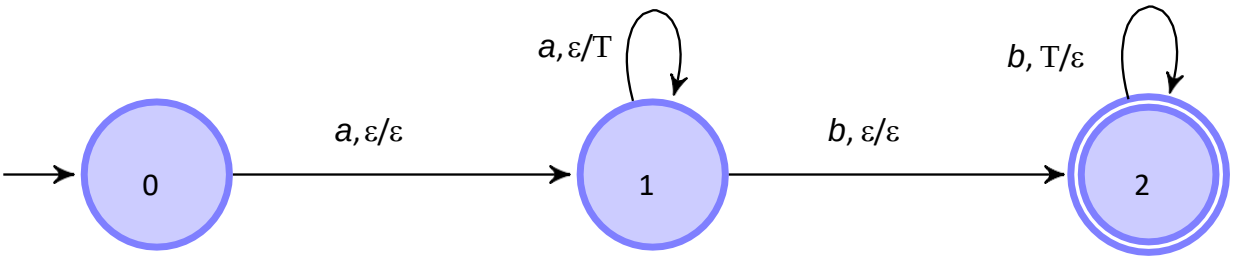
Remarque : un Automate fini "traditionnel" est un automate à pile particulier, défini sur un alphabet de pile vide (Γ = ∅) et dont toutes les transitions laissent la pile inchangée.

Transition (r , T)∈ δ(q, a, Y)



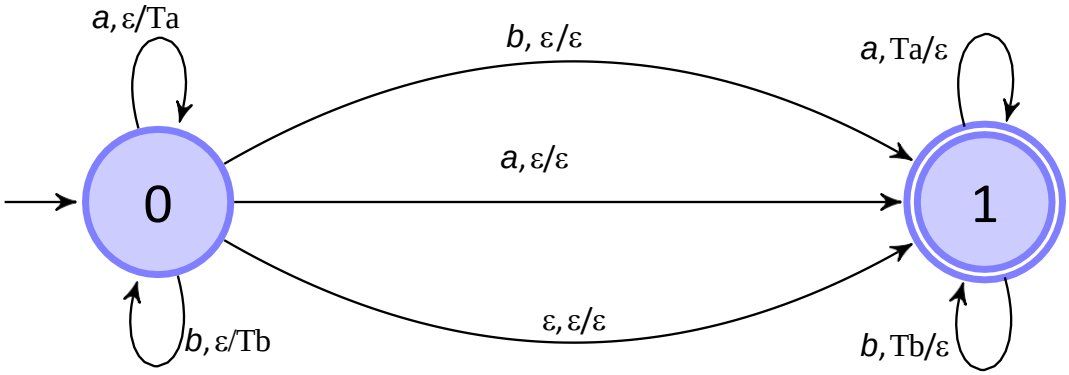
Exemple 1

Soit l'automate à pile suivant qui reconnaît le langage $\{a^n b^n / n > 0\}$



Exemple 2

Soit l'automate à pile suivant qui reconnaît le langage {w ∈ Σ∗|w est un palindrome}



4. Configuration et Exécution

Une exécution est une suite de configurations.
Une configuration est définie par :

- Le mot restant à lire m ∈ Σ^*
- L'état courant q ∈ Q
- Le contenu de la pile, l'élément le plus à gauche étant le sommet de pile

Exemple : (abbb, q, TaTbTbZ0)

La pile contient, à tout moment, un mot h sur Γ. L'automate se trouve dans un état q, et doit lire encore le mot m ∈ Σ^*
Le couple (m, q, h) est appelé une configuration de l'automate.
L'ensemble des configurations est Σ × Q × Γ .
La configuration initiale (m, q0, Z0) ∈ Σ × Q × Γ est formée de l'état initial et du symbole initial de la pile.
Un "mouvement" de l'automate représente le passage d'une configuration à une autre.

Passage d'une configuration à une autre

Le passage d'une configuration c1 à une configuration c2 dans un automate M s'écrit :
$c1 \vdash_M c2$
On note $\vdash^*_M$ la clôture réflexive et transitive de $\vdash_M$
Il y a deux modes de transition pour changer de configuration :

- Sur une Σ-transition
- Sur une ε-transition

Σ-transition :

Transition (q1, b, TA) → (q1, TB) Configuration (bba, q1, TAZ0) On aura alors :
(bba, q1, TAZ0) ⊢M (ba, q1, TB Z0)

ε-transition :

Transition (q1, ε, Ta) → (q2, Tb) Configuration (bba, q1, TaZ0) On aura alors :
(bba, q1, TaZ0) $\vdash_M$ (bba, q2, Tb Z0)
On ne touche pas à la tête de lecture.

