Université de Mostaganem
Faculté des Sciences Exactes et Informatique

Polycopié

Théorie des langages

Dr. Karim BESSAOUD

2017 - 2018

Table des matières

Introduction

En linguistique et en informatique, on parle de théorie des langages pour décrire les langages formels. Plus généralement, la théorie des langages concerne tout langage fini ou infini qui peut être spécifié par une méthode ou un mécanisme fini, et explicite, qui permet de le produire ou de l'analyser.

La théorie des langages ne se préoccupe pas du côté sémantique (le sens), mais plutôt syntaxique (la forme) d'un langage. On définit un langage notamment grâce aux grammaires et on analyse les mots d'un langage grâce aux automates ou reconnaisseurs.

Ce cours va s'articuler autour de ces trois concepts que sont les langages, les grammaires et les automates.

La théorie des langages est une base pour un certain nombre de disciplines informatique dont principalement la compilation mais aussi, la logique, la complexité, la calculabilité...

Nous allons étudier dans cette matière tous les types de langages décrits dans la classification de Chomsky que nous allons voir plus tard ainsi que les grammaires qui permettent de les générer et les automates qui permettent de les reconnaître.

Dans le chapitre suivant, nous allons étudier les concepts de base des trois axes de notre cours, à savoir les langages, les grammaires et les automates. Nous verrons aussi la classification de Chomsky qui définit quatre grands types. Un chapitre sera par la suite réservé à chaque type de langage.

Chapitre 1

Concepts de base

1.1 Langages

Un langage est tout système permettant de s'exprimer comme par exemple les langages linguistiques, les langages mathématiques ou les langages informatiques.

La théorie des langages étudie les aspects purement syntaxiques de tels langages, c'est-à-dire leurs structures internes formelles.

Un langage est constitué de mots qui sont eux-mêmes constitués de symboles faisant partie d'un alphabet.

1.1.1 Alphabet

Un alphabet est un ensemble fini de symboles permettant de construire les mots d'un langage.

Exemple :

- $\{0, 1\}$
- $\{if, then, else, x, y, z\}$ dans cet exemple 'if', '$then$' et '$else$' ne sont pas considérés comme des mots mais bien comme des symboles.
- $\{+, -, *, /, =, a, b\}$

1.1.2 Mot

Un mot sur un alphabet est une séquence finie et ordonnée, éventuellement vide, de symboles de l'alphabet. Le mot vide est noté ε.

Voici quelques exemples de mots :

- 100111 est un mot de l'alphabet $\{0, 1\}$.
- $a = b$ est un mot de l'alphabet $\{+, -, *, /, =, a, b\}$.

La longueur d'un mot : La longueur d'un mot m est notée $\mid m \mid$ et correspond au nombre de symboles que contient ce mot.

On peut aussi avoir le nombre d'occurrence d'un symbole donnée par exemple $\mid m \mid_a$ pour le nombre de a dans le mot m.

Exemple : Soit le mot $m = 100111$ sur l'alphabet $\{0, 1\}$.
$\mid m \mid = 6$.
$\mid m \mid_1 = 4$.
$\mid m \mid_0 = 2$.

1.1.3 Définition formelle d'un langage

Un langage, défini sur un alphabet A, est un ensemble de mots définis sur A. Un langage peut être décrit de différentes manières :

- Un langage fini peut être décrit par l'énumération des mots qui le composent.
- Certains langages infinis peuvent être décrits par l'application d'opérations à des langages plus simples. Ces opérations seront décrites en section 1.1.4.
- Certains langages infinis peuvent être décrits par un ensemble de règles appelé grammaire.
- Enfin, certains langages infinis ne peuvent être décrits, ni par l'application d'opérations, ni par un ensemble de règles. Dans ce type de langages, il n'existe pas d'algorithme permettant de déterminer si un mot donné appartient à ce langage ou non.

Dans ce cours, nous allons nous intéresser uniquement aux langages qui peuvent être décrits formellement et qui sont classés par la classification de Chomsky en quatre types :

Type 3 : langages réguliers,

Type 2 : langages algébriques,

Type 1 : langages contextuels,

Type 0 : langages récursivement énumérables.

Il existe une relation d'inclusion entre ces quatre types de langages : type 3 $\subset$ type 2 $\subset$ type 1 $\subset$ type 0 (voir figure 1.1). Par contre, on entend par langage de type i le premier type en commençant par le type 3 qui vérifie les contraintes du type de langages. Un langage de type 1 sous-entend implicitement qu'il n'est pas de type 2 et donc pas de type 3.

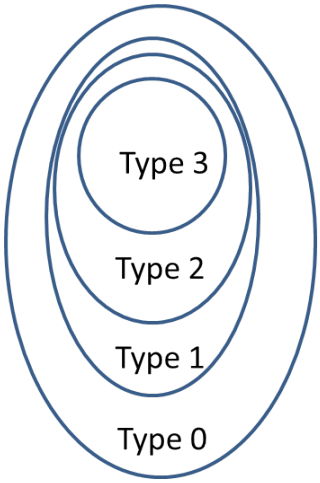


FIGURE 1.1: La relation d'inclusion entre les types de langages

1.1.4 Opérations sur les langages

Un langage étant un ensemble de mots (fini ou infini), les opérations définies sur les ensembles comme l'union, l'intersection ou encore le complément peuvent être appliquées aux langages. Il existe aussi des opérations spécifiques telles que la concaténation, l'image miroir ou l'étoile.

La concaténation : Soient deux mots u et v définis sur un alphabet A. La concaténation de u avec v, notée $u.v$ ou simplement uv, est le mot formé en faisant suivre les symboles de u par les symboles de v.

Soient deux langages $L1$ et $L2$ définis sur un alphabet A. La concaténation de $L1$ avec $L2$, notée $L1L2$, est le langage formé par les mots uv tel que u appartient à $L1$ et v appartient à $L2$.

$L1L2 = \{m/m \in uv \text{ avec } u \in L1 \text{ et } v \in L2\}$

L'étoile (la puissance) : L'opération de puissance peut être appliquée à un alphabet, un langage, un mot ou un symbole.

On note a^*, tel que a est un symbole, l'ensemble des mots constitués d'une suite de symbole a d'une taille supérieure ou égale à 0.

On note a^+ l'ensemble des mots constitués d'une suite de symbole a d'une taille supérieure 0 donc sans le mot vide. $a^* = \{\varepsilon\} \cup a^+$

On note A^* l'ensemble des mots de longueur supérieure ou égale à 0 que l'on peut construire à partir de l'alphabet A.

Soit un mot $u \in A^*$ et n un entier positif, on note u^n un mot constitué par la concaténation de n fois u.

Exemples : Soit un alphabet $A = \{a, b\}$ on note :
- $(ab)^2$ pour le mot $abab$
- ab^2 pour le mot abb
- $(a^3b)^2$ pour le mot $aaabaaab$
- a^+b^* pour le langage infini qui est formé des mots constitué d'une suite non vide de a suivie par une suite potentiellement vide de b.
- $(a^+bc^*)^*$ ici, nous avons un langage à l'étoile. Le langage $L = a^+bc^*$ pour le langage infini qui est formé des mots constitué d'une suite non vide de a suivie par un b et d'une suite potentiellement vide de c. L^* est constitué du mot vide, d'un mot de L, et de la concaténation de plusieurs mots de L. $\varepsilon, aabccc, abccaab$ sont des exemples de mots appartenant à L^*.

L'image miroir : L'image miroir d'un mot m notée m^r est le mot m inversé. On a donc : $(m^r)^r = m$. Par exemple l'image miroir du mot abb est le mot bba.

L'image miroir d'un langage L est le langage L^r qui est constitué des images inverse des mots de L.

L^r est donc défini comme suit : $L^r = \{m/m^r \in L\}$

L'union : L'union de deux langages $L1$ et $L2$ notée $L1 \cup L2$ est l'ensemble des mots qui appartiennent à au moins un des deux langages $L1$ et $L2$.

$L1 \cup L2 = \{m/m \in L1 \text{ ou } m \in L2\}$

L'intersection : L'intersection de deux langages $L1$ et $L2$ noté $L1 \cap L2$ est l'ensemble des mots qui appartiennent aux deux langages $L1$ et $L2$.

$L1 \cap L2 = \{m/m \in L1 \text{ et } m \in L2\}$

Le complément : Le complément d'un langage $L1$ est le langage $C(L1)$ constitué de l'ensemble des mots qui appartiennent à A^* et pas à $L1$.

$C(L1) = \{m/m \in A^* \text{ et } m \notin L1\}$

1.1.4.1 Priorité des opérations sur les langages

Comme vu précédemment, les langages peuvent être formés par des opérations telles que l'union ou la concaténation.

Afin d'alléger l'écriture des langages, on introduit les priorités suivantes sur les opérations les plus utilisées :

priorité$(*)$ > priorité$(.)$ > priorité$(\cup)$

Exemple : $0 \cup 10^* \Leftrightarrow (0 \cup (1(0)^*))$

1.1.4.2 Quelques règles d'équivalence

Soient $L1$, $L2$ et $L3$, trois langages définis sur un alphabet A. Voici quelques règles d'équivalence :

- $L1 \cup (L2 \cup L3) = (L1 \cup L2) \cup L3$
- $L1 \cup L2 = L2 \cup L1$
- $L1 \cup \emptyset = L1$
- $L1 \cup L1 = L1$
- $(L1 \cup L2)L3 = L1L3 \cup L2L3$
- $C(L1) \cup L1 = A^*$
- $C(L1) \cap L1 = \emptyset$
- $L1 \cap \emptyset = \emptyset$
- $\emptyset.L1 = L1.\emptyset = \emptyset$
- $\emptyset^* = \varepsilon$

1.2 Grammaires

Un langage peut être décrit par un certain nombre de règles. Pour les langages naturels, le but étant de donner une description précise des règles permettant de construire des phrases correctes d'une langue. Plus généralement, le but d'une grammaire est de donner une description précise des règles permettant de construire tous les mots d'un langage.

Notation : Une grammaire G engendre un langage $L(G)$ et un langage L est engendré par une grammaire $G(L)$.

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

1.2.1 Définition formelle

Une grammaire G est un quadruplet $G = (T, N, S, R)$ telle que :

- T **est l'ensemble du vocabulaire terminal :** c'est-à-dire l'alphabet sur lequel est défini le langage (A).
- N **est l'ensemble du vocabulaire non terminal :** c'est-à-dire l'ensemble des symboles qui n'apparaissent pas dans les mots générés, mais qui sont utilisés au cours de la génération.
- $S \in N$ **est le symbole de départ ou axiome :** c'est à partir de ce symbole non terminal que l'on commencera la génération de mots au moyen des règles de la grammaire.
- R **est l'ensemble des règles de production :** ces règles permettent de générer les mots d'un langage.

1.2.2 Les règles de production

Les règles de production d'une grammaire sont de la forme :

$$u \to v, \text{ avec } u \in (N \cup T)^+ \text{ et } v \in (N \cup T)^*$$

Dans les règles de production on ne peut pas avoir le mot vide qui génère un mot v car la partie gauche de la règle appartient à $(N \cup T)^+$.
On peut avoir des règles du type : $u \to v/w$ pour $u \to v$, $u \to w$.
Lorsqu'on a une règle du type $u \to v$, on dit que v dérive directement de u.
Lorsqu'on a $u \to u1 \to u2 \to u3\ldots \to u4 \to v$: on dit que v dérive de u et on le note $u \overset{*}{\to} v$.
Le langage L généré par une grammaire G est noté $L(G) = \{m/m \in T^* \text{ et } S \overset{*}{\to} m\}$. Cela veut dire que $L(G)$ est formé par tous les mots (formés par l'alphabet terminal) qui peuvent être dérivés à partir de l'axiome.

1.2.3 Exemple de grammaire :

Soit une grammaire $G = (T, N, S, R)$ telle que :
$T = \{a, b\}$
$N = \{S, S'\}$
$R = \{$

$S \to aS/bS'/\varepsilon$

$S' \to bS'/\varepsilon$
$\}$

Exemple de mots générés par cette grammaire :

- ε : A partir de l'axiome S, on applique la règle qui donne le mot vide $S \to \varepsilon$
- a : A partir de l'axiome S, on applique d'abord la règle $S \to aS$ puis la règle $S \to \varepsilon$. On a donc $S \to aS \to a$.
- b : A partir de l'axiome S, on applique d'abord la règle $S \to bS'$ puis la règle $S' \to \varepsilon$. On a donc $S \to bS' \to b$.
- aa, aaa, aa....a, bb, bbb, bb....b, ab, aab, abb, a..ab..b,...

Exemple de mots non générés par cette grammaire :

- ba : ce mot commence par un b, on applique donc d'abord la règle $S \to bS'$. Ensuite on a un a mais aucune règle ne permet d'avoir un a à partir de S'. Donc, le mot ba ne peut pas être généré par cette grammaire.
- aba : ce mot commence par un a, on applique donc d'abord la règle $S \to aS$. Ensuite, on a un b, on applique alors la règle $S \to bS'$. Ensuite, on a un a mais aucune règle ne permet d'avoir un a à partir de S'. Donc, le mot aba ne peut pas être généré par cette grammaire.

Le langage généré par cette grammaire est : a^*b^*.

1.2.4 Types de grammaires

En introduisant des critères sur la forme des règles des grammaires, on obtient des classes de grammaires hiérarchisées, ordonnées par inclusion. La classification de Chomsky, distingue quatre classes de grammaires :

Type 3 : grammaires régulières génèrent des langages réguliers,

Type 2 : grammaires algébriques génèrent des langages algébriques,

Type 1 : grammaire contextuelles génèrent des langages contextuels,

Type 0 : grammaire sans restriction génèrent des langages récursivement énumérables.

1.3 Automates ou reconnaisseurs

Un automate est une machine qui permet de lire un mot m et de dire formellement si m appartient à un langage donné ou non. Un automate est un reconnaisseur qui permet de reconnaitre tous les mots d'un langage L et uniquement les mots du langage L.

1.3.1 Composition d'un automate

Il existe plusieurs types d'automates. Ils ont presque tous une structure commune, mais chacun à sa particularité. Un automate peut être composé de :

Une bande de lecture/écriture : Une bande est une succession de cases. C'est dans les cases de cette bande qu'est écrit le mot à reconnaître. Chaque case pouvant contenir un seul symbole de l'alphabet d'entrée (ou l'alphabet auxiliaire d'écriture, si l'écriture est autorisé sur la bande).

Une tête de lecture/écriture : Une tête de lecture peut lire une case (ou écrire dans une case) à un instant donné. La case sur laquelle se trouve la tête de lecture à un moment donné s'appelle la case courante. La tête peut être déplacée par l'automate pour se positionner sur la case immédiatement à gauche ou à droite de la case courante.

Une mémoire : La mémoire n'est pas toujours présente dans un automate. La mémoire possède un alphabet spécial. Elle est caractérisée par des fonctions d'accès aux éléments et des fonctions de stockage.

Une unité de contrôle : L'unité de contrôle constitue le cœur d'un automate. Elle peut être vue comme un programme qui dicte à l'automate son comportement. Elle est définie par un ensemble fini d'états ainsi que par une fonction de transition qui décrit le passage d'un état à un autre en fonction du contenu de la case courante de la bande de lecture et du contenu de la mémoire. L'unité de contrôle décide aussi de la direction dans laquelle il faut déplacer la tête de lecture et choisit quels symboles stocker dans la mémoire.

1.3.2 Définition formelle d'un automate

Un automate est défini au minimum par les éléments suivants :

- A **est l'alphabet des mots en entrée :** c'est l'alphabet du langage des mots à reconnaitre.
- Q **est un ensemble non vide d'états :** l'automate utilise les états pour déterminer à quel étape il se trouve dans la reconnaissance des mots.
- $I \subseteq Q$ **est un ensemble non vide d'états initiaux :** dans la plupart des automates, on a un seul état initial par lequel l'automate commence la reconnaissance. Il peut y avoir des automates qui possèdent plusieurs états initiaux. Ces automates sont indéterministes (nous définirons dans la section suivante la notion d'indéterminisme).
- $F \subseteq Q$ **est un ensemble d'états finaux :** ces états sont utilisés pour décider de l'appartenance ou non d'un mot au langage.
- δ **est un ensemble de transitions :** les transitions permettent de faire certaines actions à la lecture d'un symbole, comme par exemple le changement d'état, l'écriture, le stockage ...

