

**L3 MIASHS 2021-2022**  
**Parcours Informatique**

**MI0C602T**

**Théorie des langages -**  
**Expressions régulières, grammaires,**  
**automates, analyse des langages**

Pierre-Jean Charrel  
UT2J – IRIT / SMART  
[pierre-jean.charrel@univ-tlse2.fr](mailto:pierre-jean.charrel@univ-tlse2.fr)

# Sommaire

<b>1. Expressions régulières</b>	<b>3</b>
<b>2. Grammaires</b>	<b>16</b>
<b>3. Principes de l'analyse descendante</b>	<b>50</b>
<b>4. Automates</b>	<b>71</b>
<b>5. Mise en œuvre de l'analyse descendante</b>	<b>144</b>
<b>6. Conclusion</b>	<b>162</b>

# Sommaire

<b>1. Expressions régulières</b>	<b>3</b>
1.1 Problèmes, mots et langages	4
1.2 Vocabulaire / Alphabet	7
1.3 Mot	7
1.4 Langage	10
1.5 Opérations sur les langages	11
2. Grammaires	16
3. Principes de l'analyse descendante	50
4. Automates	71
5. Mise en œuvre de l'analyse descendante	144
6. Conclusion	162

# 1.1 Problèmes, mots et langages

## Machines et algorithmes

L'informatique a pour but de résoudre des problèmes au moyen d'algorithmes.

Question : tous les problèmes se résolvent-ils au moyen d'un algorithme ?

Réponse : en lien avec l'analogie entre le cerveau humain et l'ordinateur

Un ordinateur suit toujours le modèle de J. Von Neumann (1940)

Unité centrale (UC) + mémoire + registres dont compteur de programme

UC : répète le traitement séquentiel

- 1- décode le compteur (adresse mémoire contient une instruction)
- 2- exécute l'instruction et met à jour les registres
- 3- incrémente le compteur

Il existe des traitements parallèles, quantiques, mais pas forcément meilleurs

# Algorithme

## Enchaînement d'actions élémentaires

- utilise toujours le **même jeu** d'instructions
- est décrit dans un **formalisme** simple valable pour **n'importe quel** langage de programmation

Origine du premier formalisme : **machine de Turing.**

## Principes d'une machine de Turing

1. données et instructions = suites de caractères = **mots**
2. **caractères** d'un **mot** inscrits en **séquence** sur un support (image d'un ruban ou d'une bande magnétique)
3. un mot est **lu** et **reconnu caractère par caractère** ou **écrit** caractère par caractère par une "**tête**" de lecture-écriture
4. La machine ne sait exécuter que **4 actions élémentaires** :
  - **lecture, écriture d'un caractère** sur le ruban,
  - **déplacement** de la tête de lecture le long du ruban, d'un caractère au suivant, en avant ou en arrière
  - **transition entre états** de la tête de lecture-écriture

Résoudre un problème revient donc à

- résoudre un problème de traitement d'un ensemble de mots
- donner la réponse sous la forme d'un mot parmi  
"oui", "non", "erreur", ...

→ Trouver tous les mots dont la réponse est "oui"

Un ensemble de mots définit un langage

→ Résoudre un problème revient donc à  
reconnaître un langage

Quels outils permettent de décrire un langage ?

## 1.2 Vocabulaire / Alphabet

### Définition

Un **vocabulaire**, ou **alphabet**, noté  $V$ , est un ensemble fini, non vide, d'éléments, appelés **lettres** ou **symboles**

### Exemples :

$$V_1 = \{ a, b, \dots, z \}$$

$$V_2 = \{ 0, 1 \}$$

$$V_3 = \{ 0, 1, \text{et}, \text{ou}, \text{non}, p \}$$

## 1.3 Mot

### Définition

Un **mot** est une **suite finie de lettres**.

On note  **$V^*$  l'ensemble des mots** qui utilisent les lettres de l'alphabet  $V$

### Exemple :

$p \quad \text{et} \quad 0 \quad \text{ou} \quad 1$       mot sur  $V_3$

## Cas particulier

Le **mot vide**, qui ne contient aucun caractère est noté  $\epsilon$  et appartient à **tous** les  $V^*$

## Remarque

$$V^* = \bigcup V^n \quad (n = 0, \dots, \omega)$$

$V^n$  = mots de longueurs  $n$ , produit cartésien  $n$  fois de  $V$

## Définition récursive d'un mot sur $V$

- (i)  $\epsilon$  est un **mot**
- (ii) si  $w$  est un mot et  $x$  appartient à  $V$   
alors  $x \cdot w$  est aussi un **mot**,  
où "." est un **opérateur** qui **ajoute** une **lettre  $x$**  au **début** d'un **mot  $w$**
- (iii) **rien** n'est un **mot hors** (i) et (ii)



# Concaténation de mots " $\wedge$ "

## Définition

La **concaténation de mots** est une loi de composition interne sur  $V^*$  :

$(v, w) \rightarrow v \wedge w$  on place toutes les lettres de  $v$  **devant**  $w$

## Définition récursive

(i)  $\varepsilon \wedge v = v$

(ii)  $(x \cdot v) \wedge w = x \cdot (v \wedge w)$

"  $\wedge$  " est associative et possède un élément neutre

$V^*$  muni de sa loi de composition "  $\wedge$  " est un monoïde.

Le monoïde est "libre" : **pas d'équivalence** entre les **mots**

C'est-à-dire :

- **toutes les suites de lettres sont différentes**
- **Il n'existe qu'une seule façon d'écrire un mot**

## 1.4 Langage

### Définition

Un **langage** sur un vocabulaire  $V$  est une **partie** quelconque de  $V^*$

### Exemples

- $\emptyset$  est un langage pour tous les alphabets  $V$
- $\{\epsilon\}$  est un langage pour tous les alphabets  $V$
- $\{a, b, \dots, z\}$  est un langage sur  $V_1 \rightarrow$  mots de longueur 1
- $\{\text{représentations binaires d'entiers pairs}\}$  est un langage sur  $V_2$
- $\{\text{assemblage de parenthèses bien équilibré}\}$  est un langage sur  $\{ (, ) \}$  langage des mots de Dyck

**Problème** : trouver un moyen fini (qui tient en une suite finie de symboles) pour décrire un ensemble potentiellement infini

## 1.5 Opérations sur les langages

## Les constructions suivantes sont des langages

- $L_1 \cup L_2$
- $L_1 \cdot L_2 = \{ v \wedge w, v \text{ mot de } L_1, w \text{ mot de } L_2 \}$
- Fermeture itérative (de Kleene) de  $L$   
 $L^* = \{ w, \exists k \text{ non nul tel que } w = w_1 \wedge w_2 \wedge \dots \wedge w_k$   
et  $\forall i, w_i \text{ mot de } L \}$
- Complément de  $L$  à  $V^*$

# On note

$L^n = L . L . \dots . L$  la concaténation  $n$  fois de  $L$

$$L^* = \bigcup L^n \quad (n = 1, \dots, \text{infini})$$

## Langage régulier

- (i)  $\phi$  et  $\{\varepsilon\}$  sont des langages réguliers
- (ii) Pour tout  $x$  de  $V$ ,  $\{x\}$  est un langage régulier
- (iii) Si  $L_1$  et  $L_2$  sont des langages réguliers,
  - $L_1 \cup L_2$
  - $L_1 \cdot L_2$
  - $L_1^*$sont des langages réguliers

## Expression régulière

Expression littérale qui désigne – **dénote** – un langage régulier

- (i)  $\phi$  et  $\varepsilon$  sont des expressions régulières
- (ii) si  $a$  et  $b$  sont des expressions régulières, alors
  - $(a + b)$  ou  $a + b$
  - $(a \cdot b)$  ou  $(a b)$  ou  $a b$
  - $(a)^*$  ou  $a^*$sont aussi des expressions régulières

## Notation – expressions régulières qui dénotent les langages

$\phi$  dénote  $L(\phi)$

$\varepsilon$  dénote  $L(\{\varepsilon\})$

$a$  dénote  $L(\{a\})$  pour tout  $a$  de  $V$

$a + b$  dénote  $L(\{a\}) \cup L(\{b\})$

$ab$  dénote  $L(\{a\}) \cdot L(\{b\})$

$a^*$  dénote  $L(\{a\})^*$

### Exemple :

Expression régulière qui dénote le langage  $L$  :

" tous les mots sur l'alphabet  $V = \{a, b, c\}$  contenant au moins une fois 4  
"a" qui se suivent "

$(a + b + c)^* a a a a (a + b + c)^*$

### Théorème

Un langage est régulier si et seulement si il est dénoté par une expression régulière

# Propriétés

Soit  $a$  et  $b$  2 expressions régulières

1.  $a + b = b + a$
2.  $a + \phi = \phi + a = a$
3.  $a + a = a$
4.  $(a + b) + c = a + (b + c)$
5.  $a \varepsilon = \varepsilon a = a$
6.  $a \phi = \phi a = \phi$
7.  $(a b) c = a (b c)$
8.  $a (b + c) = a b + a c$
9.  $(a + b) c = a c + b c$
10.  $a^* = a^* a^* = (a^*)^* = (\varepsilon + a)^*$

$$11. \phi^* = \varepsilon^* = \varepsilon$$

$$12. (a + b)^* = (a^* + b^*)^* = (a^* b^*)^*$$

$$13. a^* a = a a^*$$

$$14. a (b a)^* = (a b)^* a$$

**Preuve :**

Soit e14 mot du langage L14 dénoté par l'expression régulière

$$e14 = a (b a)^*$$

Alors il existe un nombre entier " n " tel que

$$\begin{aligned} e14 &= a_0 (b_0 a_1) \dots (b_{n-1} a_n) \\ &= (a_0 b_0) (a_1 b_1) \dots (a_{n-1} b_{n-1}) a_n \end{aligned}$$

$$e14 = (a b)^* a$$

**Notations**

$$a^+ = a + a^2 + \dots + a^k + a^{k+1} a^*$$

$$a^* = \varepsilon + a a^* = \varepsilon + a^+$$

# Sommaire

1.	Expressions régulières	3
2.	<b>Grammaires</b>	<b>16</b>
	2.0 Introduction	17
	2.1 Définition	20
	2.2 Mot engendré par une grammaire	23
	2.3 Dérivation immédiate	25
	2.4 Dérivation	26
	2.5 Langage engendré par une grammaire	29
	2.6 Types de règles - types de grammaires – hiérarchie de Chomsky	30
	2.7 Grammaire et définition récursive	36
	2.8 Arbre de dérivation dans une grammaire hors-contexte	38
	2.9 Ambiguïté syntaxique	45
3.	Principes de l'analyse descendante	50
4.	Automates	71
5.	Mise en œuvre de l'analyse descendante	144
6.	Conclusion	162
7.	Bibliographie	176



## 2.0 Introduction

Dans la **langue naturelle**, une **phrase** S (*Sentence*) est composée d'un **sujet** suivi d'un **verbe** suivi d'un **complément**.

**Exemple :**

la jeune pianiste virtuose	joue	sans partition
----------------------------	------	----------------

On peut lire une construction basée sur des **termes qui obéissent aux règles de syntaxe d'une grammaire**

*phrase*            →        *sujet*   *verbe*   *complément*

Ensuite, il faut "**expliquer**" les termes **sujet**, **verbe**, **complément** *pour justifier la structure de l'exemple.*

## Exemple :

*sujet*

→ *article adjectif nom*  
| *article nom adjectif*  
| **article adjectif nom adjectif**  
| *article nom*

*article ...*

→ **le** | **la** | **un** | **des** | **l'**

*adjectif*

→ **studieux** | *qualité* | *couleur*

*qualité*

→ **virtuose** | **débutant** | **jeune**

*nom*

→ **violoniste, violoncelliste, pianiste**

ainsi de suite ...

Expression conditionnelle dans le langage C :

**if** ( *expression* ) *instruction*

**Exemple :**

<b>if</b>	<b>(</b>	<b><i>x &lt; 10</i></b>	<b>)</b>	<b><i>a = a + b</i></b>
-----------	----------	-------------------------	----------	-------------------------

Il faut encore « **expliquer** » *expression* et *instruction* ...

Dans une langue ou un langage, 2 catégories de termes

- les **terminaux** :

**termes** qui permettent de construire les phrases des langages (Français, C) :

**le, la, if** ...

- les **non-terminaux**

**termes** qu'il faut « **expliquer** » *pour construire une phrase syntaxiquement correcte:*

*sujet, adjectif, instruction*

## 2.1 Définition

Une **grammaire**  $G$  est un quadruplet

$$G = ( V_N, V_T, S, R )$$

où :

- $V_N$  : vocabulaire (alphabet) **non terminal**
- $V_T$  : vocabulaire (alphabet) **terminal**
- $S \in V_N$  : axiome ou **racine**
- $R$  : ensemble de règles de **production** ou de **réécriture**
- $V_N \cap V_T = \emptyset$

Une **règle de réécriture** est un couple de mots  $(v, w)$

où :  $v \in (V_N \cup V_T)^* - \{\varepsilon\}$  et  $w \in (V_N \cup V_T)^*$

on note :  $v \rightarrow w$

On dit que "**v se réécrit w**" ou "**v produit w**" ou "**v donne w**"  
dans la grammaire G

**v est la partie gauche de la règle, w la partie droite**

## Notation

S'il existe plusieurs règles de réécriture de même partie gauche v :

$v \rightarrow w_1, v \rightarrow w_2, \dots, v \rightarrow w_n,$

on peut écrire les n règles en 1 seule ligne :

$v \rightarrow w_1 \mid w_2 \mid \dots \mid w_n$  " v se réécrit  $w_1$  **ou**  $w_2$  **ou** ... **ou**  $w_n$  "

## Exemple

$$G_0 = (V_N, V_T, S, R)$$

$$V_N = \{S, A, B\}$$

$$V_T = \{0, 1\}$$

$S \in V_N$ : racine

$$R = \{(1), (2), (3), (4), (5)\}$$

$$= \{ S \rightarrow 0 A 1 B \quad (1)$$

$$1 B \rightarrow 1 A B B \quad (2)$$

$$1 A \rightarrow A 1 \quad (3)$$

$$1 B \rightarrow 1 1 \quad (4)$$

$$0 A \rightarrow 0 0 \quad (5)$$

}

## 2.2 Mot engendré par une grammaire

Intuitivement, un mot  $w \in V_T^*$  est **engendré par une grammaire**  $G$  si on peut **l'assembler** au bout d'un **nombre fini de réécritures** à partir de la **racine**  $S$ .

## Exemple intuitif

$G_0 = (V_N, V_T, S, R)$

$R =$

$\{ S \rightarrow 0 A 1 B (1)$   
 $1 B \rightarrow 1 A B B (2)$   
 $1 A \rightarrow A 1 (3)$   
 $1 B \rightarrow 1 1 (4)$   
 $0 A \rightarrow 0 0 (5)$   
 $\}$

le mot  $v = 0 0 0 0 1 1 1 1$   
 est engendré par cette  
 grammaire ?

$S \rightarrow 0 A 1 B \quad 1$   
 $\rightarrow 0 A 1 A B B \quad 2$   
 $\rightarrow 0 A A 1 B B \quad 3$   
 $\rightarrow 0 A A 1 A B B B \quad 2$   
 $\rightarrow 0 A A A 1 B B B \quad 3$   
 $\rightarrow 0 A A A 1 1 B B \quad 4$   
 $\rightarrow 0 A A A 1 1 1 B \quad 4$   
 $\rightarrow 0 A A A 1 1 1 1 \quad 4$   
 $\rightarrow 0 0 A A 1 1 1 1 \quad 5$   
 $\rightarrow 0 0 0 A 1 1 1 1 \quad 5$   
 $\rightarrow 0 0 0 0 1 1 1 1 \quad 5$



## 2.3 Dérivation immédiate

$v \rightarrow w$

**si et seulement si :**

$x, y \in V_T$  et  $u, v \in (V_N \cup V_T)^*$

- $v = x u y, u \neq \varepsilon$
- $w = x v y$
- $(u \rightarrow v) \in R$

## 2.4 Dérivation

**La dérivation** est la fermeture réflexive et transitive de  $\rightarrow$   
 $v \rightarrow^* w$

**si et seulement si :**

Il existe  $k \geq 0$  et une suite  $v_0, v_1, \dots, v_k$ , d'éléments de

$(V_N \cup V_T)^*$  tels que

$$v = v_0$$

$$w = v_k$$

$$\forall i, 0 \leq i \leq k - 1, \quad v_i \rightarrow v_{i+1}$$

## Exemples sur G0

- S → 0 A 1 B
- 0 A 1 A B B
- 0 A A 1 B B
- 0 A A 1 A B B B
- 0 A A A 1 B B B
- 0 A A A 1 1 B B
- 0 A A A 1 1 1 B
- 0 A A A 1 1 1 1
- 0 0 A A 1 1 1 1
- 0 0 0 A 1 1 1 1
- 0 0 0 0 1 1 1 1

S → \* 0 0 0 0 1 1 1 1

- 0 A 1 A B B → \* 0 0 0 A 1 1 1 1
- 0 A 1 A B B → \* 0 A 1 A B B

## Remarque

**$v \rightarrow^* v$  : autrement dit, la relation  $\rightarrow^*$  est réflexive.**

Démonstration :

dans la définition, prendre  $k = 0$ , alors :

$$v = v_0 = v_k = v$$

La condition

$$\forall i, 0 \leq i \leq k - 1 \Rightarrow v_i \rightarrow v$$

est vérifiée car aucun  $i$  ne satisfait  $0 \leq i \leq -1$

## 2.5 Langage engendré par une grammaire

Soit une grammaire

$$G = (V_N, V_T, S, R)$$

$$L(G) = \{v \in V_T^* ; S \xrightarrow{*} v\}$$

**Exemple sur G0:**

$$v = 00001111 \in L(G_0)$$

## 2.6 Types de règles – types de grammaires – hiérarchie de Chomsky

**Type 0** : toutes les règles

**Type 1** : non raccourcissantes

$v \rightarrow w$  avec  $\text{longueur}(v) \leq \text{longueur}(w)$

**Type 2** : hors-contexte

$X \rightarrow w$  avec  $X \in V_N$

**Type 3** : linéaires.

linéaires gauches

$X \rightarrow xw$  avec  $X \in V_N$ ,  $x \in V_T$ , et  $w \in (V_N \cup V_T)^*$

linéaires droites

$X \rightarrow wx$  avec  $X \in V_N$ ,  $x \in V_T$ , et  $w \in (V_N \cup V_T)^*$

## Types de grammaires

Une grammaire est de type T si **toutes** ses règles de réécriture sont de type T

## Exemples

- Grammaire de typer 1

$$\begin{aligned} G_1 &= \{ V_N, V_T, S, R \} \\ V_N &= \{ S, B, C \} \\ V_T &= \{ a, b, c \} \\ R &= \{ \begin{array}{ll} S & \rightarrow a S B C \\ S & \rightarrow a b C \\ b B & \rightarrow b b \\ b C & \rightarrow b c \\ c C & \rightarrow c c \end{array} \\ &\quad \} \end{aligned}$$

- Grammaire de type 2, ou "hors-contexte"

$$G_2 = \{ V_N, V_T, S, R \}$$

$$V_N = \{ S \}$$

$$V_T = \{ a, b \}$$

$$R = \left\{ \begin{array}{l} S \rightarrow a S b \\ S \rightarrow a b \end{array} \right\}$$

$$S \rightarrow a b$$

$$S \rightarrow a S b \quad \rightarrow a a S b b \quad \rightarrow a a a b b b$$

$$S \rightarrow a S b \quad \rightarrow a a S b b \quad \rightarrow a a a S b b b \rightarrow a a a a b b b b$$

$$L(G_2) = \{ a^n b^n; n \geq 1 \}$$



- Grammaire de type 3, ou linéaire

$$G3 = \{ V_N, V_T, S, R \}$$

$$V_N = \{ S, A \}$$

$$V_T = \{ a, b \}$$

$$R = \left\{ \begin{array}{ll} S & \rightarrow a S \\ S & \rightarrow a A \\ A & \rightarrow b A \\ A & \rightarrow b \end{array} \right.$$