Σ-transition : exemple

Soit l'automate à pile qui reconnaît le langage $\{a^n b^n | n > 0\}$
(aabb, 0, Z0) $\vdash_M$ (abb, 1, Z0)
$\vdash_M$ (bb, 1, TZ0)
$\vdash_M$ (b, 2, TZ0)
$\vdash_M$ (ε, 2, Z0)

ε-transition : exemple

Soit l'automate à pile suivant qui reconnaît le langage {w ∈ Σ∗|w est un palindrome}
(baaab, 0, Z0)⊢M(aaab, 0, TbZ0)
$\vdash_M$(aab, 0, $T_aT_bZ_0$)
$\vdash_M$(ab, 1, $T_aT_bZ_0$)
$\vdash_M$(b, 1, T_bZ_0)
$\vdash_M$(ε, 1, Z_0)

5. Les critères d'acceptation

Dans nos exemples, on accepte un mot si le ruban vide, on est sur l'état final et la pile vide
Ce sont des cas particuliers
Il y a deux critères d'acceptation possibles :

- Acceptation par état final (quelle que soit la pile quand on s'arrête) ;
- Acceptation par pile vide (quel que soit l'état dans lequel on s'arrête) ;

Mais le ruban doit toujours être vide ! Ces deux critères sont équivalents

Acceptation par état final

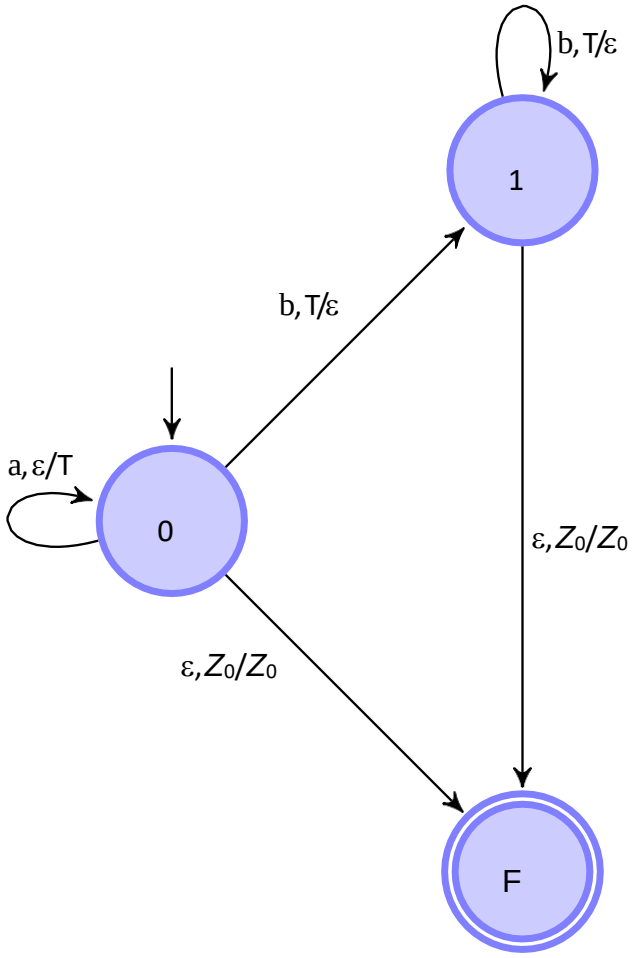
Un mot m ∈ Σ^* est accepté par état final par un automate à pile
M = (Σ, Γ, Z_0, Q, q_0, F , δ) si pour la configuration (m, q_0, Z_0), il existe
un état qf ∈ F et un mot z ∈ Γ∗ tel que : (m, q0 , Z0) ⊢∗M (ε, qf , z).

Langage accepté par état final

Le langage accepté par état final par un automate à pile est l'ensemble des mots acceptés par cet automate. L^F (M) = {m ∈ Σ^*|(m, q0 , Z0) $\vdash^{*M}$ (ε, qf , z)}

Acceptation par état final : exemple

Soit l'automate à pile suivant qui reconnaît le langage $\{a^n b^n | n \geq 0\}$



Acceptation par pile vide

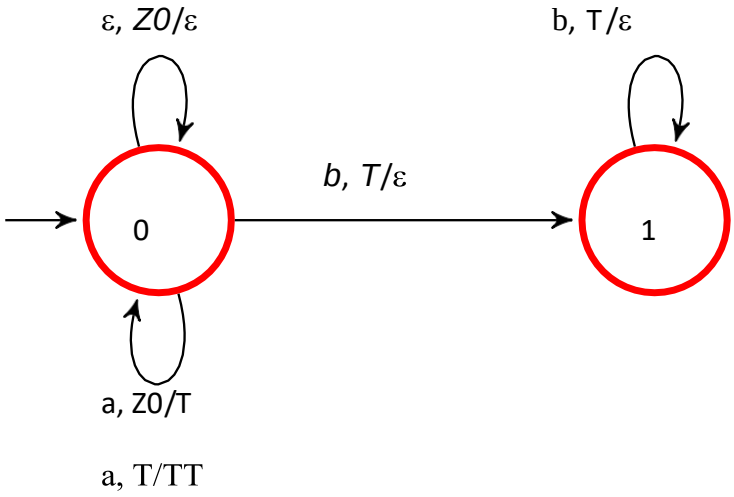
Un mot m ∈ Σ∗ est accepté par pile vide par un automate à pile M = (Σ, Γ, Z0, Q, q0, F , δ) si pour la configuration (m, q0, Z0), il existe un ´état q ∈ Q tel que
(m, q0 , Z0) ⊢∗M (ε, q, ε)