1.3.3 Déterminisme des automates

Un automate est un reconnaisseur des mots d'un langage donné L. L'objectif de l'automate est de lire un mot en entrée et de dire formellement si ce mot appartient ou non à L. Nous avons une tête de lecture qui examine successivement chaque symbole du mot, et selon la valeur du symbole déclenche la transition appropriée. Si à la fin, l'automate se trouve dans une configuration considérée comme final, le mot appartient à L sinon, quel que soit la configuration autre dans laquelle il se trouve, le mot n'appartient pas à L. Par contre l'automate peut être confronté à deux types de problème :

1. Si l'automate possède plusieurs états initiaux, par quel état initial va-t-il commencer la lecture ? Il doit alors essayer tous les états initiaux ; ce qui va générer plusieurs chemins, et s'il y a au moins un chemin qui se termine dans une configuration considérée comme final, cela veut dire que le mot appartient à L.
2. Si à un moment donné, à la lecture d'un symbole, plusieurs transitions sont possibles. Comment alors savoir quelle transition prendre ? Il faut alors essayer toutes les transitions possibles ce qui va générer plusieurs chemins, et s'il y a au moins un chemin qui se termine dans une configuration considérée comme final, cela veut dire que le mot appartient à L.

Dans les deux cas cité ci-dessus, on se trouve devant un automate indéterministe. Par opposition, on dit qu'un automate est déterministe si à tout instant de la lecture d'un symbole du mot, il existe un seul choix possible.

1.4 Classification de Chomsky

Il y a une forte relation entre les langages, les grammaires et les automates. Cette relation est résumée dans le tableau 1.1 qui représente la classification de Chomsky.

Type du Langages	Langages	Grammaires	Automates
Type 3	Langages réguliers	Grammaires régulières	Automates à états finis
Type 2	Langages algébriques	Grammaires algébriques	Automates à pile
Type 1	Langages contextuels	Grammaires contextuelles	Machines de Turing à bornes linéaires
Type 0	Langages récursivement énumérables	Grammaires sans restriction	Machines de Turing

TABLE 1.1: Classification de Chomsky

Dans les quatre chapitres suivants, nous détaillerons chaque type de langage, avec le type de grammaire qui permet de générer les mot du langage et le type d'automate qui permet la reconnaissance des mots du langage.

1.5 Exercices

Exercice 1 : Soit l'alphabet $A = \{0, 1\}$

1. Étant donnés les mots $u = 11$ et $v = 100$, écrire les mots uv, u^3, u^2v^2 et $(uv)^2$.
2. Soit $E1$ l'ensemble de tous les mots m définis sur A tel que $\mid m \mid = 3$. Que vaut $E1$?
3. Soit $E2$ l'ensemble de tous les mots m définis sur A tel que $\mid m \mid < 5$ et $\mid m \mid mod\ 2 = 0$. Que vaut $E2$?
4. Soit le langage $L = 1^+0^* \cup 0^*1^*$. Soit $E3$ l'ensemble de tous les mots m tel que $m \in L$ et $\mid m \mid = 3$. Que vaut $E3$?
5. Soit les langages $L1 = 01^+$ et $L2 = 10^+$. Définir et essayer de simplifier les langages $L1L2$, $(L2)^*L1$ et $L1 \cup C(L1 \cap L2)$.

Exercice 2 : Soit l'alphabet $A = \{a, b, c\}$.

1. Donner $L1$ le langage des mots définis sur A qui commencent par un a.
2. Donner $L2$ le langage des mots définis sur A qui se terminent par un a.
3. Donner $L3$ le langage des mots définis sur A qui contiennent au moins un a.
4. Donner $L4$ le langage des mots définis sur A qui contiennent au moins deux fois la séquence aab.
5. Donner $L5$ le langage des mots définis sur A qui ne commencent pas par un a.

Exercice 3 :

1. Comment représenter le langage régulier constitué d'aucun mot ?
   a) ε
   b) $\{\varepsilon\}$
   c) $\emptyset$
   d) $\{\emptyset\}$
2. Soit l'alphabet $A = \{a, b\}$, parmi ces langages, lequel représente le langage des mots qui commence par un b.
   a) $(ba^*)^*$
   b) $b(a^*b^*)^*$
   c) $(ba^*)^+$
   d) $(ba^*b^*)^+$
3. Quels sont les équivalences correctes
   a) $(ab)^*a = a(ba)^*$
   b) $(a^*b^*)^* = (a \cup b)^*$
   c) $(ab^*)^* = a(a^*b^*)^*$
   d) $a^* \cup \varepsilon = a^+$

4. Quel est la relation correct dans les langages ?
   a) Type 3 $\in$ Type 2
   b) Type 2 $\in$ Type 3
   c) Type 3 $\subset$ Type 2
   d) Type 2 $\subset$ Type 3

Exercice 4 : Remplir la grille de la figure 1.2 en respectant les langages des lignes et des colonnes. Chaque ligne contient un mot du langage décrit pour cette ligne et la même chose pour les colonnes. Chaque case contient un seul symbole. Les mots doivent donc être de longueurs 10 ou 9 (Pour la dernière ligne et la dernière colonne).

	L11={$a^n da^n c$; n>1}	L12={$a^n bcc$; n>1}	L13={$a^{(n+1)} ca^n$; n>0}	L14={$c^m a^n$; m,n>1}	L15=dbadd$^+$c$^+$a	L16=bdadb$^+$(ca)$^+$	L17={$b^n dc^n dcaba$; n>1}	L18=b$^+$ad$^+$b$^+$a	L19={$b^n d^n a^n (b^*d^*)^* a$; n >1}	L20={m/m$\in$(a*b*d*)* et $\lvert m\rvert_a = \lvert m\rvert_b = \lvert m\rvert_d$}
L1={m/m$\in$a*cdb* et |m| mod 2 =0}										
L2= a*c(b*d*)*d										
L3 = {m/m$\in$ (a*b*c*d*)* et $\lvert m\rvert_c = \lvert m\rvert_b$}										
L4={m/m$\in$a*(b*c*d*)*d et $\lvert m\rvert_a + \lvert m\rvert_c = \lvert m\rvert_b + \lvert m\rvert_d$}										
L5= a*daa(a*b*c*d*)*bab										
L6=a$^+$c$^+$bda$^+$	a									
L7={a*c^{n}d^n; n>0}										
L8=(a*b*c*d*)*caab$^+$										
L9={aca^{n}c^{n}b$^{(n+1)}$a et n>1}										
L10=c(a U b U c)a*										

FIGURE 1.2: Grille

Chapitre 2

Langages réguliers

Dans ce chapitre, nous allons étudier les langages de type 3 appelés aussi langages réguliers. Nous allons voir comment sont constitués ces langages, quelles sont les spécificités des grammaires qui permettent de générer les langages réguliers et quels sont les automates qui permettent de reconnaitre les mots qui appartiennent à un langage régulier.

2.1 Langages réguliers

Un langage est régulier, si et seulement si, il existe une grammaire régulière générant ce langage.

2.1.1 Exemples de langages réguliers

Étant donné un alphabet A,
- $\emptyset$(l'ensemble vide) est un langage régulier sur A.
- $\{\varepsilon\}$ est un langage régulier sur A.
- $\{a\}$ est un langage régulier sur A pour tout $a \in A$.
- Tout langage fini est régulier.
- L'ensemble de tous les mots A^* est régulier.
- a^*b^* est régulier.

2.1.2 Propriétés de fermeture des langages réguliers

Si $L1$ et $L2$ sont des langages réguliers sur un alphabet A, alors les langages suivants sont des langages réguliers :
- $L1 \cup L2$ (L'union)
- $L1L2$ (La concaténation)
- $L1^*$ (L'étoile)
- $L1 \cap L2$ (L'intersection)
- $C(L1)$ (Le complément)

2.2 Grammaires régulières

Les grammaires régulières permettent de générer des langages réguliers.

2.2.1 Définition formelle

Il existe deux types de grammaires régulières équivalentes :

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

$$S \to aS' \text{ ou } S \to a \text{ ou } S \to \varepsilon.\ \forall (S, S') \in N \text{ et } \forall a \in T$$

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

$$S \to S'a \text{ ou } S \to a \text{ ou } S \to \varepsilon.\ \forall (S, S') \in N \text{ et } \forall a \in T$$

Remarque : Dans une grammaire régulière $G = (T, N, S, R)$, on ne peut pas avoir une règle où le symbole non terminal se trouve à gauche du symbole terminal et une autre règle où le symbole non terminal se trouve à droite du symbole terminal comme par exemple : $S \to aS/Sb$.

2.2.2 Exemple de grammaire régulière

Soit un langage $L = a^*b^*$ défini sur l'alphabet $A = \{a, b\}$.

Grammaire régulière à droite : $G_d(L)$ est une grammaire régulière à droite :
$G_d(L) = (T, N_d, S_d, R_d)$
$T = \{a, b\}$
$N_d = \{S_d, S'_d\}$
$R_d = \{$

$S_d \to aS_d/bS'_d/\varepsilon,$

$S'_d \to bS'_d/\varepsilon$
$\}$

Exemple de mot appartenant au langage $L = a^*b^*$: aab
Pour générer ce mot à partir de $G_d(L)$ on applique d'abord la règle $S_d \to aS_d$ pour générer le premier a. Pour générer le deuxième a, on applique la même règle $S_d \to aS_d$. Ensuite, on a un b, on applique alors la règle $S_d \to bS'_d$. Pour finir, on applique la règle $S'_d \to \varepsilon$. On a donc $S_d \to aS_d \to aaS_d \to aabS'_d \to aab$

Exemple de mot n'appartenant pas au langage $L = a^*b^*$: ba
Ce mot commence par un b, on applique d'abord la règle $S_d \to bS'_d$ pour générer le premier b. Ensuite on a un a mais aucune règle ne permet de générer un a à partir de S'_d. Donc, le mot ba ne peut pas être généré par cette grammaire.

Grammaire régulière à gauche : $G_g(L)$ est une grammaire régulière à gauche :
$G_g(L) = (T, N_g, S_g, R_g)$
$T = \{a, b\}$
$N_g = \{S_g, S'_g\}$
$R_g = \{$
$S_g \rightarrow S_g b / S'_g a / \varepsilon,$
$S'_g \rightarrow S'_g a / \varepsilon$
$\}$

Exemple de mot appartenant au langage $L = a^*b^*$: aab
Ce mot se termine par un b, on applique d'abord la règle $S_g \rightarrow S_g b$. Ensuite, on a un a, on applique alors la règle $S_g \rightarrow S'_g a$. Ensuite, on a un autre a, on applique alors la règle $S'_g \rightarrow S'_g a$. Le mot est formé après application de la règle $S'_g \rightarrow \varepsilon$. On a donc $S_g \rightarrow S_g b \rightarrow S'_g ab \rightarrow S'_g aab \rightarrow aab$

Exemple de mot n'appartenant pas au langage $L = a^*b^*$: ba
ce mot se termine par un a, on applique d'abord la règle $S_g \rightarrow S'_g a$. Ensuite on a un b mais aucune règle ne permet de générer un b à partir de S'_g. Donc, le mot ba ne peut pas être généré par cette grammaire.

Remarque : On voit bien à travers cet exemple la différence entre une grammaire régulière à droite et une grammaire régulière à gauche. Le principe dans la grammaire régulière à droite est de faire la génération des mots du langage du début vers la fin, contrairement à la grammaire régulière à gauche où le raisonnement est inverse : la génération se fait de la fin vers le début des mots.

2.3 Automates à états finis

Un automate est une machine qui permet de lire un mot m et de dire formellement si m appartient à un langage donné.

Un automate est un reconnaisseur qui permet de reconnaître tous les mots d'un langage donné.

Les automates qui permettent de reconnaitre les langages réguliers sont les automates à états finis.

Remarque : Pour un langage régulier L, on peut avoir plusieurs automates à états finis qui reconnaissent les mots de L. Par contre, un automate à états finis ne peut reconnaitre les mots que d'un seul langage.

2.3.1 Composition d'un automate à états finis

Un automate à états finis est composé de :

Une bande de lecture : La bande est composée d'une succession de cases. C'est dans les cases de cette bande qu'est écrit le mot à reconnaître. Chaque case pouvant contenir un seul symbole de l'alphabet d'entrée.

Une tête de lecture : La tête de lecture lit à un instant donné la case courante avant de se déplacer sur la case suivante. La case courante est la case sur laquelle est positionnée la tête de lecture. La tête de lecture est initialement positionnée sur le premier symbole du mot et se déplace toujours d'une seule case à la fois vers la droite.

Une unité de contrôle : L'unité de contrôle constitue le cœur d'un automate. Elle peut être vue comme un programme qui dicte à l'automate son comportement. Elle est définie par un ensemble fini d'états ainsi que par une fonction de transition qui décrit le passage d'un état à un autre en fonction du contenu de la case courante et de l'état courant. A chaque fois qu'une transition s'exécute, la tête de lecture se déplace d'une seule case vers la droite.

2.3.2 Définition formelle d'un automate à états finis

Un automate à états finis AEF est un quintuplet $AEF = (A, Q, I, F, \delta)$:

A **est l'alphabet des mots en entrée :** c'est l'alphabet du langage des mots à reconnaitre.

Q **est un ensemble non vide d'états :** l'automate à états finis utilise les états pour déterminer à quel étape il se trouve dans la reconnaissance des mots.

$I \subseteq Q$ **est un ensemble non vide d'états initiaux :** dans la plupart des automates à états finis, il n'y a qu'un seul état initial par lequel l'automate à états finis commence la reconnaissance. Il peut y avoir des automates à états finis qui possèdent plusieurs états initiaux. Ces automates à états finis sont des automates à états finis indéterministes.

$F \subseteq Q$ **est un ensemble d'états finaux :** l'ensemble des états finaux est utilisé pour la reconnaissance des mots. Si la reconnaissance se termine dans un état final, le mot est reconnu comme appartenant au langage. Sinon, le mot n'est pas reconnu par l'automate à états finis comme appartenant au langage.

δ **est un ensemble de transitions :** les transitions permettent de faire les changements d'états à la lecture d'un symbole.

2.3.3 Procédure de reconnaissance dans les automates à états finis

La reconnaissance des mots par l'automate à états finis commence par un état initial. A chaque symbole lu, l'automate à états finis peut changer l'état dans lequel il se trouve et la tête de lecture se déplace d'une case à droite.

Un mot est reconnu par un automate à états finis si est seulement si à la fin de la lecture du mot, l'automate à états finis se trouve dans un état final.

2.3.4 Représentation des automates à états finis

Nous allons voir dans ce qui suit trois façons de représenter les automates à états finis :

— Représentation formelle
— Représentation graphique
— Représentation tabulaire

Ces trois types de représentation sont équivalentes, elles contiennent exactement les mêmes informations.

2.3.4.1 Représentation formelle

Un automate à états finis AEF peut être représenté d'une façon formelle en définissant tous les ensembles qui définissent un automate à états finis $AEF = (A, Q, I, F, \delta)$.

Les transitions sont écrites sous la forme suivante : (état source, symbole lu) $\rightarrow$ (état destination)

Exemple : Soit un langage $L = a^*b^*$ défini sur l'alphabet $A = \{a, b\}$. Nous allons donner une représentation formelle de l'automate à états finis $AEF(L)$.
$AEF(L) = \{A, Q, I, F, \delta)$
$A = \{a, b\}$
$Q = \{e_0, e_1\}$
$I = \{e_0\}$
$F = \{e_0, e_1\}$
$\delta = \{$

$(e_0, a) \rightarrow (e_0)$,

$(e_0, b) \rightarrow (e_1)$,

$(e_1, b) \rightarrow (e_1)$
$\}$

Exemple de mot appartenant au langage $L = a^*b^*$: aab
Au début, l'automate se trouve à l'état initial e_0 (il y a un seul état initial) et la tête de lecture sur le premier symbole du mot à savoir a. Il existe une transition qui va de cette configuration vers l'état e_0 ($(e_0, a) \rightarrow (e_0)$). La tête de lecture se situe alors sur le deuxième symbole du mot à savoir un a, et l'automate est resté dans l'état e_0. Il va alors faire la même chose que pour le premier symbole à savoir $(e_0, a) \rightarrow (e_0)$. La tête de lecture se situe alors sur le troisième symbole du mot à savoir un b, et l'automate est resté dans l'état e_0. Il existe une transition qui va de cette configuration vers l'état e_1 ($(e_0, b) \rightarrow (e_1)$). Le mot est terminé et l'automate se trouve dans l'état e_1 qui est un état final, le mot est alors reconnu par l'automate.