(linéaire **gauche**)

exemples de dérivation

$$S \rightarrow a A \rightarrow a b$$

$$S \rightarrow a S \rightarrow a a S \rightarrow a a a S \rightarrow a a a a A \rightarrow a a a a b$$

$$S \rightarrow a S \rightarrow a a A \rightarrow a a b$$

$$S \rightarrow a S \rightarrow a a A \rightarrow a a b A \rightarrow a a b b A \rightarrow a a b b b$$

$$L(G3) = \{ a^n b^m ; n, m \geq 1 \}$$

langage dénoté par l'expression régulière  $a a^* b b^* = a^+ b^+$

**Type i étendu = type i + règles avec  $\varepsilon$  possible en partie droite**

Grammaires de type hors-contexte ( type 2 ) étendu

$G_{21} = \{ V_N, V_T, S, R \}$

$V_N = \{ S \}$

$V_T = \{ a, b \}$

$R = \left\{ \begin{array}{ll} S & \rightarrow a S b \\ S & \rightarrow \varepsilon \end{array} \right\}$

$S \rightarrow \varepsilon$

$S \rightarrow a S b \quad \rightarrow a a S b b \quad \rightarrow a a b b$

$S \rightarrow a S b \quad \rightarrow a a S b b \quad \rightarrow a a a S b b b \quad \rightarrow a a a b b b$

$L(G_{21}) = \{ a^n b^n; n \geq 0 \}$

# Grammaire de type linéaire ( type 3 à étendu

G31 = {  $V_N$ ,  $V_T$ , S, R }

$V_N$  = { S, A }

$V_T$  = { a, b }

R = {  
     S       $\rightarrow$  a S  
     S       $\rightarrow$  b A  
     S       $\rightarrow$   $\varepsilon$   
     A       $\rightarrow$  b A  
     A       $\rightarrow$   $\varepsilon$   
     }

linéaire gauche

S $\rightarrow$ a S	$\rightarrow$ a		
S $\rightarrow$ a S	$\rightarrow$ a a S	$\rightarrow$ a a	
S $\rightarrow$ a S	$\rightarrow$ a a S	$\rightarrow$ a a a S	$\rightarrow$ a a a b A $\rightarrow$ a a a b
S $\rightarrow$ a S	$\rightarrow$ a a S	$\rightarrow$ a a a S	$\rightarrow$ a a a b A
	$\rightarrow$ a a a b b A	$\rightarrow$ a a a b b	
S $\rightarrow$ $\varepsilon$			

L (G31) = a \* b \*

## 2.7 Grammaire et définition récursive

### Exemple

Langage des mots de Dyck ou des  
Structures de Parenthèses Equilibrées (SPE)

(1)  $\varepsilon$  est une SPE

(2) Schéma d'induction :

- Si A est une SPE, alors ( A ) est une SPE
- Si A et B sont des SPE, alors A B est une SPE

(3) Rien n'est une SPE hormis par (1) et (2)

# traduction sous forme de grammaire

1 17 01 2022 8h20 – 10h20

$$G_{\text{Dyck}} = \{ V_N, V_T, S, R \}$$

$$V_N = \{ S \}$$

$$V_T = \{ (, ) \}$$

$$R = \left\{ \begin{array}{ll} S & \rightarrow \varepsilon \quad (1) \\ S & \rightarrow ( S ) \quad (2) \\ S & \rightarrow S S \quad (3) \end{array} \right\}$$

Exemple de dérivation

$$\begin{aligned} S &\rightarrow ( S ) \rightarrow (( S )) \rightarrow (( S S )) \rightarrow ((( S ) S )) \\ &\rightarrow ((( ( ) ( S ) )) \rightarrow ((( ( ) ( S S ) )) \\ &\rightarrow ((( ( ) ( ( S ) S ) )) \rightarrow ((( ( ) ( ( ) S ) )) \rightarrow ((( ( ) ( ( ) ( ) ) )) \end{aligned}$$

## 2.8 Arbre de dérivation dans une grammaire hors-contexte

Soit  $G = (V_N, V_T, S, R)$  une grammaire hors-contexte

**Un arbre de dérivation** dans  $G$  pour  $v \in V_T^*$  est un arbre :

- **Racine** :  $S$
- **Feuilles** : symboles terminaux ou  $\varepsilon$
- **Sommets** : symboles non terminaux

$v =$  **concaténation** des feuilles

Si un sommet  $N$  a pour descendants immédiats

$N_1, N_2, \dots, N_k,$

alors

$N \rightarrow N_1 N_2 \dots N_k$  est une **règle** de  $R$

## Théorème

Si  $v \in L(G)$

alors il existe un **arbre** de dérivation A pour **v** dans G

## Démonstration

- $\Rightarrow$  : récurrence sur le nombre de pas dans la dérivation  
(démontrer que pour tout non-terminal X et tout mot v,  
si  $X \rightarrow^* v$ ,  
alors il existe un arbre de dérivation pour v de racine X)
- $\Leftarrow$  : récurrence sur la profondeur de l'arbre

## Exemple 1

$G_{21} = \{ V_N, V_T, S, R \}$

$V_N = \{ S \}$

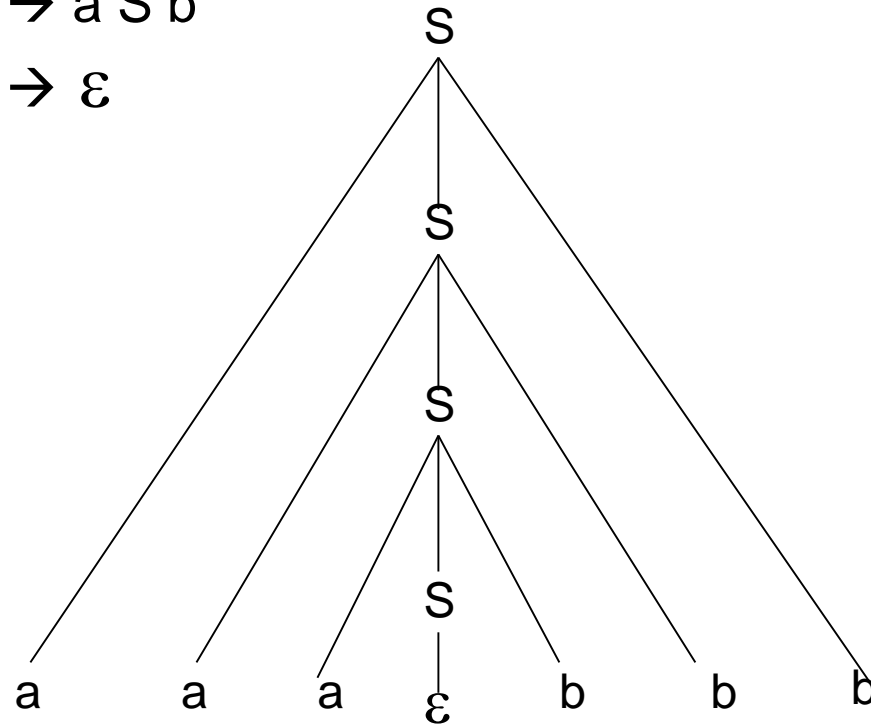
$V_T = \{ a, b \}$

$R = \left\{ \begin{array}{l} S \\ S \\ \varepsilon \end{array} \right\}$

$\rightarrow a S b$

$\rightarrow \varepsilon$

$v = a a a b b b$   
appartient-il à  $L(G_{21})$  ?



Concaténation des feuilles :  
 $a a a \varepsilon b b b = a a a b b b$

Réponse ;  
**OUI**



## Exemple 2

G22 =  $(V_N, V_T, S, R)$

$V_N = \{S, SN, SV, Det, N', N, Vt, Vi\}$

$V_T = \{\text{qui, un, le, chat, garçon, lait, ballon, regarde, boit, frappe, boue}\}$

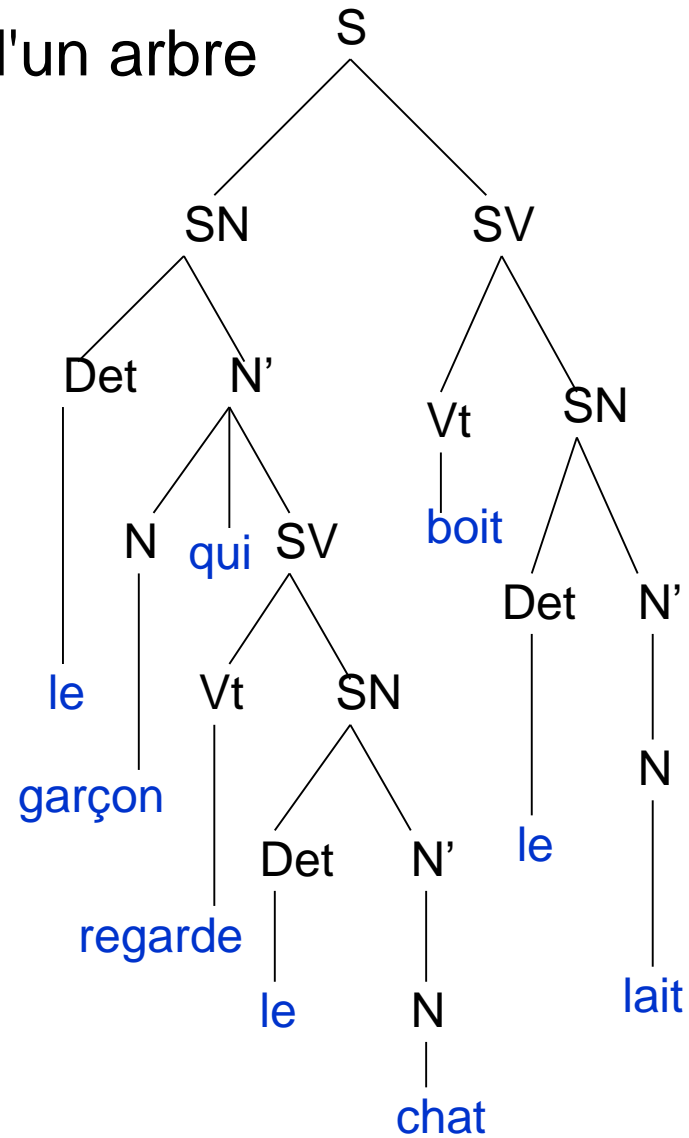
$R = \{$

S	$\rightarrow$ SN SV
SN	$\rightarrow$ Det N'
N'	$\rightarrow$ N   N qui SV
SV	$\rightarrow$ Vt SN   Vi
Det	$\rightarrow$ un   le
N	$\rightarrow$ chat   garçon   lait   ballon
Vt	$\rightarrow$ regarde   boit   frappe
Vi	$\rightarrow$ boue

$\}$

v = le garçon qui regarde le chat boit le lait  
généré par G ?

## Construction d'un arbre de dérivation



Réponse ;  
OUI

## Symbole récursif

**SV**       $\rightarrow$  V SN                       $\rightarrow$  V Det N'                       $\rightarrow$  V Det N **qui** **SV**

donc

**SV**       $\rightarrow$  \* V Det N **qui** **SV**

## **SV est récursif**

d'où :

**SV**       $\rightarrow$  \* V Det N **qui** **SV**  
             $\rightarrow$  \* V Det N **qui** V Det N **qui** **SV**  
             $\rightarrow$  \*  
            ...  
             $\rightarrow$  \* V Det N **qui** V Det N **qui** ... V Det N **qui** **SV**

Langage infini

Une sous-expression E d'un mot v **est un constituant de type X** si et seulement si E est la concaténation des feuilles d'un **sous-arbre** dans l'arbre de dérivation de E de **racine X**

## Exemples

regarde le chat

est un constituant de **type SV**

garçon qui regarde le chat

est un constituant de **type N'**

garçon qui regarde le chat boit le lait

**n'est pas** un constituant

On peut écrire le mot w en indexant ses constituants :

$$[ [ [ \text{le} ]_{\text{Det}} [ [ \text{garçon} ]_{\text{N}} \text{ qui } [ [ \text{regarde} ]_{\text{Vt}} [ [ \text{le} ]_{\text{Det}} [ [ \text{chat} ]_{\text{N}} ]_{\text{N}'} ]_{\text{SN}} ]_{\text{SV}} ]_{\text{N}'} ]_{\text{SN}}$$

$$[ [ \text{boit} ]_{\text{Vt}} [ [ \text{le} ]_{\text{Det}} [ [ \text{lait} ]_{\text{N}} ]_{\text{N}'} ]_{\text{SN}} ]_{\text{SV}} ]_{\text{S}}$$

## 2.9 Ambiguïté syntaxique

Un mot  $v \in L(G)$  est dit **ambigu** si et seulement si  $v$  admet **plusieurs arbres** de dérivation

$G$  est ambiguë ssi elle engendre des **mots ambigus**

$L$  est ambigu ssi  $L$  n'admet **que des grammaires ambiguës**

**Degré d'ambiguïté d'un mot**

**Nombre d'arbres** de dérivations admis par ce mot

## Exemple 1

$G_{23} = (V_N, V_T, S, R)$

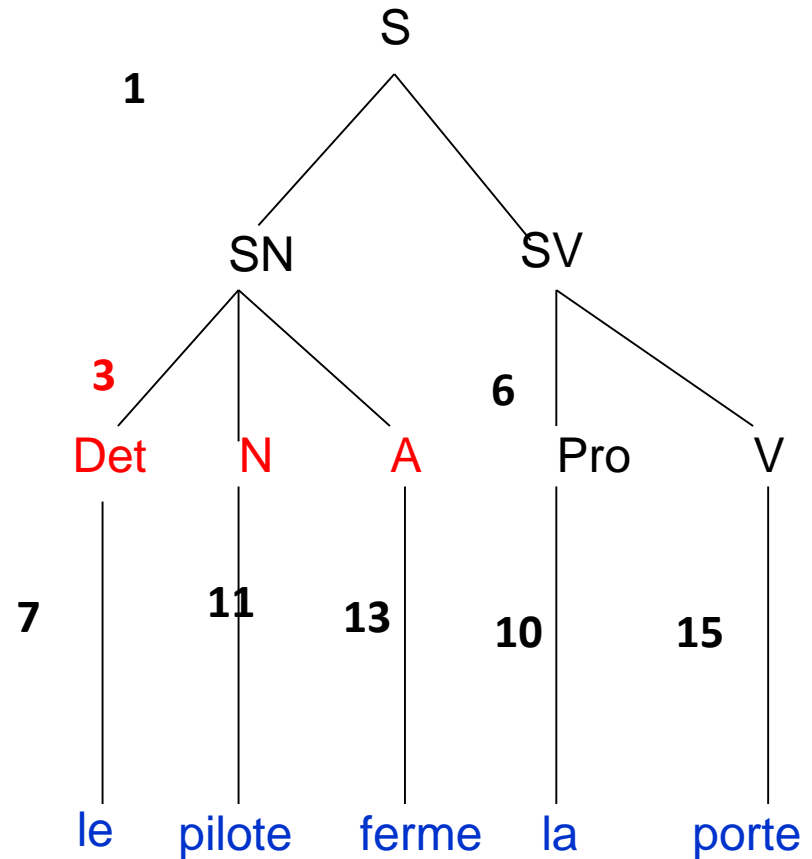
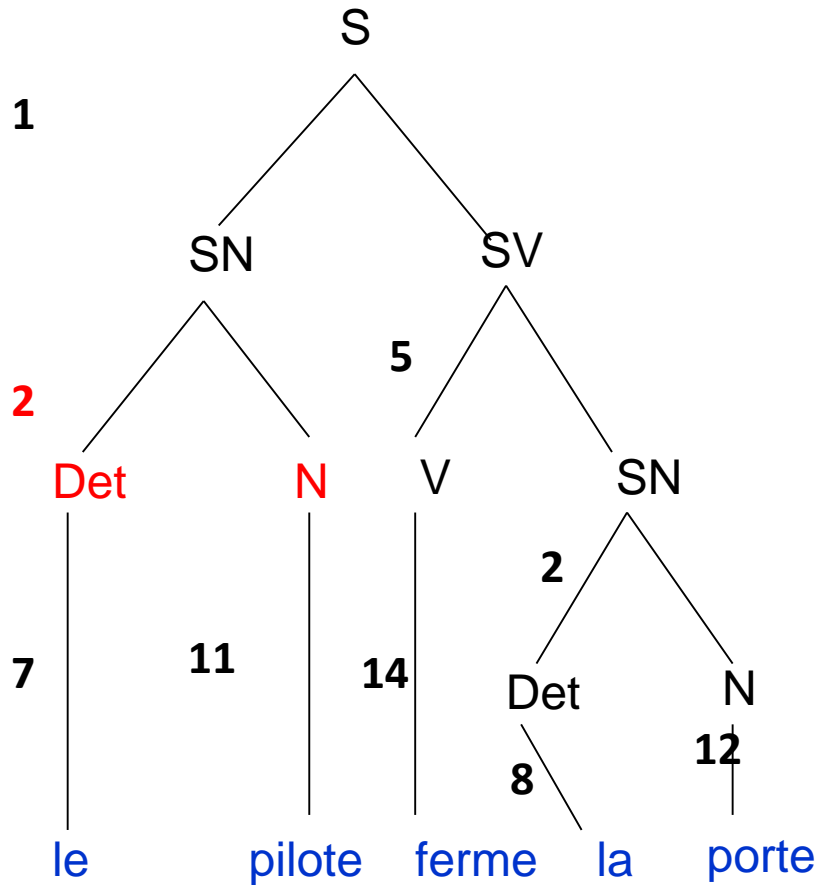
$V_N = \{S, SN, SV, Det, N, A, V, Pro\}$

$V_T = \{le, la, pilote, porte, ferme\}$

$R =$

1	S	$\rightarrow$ SN SV
2 3 4	SN	$\rightarrow$ Det N   Det N A   Det A N
5 6	SV	$\rightarrow$ V SN   Pro V
7 8	Det	$\rightarrow$ le   la
9 10	Pro	$\rightarrow$ le   la
11 12	N	$\rightarrow$ pilote   porte
13	A	$\rightarrow$ ferme
14 15	V	$\rightarrow$ ferme   porte
		}

$v = le\ pilote\ ferme\ la\ porte$  mot de G23 ?



$v$  est un mot de G23 et admet 2 arbres de dérivation

## Exemple 2

### Extrait de la grammaire d'un langage de programmation

$G = (V_N, V_T, \text{instruction}, R)$

$V_N = \{\text{instruction}, \text{expression}\}$

$V_T = \{\text{if}, \text{else}, (, )\}$

Racine = Instruction

$R = \{$   
    1   instruction        $\rightarrow \text{if ( expression ) instruction else instruction}$   
    2   instruction        $\rightarrow \text{if ( expression ) instruction}$   
    3   instruction        $\rightarrow \dots$   
    4   instruction        $\rightarrow \dots$   
     $\}$

G est ambiguë

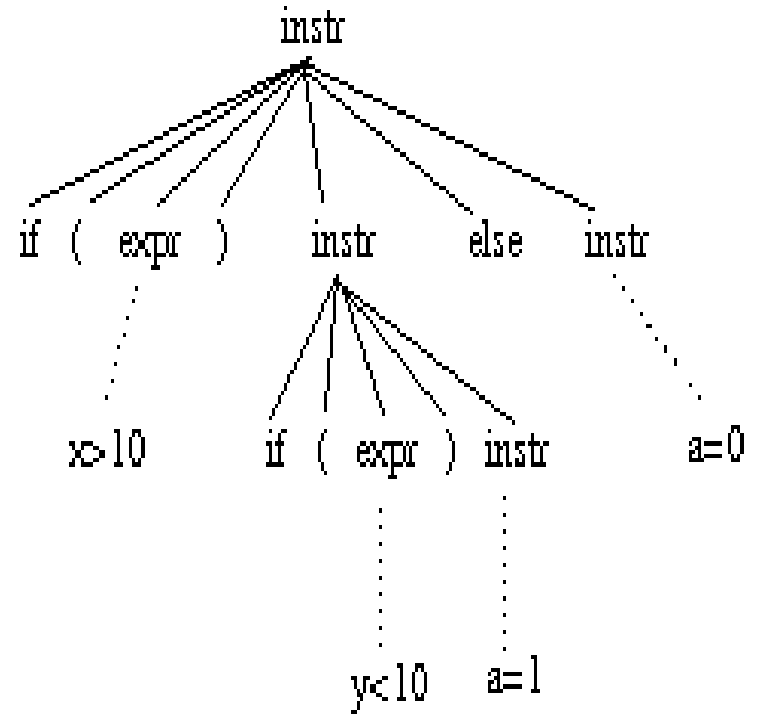
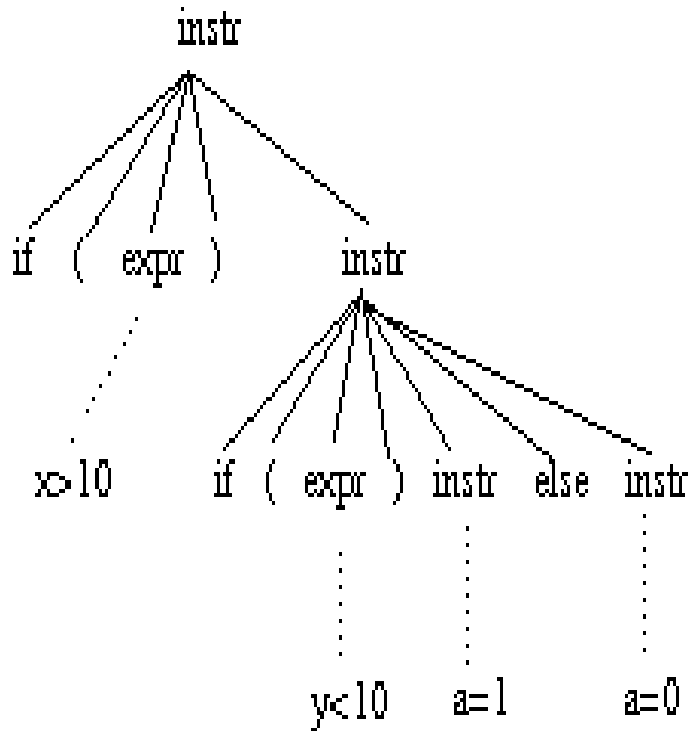
### Exemple :

Le mot

$v = \text{if (} x > 10 \text{ ) if (} y < 0 \text{ ) } a = 1 \text{ else } a = 0$

possède deux arbres de dérivation différents :





deux " **interprétations syntaxiques** " :

```

if (x>10)
    if (y<0)
        a=1
    else
        a=0
    // finsi
// finsi
  
```

```

if (x>10)
    if (y<0)
        a=1
    // finsi
else
    a = 0
// finsi
  
```

# Sommaire

1. Expressions régulières	3
2. Grammaires	16
<b>3. Principes de l'analyse descendante</b>	<b>50</b>
3.0 Introduction	51
3.1 Grammaires LL (1)	55
3.2 Récursivité à gauche	56
3.3 Factorisation à gauche	66
3.4 Grammaire propre	69
4. Automates	71
5. Mise en œuvre de l'analyse descendante	144
6. Conclusion	162
7. Bibliographie	176

### 3.0 Introduction

**Principe** : construire l'arbre de dérivation d'un mot  $w$  en partant **de la racine** – le haut - **vers les feuilles** - le bas.