Langage accepté par pile vide

Le langage accepté par pile vide par un automate à pile est l'ensemble des mots acceptés par cet automate.
LV (M) = {m ∈ Σ∗|(m, q0 , Z0) ⊢∗ M (ε, q, ε)}

Exemple

Soit l'automate à pile suivant qui reconnaît le langage $\{a^n b^n | n \geq 0\}$



Les deux critères d'acceptation (par état final et par pile vide) sont équivalents

Théorème

Un langage est accepté par un automate à pile avec le critère d'acceptation sur pile vide si et seulement s'il est accepté par un automate à pile avec acceptation par état final.

⇒ Chaque transition dans laquelle Z0 est dépilé est remplacée par une transition vers un nouvel état final
⇐ Après avoir atteint un état final, on vide entièrement la pile.

6. Automates à pile déterministes

Les automates à pile que nous avons définis jusqu'à maintenant sont indéterministes.
Un mot est accepté s'il existe au moins une suite de configurations qui conduit à l'acceptation
Mais il peut y en avoir plusieurs
Et il peut il y avoir plusieurs suites de configuration qui mènent à l'échec
⇒ Automate à pile déterministe ?
Un automate à pile M est déterministe si :

- pour un état q donné, pour un symbole d'entrée x donné, pour un sommet de pile z donné, il existe au plus une transition partant de (q, x , z)
- pour un état q donné, pour un sommet de pile z donné, s'il existe une transition partant de (q, ε, z), elle est unique et pour toute lettre x , il n'en existe pas partant de (q, x , z).

⇒ Dans une configuration donnée, on ne peut pas avoir le choix sur la transition à appliquer

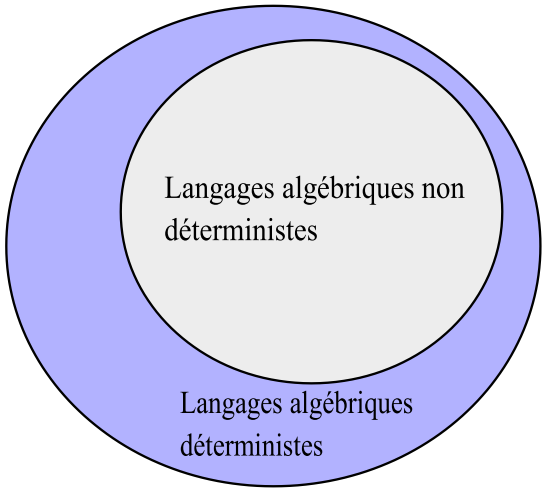
7. Automates à pile et langages algébriques (hors-contextes)

Théorème

Un langage est algébrique si et seulement s'il est reconnu par un automate à pile.

Théorème

Tout langage algébrique n'est pas nécessairement reconnu par un automate à pile déterministe.



Langage algébrique déterministe

Un langage algébrique L est déterministe s'il existe un automate à pile M acceptant par état final déterministe tel que $L^F(M) = L$

Par exemple, $\{m \in (a + b)^* | m \text{ est un palindrome}\}$ est un langage algébrique non déterministe.
Intuitivement, on ne sait pas deviner où est le milieu du mot.
$\{m1cm2 | m1m2 \in (a + b)^* \text{ est un palindrome}\}$ est un langage algébrique déterministe.

Forme normale de Greibach

Une grammaire algébrique G = (V , Σ, P, S) est sous la forme normale de Greibach si toute production est de la forme :

$A \rightarrow aA_1 \ldots A_n$

$A \rightarrow a$

avec $A, A_i \in V \setminus \Sigma$, et $a \in \Sigma$

Algorithme de passage d'une grammaire algébrique sous forme normale de Greibach vers un automate à pile non déterministe
Idée :
Empiler l'axiome
A chaque symbole lu de la chaîne d'entrée, remplacer la partie gauche de la production concernée par le reste de la partie droite

8. Grammaire algébrique vers automate à pile

Soit $L = L \setminus \varepsilon$ le langage engendré par $G = (V , \Sigma, P, S)$ sous forme normale de Greibach.
On construit l'automate à pile $M = (\Sigma, \Gamma, S , Q, q0, F , \delta)$ tel que
$\Gamma = V \setminus \Sigma$
$Q = \{q0\}$
$F = \emptyset$ (reconnaissance sur pile vide)
S le symbole initial de la pile
On construit δ itérativement de la fac¸on suivante :
1. $\delta \leftarrow \emptyset$
2. Pour toute règle $A \rightarrow aA1A2 \ldots An$,
$\delta \leftarrow \delta \cup \{(q0, a, A) \rightarrow (q0, A1A2 \ldots An)\}$
On lit a, on dépile A et on empile A1A2 . . . An (A1 est en sommet de la pile)