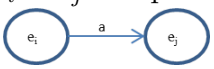
Exemple de mot n'appartenant pas au langage $L = a^*b^*$: ba
Au début, l'automate se trouve à l'état initial e_0 et la tête de lecture sur le premier symbole du mot à savoir b. Il existe une transition qui va de cette configuration vers

l'état e_1 ($(e_0, b) \rightarrow (e_1)$). La tête de lecture se situe alors sur le deuxième symbole du mot à savoir un a, et l'automate est à l'état e_1. A partir de cette configuration, il n'existe pas de transition possible. Donc ce mot n'est pas reconnu par l'automate.

2.3.4.2 Représentation graphique

Un automate à états finis peut être représenté par un graphe dans lequel :

- Les nœuds du graphe sont les états 

- Les arcs sont les transitions possibles avec la fonction de transition
- La pondération de chaque arc est le symbole ou les symboles (séparé par des virgules) qui peuvent être lus lors du changement d'état de e_i à e_j. Le passage d'un état à un autre se fait avec la lecture d'un seul symbole. 

- Les états initiaux sont identifiés par une flèche entrante.
- Les états finaux sont identifiés par un double cercle. 

Exemple : Soit un langage $L = a^*b^*$ défini sur l'alphabet $A = \{a, b\}$. Nous donnons dans la figure 2.1 une représentation graphique de l'automate à états finis $AEF(L)$.

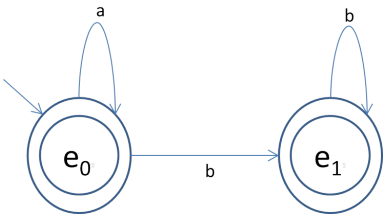


FIGURE 2.1: AEF(L)

Exemple de mot appartenant au langage $L = a^*b^*$: aab
Au début, l'automate se trouve à l'état initial e_0 (l'état initial identifié par une flèche entrante) et la tête de lecture sur le premier symbole du mot à savoir a. Il existe une transition qui va de cette configuration vers l'état e_0 (il existe une boucle dans e_0 avec le symbole a). La tête de lecture se situe alors sur le deuxième symbole du mot à savoir un a, et l'automate reste à l'état e_0. Il boucle à nouveau sur e_0 avec le a. La tête de lecture se situe alors sur le troisième symbole du mot à savoir un b, et l'automate reste à l'état e_0. Il existe une transition qui va de cette configuration vers l'état e_1 (il existe un arc pondéré par un b qui va de e_0 vers e_1). Le mot est terminé et l'automate se trouve à l'état e_1 qui est un état final (identifié par un double cercle), le mot est alors reconnu par l'automate.

Exemple de mot n'appartenant pas au langage $L = a^*b^*$: ba
Au début, l'automate se trouve à l'état initial e_0 et la tête de lecture sur le premier symbole du mot à savoir b. Il existe une transition qui va de cette configuration vers l'état e_1 (il existe un arc pondéré par un b qui va de e_0 vers e_1). La tête de lecture se situe alors sur le deuxième symbole du mot à savoir un a, et l'automate est à l'état e_1. A partir de cette configuration, il n'existe pas de transition possible (il n'existe aucun arc sortant de e_1 avec le symbole a). Donc ce mot n'est pas reconnu par l'automate.

2.3.4.3 Représentation tabulaire

La représentation tabulaire s'appuie sur une matrice où chaque ligne représente un état source, chaque colonne représente un symbole de l'alphabet et chaque case de la matrice représente le ou les états destinations des transitions.
Les états initiaux sont marqués par des flèches entrantes et les états finaux par des flèches sortantes.

Exemple : Soit un langage $L = a^*b^*$ défini sur l'alphabet $A = \{a, b\}$. Nous donnons dans la table 2.1 une représentation tabulaire de l'automate à états finis $AEF(L)$.

	a	b
$\overrightarrow{\leftarrow} e_0$	e_0	e_1
$\leftarrow e_1$		e_1

TABLE 2.1: AEF(L)

Vous pouvez vous exercer à faire le déroulement d'exemples de mots qui appartiennent et de mots qui n'appartiennent pas au langage L.

2.3.5 Caractéristiques des automates à états finis

2.3.5.1 Automates à états finis complets / incomplets

Un automate est complet si sa fonction de transition δ est définie pour tout état de l'automate et tout symbole de l'alphabet.

Pour tout automate à états finis incomplet, il existe un automate à états finis complet équivalent.

Pour rendre un automate à états finis complet, il peut être nécessaire de rajouter un état supplémentaire appelé état puits (e_p), vers lequel vont toutes les transitions impossibles. L'état puits n'est évidemment pas un état final. Donc, si la reconnaissance d'un mot se termine dans un état puits, on peut conclure que ce mot ne fait pas partie du langage.

Exemples : Soit un langage $L = a^*b^*$ défini sur l'alphabet $A = \{a, b\}$. Nous allons donner une représentation graphique de l'automate à états finis complet de L (figure 2.2).

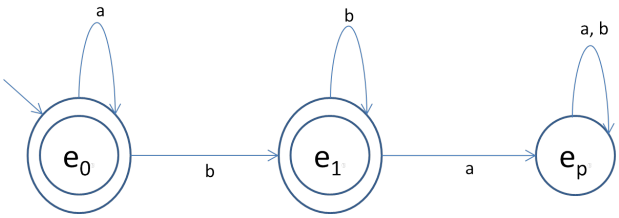


FIGURE 2.2: AEF(L)

Pour faire l'automate complet de L, il a été nécessaire de rajouter un nouvel état (e_p).

Soit un langage $L' = (a^*b^*)^*$ défini sur l'alphabet $A = \{a, b\}$. Nous allons donner une représentation graphique de l'automate à états finis complet de L' (figure 2.3).

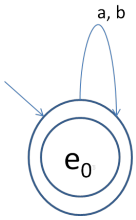


FIGURE 2.3: AEF(L')

Contrairement à l'automate de L, l'automate de L' est déjà complet sans l'ajout d'un état puits.

2.3.5.2 Automates à états finis sans états inaccessibles / avec des états inaccessibles

Un automate à états finis est un automate sans états inaccessibles si tous les états de l'automate sont accessibles à partir de l'ensemble des états initiaux. Un état e_i est accessible à partir de l'ensemble des états initiaux s'il existe au moins un chemin menant d'un état initial vers l'état e_i dans le graphe de l'automate. Plus formellement, cela implique qu'il existe une suite de transitions menant d'un état initial vers l'état e_i.

Pour tout automate à états finis avec des états inaccessibles, il existe un automate à états finis sans états inaccessibles équivalent.

Pour passer d'un automate avec des états inaccessibles à un automate sans états inaccessibles, il suffit de supprimer tous les états inaccessibles ainsi que les transitions entrantes et sortantes de ces états.

Exemple : Soit l'automate à états finis de la figure 2.4 :
Dans l'exemple, l'état e_0 est un état initial et donc forcément e_0 est un état accessible. e_1 est accessible à partir de e_0 avec un b. Par contre, e_3 et e_4 ne sont pas accessibles à partir de e_0.

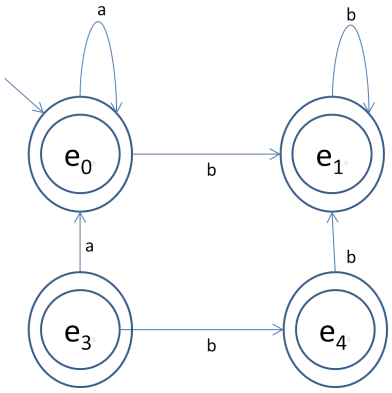


FIGURE 2.4: AEF avec des états inaccessibles

Pour passer à l'automate à états finis sans états inaccessibles, il suffit de supprimer les états e_3 et e_4. On retrouve l'automate à états finis de la figure 2.1 reconnaissant les mots du langage $L = a^*b^*$.

2.3.5.3 Automates à états finis déterministes / indéterministes

Un automate à états finis est dit indéterministe si au moins une des conditions suivantes est vraie :

- Il a plusieurs états initiaux.
- Étant donné un état $e_i \in Q$ et un symbole $a \in A$, il existe plusieurs transitions possibles.

Lorsqu'on a un automate à états finis indéterministe, pour savoir si un mot appartient au langage reconnu par l'automate, il est nécessaire de prendre tous les chemins possibles dans l'automate à chaque fois qu'il y en a plusieurs, et s'il y a au moins un chemin qui finit dans un état final, alors le mot est reconnu par l'automate à états finis comme faisant parti du langage. Le temps de calcul peut rapidement augmenter surtout si l'automate à états finis est grand avec des choix multiples dans plusieurs états.

Pour tout automate à états finis indéterministe, il existe un automate à états finis déterministe équivalent.

Il existe un algorithme (Algorithme 1) qui permet de transformer un automate à états finis indéterministe ($AEFI$) en un automate à états finis déterministe ($AEFD$) équivalent.

Pour dérouler cet algorithme, il est conseillé d'utiliser la représentation tabulaire qui permet de réduire le risque d'erreurs.

Algorithme 1 : rendDéterministe

Entrée : $AEFI = (A, Q, I, F, \delta)$

Sortie : $AEFD = (A, Q_d, e_{0d}, F_d, \delta_d)$ {tel que $L(AEFI) = L(AEFD)$ }

$e_{0d} \leftarrow I$; {e_{0d} est un état qui regroupe tous les états initiaux de $AEFI$}

$Q_d \leftarrow e_{0d}$; {initialement Q_d contient l'état initial e_{0d}}

$vus \leftarrow \emptyset$; {vus est l'ensemble des états de Q_d qui ont été traités. Initialement, aucun état n'est traité}

$\delta_d \leftarrow \emptyset$; {initialisation de δ_d}

Tant que $Q_d \neq vus$ **faire**

soit $e_{id} \in Q_d$ **et** $e_{id} \notin vus$; {prendre un état de Q_d pas encore traité}

Pour tout $a \in A$ **faire**

$e_{kd} \leftarrow \{e_k / \exists e_i \in e_{id}, (e_i, a) \rightarrow (e_k) \in \delta\}$; {e_{kd} est l'état formé par l'ensemble des états e_k qui sont les destinataires des transitions sortants avec le symbole a des états e_i formant e_{id}}

$Q_d \leftarrow Q_d \cup \{e_{kd}\}$; {e_{kd} est ajouté à Q_d s'il n'existe pas déjà}

$\delta_d \leftarrow \delta_d \cup \{(e_{id}, a) \rightarrow (e_{kd})\}$; {une nouvelle transition est ajoutée à δ_d}

Fin Pour

$vus \leftarrow vus \cup \{e_{id}\}$; {ajouter e_{id} à l'ensemble des états traités}

Fin Tant que

$F_d \leftarrow \{e_{id} \in Q_d / e_{id} \cap F \neq \emptyset\}$; {l'ensemble des états finaux est composé des états qui contiennent au moins un état final de F}

Exemple : Soit l'automate à états finis indéterministe ci-dessous reconnaissant les mots du langage $L = a^+c^+ \cup b^+c^+$.

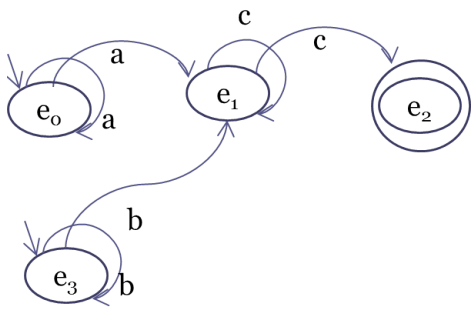


FIGURE 2.5: AEFI du langage $L = a^+c^+ \cup b^+c^+$

Ci-dessous, la représentation tabulaire de cet automate $AEFI$ qui permet de faciliter le déroulement de l'algorithme 1.

On a deux états initiaux (deux flèches entrantes dans la représentation tabulaire) et avec le même symbole et à partir d'un même état, on peut aller vers plusieurs états (des cases qui contiennent deux états dans la représentation tabulaire). Il suffisait qu'une des

	a	b	c
$\rightarrow e_0$	$\{e_0, e_1\}$		
e_1			$\{e_1, e_2\}$
$\leftarrow e_2$			
$\rightarrow e_3$		$\{e_1, e_3\}$	

TABLE 2.2: La représentation tabulaire de l'AEFI du langage $L = a^+c^+ \cup b^+c^+$

deux conditions soit vérifiée pour que l'automate soit indéterministe. Dans cet exemple, on a les deux conditions.

Déroulement de l'algorithme 1 : L'état initial e_{0d} regroupe tous les états initiaux.

	a	b	c
$\rightarrow e_{0d} = \{e_0, e_3\}$			

A partir de e_0 et e_3 (on regarde les lignes de e_0 et de e_3 de la représentation tabulaire de $AEFI$), avec le symbole a (on regarde la colonne de a), on va vers $\{e_0, e_1\}$ qui représente un nouvelle état e_{1d}.

	a	b	c
$\rightarrow e_{0d} = \{e_0, e_3\}$	e_{1d}		
$e_{1d} = \{e_0, e_1\}$			

A partir de e_0 et e_3, avec le symbole b, on va vers $\{e_1, e_3\}$ qui représente un nouvelle état e_{2d}. Avec un c, on va vers aucun état, on laisse alors la case vide.

	a	b	c
$\rightarrow e_{0d} = \{e_0, e_3\}$	e_{1d}	e_{2d}	
$e_{1d} = \{e_0, e_1\}$			
$e_{2d} = \{e_1, e_3\}$			

On continue ensuite avec les états qui sont ajouter au fur et à mesure pour arriver à l'automate à états finis de la figure 2.3 :

On remarque facilement que l'automate à états finis résultant (Table 2.3) est un automate déterministe, car on a une seule flèche entrante et au maximum un état par case du tableau.

	a	b	c
$\rightarrow e_{0d} = \{e_0, e_3\}$	e_{1d}	e_{2d}	
$e_{1d} = \{e_0, e_1\}$	e_{1d}		e_{3d}
$e_{2d} = \{e_1, e_3\}$		e_{2d}	e_{3d}
$\leftarrow e_{3d} = \{e_1, e_2\}$			e_{3d}

TABLE 2.3: La représentation tabulaire de l'AEFD du langage $L = a^+c^+ \cup b^+c^+$

2.3.5.4 Automates minimaux / non minimaux

Tout langage régulier est reconnu par au moins un automate à états finis déterministe et complet.

Il en existe un ayant un nombre minimal d'états. Cet automate est unique : c'est l'automate minimal du langage. L'intérêt d'un automate minimal est que le coût en espace de la représentation est minimisé.

Pour tout automate à états finis non minimal, il existe un automate à états finis minimal équivalent.

Il existe un algorithme (Algorithme 2) qui permet de passer d'un automate à états finis complet, sans états inaccessibles et déterministe ($AEFD$) vers un automate à états finis minimal ($AEFM$). L'idée est d'essayer de faire du regroupement d'état, en fusionnant plusieurs états.

Algorithme 2 : rendminimal

Entrée : $AEFD = (A, Q, e_0, F, \delta)$ {un automate à états finis déterministe, complet et sans états inaccessibles}

Sortie : $AEFM = (A, Q_m, e_{0m}, F_m, \delta_m)$ {tel que $L(AEFD) = L(AEFM)$ }

Séparer les états en deux groupes si c'est possible : $G1$ contenant les états finaux et $G2$ contenant les états non finaux ;

Répéter

Si il existe un symbole a et deux états $e_i \in Q$ et $e_j \in Q$ d'un même groupe tel que $\delta(e_i, a)$ et $\delta(e_j, a)$ n'appartiennent pas au même groupe **alors**

séparer e_i et e_j en deux groupes différents ; {On laisse dans le même groupe tous les états qui donnent un état d'arrivée dans le même groupe}

Fin Si

Jusqu'à il n'y a plus de groupes à séparer

Les groupes sont les états de Q_m ;

$e_{0m} \leftarrow Gi \in Q_m / e_0 \in Gi$; {L'état initial est le groupe qui contient l'état initial de $AEFD$}

F_m sont tous les groupes qui dérivent de $G1$;

δ_m est l'ensemble δ en remplaçant les états de Q par le groupe qui les contient dans Q_m et en supprimant les transitions redondantes ;

Pour la condition d'arrêt de la boucle de l'algorithme 2, on peut distinguer trois cas de figure :

1. La dernière itération n'a pas changé les groupes.
2. Il existe un seul groupe. Ce groupe ne peut pas être divisé parce que toutes les transitions vont vers des états de ce même groupe. Il existe deux cas de figure :
   (a) Tous les états sont des états finaux : vu que l'automate est complet, cela revient à reconnaitre tous les mots définis sur l'alphabet A. Le langage est alors A^*.
   (b) Aucun état n'est un état final : vu que l'automate est complet, cela revient à ne reconnaitre aucun mot défini sur l'alphabet A. Le langage est alors $\emptyset$.
3. Tous les groupes sont constitués d'un seul état. Cela veut dire que l'automate était minimal avant l'exécution de l'algorithme.