### Exemples

#### Exemple 1 :

$G1 = (V_N, V_T, S, R)$

$V_N = (S, T)$

$V_T = (a, b, c, d)$

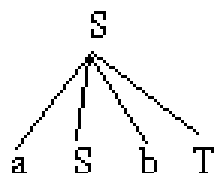
Racine =  $S$

$R = \left\{ \begin{array}{lll} 1, 2, 3 & S & \rightarrow a S b T \mid c T \mid d \\ 4, 5, 6 & T & \rightarrow a T \mid b S \mid c \end{array} \right.$

Soit le mot :  $w = a c c b b a d b c$

On démarre avec l'arbre contenant la racine  $S$  et on cherche une règle qui a la 1<sup>ère</sup> lettre de  $w$  comme 1<sup>er</sup> symbole de sa partie droite

La lecture de la première lettre du mot "a" permet d'avancer la construction



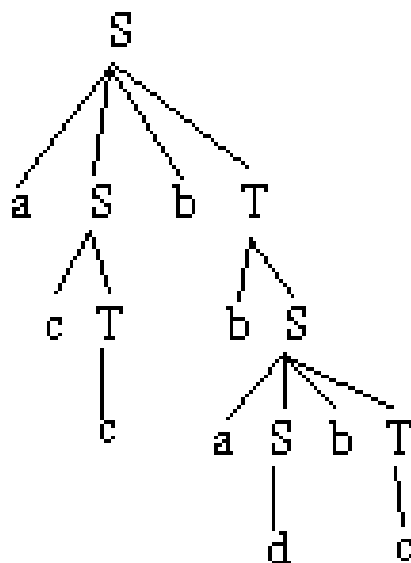
$w = \cancel{a} c c b b a d b c$

et ainsi de suite jusqu'à

Puis la deuxième lettre "c" amène à la situation



$w = \cancel{a} \cancel{c} c b b a d b c$



$w = \cancel{a} \cancel{c} \cancel{c} \cancel{b} \cancel{b} \cancel{a} \cancel{d} \cancel{b} \cancel{c}$

On a trouvé un arbre de dérivation, donc  $w$  appartient au langage de  $G1$ .

Facile : chaque règle commence par un terminal différent, on n'a pas à choisir.

### Exemple 2 :

$G_2 = (V_N, V_T, S, R)$

$V_N = (S, A)$

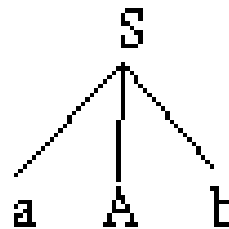
$V_T = (a, b, c, d)$

Racine = S

$R = \{$   
    1    S        $\rightarrow a A b$   
    2, 3 A        $\rightarrow c d \mid c$   
     $\}$

Soit le mot  $w = a c b$

On se retrouve avec



$w = \underline{a} c b$

En lisant  $c$ , on ne sait pas s'il faut choisir

la règle 2  $A \rightarrow c d$

ou la règle 3  $A \rightarrow c$ .

Pour le savoir, il faut :

1- soit lire **aussi la lettre suivante** du mot :  $b$ ,

2- soit faire des **retours en arrière** : on essaie la règle 2  $A \rightarrow c d$ , on aboutit à un échec, alors on retourne en arrière et on essaie 3  $A \rightarrow c$  et ça marche.

### Exemple 3 :

$G_3 = (V_N, V_T, S, R)$

$V_N = (S)$

$V_T = (a, b, c, d)$

$R = \{ S \rightarrow a S b \mid a S c \mid d \} \quad 1, 2, 3 \}$

Soit le mot  $w = a a a a a a d b b c b b b c$

Pour savoir quelle règle utiliser, il faut connaître aussi la **dernière** lettre du mot.

### Exemple 4 : grammaire des expressions arithmétiques

$G_4 = (V_N, V_T, E, R)$

$V_N = (E, E', T, T', F)$

$V_T = (+, *, (, ), \text{nb})$

Racine  $= E$

$R = \{$   
1  $E \rightarrow T E'$   
2, 3  $E' \rightarrow + T E' \mid \varepsilon$   
4  $T \rightarrow F T'$   
5, 6  $T' \rightarrow * F T' \mid \varepsilon$   
7, 8  $F \rightarrow ( E ) \mid \text{nb}$   
 $\}$

Soit le mot  $w = 3 * 4 + 10 * ( 5 + 11) \quad \text{??????}$

Ce serait pratique d'avoir une " **table de correspondance** " qui indique :

"quand je lis tel symbole et que j'en suis à dériver tel symbole non-terminal,  
alors j'applique telle règle".

## 3.1 Grammaires LL (1)

### Définition

On appelle **grammaire LL(1)** une grammaire pour laquelle la reconnaissance d'un mot peut se faire de la manière suivante :

- on parcourt le mot de gauche à droite (L pour *Left to right scanning*),
- on développe l'arbre de dérivation en choisissant d'abord les non terminaux les plus à gauche (L pour *Leftmost derivation*),
- on ne lit pas plus d'un symbole du mot à la fois (1).

## Contre-exemple :

$G_2 = (V_N, V_T, S, R)$

$V_N = (S, A)$

$V_T = (a, b, c, d)$

Racine = S

$R = \{$   
    1        S         $\rightarrow a A b$   
    2, 3    A         $\rightarrow c d \mid c$   
     $\}$

Pour pouvoir choisir entre

$A \rightarrow c d$

et

$A \rightarrow c$

il faut lire la lettre qui suit.

grammaire LL (2) :

**Il faut lire 2 symboles à la fois** pour décider



## Théorème

Une grammaire **récursive à gauche** ou **non factorisée à gauche** ou **ambiguë** n'est pas LL(1).

### 3.2 Récursivité à gauche

**Définition** ; récursivité à gauche immédiate

Une grammaire est **immédiatement récursive à gauche** si elle contient un non-terminal A tel qu'il existe une **règle**

$$A \rightarrow A v$$

où v est un mot quelconque composé de symboles terminaux ou non-terminaux

**Exemple :**

$$G5 = (V_N, V_T, S, R)$$

$$V_N = (S, A, B)$$

$$V_T = (a, b, c, d, e)$$

$$\text{Racine} = S$$

$$R = \left\{ \begin{array}{lll} 1, 2 & S & \rightarrow S c A \quad | B \\ 3, 4 & A & \rightarrow A a \quad | \varepsilon \\ 5, 6, 7 & B & \rightarrow B b \quad | d \quad | e \end{array} \right\}$$

3 récursivités à gauche immédiates : règles **1, 3, 5**

## Construction d'une grammaire non immédiatement récursive à gauche équivalente

Remplacer tout schéma de règles

$$A \rightarrow A w_1 \mid \dots \mid A w_n \mid x_1 \mid \dots \mid x_m \mid$$

où les  $x_i$  ne commencent pas par  $A$

en appliquant la procédure :

1 - ajouter le symbole  $A_1$  à  $V_N$

2 – remplacer le schéma de règles par :

$$\begin{aligned} A &\rightarrow x_1 A_1 \mid \dots \mid x_m A_1 \\ A_1 &\rightarrow w_1 A_1 \mid \dots \mid w_m A_1 \mid \varepsilon \end{aligned}$$

**Théorème** La grammaire obtenue reconnaît **le même langage** que la grammaire initiale.

## Exemple : règles de G51 issue de G5

S	→ B S1	
S1	→ c A S1	$\varepsilon$
A	→ $\varepsilon$ A1	= A1
A1	→ a A1	e
B	→ d B1	e B1
B1	→ b B1	$\varepsilon$

Ici, A1 est un symbole inutile

S	→ B S1	
S1	→ c A S1	$\varepsilon$
A	→ a A	$\varepsilon$
B	→ d B1	e B1
B1	→ b B1	$\varepsilon$

## Exemple : reconnaissance d'un mot avec G5 et G51

$w = \mathbf{d b b c a a}$  s'obtient à partir de G5 par la suite de dérivations immédiates :

$$\begin{aligned} S &\rightarrow S c A && \rightarrow B c A && \rightarrow B b c A \\ &\rightarrow B b b c A && \rightarrow d b b c A && \rightarrow d b b c A a \\ &\rightarrow d b b c A a a \\ &\rightarrow d b b c a a \end{aligned}$$

$w = \mathbf{d b b c a a}$  s'obtient à partir de G51 par la suite de dérivations immédiates :

$$\begin{aligned} S &\rightarrow B S1 && \rightarrow d B1 S1 && \rightarrow d b B1 S1 \\ &\rightarrow d b b B1 S1 && \rightarrow d b b S1 && \rightarrow d b b c A S1 \\ &\rightarrow d b b c a A S1 && \rightarrow d b b c a a A S1 \\ &\rightarrow d b b c a a S1 \\ &\rightarrow d b b c a a \end{aligned}$$

**Remarque** : ici on peut se passer de A1.

$A \rightarrow A a \mid \varepsilon$  est équivalent à :

$A \rightarrow a A \mid \varepsilon$  **immédiatement réursive à droite**, ne pose pas de problème.

## Définition : récursivité à gauche

Une grammaire est **récursive à gauche** si elle contient un non-terminal **A** tel qu'il existe une **dérivation**

$$A \rightarrow^+ A v$$

où  $v$  est un mot quelconque composé de symboles terminaux ou non-terminaux

### Exemple :

$$G_6 = (V_N, V_T, S, R)$$

$$V_N = (S, A)$$

$$V_T = (a, b, c, d)$$

$$\text{Racine} = S$$

$$R = \left\{ \begin{array}{lcl} S & \rightarrow & A a \mid b \\ A & \rightarrow & A c \mid S d \mid \varepsilon \end{array} \right\}$$

$S$  n'est pas immédiatement récursif à gauche mais récursif à gauche :  
en 2 dérivations immédiates

$$\begin{array}{lcl} S & \rightarrow & A a \\ & \rightarrow & S d a \end{array}$$

## Construction d'une grammaire non réursive à gauche équivalente

Renommer et ordonner les non-terminaux  $A_1, A_2, \dots, A_n$

**pour**  $i = 1$  à  $n$

**pour**  $k = 1$  à  $i - 1$

        remplacer chaque ensemble de règles de la forme

$$A_i \rightarrow A_k v$$

$$\text{avec } A_k \rightarrow w_1 \mid \dots \mid w_m \quad w_i \in (V_N \cup V_T)^+$$

$$\text{et } v = x_1 \mid \dots \mid x_p \quad x_j \in V_T^*$$

    par

$$A_i \rightarrow w_1 v \mid \dots \mid w_m v$$

    -- on supprime les  $A_k$   $k = 1$  à  $i - 1$

**fin pour**

        éliminer les récursivités à gauche immédiates des règles

$$A_i \rightarrow w_1 v \mid \dots \mid w_m v$$

**fin pour**

**Théorème** La grammaire obtenue reconnaît **le même langage** que la grammaire initiale.

## Exemple – G6

On **renomme** S et A :  $A1 = S$  et  $A2 = A$   
 $i = 1$

il n'existe aucun k possible  
ni de récursivité immédiate dans

$$S \rightarrow A a \mid b \quad \text{--- } A1 \rightarrow A2 a \mid b$$

$i = 2$  et  $k = 1$

$$A \rightarrow S d \quad \text{-- } A2 \rightarrow A1 x$$

$$S \rightarrow A a \mid b \quad \text{-- } A1 \rightarrow w1 \mid w2$$

devient

$$A \rightarrow A a d \mid b d \quad \text{--- } A2 \rightarrow w1 x \mid w2 x$$

En rassemblant les règles dont A est partie gauche

$$A \rightarrow A c \mid A a d \mid b d \mid \varepsilon$$

on **élimine** la récursivité **immédiate** :

1- on **introduit** A1

2- on **remplace** le schéma de règles  $A \rightarrow A c \mid A a d \mid b d \mid \varepsilon$

par le schéma :

$$A \rightarrow b d A1 \mid \varepsilon A1 = A1$$

$$A1 \rightarrow c A1 \mid a d A1 \mid \varepsilon$$

On a obtenu la grammaire :

$$G_{61} = ( V_N , V_T , S, R )$$

$$V_N = ( S, A, A1 )$$

$$V_T = ( a, b, c, d )$$

$$\text{Racine} = S$$

$$R = \left\{ \begin{array}{lcl} S & \rightarrow & A a \mid b \\ A & \rightarrow & b d A1 \mid A1 \\ A1 & \rightarrow & c A1 \mid a d A1 \mid \varepsilon \end{array} \right\}$$



## Exemple – G7

$G7 = (V_N, V_T, S, R)$

$V_N = (S, T)$

$V_T = (a, b, c, d)$

Racine = S

$R = \left\{ \begin{array}{ll} S & \rightarrow S a \mid T S c \mid d \\ T & \rightarrow T b T \mid \varepsilon \end{array} \right\}$

On obtient la grammaire :

$G71 = (V_N, V_T, S, R)$

$V_N = (S, S1, T, T1)$

$V_T = (a, b, c, d)$

Racine = S

$R = \left\{ \begin{array}{ll} S & \rightarrow T S c S1 \mid d S1 \\ S1 & \rightarrow a S1 \mid \varepsilon \\ T & \rightarrow T1 \\ T1 & \rightarrow b T T1 \mid \varepsilon \end{array} \right\}$

Mais :

$S \rightarrow T S c S1 \rightarrow T1 S c S1 \rightarrow S c S1$  !!! récursivité à gauche !!!!

L'algorithme n'est pas toujours opérant lorsque la grammaire possède une  $\varepsilon$  production  $A \rightarrow \varepsilon$

### 3.3 Factorisation à gauche

Objectif : **supprimer les ambiguïtés** dans la construction d'un arbre de dérivation

Pour développer le nœud du non-terminal A quand il n'est pas évident de choisir la production à utiliser, on doit réécrire les règles de la partie gauche A pour **différer** le choix jusqu'à ce que suffisamment de texte ait été lu pour choisir.

#### Exemple – G8

G8 = (  $V_N$  ,  $V_T$  , S, R )

$V_N$  = ( S, A, B )

$V_T$  = ( a, b, c, d )

Racine = S

R = {  
    S       →       b a c d A b d       |   b a c d B c c a  
    A       →       a D  
    B       →       c E  
    C       → ...  
    E       → ...  
}

Pour construire l'arbre de dérivation, il faut choisir entre

S → b a c d **A** b d

et

S → b a c d **B** c c a

il faut avoir lu la **5ième lettre** du mot (A ou B). G8 est une grammaire LL(5).

## Construction d'une grammaire factorisée à gauche équivalente

**pour** chaque symbole non-terminal A

trouver le plus long préfixe w commun à au moins deux parties droites des règles dont A est partie gauche

**si** w est différent de  $\varepsilon$

**alors**

Ajouter un nouveau symbole A1 à VN

remplacer

$A \rightarrow w v_1 \mid \dots \mid w v_n \mid u_1 \mid \dots \mid u_p$

où les  $u_i$  ne commencent pas par w

par les 2 ensembles de règles

$A \rightarrow w A_1 \mid u_1 \mid \dots \mid u_p$

$A_1 \rightarrow v_1 \mid \dots \mid v_n$

**finpour**

Recommencer jusqu'à ne plus trouver de préfixe commun.