Grammaire algébrique vers automate à pile : exemple

Soit $G = (V , \Sigma, P, S)$ avec
$V = \{a, b, S , B\}$
$\Sigma = \{a, b\}$
$P = \{S \rightarrow aSB|aB; B \rightarrow b\}$

On construit l'automate $M = (\Sigma, \Gamma, S , Q, q0, F , \delta)$ tel que
$\Gamma = \{S , B\}$
$Q = \{q0\}$, $F = \emptyset$
δ contient
$(q0, a, S) \rightarrow (q0, SB)$
$(q0, a, S) \rightarrow (q0, B)$
$(q0, b, B) \rightarrow (q0, \varepsilon)$

Grammaire algébrique vers automate à pile : exemple

Soit $G = (V , \Sigma, P, S)$ avec $V = \{a, +, *, (,), S , A, B, C \}$, $\Sigma = \{a, b, +, *, (,)\}$

$S \rightarrow a$
$S \rightarrow aAS$
$S \rightarrow (SCAS$
$S \rightarrow aBS$
$S \rightarrow (SC$
$A \rightarrow +$
$B \rightarrow *$
$C \rightarrow)$

S ⟶ (SCBS
On construit l'automate M = (Σ, Γ, S , Q, q0, F , δ) tel que Γ = {S , A, B, C }, Q = {q0}, F = ∅ et δ contient

(q0, a, S) ⟶ (q0, ε)
(q0, a, S) ⟶ (q0, AS)
(q0, (, S) ⟶ (q0, SCAS)
(q0, a, S) ⟶ (q0, BS)
(q0, (, S) ⟶ (q0, SCBS)
(q0, (, S) ⟶ (q0, SC)
(q0, +, A) ⟶ (q0, ε)
(q0, ∗, B) ⟶ (q0, ε)
(q0,), C) ⟶ (q0, ε)

9. Automate à pile vers grammaire algébrique

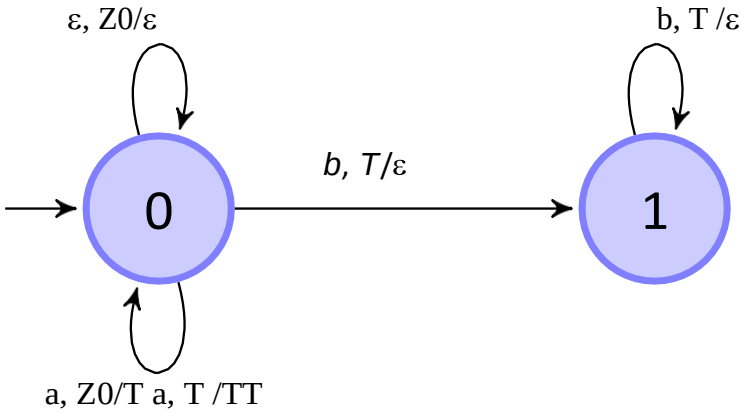
On présente un algorithme de passage d'un automate à pile à une grammaire algébrique
A tout couple (p, q) d'états de l'automate, et à tout X de la pile, on associe un non terminal de la forme (p, X , q).
On associe toutes les lectures possibles dans l'automate pour obtenir la grammaire sans savoir a priori lesquelles vont vider la pile
A la fin, on nettoie la grammaire obtenue

Automate à pile vers grammaire algébrique : algorithme

Soit M = (Σ, Γ, Z0, Q, q0, F , δ). On veut construire G = (V , Σ, P, S) équivalente.
V = {(q, X , p)|p et q ∈ Q, X ∈ Γ} ∪ {S }
P ← ∅
Pour tout état q ∈ Q, P ← P ∪ {S ⟶ (q0, Z0, q)}
Pour toute transition (q, a, X) ⟶ (p, ε) de δ faire P ← P ∪ {(q, X , p) ⟶ a}
Pour toute transition (q, a, X) ⟶ (p, Bm . . . B1) de δ faire Pour tout m-uplet d'états q1, . . . , qm de Q faire

P ← P ∪ {(q, X , qm) ⟶ a(p, Bm, q1)(q1, Bm−1, q2) . . . (qm−1, B1, qm)}

Automate à pile vers grammaire algébrique : exemple



V = {(q, X , p)|p et q ∈ Q, X ∈ Γ} ∪ {S }

V = {S , (q0, Z0, q0), (q0, Z0, q1), (q1, Z0, q0), (q1, Z0, q1),
(q0, T , q0), (q0, T , q1), (q1, T , q0), (q1, T , q1)}

P ← ∅
Pour tout état q ∈ Q, P ← P ∪ {S → (q0, Z0, q)} P ← { S→ (q0, Z0, q0) , S→ (q0, Z0, q1) }
Pour toute transition (q, a, X) → (p, ε) de δ faire