Pour dérouler cet algorithme, il est aussi pratique d'utiliser la représentation tabulaire pour réduire le risque d'erreurs.

Exemples : Soit l'automate à états finis de la figure 2.6 à minimiser.

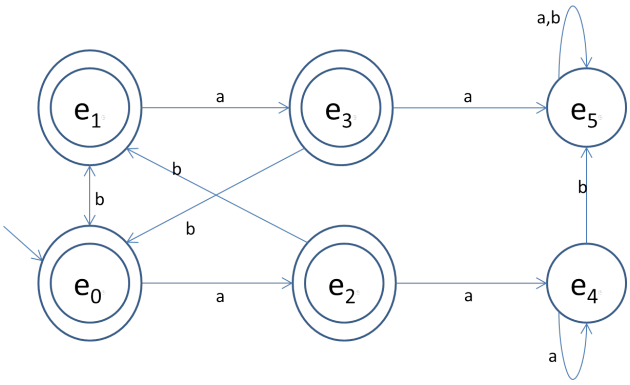


FIGURE 2.6: AEF à minimiser

On remarque qu'il n'existe pas d'état inaccessible, on passe alors à la représentation tabulaire.

	a	b
$\overrightarrow{\leftarrow}\ e_0$	e_2	e_1
$\leftarrow e_1$	e_3	e_0
$\leftarrow e_2$	e_4	e_1
$\leftarrow e_3$	e_5	e_0
e_4	e_4	e_5
e_5	e_5	e_5

TABLE 2.4: Représentation tabulaire de l'AEF à minimiser

Avec la représentation tabulaire, on voit au premier coup d'œil que l'automate est déterministe (une seule flèche entrante et maximum un état pas case) et complet (aucune case vide).

On peut alors commencer le déroulement de l'algorithme de minimisation.

- $G1 = \{e_0, e_1, e_2, e_3\}, G2 = \{e_4, e_5\}$ // initialement $G1$ contient les état finaux et $G2$ le reste des états.
- $G1\_1 = \{e_0, e_1\}, G1\_2 = \{e_2, e_3\}, G2 = \{e_4, e_5\}$ // ici, le premier groupe a été divisé en deux groupes à cause des transitions avec le symbole a, les transitions de e_0 et e_1 avec le a partent vers $G1$ alors que les transitions de e_2 et e_3 avec le a partent vers $G2$. Donc l'ancien $G1$ a été divisé en deux groupes $G1\_1$ et $G1\_2$ alors que $G2$ ne s'est pas divisé.
- $G1\_1 = \{e_0, e_1\}, G1\_2 = \{e_2, e_3\}, G2 = \{e_4, e_5\}$ // aucun changement lors de la dernière itération, on sort de la boucle.

L'état initial est maintenant $G1\_1$ (le groupe de e_0).

Les états finaux sont $G1\_1$ et $G1\_2$ (les groupes qui dérivent de $G1$).

On obtient alors l'automate à états finis de la figure 2.7 :

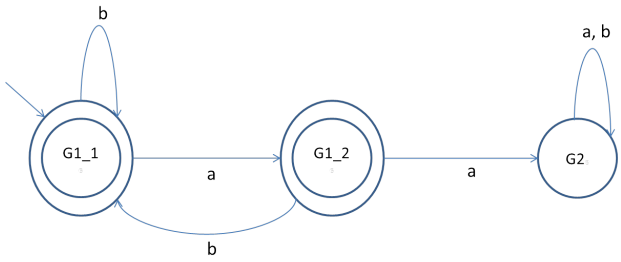


FIGURE 2.7: AEF minimisé

Nous allons voir maintenant un second exemple avec l'automate à états finis suivant :

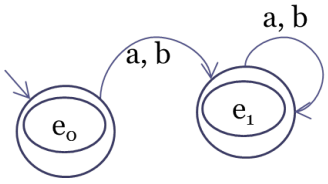


FIGURE 2.8: Exemple 2 d'AEF à minimiser

On remarque qu'il n'existe pas d'état inaccessible, on passe alors à la représentation tabulaire.

Avec la représentation tabulaire, on remarque sans effort que l'automate est déterministe et complet. On peut alors commencer le déroulement de l'algorithme de minimisation.
$G1 = \{e_0, e_1\}$

	a	b
$\overrightarrow{\leftarrow} e_0$	e_1	e_1
$\leftarrow e_1$	e_1	e_1

TABLE 2.5: Représentation tabulaire de l'automate de la figure 2.8

Fin de l'algorithme car il y a un seul état, on ne peut pas réduire plus, cet état est l'état initial et l'état final. On obtient alors l'automate suivant :

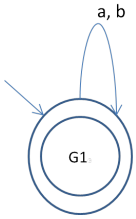


FIGURE 2.9: AEF minimisé

2.4 Conclusion

Les langages réguliers sont générés par des grammaires régulières et les mots d'un langage régulier peuvent être reconnus par un automate à états finis.

Il existe deux sortes de grammaires régulières équivalentes : les grammaires régulières à droite et les grammaires régulières à gauche. Pour tout langage régulier, il est possible d'avoir les deux sortes de grammaires.

Pour un langage régulier donné, il peut y avoir plusieurs automates à états finis déterministes et / ou indéterministes qui reconnaissent les mots du langage. Il existe un algorithme qui permet de passer de n'importe quel automate à états finis indéterministe à un automate à états finis déterministe équivalent. Il existe aussi un algorithme qui permet de passer à l'automate à états finis minimal qui est unique à partir d'un automate à états finis déterministe, complet et sans états inaccessibles.

2.5 Exercices

Exercice 1 : Donner une grammaire régulière à droite, une grammaire régulière à gauche un automate à états finis pour chacun des langages suivants :

- $L1 = a^+b^+$
- $L2 = a^*cb^*$
- $L3 = (ab^*)^+$
- $L4 = a^*b^*c^*$
- $L5 = \{m/m \in (a^*b^*)^*et \mid m \mid \text{mod } 3 = 0\}$
- $L6 = abc^*$
- $L7 = abc^+$
- $L8 = (a^*b^*)(abc^* \cup c^*)$
- $L9 = (ab \cup ccb)(bac^+ \cup cc)$
- $L10 = \{m/m \in (a^*b^*)^*et \mid m \mid_a \text{mod } 3 = 0\}$
- $L11 = C(L1)$

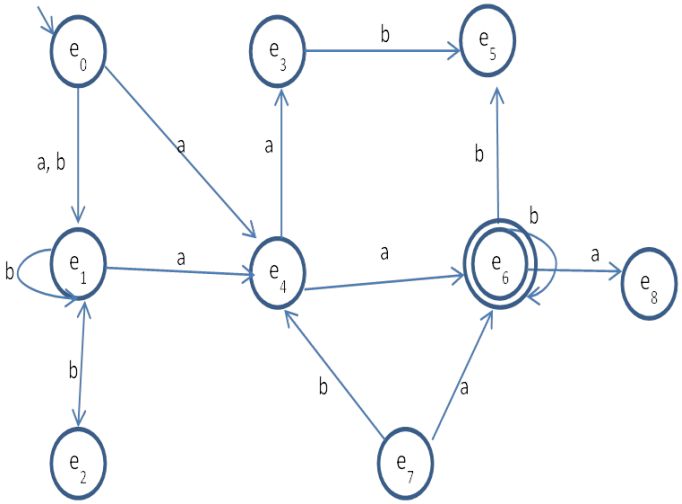
Exercice 2 : Quelles sont les langages générés par ces grammaires ?

1. $G = (T, N, S, R)$
   $T = \{a, b, c\}$
   $N = \{S, S', S''\}$
   $R = \{S \to aS/bS'/cS''/\varepsilon, S' \to bS'/cS''/\varepsilon, S'' \to cS''/\varepsilon\}$
2. $G = (T, N, S', R)$
   $T = \{a, b, c\}$
   $N = \{S, S'\}$
   $R = \{S \to aS/\varepsilon, S' \to aS'/bS'/\varepsilon\}$
3. $G = (T, N, S, R)$
   $T = \{a, b, c\}$
   $N = \{S, S', S'', S''', S'''', S'''''\}$
   $R = \{$
   $S \to aS/bS'/cS''/aS'''$
   $S' \to bS'/cS''''$
   $S'' \to cS''/aS'''$
   $S''' \to bS''''$
   $S'''' \to b$
   $S''''' \to c$
   $\}$
4. $G = (T, N, S, R)$
   $T = \{a, b, c\}$
   $N = \{S, S', S''\}$
   $R = \{S \to aS/bS'/cS''/c, S' \to aS/bS'', S'' \to cS''/cS\}$

Exercice 3 : Soit la grammaire régulière à droite suivante $G = (T, N, S, R)$ avec :
$T = \{a, b, c\}$
$N = \{S, S', S''\}$
$R = \{S \to aS/aS', S' \to bS/cS'', S'' \to cS''/\varepsilon\}$

1. Les mots *aabac*, *aaacc*, *aabbccc*, *ab* appartiennent-ils à $L(G)$.
2. Quel est le langage généré par cette grammaire ?
3. Donner une grammaire régulière à gauche pour ce langage.

Exercice 4 : Minimiser l'automate à états finis suivant :



Chapitre 3

Langages algébriques

Les langages algébriques sont les langages de type 2. Ils sont générés par des grammaires algébriques. La reconnaissance des mots des langages algébriques se fait grâce aux automates à pile. Un automate à pile a la particularité de posséder une mémoire structurée en pile.

3.1 Langages algébriques

Un langage est algébrique si et seulement si : il existe une grammaire algébrique générant ce langage.

3.1.1 Exemples de langages algébriques

La classe des langages algébriques inclus tous les langages réguliers ainsi que de nouveaux langages comme par exemple :

- $a^n b^n, n \geq 0$
- Le langage des mots palindromes : $L = \{m/m \in (a^*b^*)^* \text{ et } m = m^r\}$
- Les expressions bien parenthésées : $L = \{m/m \in [(^*)^*]^* \text{ et } \mid m \mid_) = \mid m \mid_(\text{ et } \forall u, v \text{ tel que } m = uv \text{ alors } \mid u \mid_) \leq \mid u \mid_(\}$

3.1.2 Propriétés de fermeture des langages algébriques

Si $L1$ et $L2$ sont des langages algébriques sur un alphabet A, alors les langages suivants sont des langages algébriques :

- $L1 \cup L2$ (L'union)
- $L1L2$ (La concaténation)
- $L1^*$ (L'étoile)
- $L1^r$ (L'image miroir)

3.2 Grammaires algébriques

Les grammaires algébriques permettent de générer des langages algébriques.

3.2.1 Définition formelle

Une grammaire $G = (T, N, S, R)$ est algébrique si les règles de R sont de la forme :

$$S \rightarrow m \text{ avec } S \in N \text{ et } m \in (T \cup N)^*$$

La seule restriction concerne la partie gauche de la règle qui doit être constituée d'un seul symbole non terminal.

3.2.2 Exemple de grammaire algébrique

Soit un langage $L = \{a^n b^n, n \geq 0\}$ défini sur l'alphabet $A = \{a, b\}$. Nous allons donner une grammaire algébrique $G(L)$.
$G(L) = (T, N, S, R)$
$T = \{a, b\}$
$N = \{S\}$
$R = \{S \rightarrow aSb/\varepsilon\}$

Exemple de mot appartenant au langage $L = \{a^n b^n, n \geq 0\}$: $aabb$
Ce mot est obtenu en appliquant les règles de dérivation de R comme suit : $S \rightarrow aSb \rightarrow aaSbb \rightarrow aabb$

Exemple de mot n'appartenant pas au langage $L = \{a^n b^n, n \geq 0\}$: aab
ba : $S \rightarrow aSb$. Ensuite, il reste à générer un a au milieu, mais il n'existe pas de règle permettant de générer un a du type $S \rightarrow a$. Donc, le mot aab ne peut pas être généré par cette grammaire.

3.3 Automates à pile

Pour reconnaitre les mots des langages algébriques, on utilise des automates à pile.

Le principe de fonctionnement de l'automate à pile est le même que pour l'automate à états finis auquel on va rajouter une mémoire structurée en pile.

La pile possède son propre alphabet noté P. On ne peut empiler et dépiler que les symboles appartenant à P. De plus, on ne connait pas la taille de la pile et on a accès uniquement à l'élément *sommet* de la pile. Lorsque le *sommet* est le mot vide (ε), cela veut dire que la pile est vide.

3.3.1 Composition d'un automate à pile

Un automate à pile est composé de :

Une bande de lecture : La bande est composée d'une succession de cases. C'est dans les cases de cette bande qu'est écrit le mot à reconnaître. Chaque case pouvant contenir un seul symbole de l'alphabet d'entrée.

Une tête de lecture : La tête de lecture peut lire une case à un instant donné.

Une mémoire (une pile) : La pile possède un alphabet spécial. On peut empiler, dépiler ou accéder au *sommet* de pile.

Une unité de contrôle : Elle est définie par un ensemble fini d'états ainsi que par une fonction de transition qui décrit le passage d'un état à un autre en fonction du contenu de la case courante, de l'état courant et du *sommet* de la pile. A chaque fois qu'une transition s'exécute, la tête de lecture se déplace d'une seule case vers la droite.

3.3.2 Définition formelle d'un automate à pile

Un automate à pile AP est un sextuplet $AP = (A, P, Q, e_0, F, \delta)$:

A **est l'alphabet des mots en entrée :** c'est l'alphabet du langage des mots à reconnaitre.

P **est l'alphabet de pile :** c'est les symboles qui sont stockés dans la pile.

Q **est un ensemble non vide d'états :** l'automate utilise entre autre les états pour déterminer à quel étape il se trouve dans la reconnaissance des mots.

$e_0 \in Q$ **est l'état initial :** dans les automates à pile, on utilisera un seul état initial par lequel l'automate commence la reconnaissance.

$F \subseteq Q$ **est un ensemble d'états finaux :** ces états sont utilisé pour la reconnaissance des mots du langage.

δ **est un ensemble de transitions :** les transitions permettent de faire des éventuelles changements d'états à la lecture d'un symbole et/ou de faire des actions sur la pile.

3.3.3 Procédure de reconnaissance dans les automates à pile

La reconnaissance des mots par l'automate à pile commence par l'état initial et une pile vide.

Un mot est reconnu par un automate à pile si est seulement si à la fin de la lecture du mot, on est dans un état final et la pile est vide.

Il existe d'autres type d'automates à pile qui font la reconnaissance par pile vide ou par état final. Ces types ne sont pas traités dans ce cours.

3.3.4 Représentation des automates à pile

Pour les automates à pile, nous n'utiliserons que la représentation formelle.

Dans un automate à pile, à la lecture d'un symbole d'un mot, l'automate se trouve dans un état appartenant à Q et le sommet de pile contient un symbole de P (ou bien ε lorsque la pile est vide). Il effectue alors une transition pour éventuellement changer d'état et / ou l'état de la pile. Les transitions sont écrites sous la forme suivante :
(état source, symbole lu, *sommet* de pile) $\rightarrow$ (état destination, opération sur la pile)

3.3.5 Opérations sur la pile lors des transitions dans les automates à pile

Soit un automate à pile $AP = (A, P, Q, e_0, F, \delta)$ avec $a \in A$; $e_i, e_j \in Q$ et $A, B \in P$

La fonction empiler :

- $(e_i, a, A) \rightarrow (e_j, AB)$ **:** si on est dans l'état e_i et on lit un a et le *sommet* de pile est un A alors on passe à e_j et on empile un B.
- $(e_i, a, \varepsilon) \rightarrow (e_j, B)$ **:** si on est dans l'état e_i et on lit un a et la pile est vide alors on passe à e_j et on empile un B.