## Exemple :

$$G_9 = (V_N, V_T, S, R)$$

$$V_N = (S, E, B)$$

$$V_T = (a, b, c, d)$$

$$\text{Racine} = S$$

$$R = \left\{ \begin{array}{ll} S & \rightarrow a E b S \quad | \quad a E b S d B \quad | \quad a \\ E & \rightarrow b c B \quad | \quad b c a \\ B & \rightarrow b a \end{array} \right\}$$

Factorisée à gauche, R devient :

$$S \rightarrow a S_1$$

$$S_1 \rightarrow E b S S_2 \mid \varepsilon$$

$$S_2 \rightarrow d B \mid \varepsilon$$

$$E \rightarrow b c E_1$$

$$E_1 \rightarrow B \mid a$$

$$B \rightarrow b a$$

### 3.4 Grammaire propre

#### Définition

Une grammaire est dite **propre** si

1. elle ne contient aucune  $\varepsilon$  production  
 $A \rightarrow \varepsilon$
2. ou si elle ne contient qu'une  $\varepsilon$  production  $S \rightarrow \varepsilon$

#### Construction d'une grammaire propre équivalente à une grammaire

**Pour** chaque  $A$  qui apparaît dans une règle

$$A \rightarrow \varepsilon$$

**Pour** chaque règle où  $A$  apparaît dans sa partie droite,  
Ajouter à la grammaire une règle dans laquelle  $A$   
est remplacé par  $\varepsilon$ ,

**Finpour**

**Finpour**

## Exemple :

$G_{10} = (V_N, V_T, S, R)$

$V_N = (S, T, U)$

$V_T = (a, b)$

Racine = S

$R = \{$

S	→	a T b	a U
T	→	b T a T a	$\epsilon$
U	→	a U	b

$\}$

Règles de la grammaire **propre** équivalente :

S	→	a T b	a b	a U
T	→	b T a T a	b a T a	b T a a
		b a a		
U	→	a U	b	

# Sommaire

1.	Expressions régulières	3
2.	Grammaires	16
3.	Principes de l'analyse descendante	50
<b>4.</b>	<b>Automates</b>	<b>71</b>
4.0	Introduction – Notion de graphe	72
4.1	Automates à états finis déterministe AEFD	78
4.2	Automates à états finis non déterministe AEFND	94
4.3	Déterminisation d'un AEFND	97
4.4	Minimisation d'un AEFD	110
4.5	AEFD associé à une expression régulière	117
4.6	Expression régulière associée à un AEFD	122
4.7	Automates à pile	124
4.8	Configuration	131
4.9	Transformation d'une grammaire hors-contexte en un automate à pile	142
5.	Mise en œuvre de l'analyse descendante	145
6.	Conclusion	163
7.	Bibliographie	177

## 4.0 Introduction – Notion de graphe

### - Graphe orienté

Un **graphe orienté G** est défini par un quadruplet

$G = (S, A, \text{INITIAL}, \text{TERMINAL})$

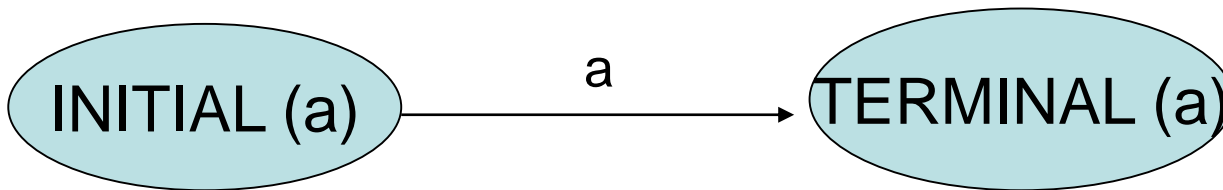
où

S = ensemble fini d'éléments, appelés **sommets**

A = ensemble fini d'éléments, appelés **arcs**

INITIAL, TERMINAL = applications de A vers S telles que  
pour tout arc a de A, son **sommet initial INITIAL (a)** et son **sommet terminal TERMINAL (a)** sont définis

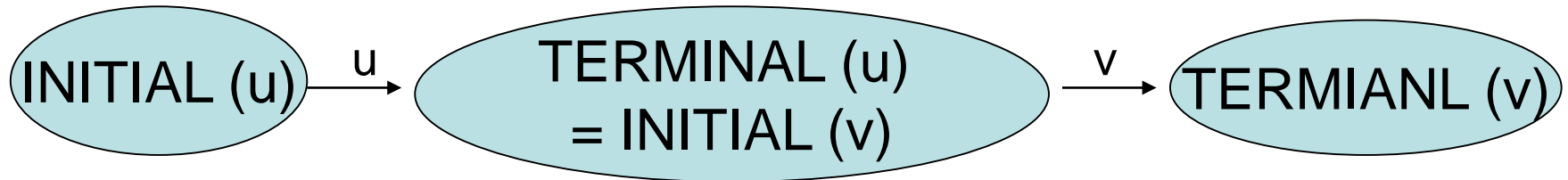
**Exemple 1** : 1 arc "a" et ses 2 sommets





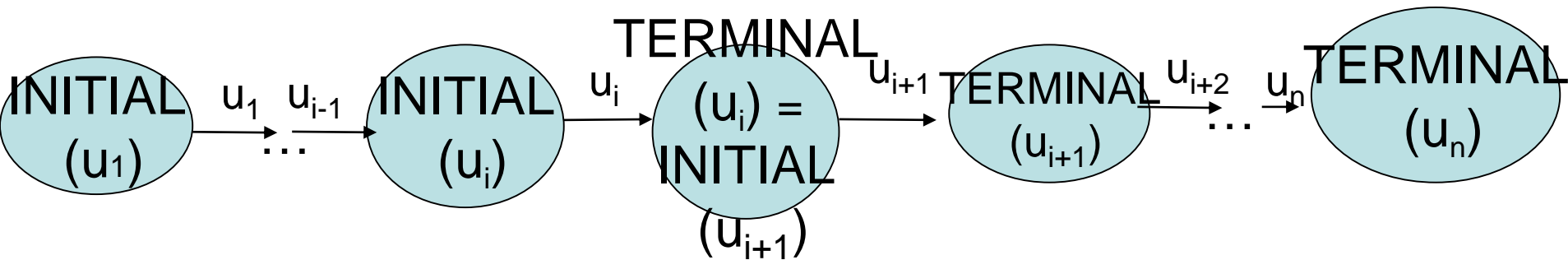
## - Arcs adjacents

2 arcs  $u$  et  $v$  sont dits **adjacents** ssi  $\text{TERMINAL}(u) = \text{INITIAL}(v)$



## - Chemin dans un graphe orienté

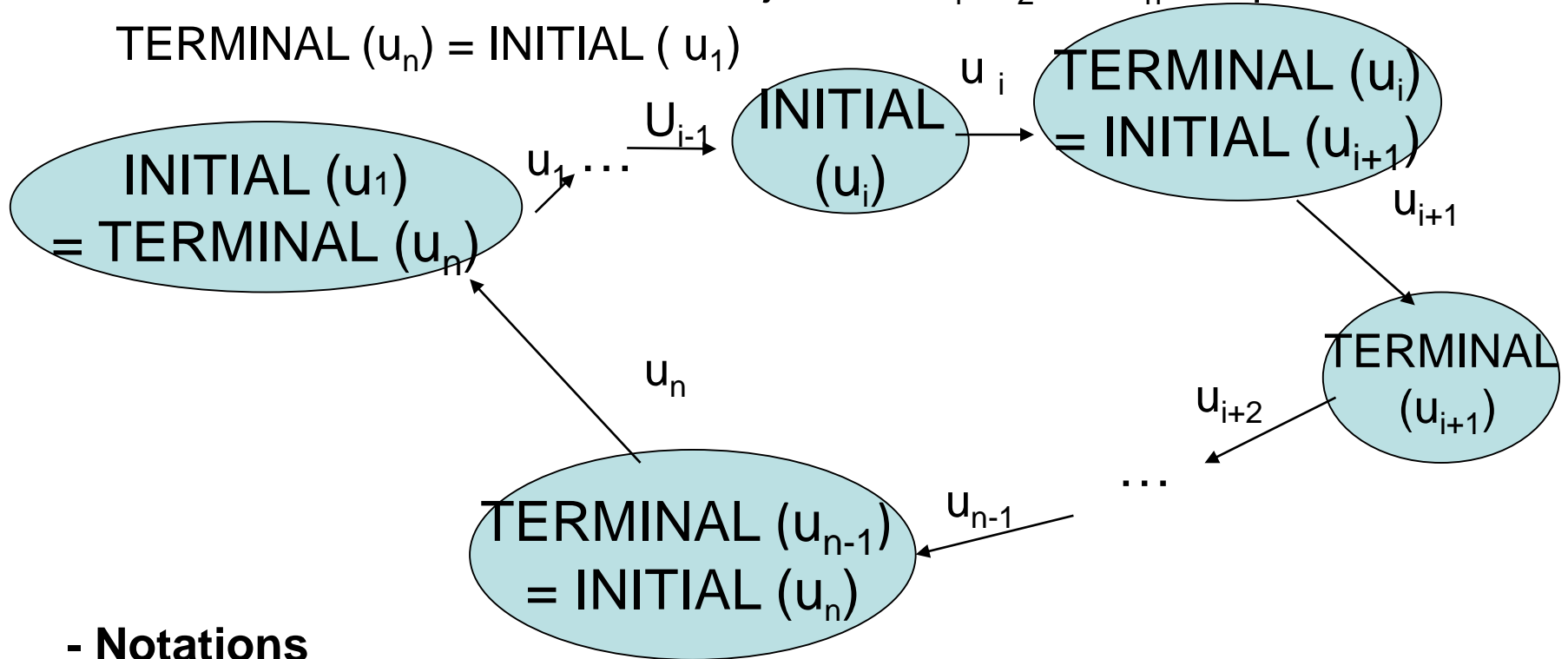
Un **chemin** est une suite d'arcs  $u_1, u_2, \dots, u_n$  telle que pour tout  $i \geq 1$ ,  $u_i$  et  $u_{i+1}$  sont adjacents



## - Circuit dans un graphe orienté

Un **circuit** est un chemin d'arcs adjacents  $u_1, u_2, \dots, u_n$  tel que

$$\text{TERMINAL}(u_n) = \text{INITIAL}(u_1)$$



## - Notations



**Sommet  
initial**



**Sommet  
terminal**

## - Expressions régulières et graphes

Une **expression régulière**  $E$  peut être représentée par un **graphe orienté** dont tout **arc** est **étiqueté** par un **symbole** de son alphabet  $V$  ou par  $\varepsilon$

Un **mot**  $v$  du **langage**  $L$  **dénoté par**  $E$  peut être représenté par un **chemin** reliant

- un **sommet initial**
- un **sommet terminal**

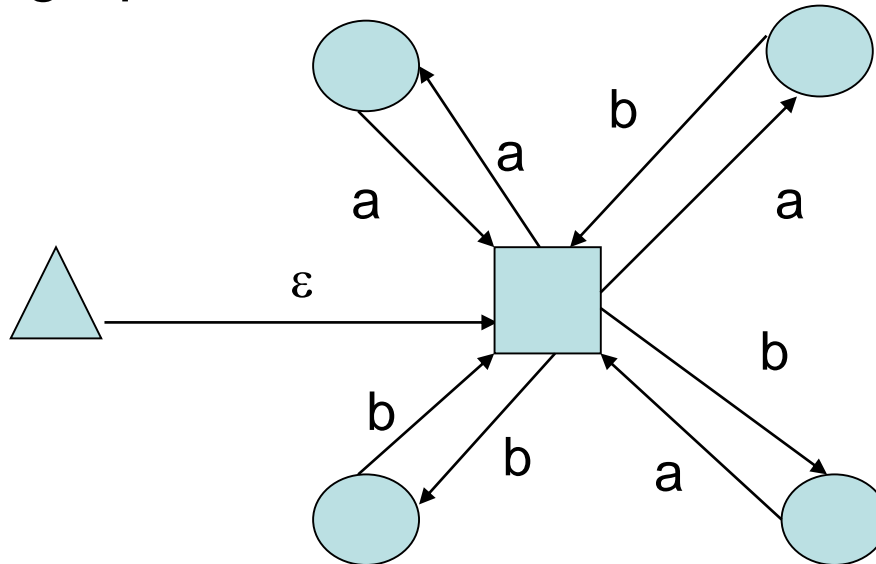
### **Exemple 2 :**

$v = pr(en(d(s + \varepsilon) + ons + ez + nent) + i(s + t + mes + tes + rent))$

Construire un graphe orienté  $G_1$  qui représente  $v$

### Exemple 3

Soit G2 le graphe



Expression régulière représentant G2

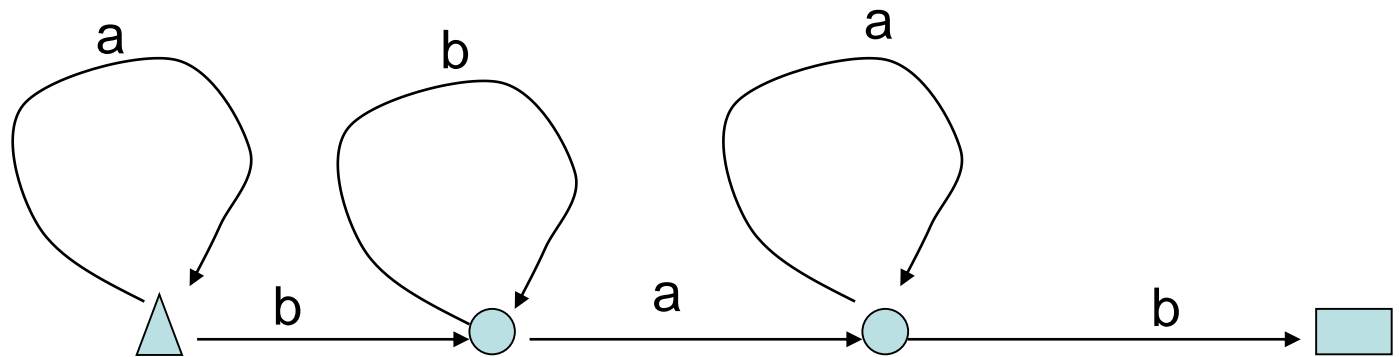
$V2 = \{a, b\}$

$E2 = (a a + a b + b a + b b)^*$

"Tous les chemins sont étiquetés par un mot dont le nombre de lettres est pair"

## Exemple 3

Soit G3 le graphe



Expression régulière représentant G3

$V3 = \{a, b\}$

$E3 = a^* b b^* a a^* b = a^* b + a + b$

## 4.1 Automates à états finis déterministe AEFD

### Définition

Un AEFD  $M$  est la donnée d'un quintuplet

$$M = ( Q, V, q_0, F, T )$$

où :

- $Q$  : ensemble d'états
- $V$  : vocabulaire / alphabet
- $q_0 \in Q$  : état initial
- $F \subseteq Q$  : ensemble d'états finaux / terminaux
- $T$  : ensemble de transitions

$$(q, x, q') \in (Q, V, Q)$$

ou fonction de transition

$$(Q, V) \rightarrow Q$$

Machine composée de :

1. Une **mémoire** = ruban où on place les symboles d'un mot construit sur un alphabet  $V$
2. Une **tête de lecture** qui peut lire un symbole  $x$  de  $V$  à la fois, dans la même direction et qui est dans l'un des états  $q$  d'un ensemble d'états  $Q$ , dont un **état initial**  $q_0$  et un **sous-ensemble d'état terminaux**  $F \subseteq Q$
3. Un **automate** qui commande la tête de lecture

La **tête** est dans un état  $q_0$  **initial**, et placée en face du 1<sup>er</sup> **symbole**  $x_0$  de  $V$  inscrit sur le ruban

- en fonction de son **état**  $q$  et du **symbole lu**  $x$  de  $V$ , la tête adopte un **nouvel état**  $q'$  de  $Q$  s'il existe une transition de  $T$   
 $(q, x, q')$  ou  $(q, x) \rightarrow q'$
- la lecture **s'arrête** quand il n'y a **plus de symbole** à lire
- La suite de symboles de  $V$  "lus" est un **mot "reconnu"** par l'automate si la tête est dans l'un des états **terminaux** et qu'il n'y a **plus de symbole à lire sur le ruban**

## Configuration – dérivabilité d'une configuration

Tout "instant" du fonctionnement d'un automate

$M = (Q, V, q_0, F, T)$

est caractérisé par

- un état  $q$  de  $Q$
- le mot  $v$ , suite des symboles qui restent à lire à cet instant

### Configuration

élément  $(q, v)$  de  $Q \times V^*$

### Configuration initiale

$(q_0, v_0)$ , où  $v_0$  est le mot à lire



Une configuration  $(q', w)$  **est une dérivation immédiate** d'une configuration  $(q, v)$

$$(q, v) \rightarrow (q', w)$$

s'il existe un symbole "a" de  $V$  tel que :  $v = a . w$

et si  $(q, a) \rightarrow (q') \in T$  ou  $(q, a, q') \in T$  ou  $q' = T(q, a)$

Une configuration  $(q', w)$  **est dérivable** à partir de  $(q, v)$  s'il existe une suite finie de dérivations immédiates

$$(q, v) \rightarrow (q_1, w_1) \rightarrow \dots \rightarrow (q_k, w_k) = (q', w)$$

on note

$$(q, v) \rightarrow^* (q', w)$$

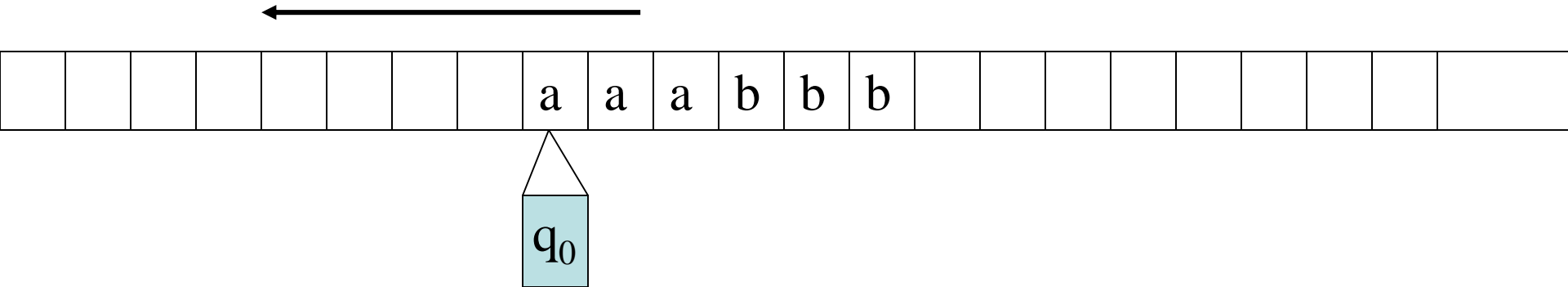
**Un mot  $v$  sur  $V^*$  est reconnu par un automate**

$M = (Q, V, q_0, F, T)$  s'il existe une dérivation

$$(q_0, v) \rightarrow^* (q_f, \varepsilon) \text{ où } q_f \in F \text{ état final (ou terminal)}$$

# Illustration

Configuration initiale  
(  $q_0$ , a a a b b b )



Tête de lecture

Transitions de l'automate :

$(q_0, a) \rightarrow (q)$

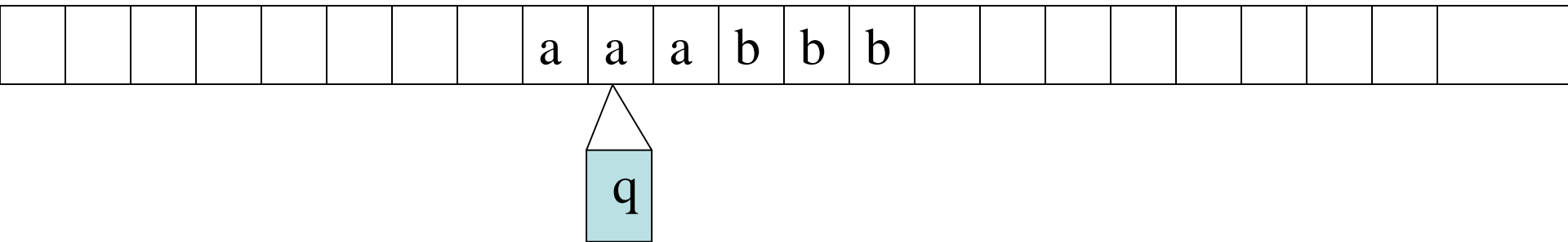
$(q, a) \rightarrow (q)$

$(q, b) \rightarrow (q)$

$(q, \varepsilon) \rightarrow (q_f)$

a a a b b b est-il un mot reconnu ?