P ← P ∪ {(q, X , p) → a} P ←
(q0, Z0, q0)→ ε
(q0, T , q1)→ b
(q1, T , q1)→ b
Pour toute transition (q, a, X) → (p, Bm . . . B1) de δ faire Pour tout m-uplet d'états q1, . . . , qm de Q faire
P ← P ∪ {(q, X , qm) → a(p, B1, q1)(q1, B2, q2) . . . (qm−1, Bm, qm)}

Pour toute transition (q, a, X) → (p, Bm . . . B1) de δ faire Pour tout m-uplet d'états q1, . . . , qm de Q faire
P ← P ∪ {(q, X , qm) → a(p, B1, q1)(q1, B2, q2) . . . (qm−1, Bm, qm)}

(q0, Z0, q0)→ a(q0, T , q0)
(q0, Z0, q1)→ a(q0, T , q1)
Pour toute transition (q, a, X) → (p, Bm . . . B1) de δ faire Pour tout m-uplet d'états q1, . . . , qm de Q faire
P ← P ∪ {(q, X , qm) → a(p, B1, q1)(q1, B2, q2) . . . (qm−1, Bm, qm)}
(q0, T , q0)→ a(q0, T , q0)(q0, T , q0)
(q0, T , q0)→ a(q0, T , q1)(q1, T , q0)
(q0, T , q1)→ a(q0, T , q0)(q0, T , q1)
(q0, T , q1)→ a(q0, T , q1)(q1, T , q1)

On récapitule :
A V = {S , (q0, Z0, q0), (q0, Z0, q1), (q1, Z0, q0), (q1, Z0, q1),
(q0, T , q0), (q0, T , q1), (q1, T , q0), (q1, T , q1)}

$S \longrightarrow (q_0, Z_0, q_0)$
$S \longrightarrow (q_0, Z_0, q_1)$
$(q_0, Z_0, q_0) \longrightarrow \varepsilon$

$(q_0, T, q_1) \longrightarrow b$
$(q_1, T, q_1) \longrightarrow b$
$(q_0, Z_0, q_0) \longrightarrow a(q_0, T, q_0)$
$(q_0, Z_0, q_1) \longrightarrow a(q_0, T, q_1)$
$(q_0, T, q_0) \longrightarrow a(q_0, T, q_0)(q_0, T, q_0)$
$(q_0, T, q_0) \longrightarrow a(q_0, T, q_1)(q_1, T, q_0)$
$(q_0, T, q_1) \longrightarrow a(q_0, T, q_0)(q_0, T, q_1)$
$(q_0, T, q_1) \longrightarrow a(q_0, T, q_1)(q_1, T, q_1)$

Automate à pile vers grammaire algébrique : exemple

On renomme :
V = {S , (q0, Z0, q0), (q0, Z0, q1), (q1, Z0, q0), (q1, Z0, q1),
(q0, T , q0), (q0, T , q1), (q1, T , q0), (q1, T , q1)}

$S \longrightarrow (q_0, Z_0, q_0)$
$S \longrightarrow (q_0, Z_0, q_1)$
$(q_0, Z_0, q_0) \longrightarrow \varepsilon$
$(q_0, T, q_1) \longrightarrow b$
$(q_1, T, q_1) \longrightarrow b$
$(q_0, Z_0, q_0) \longrightarrow a(q_0, T, q_0)$
$(q_0, Z_0, q_1) \longrightarrow a(q_0, T, q_1)$
$(q_0, T, q_0) \longrightarrow a(q_0, T, q_0)(q_0, T, q_0)$
$(q_0, T, q_0) \longrightarrow a(q_0, T, q_1)(q_1, T, q_0)$
$(q_0, T, q_1) \longrightarrow a(q_0, T, q_0)(q_0, T, q_1)$
$(q_0, T, q_1) \longrightarrow a(q_0, T, q_1)(q_1, T, q_1)$

Automate à pile vers grammaire algébrique : exemple

On renomme :
V = {S , A, B, C , D, E , F , G , H}

S	→	A	B	→	aF
S	→	B	E	→	aEE
A	→	ε	E	→	aFG
F	→	b	F	→	aEF
H	→	b	F	→	aFH
A	→	aE			

Symboles productifs : {A, F , H, S , B}

- ☞ On supprime C , D, E , G

On nettoie :

Symboles productifs : {A, F , H, S , B}

- ☞ On supprime C , D, E , G

V = {S , A, B, F , H}
S→A
S→B
A→ε
B→aF
F→b
H→bF → aFH

- ☞ On nettoie :

V = {S , A, B, F , H}
S→A
S→B
A→ε
H→b
B→aF
F→b
F→aFH
Symboles productifs : {A, F , H, S , B}