La fonction dépiler :

- $(e_i, a, A) \rightarrow (e_j, \varepsilon)$ **:** si on est dans l'état e_i et on lit un a et le *sommet* de pile est un A alors on passe à e_j et on dépile un A.

  On ne peut pas dépiler si la pile est vide.

Transitions sans modifier la pile :

- $(e_i, a, A) \rightarrow (e_j, A)$ **:** si on est dans l'état e_i et on lit un a et le *sommet* de pile est un A alors on passe à e_j sans modifier la pile.
- $(e_i, a, \varepsilon) \rightarrow (e_j, \varepsilon)$ **:** si on est dans l'état e_i et on lit un a et le *sommet* de pile est vide alors on passe à e_j sans modifier la pile.

Exemple : Soit un langage $L = \{a^n b^n, n \geq 0\}$ défini sur l'alphabet $A = \{a, b\}$. Nous allons donner une représentation formelle de l'automate à pile $AP(L)$.

L'idée est de compter le nombre de symbole a grâce à la pile. A chaque lecture d'un symbole a, on empile un symbole A. Pour vérifier qu'on a le même nombre de a que de b. On dépile un A à chaque lecture d'un b. Lorsque la pile est vide, on aura atteint le même nombre de a que de b.
$AP = (A, P, Q, e_0, F, \delta)$
$A = \{a, b\}$
$P = \{A\}$
$Q = \{e_0, e_1\}$

$F = \{e_0, e_1\}$
$\delta = \{$

$(e_0, a, \varepsilon) \rightarrow (e_0, A)$,

$(e_0, a, A) \rightarrow (e_0, AA)$,

$(e_0, b, A) \rightarrow (e_1, \varepsilon)$,

$(e_1, b, A) \rightarrow (e_1, \varepsilon)$
$\}$

Exemple de mot appartenant au langage $L = \{a^n b^n, n \geq 0\}$: $aabb$
Au début, l'automate se trouve à l'état initial e_0 et la pile est vide. La tête de lecture est sur le premier symbole du mot à savoir a. Il existe une transition qui va de cette configuration vers l'état e_0 et un A est empilé dans la pile $((e_0, a, \varepsilon) \rightarrow (e_0, A))$. La tête de lecture se situe alors sur le deuxième symbole du mot à savoir un a, et l'automate est resté dans l'état e_0 et la pile contient un A. L'automate va alors faire passer par la transition $(e_0, a, A) \rightarrow (e_0, AA)$. La tête de lecture se situe alors sur le troisième symbole du mot à savoir un b, et l'automate est resté dans l'état e_0 et la pile contient deux symboles A. Il existe une transition qui va de cette configuration vers l'état e_1 et un A va être dépilé $((e_0, b, A) \rightarrow (e_1, \varepsilon))$. La tête de lecture se situe alors sur le quatrième symbole du mot à savoir un b, et l'automate est passé à l'état e_1 et la pile contient un A. Il existe une transition qui va de cette configuration vers l'état e_1 et un A va être dépilé $((e_1, b, A) \rightarrow (e_1, \varepsilon))$. Le mot est terminé et l'automate se trouve dans l'état e_1 qui est un état final, et la pile est vide. Le mot est alors reconnu par l'automate.

Exemple de mot n'appartenant pas au langage $L = \{a^n b^n, n \geq 0\}$: aab
L'automate va avoir exactement le même comportement que le premier exemple sauf que le mot va se terminer alors que l'automate se trouve sur l'état e_1 et la pile contient un A. L'état e_1 est un état final, mais la pile n'est pas vide. Le mot est alors non reconnu par l'automate.

On peut aussi avoir des mots avec plus de b que de a comme par exemple abb. Dans ce cas l'automate va faire des empilement lorsqu'il rencontre des a puis des dépilement lorsqu'il rencontre des b, sauf que lorsqu'il va faire autant de dépilement que d'empilement, l'automate va se trouver dans l'état e_1 et il va lire un b (il y a plus de b que de a) et la pile sera vide. Il va alors y avoir un blocage. Le mot est alors non reconnu par l'automate.

3.3.6 Automates à pile indéterministes

Dans les automates à pile, il y a un seul état initial. Le seul cas possible pour avoir un automate à pile indéterministe est qu'à partir d'un état donné, un état de pile donné et un symbole lu, on a deux transitions possibles. Nous allons illustrer le cas des automates à pile indéterministes par l'exercice suivant :

Exercice : Donner le langage, la grammaire et l'automate à pile du langage des mots palindromes constitués des symboles $\{a, b\}$.

Solution : Un mot palindrome est un mot qui se lit de gauche à droite ou de droite à gauche de la même façon. C'est donc un mot qui est égal à son image miroir.
$L = \{m/m = m^r \text{ et } m \in (a^*b^*)^*\}$
$G(L) = (T, N, S, R)$
$T = \{a, b\}$
$N = \{S\}$
$R = \{S \to aSa/bSb/\varepsilon/a/b\}$

Il existe une grammaire algébrique pour ce langage donc ce langage est algébrique.

L'idée pour faire un automate à pile du langage des mots palindrome est déjà de définir le langage différemment :
$L = \{mm^r \cup mam^r \cup mbm^r/m \in (a^*b^*)^*\}$

Ce dernier langage est équivalent au premier sauf qu'il nous permet de voir les différents cas possibles. Avec cette écriture, on peut remarquer que le dernier symbole de m est égal au premier symbole de m^r et l'avant dernier symbole de m est égal au deuxième symbole de m^r et ainsi de suite jusqu'au premier symbole de m qui est égal au dernier symbole de m^r. C'est le principe même de la pile (dernier entrée- premier sortie). On va alors faire des empilements à la lecture de m et des dépilements à la lecture de m^r.

Sauf qu'on a trois cas (un mot de taille paire, un mot de taille impaire avec un a au milieu, un mot de taille impaire avec un b au milieu) et qu'on est incapable de savoir à la lecture d'un mot dans quel cas on se trouve. Pire encore, on n'a pas la possibilité de savoir à quel moment on a terminé m pour arrêter d'empiler et commencer à dépiler.

La solution est de faire un automate à pile indéterministe.

L'automate à pile indéterministe de L est comme suit :
$AP(L) = (A, P, Q, e_0, F, \delta)$
$A = \{a, b\}$
$P = \{A, B\}$
$Q = \{e_0, e_1\}$
$F = \{e_0, e_1\}$
$\delta = \{$

$(e_0, a, \varepsilon) \to (e_0, A)$,

$(e_0, b, \varepsilon) \to (e_0, B)$,

$(e_0, a, \varepsilon) \to (e_1, \varepsilon)$,

$(e_0, b, \varepsilon) \to (e_1, \varepsilon)$,

$(e_0, a, A) \to (e_0, AA)$,

$(e_0, a, B) \to (e_0, BA)$,

$(e_0, b, A) \to (e_0, AB)$,

$(e_0, b, B) \to (e_0, BB)$,

$(e_0, a, A) \to (e_1, \varepsilon)$,

$(e_0, b, B) \to (e_1, \varepsilon)$,

$(e_0, a, A) \to (e_1, A)$,

$(e_0, a, B) \to (e_1, B)$,

$(e_0, b, A) \to (e_1, A)$,

$(e_0, b, B) \to (e_1, B)$,

$(e_1, a, A) \to (e_1, \varepsilon)$,

$(e_1, b, B) \to (e_1, \varepsilon)$

}

Exemple de mot appartenant au langage $L = \{mm^r \cup mam^r \cup mbm^r / m \in (a^*b^*)^*\}$: a
Au début, l'automate se trouve dans l'état initial e_0 et la pile est vide. La tête de lecture est sur le premier symbole du mot à savoir a. Il existe deux transitions à partir de cette configuration. il faut alors tester les deux chemins :

- **Chemin 1 :** $(e_0, a, \varepsilon) \to (e_0, A)$, le mot est alors terminé et l'automate se trouve sur l'état e_0 et la pile contient un A. L'état e_0 n'est pas un état final et la pile n'est pas vide. Donc, ce chemin ne permet pas de reconnaitre le mot.
- **Chemin 2 :** $(e_0, a, \varepsilon) \to (e_1, \varepsilon)$, le mot est alors terminé et l'automate se trouve sur l'état e_1 et la pile est vide. L'état e_1 est un état final et la pile est vide. Donc, ce chemin permet de reconnaitre le mot.

Le mot a est reconnu par l'automate car il y a au moins un chemin qui a reconnue le mot.

Exemple de mot n'appartenant pas au langage $L = \{mm^r \cup mam^r \cup mbm^r / m \in (a^*b^*)^*\}$: ab
Au début, l'automate se trouve dans l'état initial e_0 et la pile est vide. La tête de lecture est sur le premier symbole du mot à savoir a. Il existe deux transitions à partir de cette configuration. il faut alors tester les deux chemins :

- **Chemin 1 :** $(e_0, a, \varepsilon) \to (e_0, A)$, la tête de lecture est sur le deuxième symbole à savoir le b, l'automate est dans l'état e_0 et la pile contient un A. Il existe deux transitions à partir de cette configuration. il faut les tester les deux :
  - **Chemin 1.1 :** $(e_0, b, A) \to (e_0, AB)$, le mot est alors terminé et l'automate se trouve sur l'état e_0 et la pile n'est pas vide. L'état e_0 n'est pas un état final et la pile n'est pas vide. Donc, ce chemin ne permet pas de reconnaitre le mot.
  - **Chemin 1.2 :** $(e_0, b, A) \to (e_1, A)$, le mot est alors terminé et l'automate se trouve sur l'état e_1 et la pile n'est pas vide. L'état e_1 est un état final mais comme la pile n'est pas vide, ce chemin ne permet pas de reconnaitre le mot.
- **Chemin 2 :** $(e_0, a, \varepsilon) \to (e_1, \varepsilon)$, la tête de lecture est sur le deuxième symbole à savoir le b, l'automate est dans l'état e_1 et la pile est vide. Il n'existe aucune transition à partir de cette configuration. Donc, ce chemin ne permet pas de reconnaitre le mot.

Le mot ab n'est pas reconnu par l'automate car il n'existe aucun chemin qui permette de reconnaitre le mot ab.

Il existe un automate à pile pour reconnaitre n'importe quel langage algébrique. Par contre, il n'existe pas nécessairement un automate à pile déterministe pour tous les langages algébriques.

Il n'existe pas d'algorithme permettant de passer de l'automate à pile indéterministe à l'automate à pile déterministe.

La classe des langages reconnus par un automate à pile déterministe est incluse dans la classe des langages algébriques mais n'est pas égale. Ce qui signifie qu'il existe des langages algébriques qui ne possèdent pas d'automates à pile déterministe reconnaissant les mots de ces langages.

3.4 Conclusion

Les langages algébriques sont générés par des grammaires algébriques et les mots d'un langage algébrique peuvent être reconnus par un automate à pile.

Les grammaires algébriques ont une seule contrainte, c'est que la partie gauche de la règle doit être constituée d'un seul symbole non terminal.

Les automates à pile déterministe ne peuvent pas reconnaitre les mots de tous les langages algébriques. Par contre, pour tout langage algébrique, il existe au moins un automate déterministe ou indéterministe qui reconnait les mots du langage.

En raison de la relation d'inclusion entre les langages (voir figure 1.1), il existe une grammaire algébrique pour générer n'importe quel langage régulier et un automate à pile pour reconnaitre les mots d'un langage régulier.

3.5 Exercices

Exercice 1 : Donner les grammaires ainsi que les automates à pile des langages suivants :

— $L1 = \{a^n b^n c^m d^m, n > 0 \text{ et } m > 0\}$
— $L2 = \{a^n b^m c^m d^n, n > 0 \text{ et } m > 0\}$

Exercice 2 : Définir formellement les langages suivants puis donner leurs grammaires ainsi que leurs automates à pile :

— $L1$ = les mots palindromes constitués des symboles a, b et un seul c au milieu du mot.
— $L2 = L3L4$ tel que :

  — $L3$ est constitué des symboles a et b et le nombre de symbole a est égal au nombre de symbole b.
  — $L4$ est constitué des symboles c et d et le nombre de symbole c est égal au nombre de symbole d.

Exercice 3 : Une équipe de cryptographes décide d'introduire une règle pour les messages codés (une signature), les messages qui doivent être décryptés sont uniquement les messages constitués d'un même nombre de symboles "?" que de symboles "!".
Les autres sont rejetés. Les messages peuvent aussi contenir les symboles suivants $\{1, 2, a, b\}$.

1. Donner un automate à pile permettant de savoir si un message doit être décrypté ou non.
2. Définir formellement le langage L reconnu par l'automate à pile.
3. Donner la grammaire $G(L)$.

Exercice 4 : QCM

1. Soient les langages suivants, lesquels désignent un langage de mots palindromes ?
   a) $L1 = \{mam^r / m \in (a^*b^*)^*\}$
   b) $L2 = \{mm^r / m \in (a^*b^*)^*\}$
   c) $L3 = \{m / m \in (a^*b^*)^* \text{ et } m = m^r\}$
   d) $L4 = \{mcm^r / m \in (a^*b^*)^*\}$
2. Lorsqu'il existe un automate à pile indéterministe, il existe un automate à pile déterministe équivalent.
   a) Vrai
   b) Faux

3. Le langage $L = \{a^n b^m, n < m\}$ est reconnaissable par un automate à pile déterministe.
   a) Vrai
   b) Faux
4. Le langage $L = \{a^n b^m, n, m > 0\}$ est reconnaissable par un automate à pile déterministe.
   a) Vrai
   b) Faux
5. Le langage $L = \{a^n b^m, n > m\}$ est reconnaissable par un automate à pile déterministe.
   a) Vrai
   b) Faux

Chapitre 4

Langages contextuels

Les langages contextuels sont les langages de type 1, ils sont générés par des grammaires de type 1. Il existe deux types de grammaires de type 1, les grammaires monotones et les grammaires contextuelles. La reconnaissance des mots des langages contextuels se fait grâce aux machines de Turing à bornes linéaires.

4.1 Langages contextuels

Un langage est contextuel si et seulement si : il existe une grammaire de type 1 générant ce langage.

4.1.1 Exemples de langages contextuels

La classe des langages contextuels inclus tous les langages algébriques ainsi que de nouveaux langages comme par exemple :

— $a^n b^n c^n, n \geq 0$
— $a^{2^n}, n \geq 0$

4.1.2 Propriétés de fermeture des langages contextuels

Si $L1$ et $L2$ sont des langages contextuels sur un alphabet A, alors les langages suivants sont des langages contextuels :

— $L1 \cup L2$ (L'union)
— $L1L2$ (La concaténation)
— $L1^*$ (L'étoile)
— $L1 \cap L2$ (L'intersection)
— $C(L1)$ (Le complément)

4.2 Grammaires de type 1

Les grammaires de type 1 permettent de générer des langages contextuels.

4.2.1 Définition formelle

Il existe deux types de grammaires de type 1 équivalentes :

Grammaire contextuelle : Une grammaire $G = (T, N, S, R)$ est contextuelle si les règles de R sont de la forme :

$$uSv \rightarrow umv \text{ avec } S \in N,\ u, v \in (T \cup N)^* \text{ et } m \in (T \cup N)^+$$

Le symbole non terminal S est remplacé par m tout en gardant le préfixe u à gauche et le suffixe v à droite.

Grammaire monotone : Une grammaire $G = (T, N, S, R)$ est monotone si les règles de R sont de la forme :

$$u \rightarrow v \text{ avec } u, v \in (T \cup N)^+ \text{ et } \mid u \mid \leq \mid v \mid$$

On remarque en analysant les deux définitions précédentes qu'il n'est pas possible d'avoir de règle permettant de générer le mot vide.

En effet, dans la grammaire contextuelle, on a u et v qui peuvent prendre ε mais pas m ($m \in (T \cup N)^+$). Dans la grammaire monotone, on a $u \rightarrow v$ avec $u, v \in (T \cup N)^+$.