Configuration  
(  $q$ , a a b b b )



Transitions :

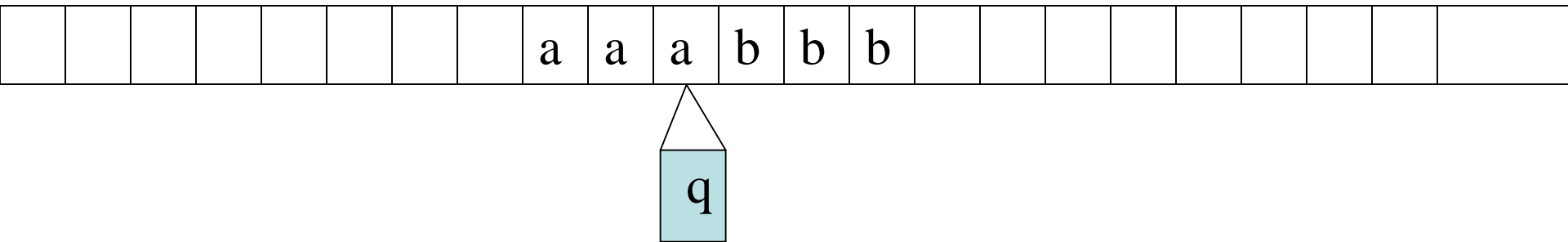
$(q_0, a) \rightarrow (q)$

$(q, a) \rightarrow (q)$

$(q, b) \rightarrow (q)$

$(q, \varepsilon) \rightarrow (q_f)$

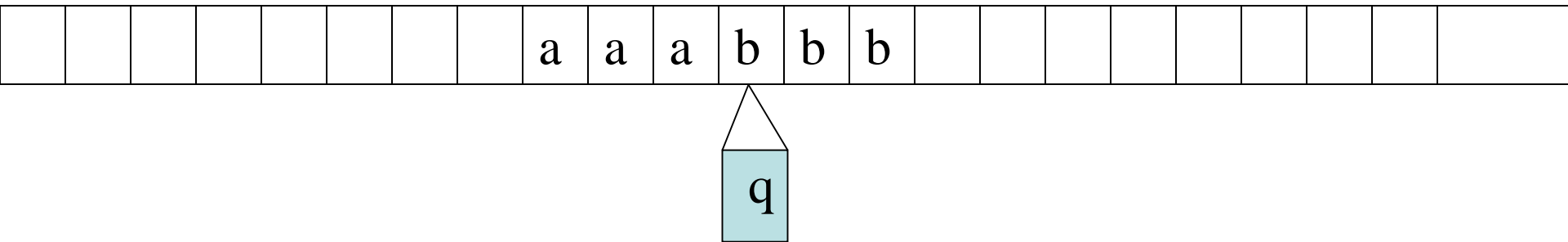
Configuration  
( q, a b b b )



Transitions :

- $(q_0, a) \rightarrow (q)$
- $(q, a) \rightarrow (q)$
- $(q, b) \rightarrow (q)$
- $(q, \varepsilon) \rightarrow (q_f)$

Configuration  
( q, b b b)



Transitions :

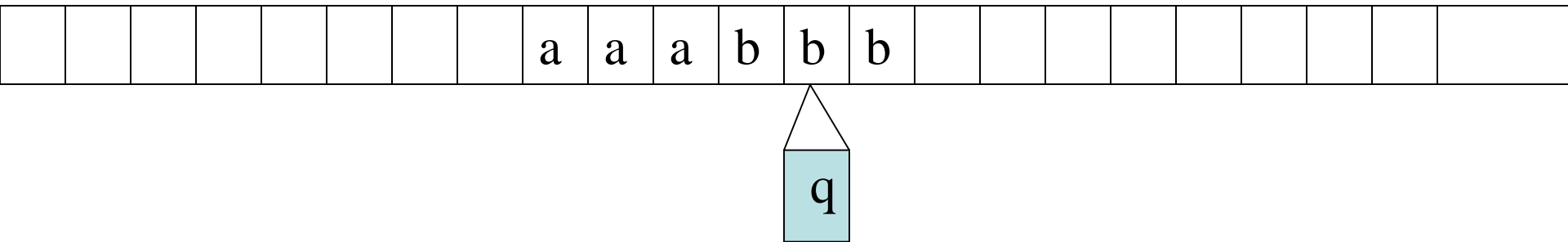
$(q_0, a) \rightarrow (q)$

$(q, a) \rightarrow (q)$

$(q, b) \rightarrow (q)$

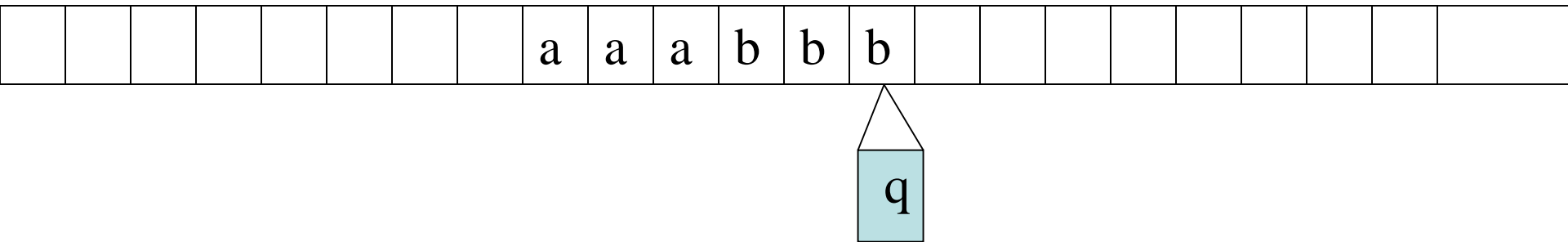
$(q, \varepsilon) \rightarrow (q_f)$

Configuration  
( q, b b)



Transitions :  
 $(q_0, a) \rightarrow (q)$   
 $(q, a) \rightarrow (q)$   
 $(q, b) \rightarrow (q)$   
 $(q, \varepsilon) \rightarrow (q_f)$

Configuration  
(  $q$ ,  $b$  )



Transitions :

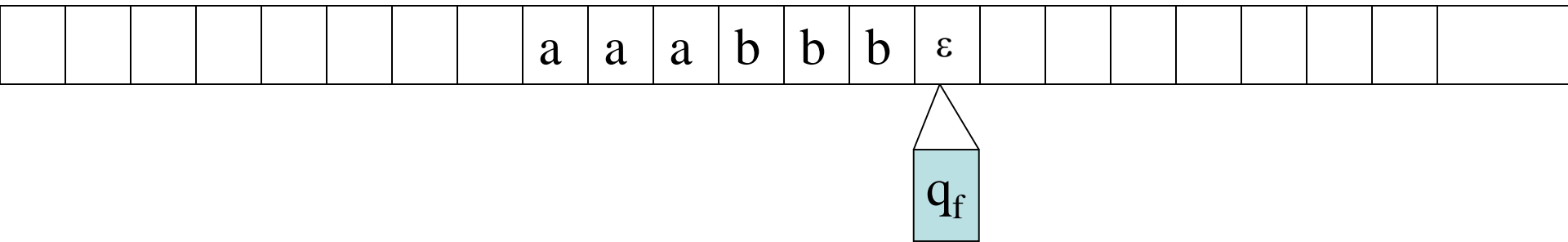
$(q_0, a) \rightarrow (q)$

$(q, a) \rightarrow (q)$

$(q, b) \rightarrow (q)$

$(q, \varepsilon) \rightarrow (q_f)$

Configuration terminale  
(  $q_f$ ,  $\varepsilon$  )



Transitions :  
( $q_0$ , a)  $\rightarrow$  (q)  
(q, a)  $\rightarrow$  (q)  
(q, b)  $\rightarrow$  (q)  
(q,  $\varepsilon$ )  $\rightarrow$  ( $q_f$ )



Tout **automate**

$M = ( Q, V, q_0, F, T )$

peut être représenté par un **graphe orienté**

$G = ( S, A, \text{INITIAL}, \text{TERMINAL} )$

- Tout **sommet**  $s$  de  $S$  est identifié à un **état**  $q$  de  $Q$
- Le sommet initial de  $S$  est identifié à l'état initial  $q_0$
- Tout arc  $a = ( \text{INITIAL} (a), \text{TERMINAL} (a) )$  de  $A$  entre 2 sommets de  $S$  est identifié à une **transition**

$$(q, a, q') \in T \text{ ou } (q, a) \rightarrow q'$$

avec

$$a \in V$$

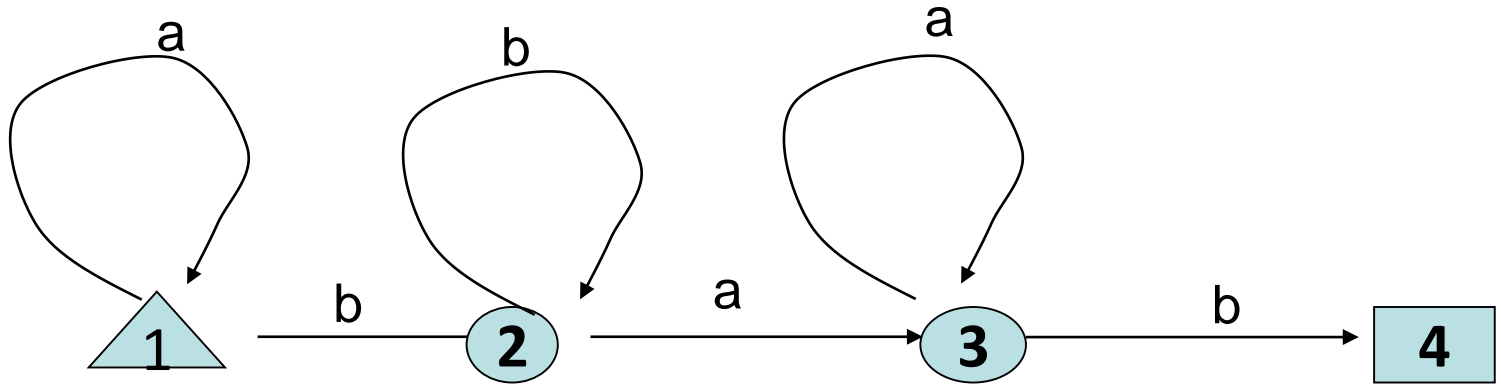
$$q = \text{INITIAL} ( a )$$

$$q' = \text{TERMINAL} ( a )$$

- On souligne les états terminaux de  $F$

## Exemple 3

Soit G3 le graphe



G3 représente l'expression régulière

$$E3 = a^* b b^* a a^* b = a^* b^+ a^+ b$$

G3 représente un automate

$$M = (Q, V, q_0, F, T)$$

$$Q = (1, 2, 3, 4)$$

$$V = (a, b)$$

$$q_0 = 1$$

$$F = \underline{4}$$

T =

V	a	b
Q		
1	1	2
2	3	2
3	3	<u>4</u>
<u>4</u>	-	-

## Example 4

$$M1 = ( Q, V, q_0, F, T )$$

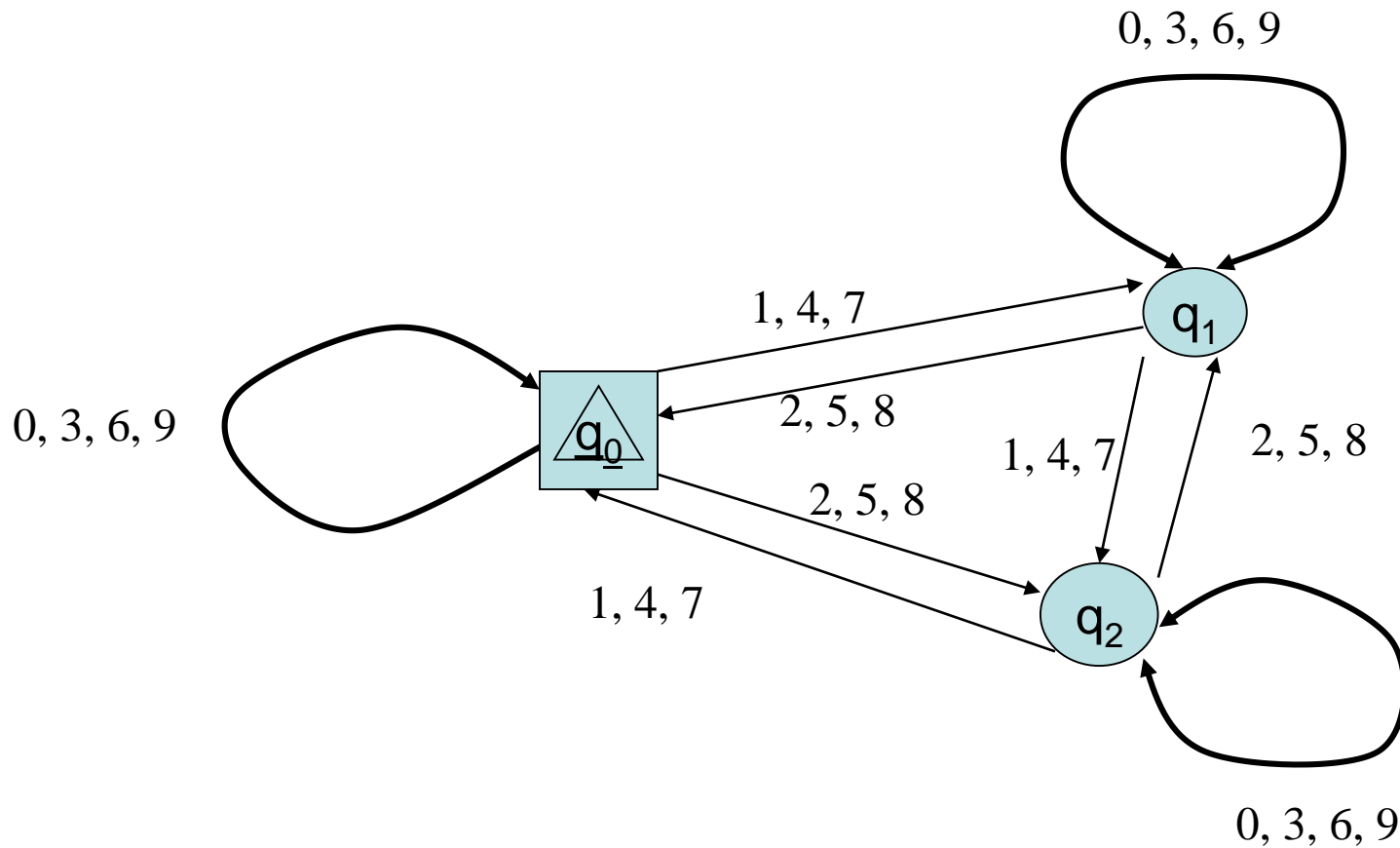
$$Q = \{ q_0, q_1, q_2 \}$$

$$V = \{ 0, \dots, 9 \}$$

$$F = \{ \underline{q_0} \}$$

**T =**

V	0, 3, 6, 9	1, 4, 7	2, 5, 8
Q			
<u>q<sub>0</sub></u>	<u>q<sub>0</sub></u>	q <sub>1</sub>	q <sub>2</sub>
q <sub>1</sub>	q <sub>1</sub>	q <sub>2</sub>	<u>q<sub>0</sub></u>
q <sub>2</sub>	q <sub>2</sub>	<u>q<sub>0</sub></u>	q <sub>1</sub>



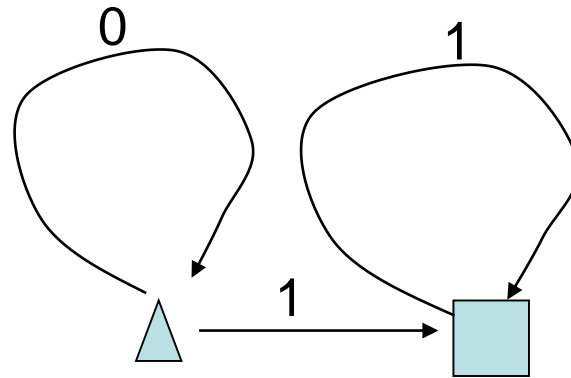
## Exemple

$v1 = 150$  est reconnu par M1,

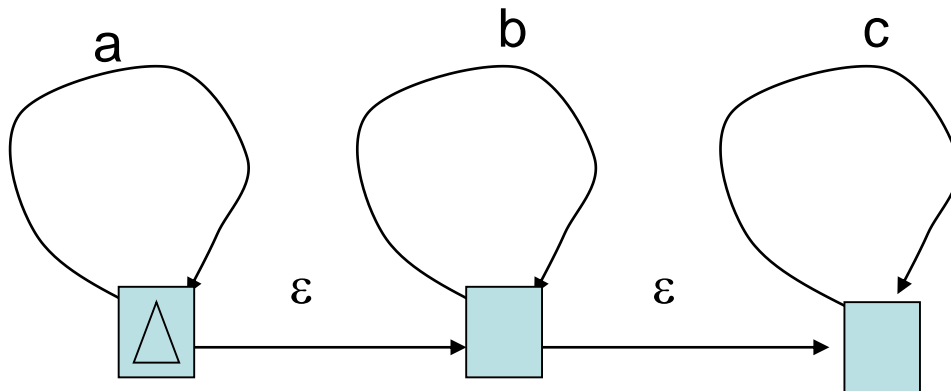
$v2 = 149$  n'est pas reconnu par M1

**Exercices** : trouver un AEFD pour les langages L1 et L2

1-  $L1 = \{ 0^p 1^n, p \geq 0, n \geq 1 \}$



2-  $L2 = \{ a^p b^k c^m, p \geq 0, k \geq 0, m \geq 0 \}$



## 4.2 Automates à états finis non déterministe AEFND

Un AEFND  $M$  est la donnée d'un quintuplet

$$(Q, V, q_0, F, \Delta)$$

où :

- $Q$  : ensemble d'états
- $V$  : vocabulaire / alphabet
- $q_0 \in Q$  : état initial
- $F \subseteq Q$  : ensemble d'états finaux / terminaux
- $\Delta$  : ensemble de transitions

$$(Q, V, Q^*)$$

ou relation de transition

$$(Q, V) \rightarrow Q^*$$

## Exemple 5 :

$M2 = (Q, V, q_0, F, \Delta)$

$Q = \{0, 1, 2, 3, 4\}$

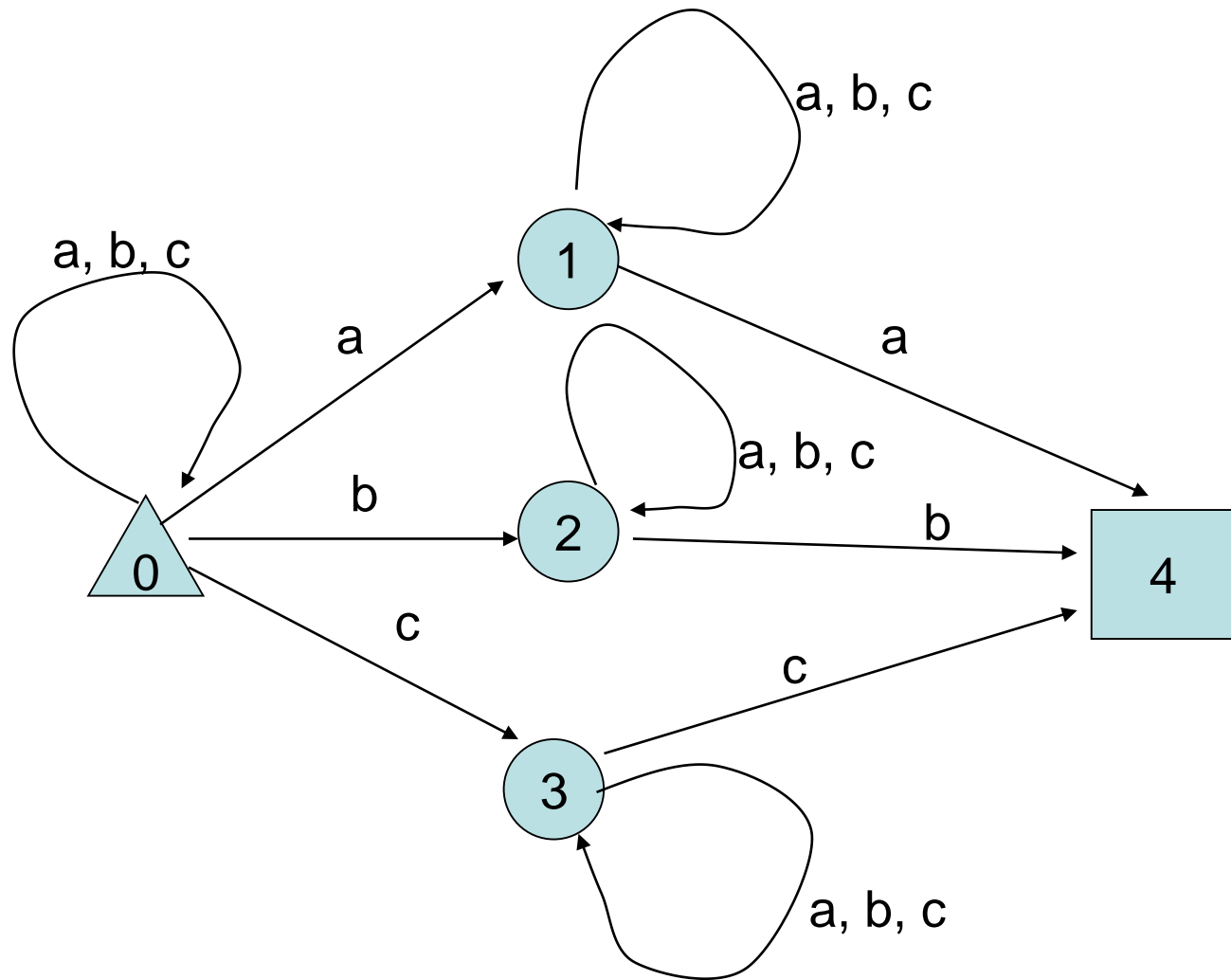
$V = \{a, b, c\}$

$q_0 = 0$

$F = \{\underline{4}\}$

$\Delta =$

Q \ V	a	b	c
0	0, 1	0, 2	0, 3
1	1, <u>4</u>	1	1
2	2	2, <u>4</u>	2
3	3	3	3, <u>4</u>
<u>4</u>	-	-	-



**Langage** = ensemble des mots dont la dernière lettre se trouve déjà dans le mot



### 4.3 "Déterminisation" d'un AEFND

**Principe :** Soit  $M = (Q, V, q_0, F, \Delta)$  un AEFND.

Les états et transitions de l'AEFD MD équivalent sont construits à partir des états et des ensembles de transitions de  $M$

#### Définition

Une  $\varepsilon$  transition est une transition  $(p, \varepsilon, q)$  qui fait changer d'état un automate de  $p$  à  $q$  sans consommer de lettre de  $V$   
– "spontanément"

### 4.3.1 Cas particulier : l'AEFND ne contient pas d' $\epsilon$ transition

- Alphabet de MD = alphabet de M = V
1. Etat initial de MD = état initial de M =  $q_0$
  2. Pour chaque lettre "a" de V, ajouter à MD un état qui rassemble l'ensemble  $\{q_1, \dots, q_n\}$  des états de Q accessibles par l'une des transitions  $(q_0, a, q_i)$ ,  $i = 1, \dots, n$  de  $\Delta$
  3. Pour chaque nouvel état de MD créé, recommencer 2 jusqu'à ce qu'aucun nouvel état soit créé
  4. Tout état de MD contenant au moins l'un des états  $q_f$  de F est un état terminal de MD
  5. Renumérotez tous les états de MD