- ☞ On supprime C , D, E , G

Symboles accessibles : {S , A, B, F , H}
Suppression des règles unitaires

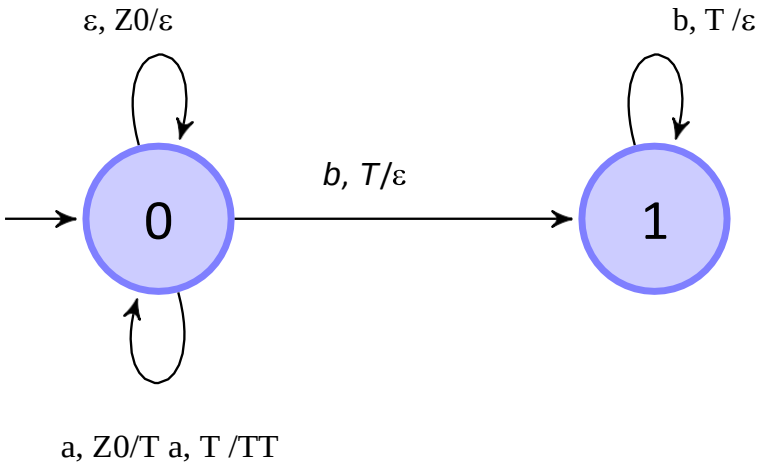
- ☞ On nettoie :

V = {S , A, B, F , H}
S→ε
S→aF
F→b
F→aFb

Symboles productifs : {A, F , H, S , B}
→ On supprime C , D, E , G

Symboles accessibles : {S , A, B, F , H}
Suppression des règles unitaires

Automate à pile vers grammaire algébrique : exemple



On obtient :

S	→	ε
S	→	aF
F	→	b
F	→	aFb

10. Clôture des langages algébriques

Clôture par union

Soient L_1 et L_2 deux langages algébriques.
Alors $L_1 \cup L_2$ est un langage algébrique.

Preuve par construction :
Soient G1 = (V1, Σ1, P1, S1) et G1 = (V2, Σ2, P2, S2) engendrant L1 et L2 et tels que (V1 \ Σ1) ∩ (V2 \ Σ2) = ∅ et S /∈ V1 ∪ V 2 (et sinon on renomme)
On construit G = (V , Σ, P, S) telle que :
V = V1 ∪ V2 ∪ {S }
Σ = Σ1 ∪ Σ2
P = P1 ∪ P2 ∪ {S → S1|S2}

Clôture par concaténation

Soient L_1 et L_2 deux langages algébriques.
Alors $L_1.L_2$ est un langage algébrique.

Preuve par construction :
Soient G1 = (V1, Σ1, P1, S1) et G1 = (V2, Σ2, P2, S2) engendrant L1 et L2 et tels que (V1 \ Σ1) ∩ (V2 \ Σ2) = ∅ et S /∈ V1 ∪ V 2 (et sinon on renomme)
On construit G = (V , Σ, P, S) telle que :
V = V1 ∪ V2 ∪ {S }
Σ = Σ1 ∪ Σ2
P = P1 ∪ P2 ∪ {S → S1S2}

Clôture par opération étoile

Soit L un langage algébrique. Alors L^* est un langage algébrique.

Preuve par construction :
Soit G1 = (V1, Σ1, P1, S1) engendrant le langage L
On construit G = (V , Σ, P, S) engendrant le langage L∗ telle que :
V = V1 ∪ {S }
Σ = Σ1
P = P1 ∪ {S → ε|S1S }

Exemple

Peut-on trouver une grammaire algébrique pour L = {ai bj ck |i = j ou j = k} ?
L se décompose en l'union de
L1 = {ai bj ck |i = j, k ≥ 0}
L2 = {ai bj ck |i ≥ 0, j = k}
L1 = {ai bj ck |i = j, k ≥ 0} est la concaténation de {ai bj |i = j} et {ck |k ≥ 0}
L2 = {ai bj ck |i ≥ 0, j = k} est la concaténation de {ai |i ≥ 0} et {bj ck |j = k}

Exemple, suite

{ai bj |i = j}
S1 → ε|aS1b
{ck |k ≥ 0}
S2 → ε|cS2
L1 = {ai bj ck |i = j, k ≥ 0}
S1 → ε|aS1b
S2 → ε|cS2
SL_1 → S1S2
L2 = {ai bj ck |i ≥ 0, j = k}
S3 → ε|aS3
S4 → ε|bS4c
SL_2 → S3S4

{ai bj |i = j}
S1 → ε|aS1b
{ck |k ≥ 0}
S2 → ε|cS2
L1 = {ai bj ck |i = j, k ≥ 0}
S1 → ε|aS1b
S2 → ε|cS2SL_1 → S1S2
L2 = {ai bj ck |i ≥ 0, j = k}
S3 → ε|aS3
S4 → |bS4c
SL_2 → S3S4
L = {ai bj ck |i = j ou j = k} G = (V , Σ, S , P)
V = {S1, S2, S3, S4, SL_1 , SL_2 , S }
Σ = {a, b, c} P :

S_1 →ε|aS_1b
S_2 → ε|cS_2
S_{L_1} → S_1S_2
S_3 →ε|aS_3
S_4 →ε|bS_4c
S_{L_2} →S_3S_4
S→S_{L_1} |S_{L_2}

Clôture par intersection

La classe des langages algébriques n'est pas close par intersection

Série d'exercices VI

Exercice 1.