Pour pouvoir générer le mot vide par une grammaire de type 1 d'un langage qui contient le mot vide, il y a une condition supplémentaire sur les règles qui est :

Si $(S \rightarrow \varepsilon) \in R$, S étant l'axiome, alors S n'apparait pas en partie droite d'une autre règle.

Et $(S' \rightarrow \varepsilon) \notin R, \forall S' \in N$ et $S' \neq S$.

4.2.2 Exemples de grammaires de type 1

Soit un langage $L = a^n b^n c^n, n \geq 0$ défini sur l'alphabet $A = \{a, b, c\}$. Nous donnons une grammaire monotone $G_m(L)$ et une grammaire contextuelle $G_c(L)$.

La grammaire monotone $G_m(L)$:
$G_m(L) = (T, N_m, S_m, R_m)$
$T = \{a, b, c\}$
$N_m = \{S_m, X, Y\}$
$R_m = \{$

$S_m \rightarrow aXbc/abc/\varepsilon,$

$X \rightarrow aXYb/aYb,$

$Yb \rightarrow bY,$

$Yc \rightarrow cc$
$\}$

Voici un exemple de génération de mot ($aaabbbccc$) avec la grammaire monotone $G_m(L)$:

$$S_m \to aXbc \overset{X\to aXYb}{\to} aaXYbbc \overset{X\to aYb}{\to} aaaYbYbbc \overset{Yb\to bY}{\to} aaabYYbbc \overset{Yb\to bY}{\to} aaabYbYbc$$

$$\overset{Yb\to bY}{\to} aaabbYYbc \overset{Yb\to bY}{\to} aaabbYbYc \overset{Yb\to bY}{\to} aaabbbYYc \overset{Yc\to cc}{\to} aaabbbYcc \overset{Yc\to cc}{\to} aaabbbccc$$

On peut vérifier que cette grammaire est une grammaire monotone. En effet, on a :
- $S_m \to \varepsilon$. S_m est bien l'axiome, et S_m n'apparait pas en partie droite d'une autre règle.
- $S_m \to aXbc.$ | S_m |≤| $aXbc$ |
- $S_m \to abc.$ | S_m |≤| abc |
- $X \to aXYb.$ | X |≤| $aXYb$ |
- $X \to aYb.$ | X |≤| aYb |
- $Yb \to bY.$ | Yb |≤| bY |
- $Yc \to cc.$ | Yc |≤| cc |

La grammaire contextuelle $G_c(L)$:
$G_c(L) = (T, N_c, S_c, R_c)$
$T = \{a, b, c\}$
$N_c = \{S_c, A, B, C, D, E\}$
$R_c = \{$

$S_c \to aABc/abc/\varepsilon,$

$A \to aACB/aCB,$

$B \to b,$

$Cc \to cc,$

$CB \to DB,$

$DB \to DE,$

$DE \to BE,$

$BE \to BC$
$\}$

Dans la grammaire contextuelle, il n'est pas possible d'avoir des règle du type $CB \to BC$, c'est pourquoi nous avons utiliser 4 règles pour passer de CB à BC en utilisant les symboles non terminaux D et E.

Voici un exemple de génération de mot ($aaabbbccc$) avec la grammaire contextuelle $G_c(L)$:

$S_c \to aABc \overset{A\to aACB}{\to} aaACBBc \overset{A\to aCB}{\to} aaaCBCBBc \overset{CB\to DB}{\to} aaaDBCBBc \overset{DB\to DE}{\to}$

$aaaDECBBc \overset{DE\to BE}{\to} aaaBECBBc \overset{BE\to BC}{\to} aaaBCCBBc \overset{*}{\to} aaaBCBCBc \overset{*}{\to}$

$aaaBBCCBc \overset{*}{\to} aaaBBCBCc \overset{*}{\to} aaaBBBCCc \overset{B\to b}{\to} aaabBBCCc \overset{B\to b}{\to} aaabbBCCc \overset{B\to b}{\to}$

$aaabbbCCc \overset{Cc\to cc}{\to} aaabbbCcc \overset{Cc\to cc}{\to} aaabbbccc$

On peut facilement vérifier que cette grammaire est une grammaire contextuelle. En effet, on va donner pour chaque règle le préfixe u et le suffixe v :

- $S_c \to \varepsilon$. S_c est bien l'axiome, et S_c n'apparait pas en partie droite d'une autre règle.
- $S_c \to aABc$. $u = v = \emptyset$
- $S_c \to abc$. $u = v = \emptyset$
- $A \to aACB$. $u = v = \emptyset$
- $A \to aCB$. $u = v = \emptyset$
- $B \to b$. $u = v = \emptyset$
- $Cc \to cc$. $u = \emptyset, v = c$
- $CB \to DB$. $u = \emptyset, v = B$
- $DB \to DE$. $u = D, v = \emptyset$
- $DE \to BE$. $u = \emptyset, v = E$
- $BE \to BC$. $u = B, v = \emptyset$

4.3 Machines de Turing à bornes linéaires

Pour reconnaitre les mots des langages contextuels, on utilise des automates particuliers appelés : machines de Turing à bornes linéaires.

Les machines de Turing à bornes linéaires utilise le principe de marquage (Les symboles des mots sont remplacés par d'autres symboles) dans la procédure de reconnaissance.

4.3.1 Composition d'une Machine de Turing à bornes linéaires

Les composants d'une machine de Turing à bornes linéaires sont :

Une bande de lecture/écriture : Contrairement aux automates vus jusque là, la bande de lecture d'une machine de Turing à bornes linéaires est aussi une bande d'écriture. Chaque case pouvant contenir un seul symbole de l'alphabet d'entrée ou de l'alphabet auxiliaire d'écriture.

Une tête de lecture/écriture : La tête de lecture peut lire une case à un instant donné mais aussi écrire dans une case. La tête peut être déplacée par la machine de Turing à bornes linéaires pour se positionner sur la case immédiatement à gauche ou à droite de la case courante.

Une unité de contrôle : Elle est définie par un ensemble fini d'états ainsi que par une fonction de transition qui décrit le passage d'un état à un autre en fonction du contenu de la case courante de la bande de lecture et de l'état dans lequel se trouve la machine de Turing à bornes linéaires. L'unité de contrôle décide aussi de la direction dans laquelle il faut déplacer la tête de lecture/écriture.

4.3.2 Définition formelle d'une Machine de Turing à bornes linéaires

Une machine de Turing à bornes linéaires $MTABL$ est un septuplet :
$AP = (A, X, \sharp, Q, e_0, e_f, \delta)$:

- A **est l'alphabet des mots en entrée :** c'est l'alphabet du langage des mots à reconnaitre.
- X **est l'alphabet auxiliaire d'écriture :** c'est les symboles qui sont utilisés pour le marquage.
- $\sharp$ **est le symbole qui marque la fin de bande à droite et à gauche :** Le mot à reconnaitre m se trouve sur la bande de lecture/écriture entre deux symboles $\sharp$. On a donc toujours $\sharp m \sharp$.
- Q **est un ensemble non vide d'états :** Utilisé entre autre pour déterminer à quel étape se trouve la machine de Turing à bornes linéaires dans la reconnaissance des mots.
- $e_0 \in Q$ **est l'état initial :** État par lequel la machine de Turing à bornes linéaires commence la reconnaissance.
- $e_f \in Q$ **est un état final :** Il y a la possibilité d'utiliser un seul état final dans machine de Turing à bornes linéaires.
- δ **est un ensemble de transitions :** Les transitions permettent de faire certaines actions à la lecture d'un symbole, comme par exemple le changement d'état, l'écriture ou le changement de direction de la tête de lecture/écriture.

4.3.3 Procédure de reconnaissance dans les machines de Turing à bornes linéaires

Le mot à reconnaitre se trouve sur la bande de lecture/écriture entre deux symboles $\sharp$. Un symbole $\sharp$ à gauche du mot et un autre à droite du mot. La reconnaissance des mots par une machine de Turing commence par l'état initial et la tête de lecture/écriture se trouve sur le premier symbole du mot ou bien sur le symbole $\sharp$ de droite si le mot est vide. Un mot est reconnu si à la fin, on se trouve dans un état final est que la tête de lecture/écriture se trouve à droite du symbole $\sharp$ de droite. Une machine de Turing à borne linéaire ne peut pas utiliser les cases qui précèdent le $\sharp$ de gauche ou les cases qui suivent le $\sharp$ de droite pour la reconnaissance.

4.3.4 Représentation des machines de Turing à bornes linéaires

Concernant les machines de Turing à bornes linéaires, nous allons utiliser uniquement la représentation tabulaire qui est comme suit :

- Chaque ligne représente un état.
- Chaque colonne représente un symbole de l'alphabet du langage ou de l'alphabet auxiliaire ou bien le symbole ♯.
- L'état initial est marqué par une flèche entrante et l'état final par une flèche sortante.
- Les cases du tableau sont constituées de triplets (état destination, le symbole écrit, la direction). La direction est soit D pour droite ou G pour gauche.

Exemple : Soit un langage $L = \{a^n b^n c^n, n \geq 0\}$ défini sur l'alphabet $A = \{a, b, c\}$. Nous allons donner une représentation tabulaire de la machine de Turing à bornes linéaires $MTABL(L)$.

L'idée est de marquer un symbole a avec un symbole A. Ensuite, il faut traverser les autres sans les marquer jusqu'à arriver au premier symbole b qui est marqué à son tour par un B. Ensuite, il faut traverser les autres sans les marquer jusqu'à arriver au premier symbole c qui est marqué à son tour par un C. Après, il faut revenir au deuxième a pour refaire la même procédure de marquage. Donc à chaque tour, un a, un b et un c sont marqués. Donc, lorsque les a se terminent, les autres symboles doivent être tous marqués (il n'y a plus de symboles a, b, c sur la bande de lecture/écriture).

	a	b	c	A	B	C	♯
$\rightarrow e_0$	(e_1, A, D)				(e_4, B, D)		$(e_f, \sharp, D)$
e_1	(e_1, a, D)	(e_2, B, D)			(e_1, B, D)		
e_2		(e_2, b, D)	(e_3, C, G)			(e_2, C, D)	
e_3	(e_3, a, G)	(e_3, b, G)		(e_0, A, D)	(e_3, B, G)	(e_3, C, G)	
e_4					(e_4, B, D)	(e_4, C, D)	$(e_f, \sharp, D)$
$\leftarrow e_f$							

Exemple de mot appartenant au langage $L = \{a^n b^n c^n, n \geq 0\}$: $aabbcc$
Le mot se trouve entre deux symboles ♯ dans la bande de lecture/écriture ($\sharp aabbcc\sharp$). Initialement la machine de Turing à bornes linéaires se trouve à l'état initial e_0 et la tête de lecture/écriture sur le premier symbole du mot à savoir a. Il existe une transition qui va de cette configuration vers l'état e_1 avec le marquage du symbole a avec le symbole A et la tête de lecture/écriture se déplace à droite($\sharp Aabbcc\sharp$). La tête de lecture se situe alors sur le deuxième symbole du mot à savoir un a, et la machine de Turing à bornes linéaires est à l'état e_1. Il existe une transition qui va de cette configuration vers l'état e_1 sans marquage et la tête de lecture/écriture se déplace à droite($\sharp Aabbcc\sharp$). La tête de lecture se situe alors sur le troisième symbole du mot à savoir un b, et la machine de Turing à bornes linéaires est à l'état e_1. Il existe une transition qui va de cette configuration vers l'état e_2 avec le marquage du symbole b avec le symbole B et la tête de lecture/écriture se déplace à droite($\sharp AaBbcc\sharp$). La tête de lecture se situe alors sur

le quatrième symbole du mot à savoir un b, et la machine de Turing à bornes linéaires est à l'état e_2. Il existe une transition qui va de cette configuration vers l'état e_2 sans marquage et la tête de lecture/écriture se déplace à droite($\sharp AaBbcc\sharp$). La tête de lecture se situe alors sur le cinquième symbole du mot à savoir un c, et la machine de Turing à bornes linéaires est à l'état e_2. Il existe une transition qui va de cette configuration vers l'état e_3 avec le marquage du symbole c avec le symbole C et la tête de lecture/écriture se déplace à gauche($\sharp AaBbCc\sharp$). La machine de Turing à bornes linéaires est à l'état e_3, elle va aller à gauche tant qu'elle ne rencontre pas de symbole A.

A la lecture d'un A, la machine de Turing à bornes linéaires se déplace à droite et revient à état e_0 pour un nouveau cycle de marquage pour obtenir le marquage suivant $\sharp AABBCC\sharp$. Après le marquage du dernier symbole c. La machine de Turing à bornes linéaires est à l'état e_3, elle va aller à gauche tant qu'elle ne rencontre pas de symbole A.

A la lecture d'un A, la machine de Turing à bornes linéaires se déplace à droite et revient à état e_0 et trouve un symbole B. A la lecture d'un B, la machine de Turing à bornes linéaires se déplace à droite et va à état e_4. Il ne reste que des symbole B et C avant le $\sharp$, ils vont être parcourus avec l'état e_4, et à la rencontre du $\sharp$ la machine de Turing à bornes linéaires se déplace à droite et va à état e_f. Le mot est donc reconnu.

4.3.5 Machines de Turing à bornes linéaires indéterministes

On peut avoir des machines de Turing à bornes linéaires indéterministes pour les langages contextuels. Dans une Machine de Turing à bornes linéaires indéterministe, on a au moins une case qui contient deux transitions (deux triplets). Par contre, contrairement aux langages de type 2, la classe des langages reconnus par une machine de Turing à bornes linéaires déterministe est équivalentes à la classe des langages contextuels. Donc, si L est un langages contextuel alors, il existe une machine de Turing à bornes linéaires déterministe reconnaissant les mots de L.

4.4 Conclusion

Les langages contextuels sont générés par des grammaires de type 1 et les mots d'un langage contextuel peuvent être reconnus par une machine de Turing à bornes linéaires.

Il existe deux sortes de grammaires de type 1 équivalentes : les grammaires monotones et les grammaires contextuelles. Pour tout langage contextuel, il est possible d'avoir les deux sortes de grammaires.

Nous nous sommes focalisés dans ce chapitre sur les machines de Turing à bornes linéaires qui représentent la partie la plus importante du chapitre. Nous avons vu qu'on pouvait avoir une machine de Turing à bornes linéaires déterministe pour tout langage contextuel, et qu'on pouvait avoir un seul état initial et un seul état final avec ce type de reconnaisseur.

4.5 Exercices

Exercice 1 : Faire une machine de Turing à borne linéaire pour chacun des langages suivants :

- $L1 = \{a^n b^m c^n d^m | n, m > 0\}$
- $L2 = \{a^{3n} b^{2n} c^n, n > 0\}$
- $L3 = \{a^{2^n}, n \geq 0\}$
- $L4 = \{m/m \in (a^* b^* c^*)^*$ et $\mid m \mid_a = \mid m \mid_b = \mid m \mid_c\}$

Chapitre 5

Langages récursivement énumérables et calculabilité

Les langages récursivement énumérables sont les langages de type 0. Ils sont générés par des grammaires sans restriction. La reconnaissance des mots des langages récursivement énumérables se fait grâce aux machines de Turing. Nous allons voir à la fin de ce chapitre comment les machines de Turing sont aussi utilisées pour la calculabilité.

5.1 Langages récursivement énumérables

Un langage L est récursivement énumérable si et seulement si il existe une grammaire de type 0 générant ce langage.

Un langage L est récursivement énumérables si et seulement si il existe une machine de Turing reconnaissant ce langage.

5.1.1 Exemples de langages récursivement énumérables

La classe des langages récursivement énumérables inclus tous les langages contextuels ainsi que de nouveaux langages comme par exemple :

— $L = \{n/n \in 1(0^*1^*)^* \cup 0$ et La valeur entière de n représenté en binaire est divisible par 3$\}$

5.1.2 Propriétés de fermeture des langages récursivement énumérables

Si $L1$ et $L2$ sont des langages récursivement énumérables sur un alphabet A, alors les langages suivants sont des langages récursivement énumérables :

— $L1 \cup L2$ (L'union)
— $L1L2$ (La concaténation)
— $L1^*$ (L'étoile)
— $L1 \cap L2$ (L'intersection)

5.2 Grammaires sans restriction

La grammaire de type 0 est une grammaire sans restriction :

5.2.1 Définition formelle

Une grammaire sans restriction est une grammaire $G = (T, N, S, R)$ telle que les règles de R sont de la forme :

$$u \to v \text{ avec } u \in (T \cup N)^+,\ v \in (T \cup N)^*$$

La seule restriction est qu'on ne peut pas générer un mot à partir du mot vide.