## Example 6 :

M3 = ( Q, V, q<sub>0</sub>, F, Δ )

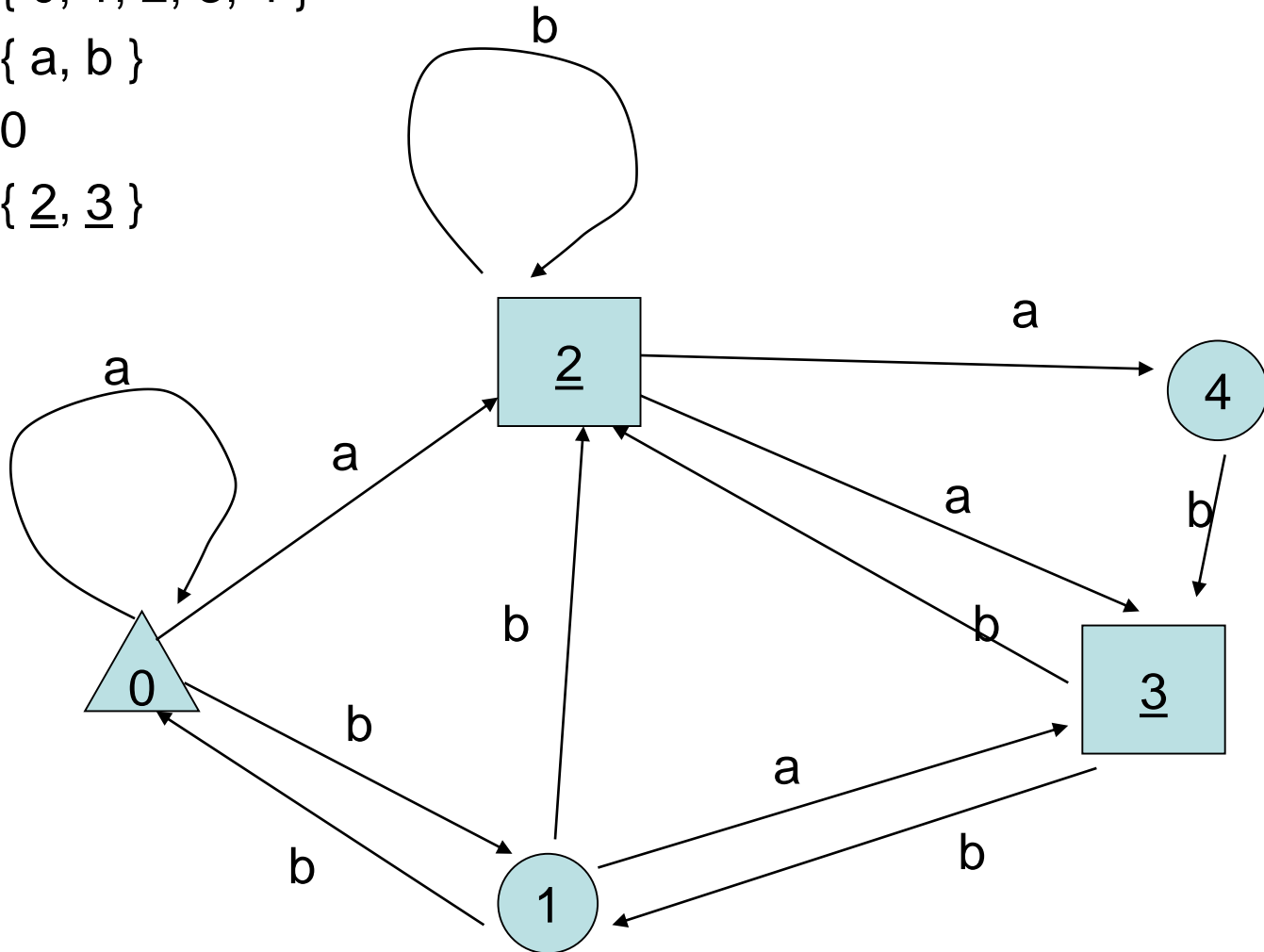
Q = { 0, 1, 2, 3, 4 }

V = { a, b }

q<sub>0</sub> = 0

F = { 2, 3 }

Δ =



$\Delta =$

	V	a	b
Q			
0		0, <u>2</u>	1
1		<u>3</u>	0, <u>2</u>
<u>2</u>		<u>3</u> , 4	<u>2</u>
<u>3</u>		-	1, <u>2</u>
4		-	<u>3</u>

Automate M3D : on applique l'algorithme

$T =$

0

1

2

3

4

5

6

7

8

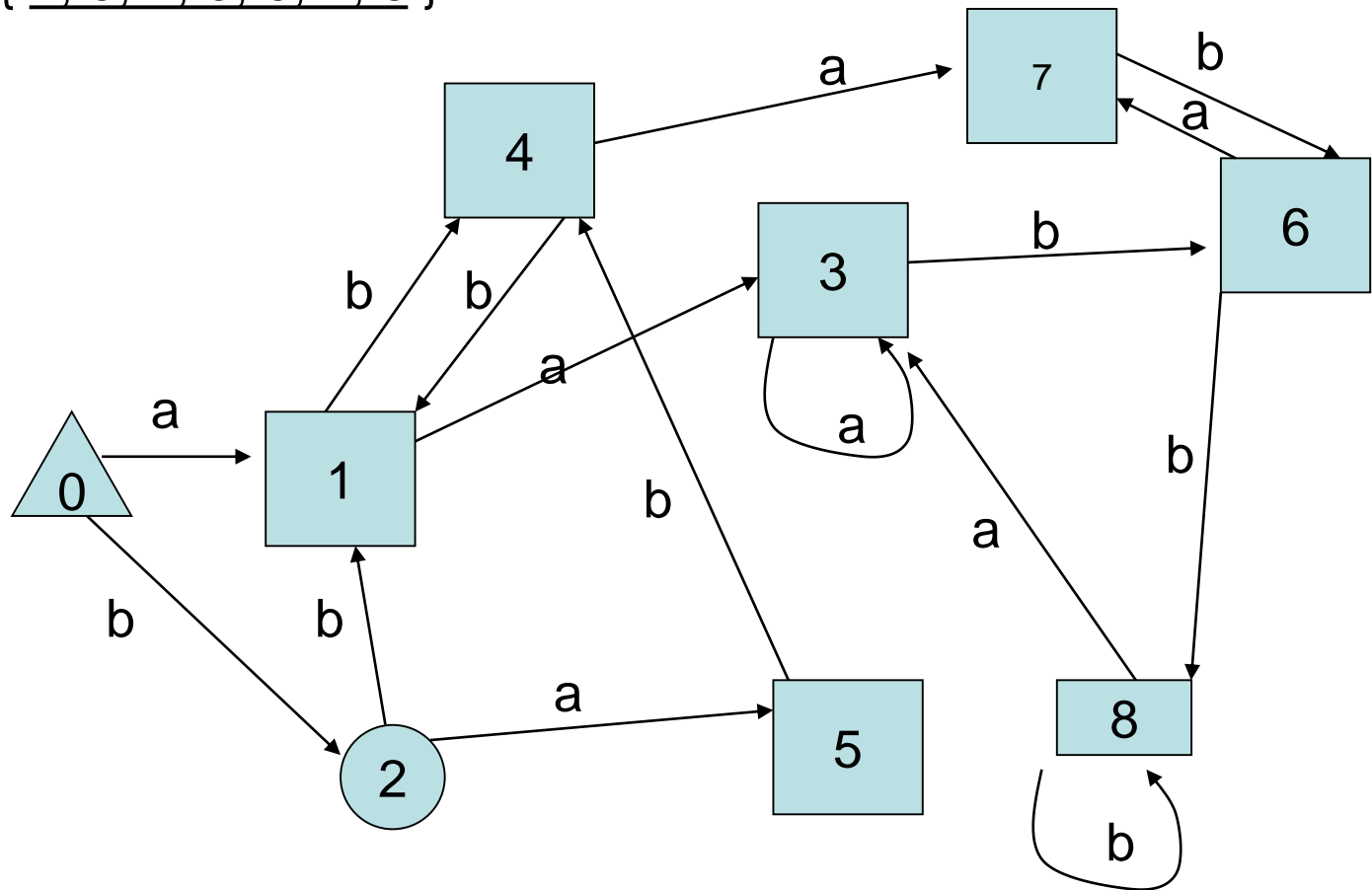
	V	a	b
Q			
0		0, <u>2</u>	1
0, <u>2</u>		0, 2, 3, 4	1, <u>2</u>
1		<u>3</u>	0, <u>2</u>
0, <u>2</u> , <u>3</u> , 4		0, <u>2</u> , <u>3</u> , 4	1, <u>2</u> , <u>3</u>
1, <u>2</u>		<u>3</u> , 4	0, <u>2</u>
<u>3</u>		-	1, <u>2</u>
1, <u>2</u> , <u>3</u>		<u>3</u> , 4	0, 1, <u>2</u>
<u>3</u> , 4		-	1, <u>2</u> , <u>3</u>
0, 1, <u>2</u>		0, <u>2</u> , 3, 4	0, 1, <u>2</u>

# Renumérotation des états du nouvel automate M3D

T =

Q	V	a	b
<u>0</u>		<u>1</u>	2
<u>1</u>		<u>3</u>	<u>4</u>
<u>2</u>		<u>5</u>	<u>1</u>
<u>3</u>		<u>3</u>	<u>6</u>
<u>4</u>		<u>7</u>	<u>1</u>
<u>5</u>		-	<u>4</u>
<u>6</u>		<u>7</u>	<u>8</u>
<u>7</u>		-	<u>6</u>
<u>8</u>		<u>3</u>	<u>8</u>

M3D = ( QD, V,  $q_0$ , FD, T )  
 QD = { 0, 1, 2, 3, 4, 5, 6, 7, 8 }  
 V = { a, b }  
 $q_0$  = 0  
 FD = { 1, 3, 4, 5, 6, 7, 8 }  
 T =



## 4.3.2 Cas général : : l'AEFND contient des $\varepsilon$ transition

**Principe** : considérer l'  $\varepsilon$  – fermeture des ensembles d'états

### Définition

$\varepsilon$  – fermeture de l'ensemble d'états  $Etats = \{ q_1, \dots, q_n \}$

= ensemble des **états accessibles** depuis un état  $q_i$  de  $Etats$   
par **des  $\varepsilon$  – transitions**

# Calcul de l' $\varepsilon$ – fermeture de Etats = $\{ q_1, \dots, q_n \}$

Placer tous les états  $q_i$  de Etats dans une pile P

$\varepsilon$  – fermeture (Etats) = Etats

**Tant que** P n'est pas vide

    Soit p sommet de P

**Pour** chaque état q tel que  $(p, \varepsilon, q) \in \Delta$

**Si** q n'est pas dans  $\varepsilon$  – fermeture (Etats)

**Alors**

$\varepsilon$  – fermeture (Etats) =  $\varepsilon$  – fermeture (Etats) + q  
            empiler q dans P

**Fin si**

**Fin pour**

**Si** il n'existe aucun q tel que  $(p, \varepsilon, q) \in \Delta$

**alors**

            dépiler p de P

**fin si**

**Fin tant que**



### Exemple 7 :

$M4 = (Q, V, q_0, F, \Delta)$

$Q = \{0, 1, 2, 3, 4\}$

$V = \{a, b, c\}$

$q_0 = 0$

$F = \{\underline{4}\}$

$\Delta =$

Q \ V	a	b	c	$\epsilon$
0	2	-	0	1
1	3	<u>4</u>	-	-
2	-	-	1, <u>4</u>	0
3	-	1	-	-
<u>4</u>	-	-	3	2

$\epsilon$  – fermeture ( $\{0\}$ ) =  $\{0, 1\}$

$\epsilon$  – fermeture ( $\{1, 2\}$ ) =  $\{1, 2, 0\}$

$\epsilon$  – fermeture ( $\{3, 4\}$ ) =  $\{3, 4, 2, 0, 1\} \dots$

## Déterminisation d'un AEFND avec des $\varepsilon$ – transitions

Alphabet de MD = alphabet de M = V

F = ensemble des états terminaux de M

Etat initial de MD =  $\varepsilon$ –fermeture (  $q_0$  )

**1. Pour** chaque lettre "a" de V

ajouter un état  $q_a$  qui rassemble l'ensemble d'états

$\{q_1, \dots, q_n\}$  de Q accessibles par une transition de

$\Delta : (q_0, a, q_i), i = 1, \dots, n$

+  $\varepsilon$ –fermeture (  $q_0$  )

+  $\varepsilon$ –fermeture (  $q_1$  )

+ . . .

+  $\varepsilon$ –fermeture (  $q_n$  )

**3. Pour** chaque état  $q_a$  de MD créé, recommencer 1 jusqu'à ce qu'aucun nouvel état soit créé

4. Tout état de MD contenant au moins l'un des états de F est un état terminal de MD

5. Renumérotez tous les états

### Exemple 8 : automate M4D = M4 déterminisé

- état **initial** de M4D =  $\varepsilon$  – fermeture ( 0 ) = ( { 0, 1 } )
- $T ( \{ 0, 1 \} , a ) =$ 
  - $\Delta ( \{ 0, 1 \} , a ) = \{ 2, 3 \}$
  - +  $\varepsilon$  – fermeture ( { 0, 1 } ) = { 0, 1 }
  - +  $\varepsilon$  – fermeture ( { **2, 3** } ) = { 0, 1 }

d'où :

$$T ( ( \{ 0, 1 \} ), a ) = ( \{ 0, 1, 2, 3 \} )$$

- $T ( \{ 0, 1 \} , b ) =$ 
  - $\Delta ( \{ 0, 1 \} , b ) = \{ 4 \}$
  - +  $\varepsilon$  – fermeture ( { 0, 1 } ) = { 0, 1 }
  - +  $\varepsilon$  – fermeture ( { **4** } ) = ( { 2, 0, 1 } )

d'où :

$$T ( ( \{ 0, 1 \} ), b ) = ( \{ 0, 1, 2, 4 \} )$$

etc.

Q	V	a	b	c
0, 1		0, 1, 2, 3	0, 1, 2, <u>4</u>	0, 1
0, 1, 2, 3		0, 1, 2, 3	0, 1, 2, <u>4</u>	0, 1, 2, <u>4</u>
0, 1, 2, <u>4</u>		0,1, 2, 3	0, 1, 2, <u>4</u>	0, 1, 2, 3, <u>4</u>
0, 1, 2, 3, <u>4</u>		0, 1, 2, 3, <u>4</u>	0, 1, 2, 3, <u>4</u>	0, 1, 2, 3, <u>4</u>

## Après renumérotation des états de M4D

Q \ V	a	b	c
0	1	<u>2</u>	0
1	1	<u>2</u>	<u>2</u>
<u>2</u>	1	<u>2</u>	<u>3</u>
<u>3</u>	<u>3</u>	<u>3</u>	<u>3</u>

$$F = \{ \underline{2}, \underline{3} \}$$

### Exercice

Dessiner les 2 graphes avant et après détermination

## 4.4 Minimisation d'un AEFD

**But** : A partir d'un AEFD  $M$  donné **complet**, construire un nouvel automate  $M_m$  équivalent ayant le **minimum d'états** possible.

**Principe** : on définit des **classes d'équivalence** entre états par raffinements successifs. Chaque **classe d'équivalence** obtenue forme un seul et même **état** du nouvel automate

1. **Compléter** l'automate : ajouter un état "**puir**" dans les cases vides de la matrice de transition
2. Constituer 2 classes
$$A = \{ \text{états terminaux} \}$$
$$B = \{ \text{états non terminaux} \}$$
3. **Pour chaque classe**  
**Si** il existe un symbole  $a$  et 2 états  $q_1, q_2$  d'une même classe tels que  
 $\Delta (q_1, a)$  et  $\Delta (q_2, a)$  n'appartiennent pas à la même classe,  
**Alors** créer une nouvelle classe  $C$   
enlever  $q_2$  de sa classe et placer  $q_2$  dans  $C$   
**Fin pour**
4. Recommencer 3 jusqu'à ce qu'il n'y ait plus de classes à séparer
5. Regrouper les classes qui ont "le même comportement"
6. **Chaque classe restante forme un état du nouvel automate  $M_m$**

## Exemple 9 :

Construction de M3Dm, automate minimal équivalent de M3D

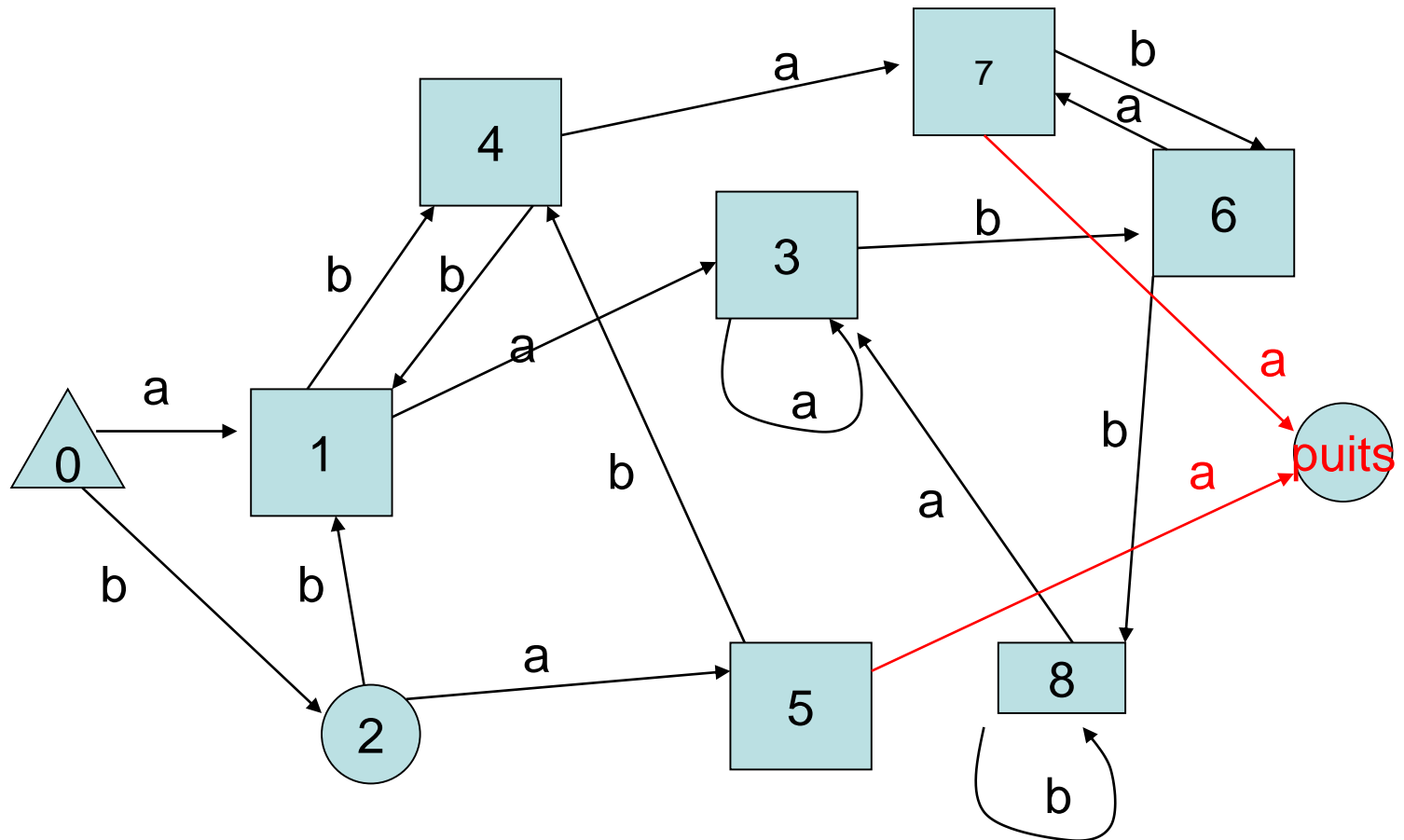
1. On complète M3D avec l'ajout d'un état "puits" pour les transitions manquantes