Donner l'automate à pile permettant de reconnaître chacun des langages suivants :

a) $\{a^n b^m \mid n,m \geq 0 \text{ et } n \leq m\}$
b) $\{a^n b^m c^2(n+m) \mid n,m \geq 0\}$
c) $\{a^n b^m \mid n,m \geq 0 \text{ et } n \leq m \leq 2n\}$
d) $\{a^m b^k a^m b^k / m \geq 1 et 1 \leq k \leq n\}$

Exercice 2.

Donner des automates à pile reconnaissant les langages suivants :

a) $L_1 = \{u \in \{a, b\}/ |u|_a = |u|_b\}$
b) $L_2 = \{u \in \{a, b\}/ |u|_a \geq |u|_b\}$
c) $L_3 = \{u \in \{a, b\}/ |u|_a = 2|u|_b\}$

Exercice 3.

Pour chacun des langages des exercices précédents, donner une grammaire qui génère le même langage.

Exercice 4

Donner l'automate `a pile qui permet de reconnaitre le langage des palindromes de longueur paire (non nulle) sur l'alphabet {a, b}. L'automate obtenu est-il d´déterministe ? Justifiez votre réponse.

Si ce n'est pas le cas, pourriez-vous proposer un automate déterministe qui reconnaisse ce langage ? Comment modifier l'automate obtenu pour reconnaitre tous les palindromes (pairs et impairs) ?

Exercice 5.

On considère l'automate à piles $P = (\{q,p\}, \{0,1\}, \{Z_0,X\},\delta,q,Z0, \{p\})$ dont on définit δ comme suit :

$\delta(q, 0,Z_0) = \{(q,XZ_0)\}$
$\delta(q, 0,X) = \{(q,XX)\}$
$\delta(q, 1,X) = \{(q,X)\}$
$\delta(q,\varepsilon,X) = \{(p, \varepsilon)\}$
$\delta(p, \varepsilon,X) = \{(p, \varepsilon)\}$
$\delta(p, 1,X) = \{(p,XX)\}$
$\delta(p, 1,Z_0) = \{(p, \varepsilon)\}$

Décrire le comportement de P sur les entrées suivantes : 01 ; 0011 ; 010

Exercice 6.

On considère l'automate à pile P de l'exercice précédent. Soit L le langage reconnu par P par état final et L_0 celui qu'il reconnait par pile vide.

1. Construire un automate à pile P_1 reconnaissant L par pile vide.
2. Construire un automate à pile P_2 reconnaissant L_0 par état final.

Exercice 7.

Décrire un automate à piles acceptant les langages suivants (l'acceptation peut se faire par état final ou par pile vide) :

1. $\{0^n1^n / n \geq 1\}$
2. L'ensemble des mots sur l'alphabet {0,1} tels qu'aucun de leur préfixe n'a plus de 1 que de 0.
3. L'ensemble des mots sur l'alphabet {0,1} ayant autant de 0 que de 1.
4. $\{a^i b^j c^k / i = j \text{ or } j = k\}$

Exercice 8.

Soit P un automate à pile reconnaissant un langage L =N(P) par pile vide.

Supposons que $\varepsilon \notin L$. Montrer comment modifier P pour qu'il reconnaisse L U{ε} par pile vide.

Exercice 9.

On considère un automate à piles P = ({q_0,q_1,q_2,q_3, f }, {a,b}, {Z_0, A,B},δ,q_0,Z_0, { f }).

Les règles définissant δ sont définies ci-dessous :

δ (q_0,a,Z_0) = (q_1, AAZ_0)
δ(q_0,b,Z_0) = (q_2,BZ_0)
δ(q_0, ε,Z_0) = (f , ε)
δ (q_1,a, A) = (q_1, AAA)
δ(q_1,b, A) = (q_1, ε)
δ(q_1, ε,Z0) = (q_0,Z0)
δ(q_2,a,B) = (q_3, ε)
δ(q_2,b,B) = (q_2,BB)
δ(q_2, ε,Z_0) = (q_0,Z_0)
δ(q_3, ε,B) = (q_2, ε)
δ(q_3, ε,Z_0) = (q_1, AZ_0)

1. Montrer que le mot bab ∈ L(P)
2. Montrer que abb ∈ L(P)
3. Décrire le contenu de la pile après lecture de b^7a^4
4. Quel est le langage reconnu par P ?

Exercice 10.

Grammaires ambiguës

a) Ecrire une grammaire hors-contexte qui génère le langage $L = \{a^i b^j c^k | i = j \text{ ou } i = k\}$
b) Votre grammaire est-elle ambiguë? pourquoi?

Exercice 11.