5.2.2 Exemple de grammaire sans restriction

Soit un langage $L = a^{2^n}, n \geq 0$ défini sur l'alphabet $A = \{a\}$. Nous allons donner deux grammaire pour ce langage $G_1(L)$ et $G_2(L)$.

$G_1(L) = (T, N_1, S_1, R_1)$
$T = \{a\}$
$N_1 = \{S_1, D_1, X_1, F_1, Y_1, Z_1\}$
$R_1 = \{$

$S_1 \to D_1X_1aF_1/a,$

$X_1a \to aaX_1,$

$X_1F_1 \to Y_1F_1,$

$X_1F_1 \to Z_1,$

$aY_1 \to Y_1a,$

$D_1Y_1 \to D_1X_1,$

$aZ_1 \to Z_1a,$

$D_1Z_1 \to \varepsilon$
$\}$

Voici un exemple de génération de mot ($aaaaaaaa$) avec la grammaire $G_1(L)$:

$$S_1 \to D_1X_1aF_1 \overset{X_1a\to aaX_1}{\to} D_1aaX_1F_1 \overset{X_1F_1\to Y_1F_1}{\to} D_1aaY_1F_1 \overset{aY_1\to Y_1a}{\to} D_1aY_1aF_1 \overset{aY_1\to Y_1a}{\to}$$

$$D_1Y_1aaF_1 \overset{D_1Y_1\to D_1X_1}{\to} D_1X_1aaF_1 \overset{X_1a\to aaX_1}{\to} D_1aaX_1aF_1 \overset{X_1a\to aaX_1}{\to} D_1aaaaX_1F_1 \overset{X_1F_1\to Y_1F_1}{\to}$$

$$D_1aaaaY_1F_1 \overset{aY_1\to Y_1a}{\to} D_1aaaY_1aF_1 \overset{*}{\to} D_1Y_1aaaaF_1 \overset{D_1Y_1\to D_1X_1}{\to} D_1X_1aaaaF_1 \overset{X_1a\to aaX_1}{\to}$$

$D_1aaX_1aaaF_1 \overset{*}{\to} D_1aaaaaaaaX_1F_1 \overset{X_1F_1\to Z_1}{\to} D_1aaaaaaaaZ_1 \overset{aZ_1\to Z_1a}{\to} D_1aaaaaaaZ_1a$

$\overset{aZ_1\to Z_1a}{\to} D_1aaaaaaZ_1aa \overset{*}{\to} D_1Z_1aaaaaaaa \overset{D_1Z_1\to\varepsilon}{\to} aaaaaaaa$

De quel type est cette grammaire ?
- Cette grammaire n'est pas de type 3 ($S_1 \to D_1X_1aF_1$)
- Cette grammaire n'est pas de type 2 ($X_1a \to aaX_1$)
- Cette grammaire n'est pas de type 1 ($X_1F_1 \to Z_1$)
- Cette grammaire est de type 0

$G_2(L) = (T, N_2, S_2, R_2)$
$T = \{a\}$
$N_2 = \{S_2, D_2, X_2, F_2, Y_2\}$
$R_2 = \{$

$S_2 \to D_2Y_2F_2/aa/a,$

$Y_2 \to X_2Y_2,$

$X_2aa \to aaX_2a,$

$D_2aaa \to aaD_2aa,$

$Y_2 \to aa,$

$X_2aF_2 \to aaF_2,$

$D_2aaF_2 \to aaaa$
$\}$

Voici un exemple de génération de mot ($aaaaaaaa$) avec la grammaire $G_2(L)$:

$S_2 \to D_2Y_2F_2 \overset{Y_2\to X_2Y_2}{\to} D_2X_2Y_2F_2 \overset{Y_2\to aa}{\to} D_2X_2aaF_2 \overset{X_2aa\to aaX_2a}{\to} D_2aaX_2aF_2 \overset{X_2aF_2\to aaF_2}{\to}$

$D_2aaaaF_2 \overset{D_2aaa\to aaD_2aa}{\to} aaD_2aaaF_2 \overset{D_2aaa\to aaD_2aa}{\to} aaaaD_2aaF_2 \overset{D_2aaF_2}{\to} aaaaaaaa$

De quel type est cette grammaire ?
- Cette grammaire n'est pas de type 3 ($S_2 \to D_2Y_2F_2$)
- Cette grammaire n'est pas de type 2 ($X_2aa \to aaX_2a$)
- Cette grammaire est de type 1 (grammaire monotone)

Pour le même langage L, nous avons une grammaire $G_1(L)$ de type 0 et une grammaire $G_2(L)$ de type 1. Donc, L est un langage de type 1 (voir 1.1.3).

5.3 Machines de Turing

Pour reconnaitre les mots des langages de type 0, on utilise des machines de Turing.

5.3.1 Composition d'une Machine de Turing

Les composants d'une machine de Turing sont :

Une bande de lecture/écriture infinie : La bande de lecture d'une machine de Turing est aussi une bande d'écriture. Chaque case pouvant contenir un seul symbole de l'alphabet d'entrée ou de l'alphabet auxiliaire d'écriture. Cette bande est infinie.

Une tête de lecture/écriture : La tête de lecture peut lire une case à un instant donné mais aussi écrire dans une case. La tête peut être déplacée par l'automate pour se positionner sur la case immédiatement à gauche ou à droite de la case courante.

Une unité de contrôle : Elle est définie par un ensemble fini d'états ainsi que par une fonction de transition qui décrit le passage d'un état à un autre en fonction du contenu de la case courante de la bande de lecture et de l'état dans lequel se trouve la machine de Turing. L'unité de contrôle décide aussi de la direction dans laquelle il faut déplacer la tête de lecture/écriture.

Avant même l'apparition des premiers ordinateurs, Alain Turing (1912 - 1954), un mathématicien et cryptographe britannique a proposé un modèle théorique en la machine de Turing en 1936 qui est une abstraction des ordinateurs modernes. Le microprocesseur est représenté par l'unité de contrôle, le nombre d'états, étant fini, peut être assimilé au nombre de configurations fini des microprocesseurs. Les mémoires (registres, mémoire vive et disques dures) représentées par une bande de lecture/écriture infinie. A noter que quel que soit le nombre et la taille des mémoires, on ne pourra jamais avoir une mémoire infinie.

5.3.2 Définition formelle d'une Machine de Turing

Une machine de Turing MT est un sextuplet $AP = (A, X, Q, e_0, e_f, \delta)$:

A **est l'alphabet des mots en entrée :** C'est l'alphabet du langage des mots à reconnaitre.

X **est l'alphabet auxiliaire d'écriture :** C'est les symboles qui sont utilisés pour le marquage.

Q **est un ensemble non vide d'états :** Utilisé entre autre pour déterminer à quel étape se trouve la machine de Turing dans la reconnaissance des mots.

$e_0 \in Q$ **est l'état initial :** État par lequel la machine de Turing commence la reconnaissance.

$e_f \in Q$ **est un état final :** Il y a la possibilité d'utiliser un seul état final dans machine de Turing.

δ **est un ensemble de transitions :** Les transitions permettent de faire certaines actions à la lecture d'un symbole, comme par exemple le changement d'état, l'écriture ou le changement de direction de la tête de lecture/écriture.

5.3.3 Représentation des Machines de Turing

Concernant les machines de Turing, nous allons utiliser uniquement la représentation tabulaire qui est comme suit :

- Chaque ligne représente un état.
- Chaque colonne représente un symbole de l'alphabet du langage ou de l'alphabet auxiliaire.
- L'état initial est marqué par une flèche entrante et l'état final par une flèche sortante.
- Les cases du tableau sont constituées de triplets (état destination, le symbole écrit, la direction). La direction est soit D pour droite ou G pour gauche.

5.3.4 Utilisations des Machines de Turing

On utilise les machines de Turing pour faire :

- la reconnaissance,
- la génération,
- le calcul.

5.3.5 Différence entre la reconnaissance avec les machines de Turing et la reconnaissance avec les machines de Turing à bornes linéaires

Dans une machine de Turing à bornes linéaires, le marquage qui est utilisé pour la reconnaissance doit être réalisé en utilisant l'espace du mot (entre les deux symboles ♯). Si l'espace du mot n'est pas suffisant pour décider si un mot appartient à un langage donné L et qu'il est nécessaire de faire un calcul qui prend plus d'espace que la taille du mot, alors L n'est pas un langage de type 1 mais de type 0. Car, la bande de lecture/écriture d'une machine de Turing n'est pas limitée, elle contient un mot ou plusieurs mots séparés par une case vide entre tout couple de mots. Les autres cases sont vides. La case vide est représentée par un « _ ». Du coup, pour reconnaitre un mot, la machine de Turing peut utiliser les cases vides pour faire un marquage supplémentaire. C'est ce qui permet de faire des calculs.

Par exemple pour le langage : $L = \{m/m \in 1(0^*1^*)^* \cup 0$ et la valeur entière de m représentée en binaire est divisible par $3\}$ Pour savoir si un mot appartient à L, on doit déterminer si ce mot écrit en binaire est divisible par 3, ce qui n'est pas possible directement. On doit écrire m d'une certaine façon qui nous permet de savoir s'il est divisible par 3 ou pas. On a donc besoin de plus d'espace que l'espace réservé au mot m écrit en binaire pour décider si m appartient au langage. Pour résoudre ce problème on doit utiliser la calculabilité.

5.4 Calculabilité

La calculabilité est l'utilisation d'une machine de Turing pour le calcul. Les nombres positifs ou nul peuvent être représenté à l'aide du symbole « | ».

exemple de nombres sur une machine de Turing

- 0 : |
- 1 : ||
- 2 : |||
- 3 : ||||
- 4 : |||||
- 5 : ||||||
- ...

5.4.1 Procédure de calcul dans les Machines de Turing

Initialement, la tête de lecture/écriture est positionné sur le premier | du premier nombre (les nombres étant positifs ou nul, on a au moins un | par nombre). les nombres sont séparés par une case vide. Il existe aussi une case vide entre le dernier nombre et le résultat. La case vide est représentée par un « _ ». A la fin de la procédure de calcul, la machine de Turing doit être sur l'état final et la tête de lecture doit être positionné sur le premier | du résultat.

5.4.2 exemple

On souhaite calculer $a + b$.

La bande de lecture/écriture contient alors deux nombres séparés par une case vide et le reste de la bande (qui est infinie) est vide.

Si $a = 3$ et $b = 1$, on aura donc ..._ |||| _ || _...

Le résultat doit se trouver après les deux nombres a et a et une case vide.

Si $a = 3$ et $b = 1$ alors $a + b = 4$, on aura donc ..._| |||| _...

L'idée est d'abord de marquer le premier | de chaque nombre et de mettre un premier symbole | dans la partie résultat qui représente le 0. Ensuite, il suffit qu'à chaque fois qu'on marque un symbole | de a ou de b, on rajoute un symbole | au résultat. Ce qui donne la machine de Turing suivante :

	$\|$	X	$\_$
$\rightarrow e_0$	(e_1, X, D)		
e_1	$(e_1, \|, D)$		$(e_2, \_, D)$
e_2	(e_3, X, D)		$(e_6, \_, G)$
e_3	$(e_3, \|, D)$		$(e_4, \_, D)$
e_4	$(e_4, \|, D)$		$(e_5, \|, G)$
e_5	$(e_5, \|, G)$	(e_2, X, D)	$(e_5, \_, G)$
e_6		(e_6, X, G)	$(e_7, \_, G)$
e_7	$(e_7, \|, G)$	(e_8, X, D)	
e_8	(e_9, X, D)		$(e_{11}, \_, D)$
e_9	$(e_9, \|, D)$		$(e_{10}, \_, D)$
e_{10}		(e_{10}, X, D)	$(e_4, \_, D)$
e_{11}		(e_{11}, X, D)	$(e_f, \_, D)$
$\leftarrow e_f$			

Exemple avec $a = 3$ et $b = 1$: on a donc sur la bande lecture ...$\_$ |||| $\_$ || $\_$....
Initialement la machine de Turing se trouve à l'état initial e_0 et la tête de lecture/écriture sur le premier | du nombre a. Il existe une transition qui va de cette configuration vers l'état e_1 avec le marquage du symbole | avec le symbole X et la tête de lecture/écriture se déplace à droite(...$\_X$ ||| $\_$ || $\_$....).

L'état e_1 est utilisé pour passer tous les autres symboles | sans les marquer jusqu'à ce que la tête de lecture se trouve sur le vide alors la machine de Turing passe à l'état e_2.

La machine de Turing se trouve alors à l'état e_2 et la tête de lecture/écriture sur le premier | du nombre b. Il existe une transition qui va de cette configuration vers l'état e_3 avec le marquage du symbole | avec le symbole X et la tête de lecture/écriture se déplace à droite(...$\_X$ ||| $\_X$ | $\_$....).

L'état e_3 est utilisé pour passer tous les autres symboles | sans les marquer jusqu'à ce que la tête de lecture se trouve sur le vide alors la machine de Turing passe à l'état e_4.

La machine de Turing se trouve alors à l'état e_4 et la tête de lecture/écriture après les deux nombres et un vide |. Il existe une transition qui va de cette configuration vers l'état e_5 avec le marquage du vide avec le symbole | et la tête de lecture/écriture se déplace à gauche (...$\_X$ ||| $\_X$ | $\_$ | $\_$....).

La suite des marquages va se dérouler comme suit :
- ...$\_X$ ||| $\_XX\_$ | $\_$....
- ...$\_X$ ||| $\_XX\_$ || $\_$....
- ...$\_XX$ || $\_XX\_$ || $\_$....
- ...$\_XX$ || $\_XX\_$ ||| $\_$....
- ...$\_XXX$ | $\_XX\_$ ||| $\_$....
- ...$\_XXX$ | $\_XX\_$ |||| $\_$....
- ...$\_XXXX\_XX\_$ |||| $\_$....
- ...$\_XXXX\_XX\_$ ||||| $\_$....

A ce moment, la machine de Turing ne peut plus marquer de symbole |. Elle va se positionner sur le premier symbole du résultat et sur l'état final. Le calcul est terminer.

5.5 Conclusion

Les langages récursivement énumérables sont le dernier type de langages qu'on a abordé dans cette matière. Ils sont générés par des grammaires sans restriction et les mots d'un langage récursivement énumérables peuvent être reconnus par une machine de Turing.

Les machines de Turing peuvent être utilisées pour la reconnaissance, la génération et le calcul. C'est cette dernière fonction qui nous a particulièrement intéressé dans ce chapitre.

5.6 Exercices

Exercice 1 : Faire une machine de Turing qui calcule :

1. $\mid a - b \mid$ avec a et b des entiers positifs ou nuls.
2. $a * b$ avec a et b des entiers positifs ou nuls.

Chapitre 6

Corrigés d'exercices

Corrigés des exercices du chapitre 1

Corrigé de l'exercice 1 :

1. $uv = 11100$
   $u^3 = 111111$
   $u^2v^2 = 1111100100$
   $(uv)^2 = 1110011100$
2. $E1 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
3. $E2 = \{\varepsilon, 00, 01, 10, 11, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111\}$
4. $E3 = \{000, 001, 011, 100, 110, 111\}$
5. $L1L2 = 01^+10^+ = 011^+0^+$
   $(L2)^*L1 = (10^+)^*01^+$
   $L1 \cup C(L1 \cap L2) = L1 \cup C(\emptyset) = L1 \cup A^* = A^*$

Corrigé de l'exercice 2 :

1. $L1 = a(a^*b^*c^*)^*$
2. $L2 = (a^*b^*c^*)^*a$
3. $L3 = (a^*b^*c^*)^*a(a^*b^*c^*)^*$
4. $L4 = (a^*b^*c^*)^*aab(a^*b^*c^*)^*aab(a^*b^*c^*)^*$
5. $L5 = C(L1)$

Corrigé de l'exercice 3 :

1. c)
2. b)c)d)
3. a)b)
4. c)

Corrigé de l'exercice 4 : .

.