T =

Q	V	a	b
<u>0</u>		<u>1</u>	2
<u>1</u>		<u>3</u>	<u>4</u>
<u>2</u>		<u>5</u>	<u>1</u>
<u>3</u>		<u>3</u>	<u>6</u>
<u>4</u>		<u>7</u>	<u>1</u>
<u>5</u>		puits	<u>4</u>
<u>6</u>		<u>7</u>	<u>8</u>
<u>7</u>		puits	<u>6</u>
<u>8</u>		<u>3</u>	<u>8</u>



## Graphe de M3D complété



## 2- Classes initiales

$$A = \{ 1, 3, 4, 5, 6, 7, 8 \}$$

$$B = \{ 0, 2 \}$$

### 3- Etude de la classe A

Les transitions qui partent des états de A restent dans A, sauf 5 qui est projeté vers l'état "puit" avec "a"

On crée une classe A1 pour 5

$$A = \{ 1, 3, 4, 6, 7, 8 \} \quad A1 = \{5\}$$

Les transitions qui partent des états de A restent dans A, sauf 7 qui est projeté vers l'état "puit" avec "a"

On crée une classe A2 pour 7

$$A = \{ 1, 3, 4, 6, 8 \} \quad A1 = \{5\} \quad A2 = \{7\}$$

Les transitions qui partent des états de A restent dans A, sauf 4 qui est projeté vers l'état 7 avec "a"

On crée une classe A3 pour 4

$$A = \{ 1, 3, 6, 8 \} \quad A1 = \{5\} \quad A2 = \{7\} \quad A3 = \{4\}$$

Les transitions qui partent des états de A restent dans A, sauf 6 qui est projeté vers l'état 7 avec "a"

On crée une classe A4 pour 6

$$A = \{ 1, 3, 8 \} \quad A1 = \{5\} \quad A2 = \{7\} \quad A3 = \{4\} \quad A4 = \{6\}$$

Les transitions qui partent des états de A restent dans A, sauf 3 qui est projeté vers l'état A4 avec "b"

On crée une classe A5 pour 3

$$A = \{ 1, 8 \} \quad A1 = \{5\} \quad A2 = \{7\} \quad A3 = \{4\} \quad A4 = \{6\} \quad A5 = \{3\}$$

8 est séparable de 1 dans A à cause de "b"

On crée une classe A6 pour 8

$$A = \{ 1 \} \quad A1 = \{5\} \quad A2 = \{7\} \quad A3 = \{4\} \quad A4 = \{6\} \quad A5 = \{3\} \quad A6 = \{8\}$$

La classe A est donc **entièrement séparable**

**Aucun regroupement de classe possible**

## 2- Etude de la classe B

$T(0, b) = 2$  et  $T(2, b) = 1$  n'appartiennent pas à la même classe, 0 et 2 ne sont pas équivalents

on crée la classe B1 pour 2

$$B = \{ 0 \}$$

$$B1 = \{ 2 \}$$

## Conclusion

Toutes les classes de l'automate M3D sont séparables et aucun regroupement de classes de même comportement

Cet automate est donc **minimal**

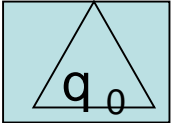
## 4.5 AEFD associé à une expression régulière

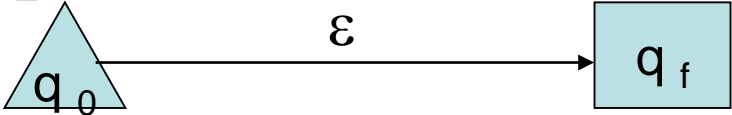
### Théorème

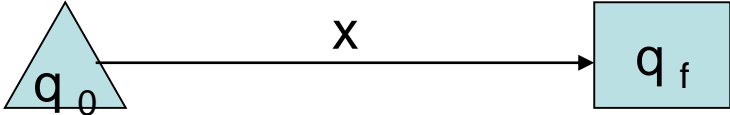
Pour toute expression régulière  $E$  définie sur un vocabulaire  $V$ , il existe un automate  $M_E$  qui accepte le langage  $L(E)$  dénoté par  $E$

### Démonstration

soit  $E$  une expression régulière

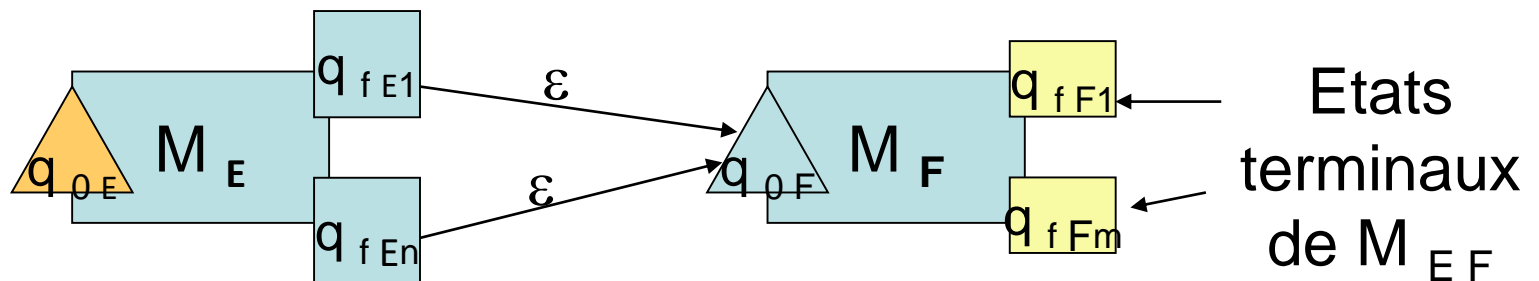
1.  $E = \emptyset : M_E =$  

2.  $E = \varepsilon : M_E =$  

3.  $E = x$  appartient à  $V : M_E =$  

Soient  $E$  et  $F$  2 expressions régulières et  $M_E$  et  $M_F$  les 2 automates associés

Automate  $M_{EF}$  associé à l'expression  $EF$



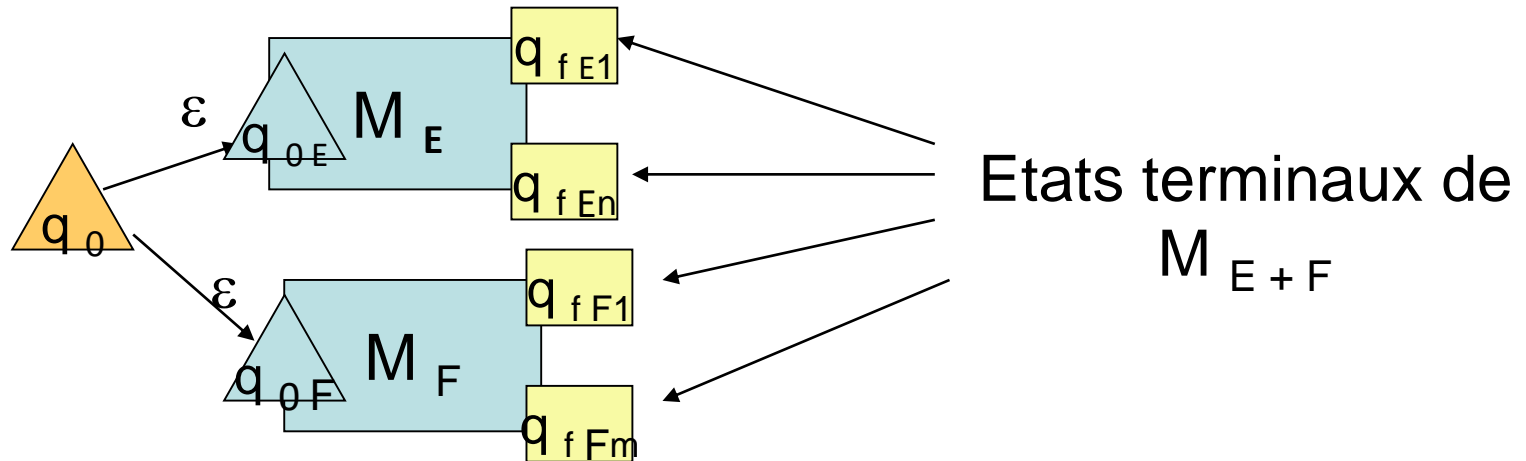
Etat initial de  $M_{EF}$

$= q_{0E}$  état initial de  $M_E$

Etats terminaux de  $M_{EF}$

$= \{ q_{fF1}, \dots, q_{fFm} \}$  états terminaux de  $M_F$

Automate  $M_{E+F}$  associé à l'expression  $E + F$



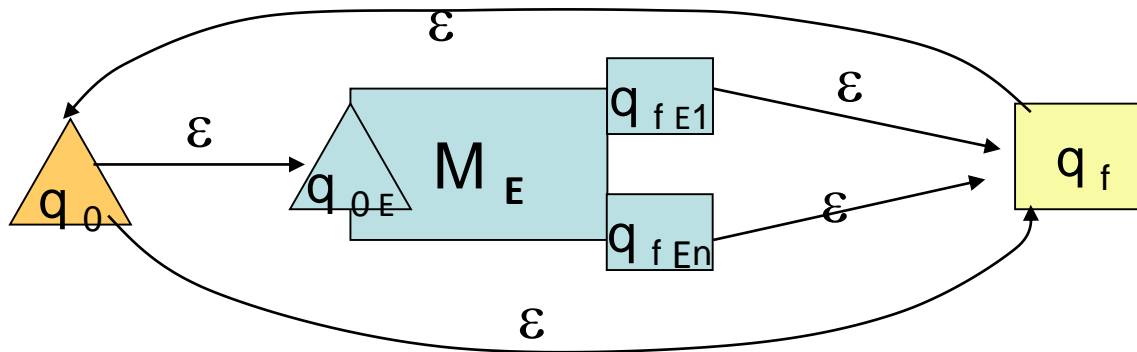
Etat initial de  $M_{E+F}$

= nouvel état  $q_0$

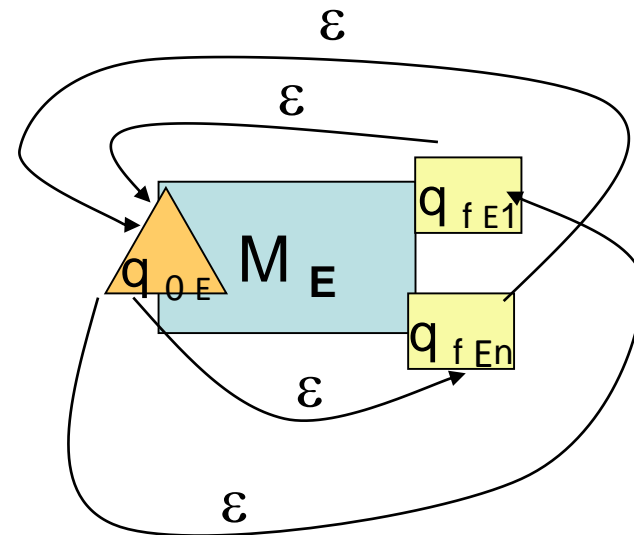
Etats terminaux de  $M_{E+F}$

= tous les états terminaux de  $M_E$  et de  $M_F$

Automate  $M_{E^*}$  associé à l'expression  $E^*$



Forme simplifiée



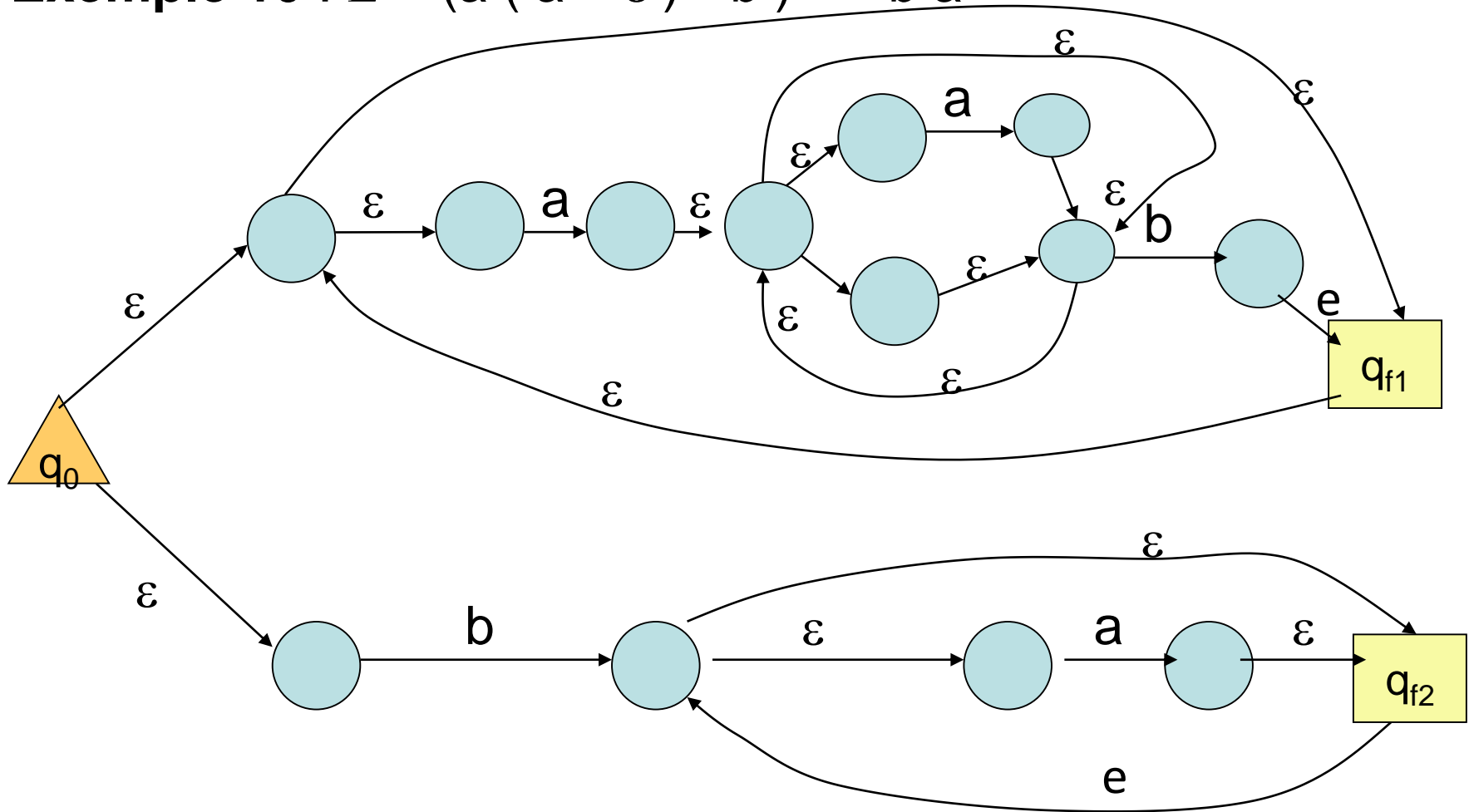
Etat **initial** de  $M_{E^*}$   
 = **nouvel** état  $q_0$

Etat **terminal** de  $M_{E^*}$   
 = **nouvel** état  $q_f$

$\epsilon$ -transitions



**Exemple 10 :**  $E = (a (a + \varepsilon)^* b)^* + b a^*$



## 4.6 Expression régulière associée à un AEFD

Soit  $L$  le langage que reconnaîtrait l'automate si  $q$  était son état initial et  $E$  l'expression régulière qui dénote  $L$

On va produire un système d'équations liant toutes les  $E_i$  associés à tous les états  $q_i$

Chaque transition  $(q_i, a) \rightarrow q_k$  produit l'équation :

$$E_i = a E_k$$

Pour chaque  $q_i$  terminal, on produit l'équation :

$$E_i = \varepsilon$$

Les équations  $E_i = a$  et  $E_i = b$  se regroupent et produisent l'équation :

$$E_i = a + b$$

Remarque

L'équation  $E_i = a E_i + b$  est équivalente à l'équation :

$$E_i = a * b$$

On cherche à réduire le système d'équation pour résoudre l'équation associée à  $E_0$

### Exemple 11 :

Trouver l'expression régulière associée à l'AEFD

$M5 = (Q, V, q_0, F, T)$

$Q = \{ 0, 1, 2, 3 \}$

$V = \{ a, b, c \}$

$q_0 = 0$

$F = \{ 1, 3 \}$

$T =$

V	a	b	c
Q			
0	1	2	-
<u>1</u>	3	0	-
2	-	-	3
<u>3</u>	2	-	-

## Système d'équations

$$E_0 = a E_1 + b E_2$$

$$E_1 = a E_3 + b E_0 + \varepsilon$$

$$E_2 = c E_3$$

$$E_3 = a E_2 + \varepsilon$$

On cherche à calculer  $E_0$

$$\begin{aligned} E_3 &= a c E_3 + \varepsilon = (a c)^* \varepsilon \\ &= (a c)^* \end{aligned}$$

$$E_2 = c (a c)^*$$

$$E_1 = a (a c)^* + b E_0 + \varepsilon$$

$$\begin{aligned} E_0 &= a E_1 + b E_2 \\ &= a (a (a c)^* + b E_0 + \varepsilon) + b c (a c)^* \\ &= a b E_0 + a a (a c)^* + a + b c (a c)^* \end{aligned}$$

$$E_0 = (a b)^* (a a (a c)^* + a + b c (a c)^*)$$

## 4.7 Automates à pile

Un automate à pile  $P$  est la donnée d'un heptuplet

$$(Q, VR, VP, \$, q_0, F, \Delta)$$

où :

- $Q$  : ensemble d'états
- $VR$  : alphabet (du "ruban")
- $VP$  : alphabet (de la "pile")
- $\$ \in VP$  : symbole initial de pile
- $q_0 \in Q$  : état initial
- $F \subseteq Q$  : ensemble d'états finaux
- $\Delta$  : ensemble de transitions

Un élément de  $\Delta$  = quintuplé

$$(q, u, \alpha) \rightarrow (q', \beta)$$

$$q, q' \in Q$$

$$u \in VR^*$$

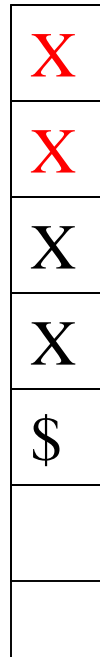
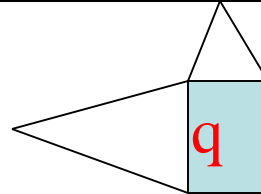
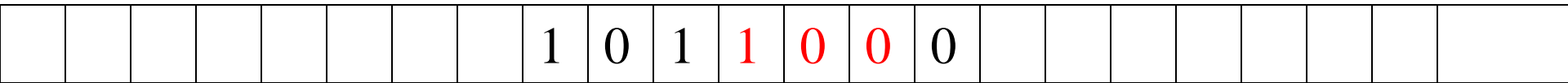
$$\alpha, \beta \in VP^*$$

si dans l'état  $q$ , l'automate peut lire le mot  $u$  sur le ruban (de gauche à droite),  
et si le mot  $\alpha$  figure en haut de la pile (on lit  $\alpha$  de haut en bas),  
alors l'automate peut

1. **passer** dans l'état  $q'$ ,
2. **lire**  $u$
3. **remplacer**  $\alpha$  par  $\beta$  au sommet de la pile

$(q, 100, XX) \rightarrow (q', Y) :$

RUBAN

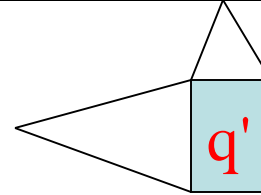
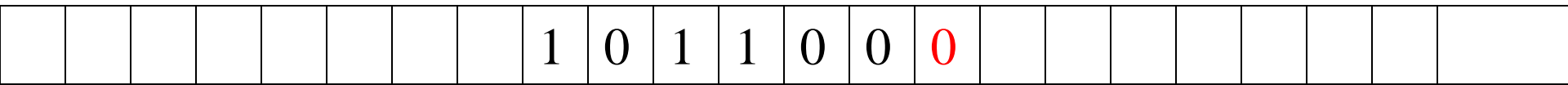


PILE



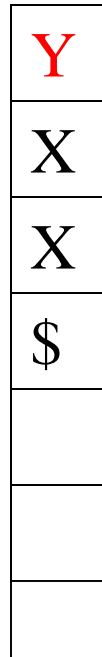
$(q, 100, XX) \rightarrow (q', Y) :$

RUBAN



$(q', 0, Y) \rightarrow \dots ?$

PILE



Cas particuliers des 2 types de transitions

$$(q, u, \varepsilon) \rightarrow (q', \beta)$$

EMPILER  $\beta$

$$(q, u, \alpha) \rightarrow (q', \varepsilon)$$

DEPILER  $\alpha$

## 4.8 Configuration

Triplet  $(q, u, v)$  où:

$q \in Q$  ;  $u \in VR^*$  ;  $v \in VP^*$

- $q$  : état courant
- $u$  : mot restant à lire (de gauche à droite)
- $v$  : contenu de la pile (de haut en bas)

- **configuration initiale** :  $(q_0, w, \$)$

$w$  = mot à reconnaître

- **configuration terminale**:

- $(q, \varepsilon, v)$  où  $q \in F$  : automate **acceptant sur état final**
- $(q, \varepsilon, \varepsilon)$  : automate **acceptant sur pile vide**

## Exemple 12

Reconnaître  $\{a^n b^n; n \geq 1\}$  "sur pile vide"

- **principe** : empiler des "a" **tant que** l'automate lit des "a",  
les dépiler **tant que** il lit des "b"
- conséquence : un automate à pile "sait" compter!