Déterminisme

a) Dessiner un automate à pile déterministe *A* qui reconnaît le langage $\{m\#m^{-1} | m \in \{a,b\}^*\}$
b) Simuler le comportement de *A* sur l'entrée *abba#abba*
c) Dessiner au automate à pile A^0 qui reconnaît le langage $\{mm^{-1} | m \in \{a,b\}^*\}$
d) Simuler le comportement de A0 sur l'entrée abbaabba — Est-il possible de déterminiser A0 ?

Exercice 13.

Automates et grammaires

a) Dessiner un automate à pile qui reconnaît le langage des expressions régulières sur l'alphabet $\{a,b\}$
b) Donner une dérivation gauche de l'expression $a^*(b+c)$
c) Dessiner un analyseur gauche pour le langage des expressions régulières sur l'alphabet $\{a,b\}$
d) Dessiner l'arbre de dérivation que produit l'analyseur pour le mot $a+b^*aa$

Exercice 14

On considère la grammaire G = ({e,p} ; {S} ; S ; { S → SpS | e }).

1) Quel est le type de cette grammaire ? Justifier.
2) Donner l'arbre de dérivation du mot epepe.
3) Montrer que G est ambiguë ?
4) Décrire le langage L(G) engendré par la grammaire G.
5) Mettre la grammaire G sous forme normale de Chomsky.
6) On considère l'automate à pile M = ({0}, {e,p}, {Z_0,X},δ,1,Z_0, ∅) dont on définit δ comme suit :

   $\delta(0, p, Z_0) = \{(0, XXZ_0)\}$
   $\delta(0, p, X) = \{(0, XXX)\}$
   $\delta(0, e, X) = \{(0, \varepsilon)\}$
   $\delta(0, e, Z_0) = \{(0, \varepsilon)\}$

   a) Construire le graphe de M.
   b) Exécuter M pour le mot pepee.
   c) Quel est le mode d'acceptation de cet automate. Justifier.

Exercice 15

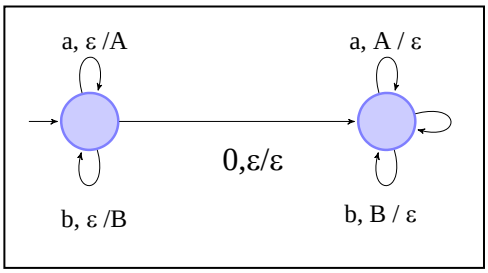
Soit la grammaire G1 = ({c,d} , {S,X} , S , {S → cXd ; X → cSd ; Xd → ε }) .

1) Montrer que G1 est de type 0.
2) Trouver une grammaire G2 de type 2 équivalente à G1.
3) Donner l'arbre de dérivation gauche du mot c^5d^4 par G2.
4) G2 est-elle ambiguë ? Justifier.
5) Calculer le langage L(G2) généré par G2.
6) Mettre G2 sous forme normale de Greibach puis construire un automate à pile associé à G2.

Exercice 16

Soit l'automate à pile ci-dessous.

1. Donner les différentes étapes de reconnaissance du mot baa0aab.
2. Quel est le mode d'acceptation de cet automate ?
3. Quel langage est accepté par cet automate ?



Références bibliographiques

1. J.E. Hopcroft, J.D. Ullman. Introduction to automata theory, languages and computation.Addison-Wesley, 1979.
2. M. Sipser. Introduction to the theory of computation. PWS Publishing Company, 1996.
3. J. Berstel. Automates et grammaires (notes de cours, Univ. Marne-la-Vallée, 2005).
4. Gire, S. (2000). Compilation théorique des langages. http://fastnet.univ-brest.fr. (Deuxième année IUP Ingénierie Informatique Université de Bretagne Occidentale)
5. Lecomte, A. (2000). Langages et communication. http://brassens.upmf-grenoble.fr/-~alecomte/alaincours.html. (Cours pour le DESS Double Compétence "Informatique et Sciences Sociales")
6. Véronis, J. (2001). Cours d'Informatique et linguistique 1 (INFZ18). (Université de Provence, Centre Informatique pour les Lettres et Sciences Humaines)
7. Perrin, D. (1995). Les débuts de la théorie des automates. Technique et science informatique, Un historique de l'émergence de la théorie des automates.
8. Sakarovitch, J. (2003). Eléments de théorie des automates. Vuibert, Paris.
9. Hopcroft, J. E. et Ullman, J. D. (1979). Introduction to automata theory, languages and computation. Addison-Wesley
10. AHO, ULLMAN, SETHI "Compilers, Principles techniques and tools" Addison-Wesley 1986
11. A. Aho, R. Sethi and J. Ullman, Compilers Principles, Techniques and Tools, Addsion Wesley, 1986
12. J. Hopcroft, R. Motwani and J. Ullman, Introduction to Automata Theory, Languages and Computation, Addsion Wesley, 2003 .
13. R. Hunter, The Essence of Compilers, Prentice Hall, 1999.