	L11={$a^n da^n c$; n>1}	L12={$a^n bcc$; n>1}	L13={$a^{(n+1)} ca^n$; n>0}	L14={$c^m a^n$; m,n>1}	L15=dbadd$^+$c$^+$a	L16=bdadb$^+$(ca)$^+$	L17={$b^n dc^n dcaba$; n>1}	L18=b$^+$ad$^+$b$^+$a	L19={$b^n d^n a^n (b*d*)*a$; n >1}	L20={m/m∈(a*b*d*)* et $\|m\|_a = \|m\|_b = \|m\|_d$}
L1={m/m∈a*cdb* et |m| mod 2 =0}	a	a	a	c	d	b	b	b	b	b
L2= a*c(b*d*)*d	a	a	a	c	b	d	b	b	b	d
L3 = {m/m∈ (a*b*c*d*)* et $\|m\|_c = \|m\|_b$}	a	a	a	c	a	a	d	b	d	a
L4={m/m∈a*(b*c*d*)*d et $\|m\|_a + \|m\|_c = \|m\|_b + \|m\|_d$}	a	a	a	c	d	d	c	b	d	d
L5= a*daa(a*b*c*d*)*bab	d	a	a	c	d	b	c	b	a	b
L6=a$^+$c$^+$bda$^+$	a	a	c	c	c	b	d	a	a	a
L7={a*c^{n}d^n; n>0}	a	a	a	a	c	c	c	d	d	d
L8=(a*b*c*d*)*caab$^+$	a	b	a	a	c	a	a	b	b	b
L9={aca^{n}c^{n}b$^{(n+1)}$a et n>1}	a	c	a	a	c	c	b	b	b	a
L10=c(a U b U c)a*	c	c	a	a	a	a	a	a	a	

Corrigés des exercices du chapitre 2

Corrigé de l'exercice 1 : .

$G_d(L1)$ est une grammaire régulière à droite de $L1$:
$G_d(L1) = (T, N_d, S_d, R_d)$
$T = \{a, b\}$
$N_d = \{S_d, S'_d\}$
$R_d = \{$

$S_d \rightarrow aS_d / aS'_d,$

$S'_d \rightarrow bS'_d / b$
$\}$

$G_g(L1)$ est une grammaire régulière à gauche de $L1$:
$G_g(L1) = (T, N_g, S_g, R_g)$
$T = \{a, b\}$
$N_g = \{S_g, S'_g\}$
$R_g = \{$

$S_g \rightarrow S_g b / S'_g b /,$

$S'_g \rightarrow S'_g a / a$
$\}$

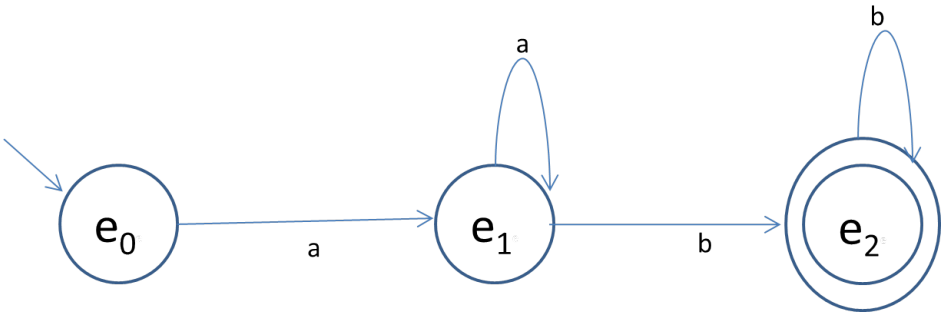


FIGURE 6.1: AEF(L1)

Corrigé de l'exercice 2 :

1. $L = a^*b^*c^*$
2. $L = (a^*b^*)^*$
3. $L = a^*(b^+cb \cup c^*abc)$
4. $L = [a^*(ba \cup bbc^+ \cup cc^+)]^*c$

Corrigé de l'exercice 4 : .

On remarque d'abord que e_7 est un état inaccessible. Il faut donc le supprimer.

Passant maintenant à la représentation tabulaire de l'automate.

	a	b
$\rightarrow e_0$	$\{e_1, e_4\}$	e_1
e_1	e_4	$\{e_1, e_2\}$
e_2	-	e_1
e_3	-	e_5
e_4	$\{e_3, e_6\}$	-
e_5	-	-
$\leftarrow e_6$	e_8	$\{e_5, e_6\}$
e_8	-	-

TABLE 6.1: La représentation tabulaire de l'AEF

On remarque que l'automate est indéterministe. Il faut donc le rendre déterministe puis complet.

	a	b
$\rightarrow e_{0d} = e_0$	e_{2d}	e_{2d}
$e_{1d} = \{e_1, e_4\}$	e_{3d}	e_{4d}
$e_{2d} = e_1$	e_{5d}	e_{4d}
$e_{3d} = \{e_3, e_4, e_6\}$	e_{6d}	e_{7d}
$e_{4d} = \{e_1, e_2\}$	e_{5d}	e_{4d}
$e_{5d} = e_4$	e_{8d}	e_p
$\leftarrow e_{6d} = \{e_3, e_6, e_8\}$	e_{9d}	e_{7d}
$\leftarrow e_{7d} = \{e_5, e_6\}$	e_{9d}	e_{7d}
$\leftarrow e_{8d} = \{e_3, e_6\}$	e_{9d}	e_{7d}
$e_{9d} = e_8$	e_p	e_p
e_p	e_p	e_p

TABLE 6.2: La représentation tabulaire de l'AEF déterministe et complet

On peut alors commencer le déroulement de l'algorithme de minimisation.

— $G1 = \{e_{6d}, e_{7d}, e_{8d}\}$,$G2 = \{e_{0d}, e_{1d}, e_{2d}, e_{3d}, e_{4d}, e_{5d}, e_{9d}, e_p\}$

— $G1 = \{e_{6d}, e_{7d}, e_{8d}\}, G2\_1 = \{e_{0d}, e_{1d}, e_{2d}, e_{4d}, e_{9d}, e_p\}, G2\_2 = \{e_{3d}\}, G2\_3 = \{e_{5d}\}$

— $G1 = \{e_{6d}, e_{7d}, e_{8d}\}, G2\_1\_1 = \{e_{0d}, e_{9d}, e_p\}, G2\_1\_2 = \{e_{1d}, e_{2d}, e_{4d}, e_{5d}, e_p\}$, $G2\_1\_3 = \{e_{2d}, e_{4d}\}, G2\_2 = \{e_{3d}\}, G2\_3 = \{e_{5d}\}$

— $G1 = \{e_{6d}, e_{7d}, e_{8d}\}, G2\_1\_1\_1 = \{e_{0d}\}, G2\_1\_1\_2 = \{e_{9d}, e_p\}, G2\_1\_2 = \{e_{1d}\}$, $G2\_1\_3 = \{e_{2d}, e_{4d}\}, G2\_2 = \{e_{3d}\}, G2\_3 = \{e_{5d}\}$

On obtient alors l'automate à états finis suivant :

$AEF(L) = \{A, Q, I, F, \delta)$
$A = \{a, b\}$

$Q = \{G1, G2\_1\_1\_1, G2\_1\_1\_2, G2\_1\_2, G2\_1\_3, G2\_2, G2\_3\}$
$I = \{G2\_1\_1\_1\}$
$F = \{G1\}$
$\delta = \{$
$(G2\_1\_1\_1\, a) \rightarrow (G2\_1\_2)$,
$(G2\_1\_1\_1\, b) \rightarrow (G2\_1\_3)$,
$(G2\_1\_2, a) \rightarrow (G2\_2)$,
$(G2\_1\_2, b) \rightarrow (G2\_1\_2)$,
$(G2\_1\_3, a) \rightarrow (G2\_3)$,
$(G2\_1\_3, b) \rightarrow (G2\_1\_3)$,
$(G2\_2, a) \rightarrow (G1)$,
$(G2\_2, b) \rightarrow (G1)$,
$(G2\_3, a) \rightarrow (G2\_1\_1\_2)$,
$(G2\_3, b) \rightarrow (G2\_1\_1\_2)$,
$(G1, a) \rightarrow (G2\_1\_1\_2)$,
$(G1, b) \rightarrow (G1)$,
$(G2\_1\_1\_2, a) \rightarrow (G2\_1\_1\_2)$,
$(G2\_1\_1\_2, b) \rightarrow (G2\_1\_1\_2)$,
$\}$

Corrigés des exercices du chapitre 3

Corrigé de l'exercice 1 : .

$G(L1) = (T, N, S, R)$
$T = \{a, b, c, d\}$
$N = \{S, S', S''\}$
$R = \{$
$S \rightarrow S'S'',$
$S' \rightarrow aS'b/ab,$
$S'' \rightarrow cS''d/cd$
$\}$

$AP(L1) = (A, P, Q, e_0, F, \delta)$
$A = \{a, b, c, d\}$
$P = \{a, c\}$
$Q = \{e_0, e_1, e_2\}$
$F = \{e_2\}$
$\delta = \{$
$(e_0, a, \varepsilon) \rightarrow (e_0, a),$
$(e_0, a, a) \rightarrow (e_0, aa),$
$(e_0, b, a) \rightarrow (e_1, \varepsilon),$
$(e_1, b, a) \rightarrow (e_1, \varepsilon),$
$(e_1, c, \varepsilon) \rightarrow (e_1, c),$
$(e_1, c, c) \rightarrow (e_1, cc),$
$(e_1, d, c) \rightarrow (e_2, \varepsilon),$
$(e_2, d, c) \rightarrow (e_2, \varepsilon)$
$\}$

$G(L2) = (T, N, S, R)$
$T = \{a, b, c, d\}$
$N = \{S, S'\}$
$R = \{$
$S \rightarrow aSd/aS'd$
$S' \rightarrow bS'c/bc$
$\}$

$AP(L2) = (A, P, Q, e_0, F, \delta)$
$A = \{a, b, c, d\}$
$P = \{a, b\}$
$Q = \{e_0, e_1\}$
$F = \{e_1\}$
$\delta = \{$

$(e_0, a, \varepsilon) \rightarrow (e_0, a)$,
$(e_0, a, a) \rightarrow (e_0, aa)$,
$(e_0, b, a) \rightarrow (e_0, ab)$,
$(e_0, b, b) \rightarrow (e_0, bb)$
$(e_0, c, b) \rightarrow (e_1, \varepsilon)$,
$(e_1, c, b) \rightarrow (e_1, \varepsilon)$,
$(e_1, d, a) \rightarrow (e_1, \varepsilon)$
}

Corrigé de l'exercice 2 : .

$L1 = \{mcm^r / m \in (a^*b^*)^*\}$
$L2 = \{m/m \in (a^*b^*)^*(c^*d^*)^*$ et $\mid m \mid_a = \mid m \mid_b$ et $\mid m \mid_c = \mid m \mid_d\}$

$G(L1) = (T, N, S, R)$
$T = \{a, b, c\}$
$N = \{S\}$
$R = \{S \rightarrow aSa/bSb/c\}$

$AP(L1) = (A, P, Q, e_0, F, \delta)$
$A = \{a, b, c\}$
$P = \{a, b\}$
$Q = \{e_0, e_1\}$
$F = \{e_1\}$
$\delta = \{$
$(e_0, a, \varepsilon) \rightarrow (e_0, a)$,
$(e_0, a, a) \rightarrow (e_0, aa)$,
$(e_0, a, b) \rightarrow (e_0, ba)$
$(e_0, b, \varepsilon) \rightarrow (e_0, b)$
$(e_0, b, a) \rightarrow (e_0, ab)$,
$(e_0, b, b) \rightarrow (e_0, bb)$
$(e_0, c, \varepsilon) \rightarrow (e_1, \varepsilon)$,
$(e_0, c, a) \rightarrow (e_1, a)$,
$(e_0, c, b) \rightarrow (e_1, b)$,
$(e_1, b, b) \rightarrow (e_1, \varepsilon)$,
$(e_1, a, a) \rightarrow (e_1, \varepsilon)$
}

$G(L2) = (T, N, S, R)$
$T = \{a, b, c, d\}$
$N = \{S, S', S''\}$
$R = \{$

$S \rightarrow S'S''$,
$S' \rightarrow aS'bS'/bS'aS'/\varepsilon$,
$S'' \rightarrow cS''dS''/dS''cS''/\varepsilon$
$\}$

$AP(L2) = (A, P, Q, e_0, F, \delta)$
$A = \{a, b, c, d\}$
$P = \{a, b, c, d\}$
$Q = \{e_0, e_1\}$
$F = \{e_0, e_1\}$
$\delta = \{$
$(e_0, a, \varepsilon) \rightarrow (e_0, a)$,
$(e_0, a, a) \rightarrow (e_0, aa)$,
$(e_0, b, a) \rightarrow (e_0, \varepsilon)$,
$(e_0, b, \varepsilon) \rightarrow (e_0, b)$,
$(e_0, b, b) \rightarrow (e_0, bb)$,
$(e_0, a, b) \rightarrow (e_0, \varepsilon)$,
$(e_0, c, \varepsilon) \rightarrow (e_1, c)$,
$(e_0, d, \varepsilon) \rightarrow (e_1, d)$,
$(e_1, c, c) \rightarrow (e_1, cc)$,
$(e_1, c, \varepsilon) \rightarrow (e_1, c)$,
$(e_1, c, d) \rightarrow (e_1, \varepsilon)$,
$(e_1, d, c) \rightarrow (e_1, \varepsilon)$,
$(e_1, d, d) \rightarrow (e_1, dd)$
$(e_1, d, \varepsilon) \rightarrow (e_1, d)$
$\}$

Corrigé de l'exercice 4 :

1. a)b)c)d)
2. b)
3. a)
4. a)
5. b)

Corrigés des exercices du chapitre 4

Corrigé de l'exercice 1 : .

$L2 = \{a^{3n}b^{2n}c^n, n > 0\}$

	a	b	c	A	B	C	$\sharp$
$\rightarrow e_0$	(e_1, A, D)				(e_7, B, D)		
$\rightarrow e_1$	(e_2, A, D)						
$\rightarrow e_2$	(e_3, A, D)						
e_3	(e_3, a, D)	(e_4, B, D)			(e_3, B, D)		
e_4		(e_5, B, D)					
e_5		(e_5, b, D)	(e_6, C, G)			(e_5, C, D)	
e_6	(e_6, a, G)	(e_6, b, G)		(e_0, A, D)	(e_6, B, G)	(e_6, C, G)	
e_7					(e_7, B, D)	(e_7, C, D)	$(e_f, \sharp, D)$
$\leftarrow e_f$							

$L3 = \{a^{2^n}, n \geq 0\}$

	a	A	$\sharp$
$\rightarrow e_0$	(e_1, A, D)	(e_0, A, D)	
e_1	(e_2, a, D)	(e_1, A, D)	$(e_f, \sharp, D)$
e_2	(e_3, A, D)	(e_2, A, D)	$(e_4, \sharp, G)$
e_3	(e_2, A, D)	(e_3, A, D)	
e_4	(e_4, A, G)	(e_4, A, G)	$(e_0, \sharp, D)$
$\leftarrow e_f$			

Corrigés des exercices du chapitre 5

Corrigé de l'exercice 1 :

1. $\mid a - b \mid$ avec a et b des entiers positifs ou nuls.

	$\mid$	X	$\_$
$\rightarrow e_0$	(e_1, X, D)		
e_1	$(e_1, \mid, D)$		(e_2, X, D)
e_2	(e_3, X, D)		
e_3	$(e_3, \mid, D)$		$(e_4, \_, D)$
e_4	$(e_4, \mid, D)$		$(e_5, \mid, G)$
e_5	$(e_5, \mid, G)$		$(e_6, \_, G)$
e_6	$(e_6, \mid, G)$	(e_6, X, G)	$(e_7, \_, D)$
e_7	(e_8, X, D)	(e_7, X, D)	$(e_f, \_, D)$
e_8	$(e_8, \mid, D)$	(e_9, X, D)	$(e_4, \_, D)$
e_9	(e_6, X, G)	(e_9, X, D)	$(e_4, \_, D)$
$\leftarrow e_f$			

Bibliographie

[1] A. V. Aho and J. D. Ullman. *Principles of Compiler Design*. Addison-Wesley, 1977.

[2] J-M. Autebert. *Théorie des langages et des automates*. MASSON, 1994.

[3] O. Carton. *Langages formels, Calculabilité et complexité*. Vuibert, 2014.

[4] M. Noureddine. *Théorie des langages*. Office des publications universitaires, 1991.