$P = (Q, VR, VP, \$, q_0, F, \Delta)$

$Q = \{0, 1, 2, 3\}$

$VR = \{a, b\}$

$VP = \{\$, a, b\}$

$q_0 = 0$

$F = 3$

$\Delta =$

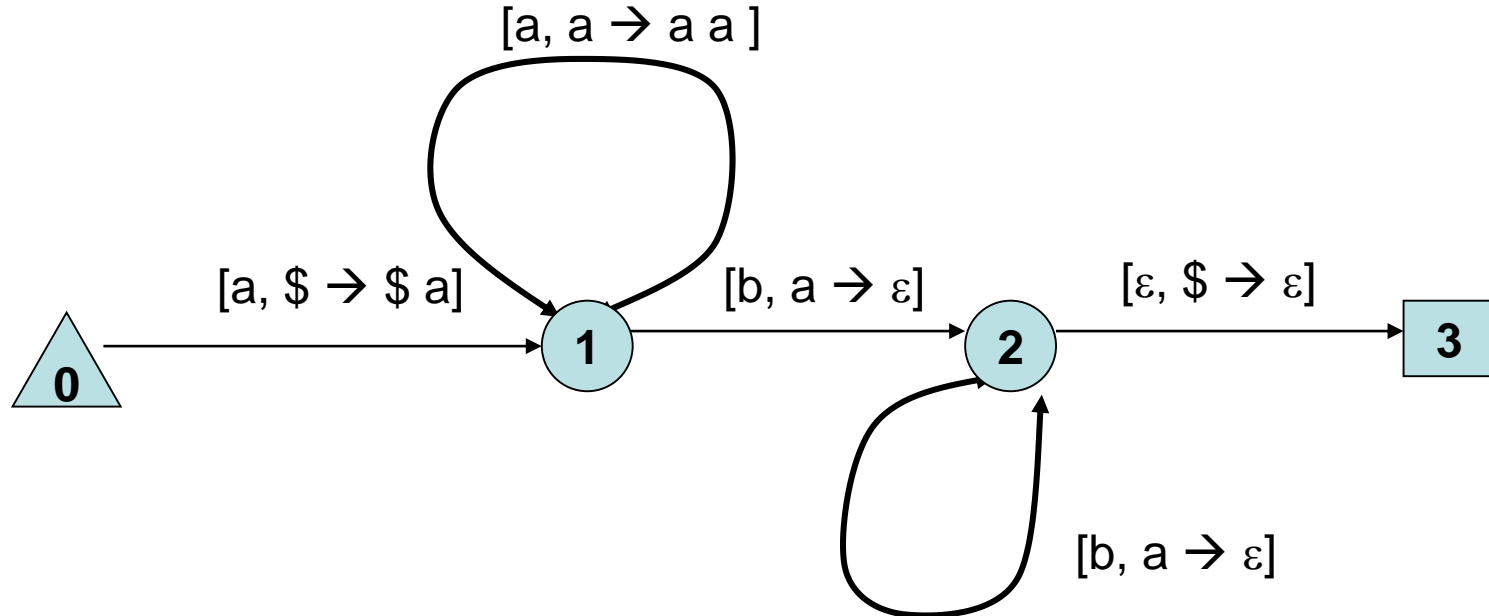
$(0, a, \$) \rightarrow (1, \$ a)$

$(1, a, a) \rightarrow (1, a a)$

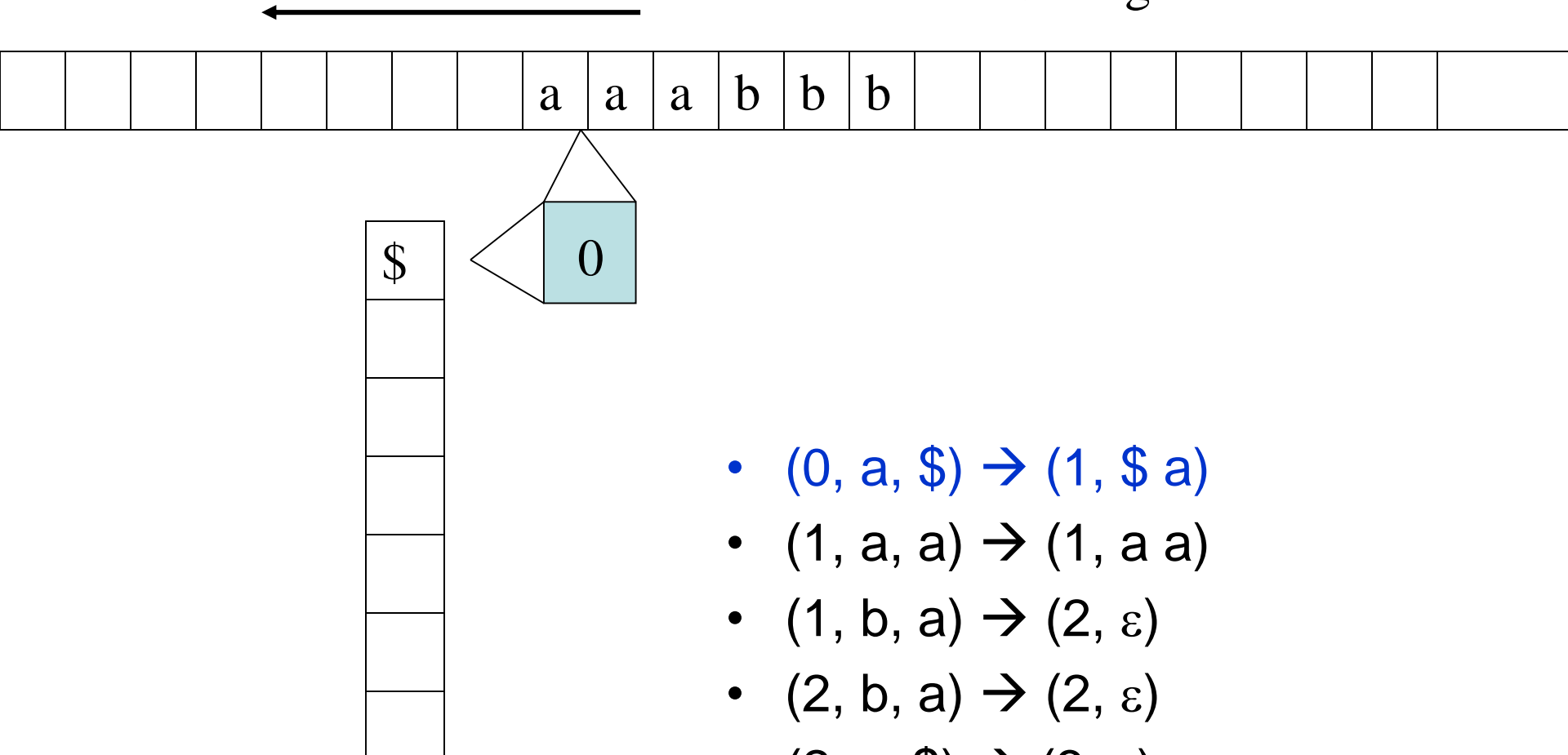
$(1, b, a) \rightarrow (2, \varepsilon)$

$(2, b, a) \rightarrow (2, \varepsilon)$

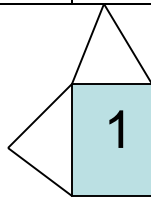
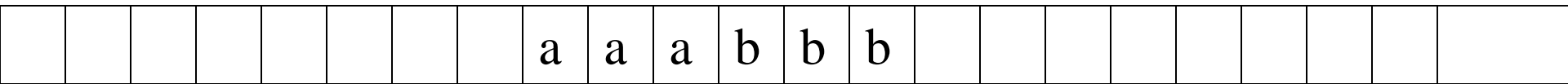
$(2, \varepsilon, \$) \rightarrow (3, \varepsilon)$



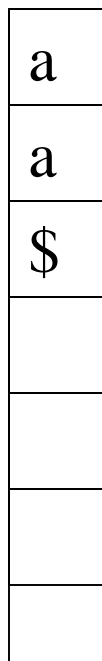
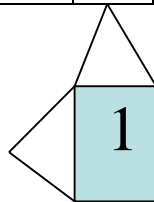
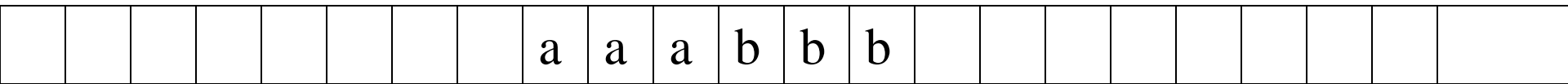
## Configuration initiale



- $(0, a, \$) \rightarrow (1, \$ a)$
- $(1, a, a) \rightarrow (1, a a)$
- $(1, b, a) \rightarrow (2, \varepsilon)$
- $(2, b, a) \rightarrow (2, \varepsilon)$
- $(2, \varepsilon, \$) \rightarrow (3, \varepsilon)$

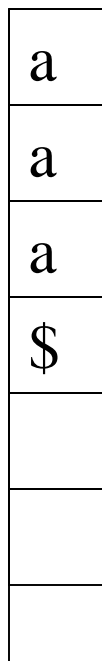
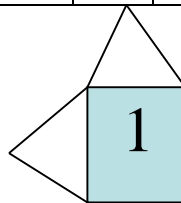
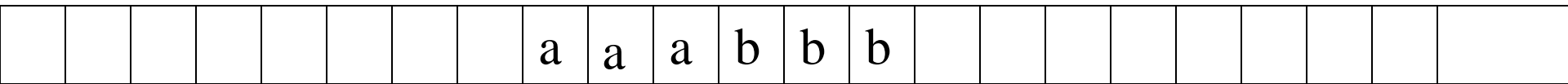


- $(0, a, \$) \rightarrow (1, \$ a)$
- $(1, a, a) \rightarrow (1, a a)$
- $(1, b, a) \rightarrow (2, \varepsilon)$
- $(2, b, a) \rightarrow (2, \varepsilon)$
- $(2, \varepsilon, \$) \rightarrow (3, \varepsilon)$

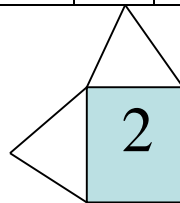
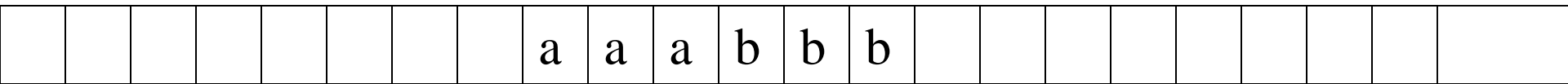


- $(0, a, \$) \rightarrow (1, \$ a)$
- $(1, a, a) \rightarrow (1, a a)$
- $(1, b, a) \rightarrow (2, \varepsilon)$
- $(2, b, a) \rightarrow (2, \varepsilon)$
- $(2, \varepsilon, \$) \rightarrow (3, \varepsilon)$

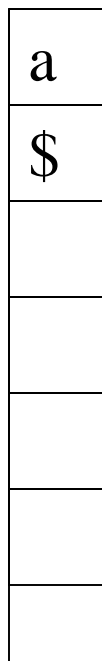
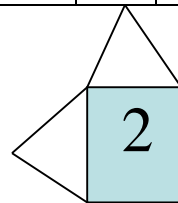
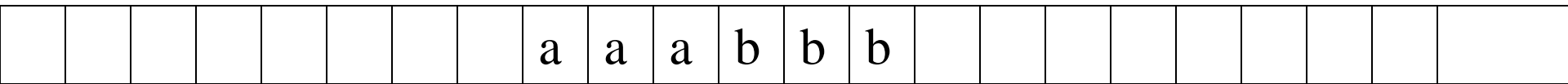




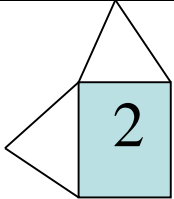
- $(0, a, \$) \rightarrow (1, \$ a)$
- $(1, a, a) \rightarrow (1, a a)$
- $(1, b, a) \rightarrow (2, \varepsilon)$
- $(2, b, a) \rightarrow (2, \varepsilon)$
- $(2, \varepsilon, \$) \rightarrow (3, \varepsilon)$



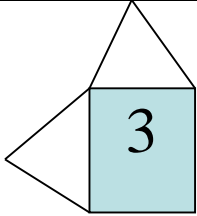
- $(0, a, \$) \rightarrow (1, \$ a)$
- $(1, a, a) \rightarrow (1, a a)$
- $(1, b, a) \rightarrow (2, \varepsilon)$
- $(2, b, a) \rightarrow (2, \varepsilon)$
- $(2, \varepsilon, \$) \rightarrow (3, \varepsilon)$



- $(0, a, \$) \rightarrow (1, \$ a)$
- $(1, a, a) \rightarrow (1, a a)$
- $(1, b, a) \rightarrow (2, \varepsilon)$
- $(2, b, a) \rightarrow (2, \varepsilon)$
- $(2, \varepsilon, \$) \rightarrow (3, \varepsilon)$



- |    |
|----|
| \$ |
|    |
|    |
|    |
|    |
|    |



## 4.9 Transformation d'une grammaire hors-contexte en un automate à pile

$$G = ( V_T, V_N, S, R )$$

Trouver P, automate à pile, qui reconnaît exactement le langage engendré par G

$$Q = \{ p, q, r \}$$

$$VR = V_T$$

$$VP = V_N \cup V_T \cup \{ \$ \} \quad ( \$ \notin V_N \text{ et } \$ \notin V_T )$$

$\$ \in VP$  : symbole initial de pile

p = état initial

r = état final

$\Delta$  : ensemble de transitions ( quintuplés )

## Algorithme de construction de l'automate à pile

-- empiler l'axiome, initialise la dérivation

ajouter la transition  $(p, \varepsilon, \varepsilon) \rightarrow (q, \$ S)$

**pour** chaque symbole terminal  $x$

-- accepte  $x$  comme l'un des symboles du mot à reconnaître

-- dépile  $x$

ajouter la transition  $(q, x, x) \rightarrow (q, \varepsilon)$

**finpour**

**pour** chaque règle de la grammaire  $A \rightarrow w$

-- dérive  $A$  en  $w$

-- remplace le sommet de pile  $A$  par  $w$

ajouter la transition  $(q, \varepsilon, A) \rightarrow (q, w)$

**finpour**

-- la pile est vide et le reste du mot à reconnaître est vide

ajouter la transition  $(q, \varepsilon, \$) \rightarrow (r, \varepsilon)$

$P = (Q, VR, VP, \$, q_0, F, \Delta)$

$Q = \{p, q, r\}$

$VR = V_T$

$VP = V_T \cup V_N \cup \{\$\}$

$q_0 = p$

$F = r$

$\Delta =$

$(p, \varepsilon, \varepsilon) \rightarrow (q, \$ S)$

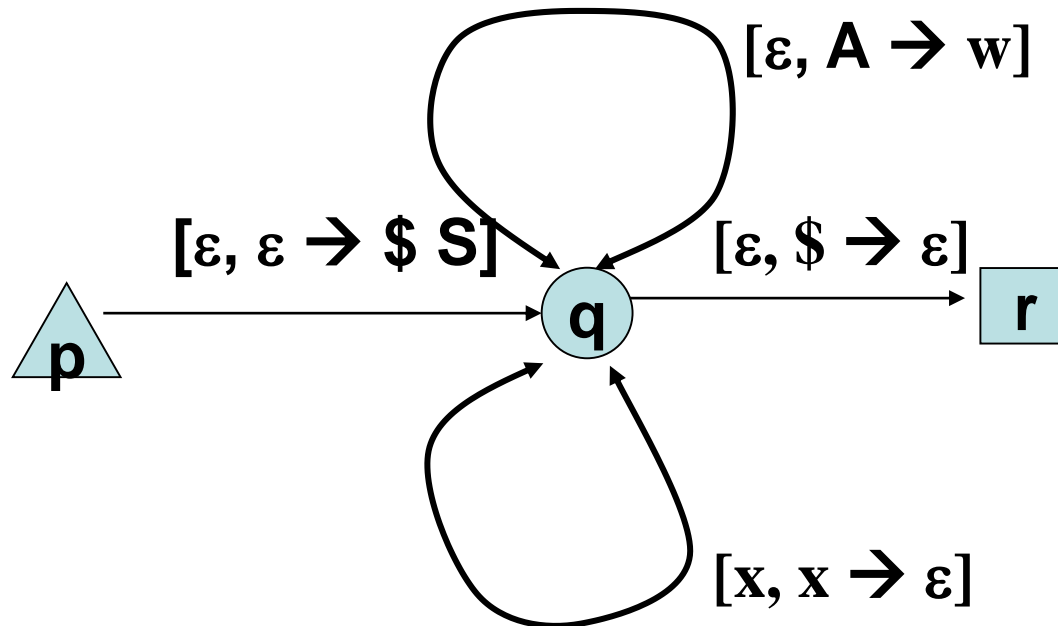
$\{ (q, x, x) \rightarrow (q, \varepsilon) \}$

pour tout  $x$  de  $VR = V_T$

$\{ (q, \varepsilon, A) \rightarrow (q, w) \}$

pour toutes les règles  $A \rightarrow w$

$(q, \varepsilon, \$) \rightarrow (r, \varepsilon)$





# Sommaire

1. Expressions régulières	3
2. Grammaires	16
3. Principes de l'analyse descendante	50
4. Automates	66
<b>5. Mise en œuvre d'une analyse descendante</b>	<b>145</b>
5.0 Principe	146
5.1 Calcul de PREMIER	147
5.2 Calcul de SUIVANT	153
5.3 Construction de la table d'analyse	157
5.4 Analyseur	158
6. Conclusion	163
Bibliographie	177

## 5.0 Principe

Construire **l'arbre** de dérivation **du haut** (la racine, c'est à dire l'axiome de départ) **vers le bas** (les feuilles).

### Table d'analyse LL(1)

Pour construire une table d'analyse, on a besoin des ensembles **PREMIER** et **SUIVANT**

## 5.1 Calcul de PREMIER

Pour toute chaîne  $\alpha$  composée de symboles terminaux et non-terminaux, on cherche  $\text{PREMIER}(\alpha)$  : l'ensemble de tous les **terminaux** qui peuvent **commencer** une chaîne qui se **dérive** de  $\alpha$  :

On cherche tous les symboles terminaux **a** tels qu'il existe une dérivation

$$\alpha \rightarrow^* \mathbf{a} \beta$$

$\beta$  étant une chaîne quelconque composée de symboles terminaux **et** non-terminaux.

## Exemple

$G1 = \{ V_N, V_T, S, R \}$

$V_N = \{ S, B, P \}$

$V_T = \{ a, b, c, d \}$

Racine = S

$R = \{$   
     $S \quad \rightarrow B a$   
     $B \quad \rightarrow c P \mid b P \mid P \mid \varepsilon$   
     $P \quad \rightarrow d S$   
     $\}$

$S \rightarrow^* a$             donc a appartient à PREMIER (S)

$S \rightarrow^* c P a$         donc c appartient à PREMIER (S)

$S \rightarrow^* b P a$         donc b appartient à PREMIER (S)

$S \rightarrow^* d S a$         donc d appartient à PREMIER (S)

Pas de dérivation  $S \rightarrow^* e$

Donc  $\text{PREMIER (S)} = \{a, b, c, d\}$

## Algorithme de construction de PREMIER (X)

1. **Si**  $X$  est un terminal,  $\text{PREMIER}(X) = \{X\}$
2. **Si**  $X$  est un non-terminal et  $X \rightarrow \varepsilon$  est une production **alors**  
ajouter  $\varepsilon$  dans  $\text{PREMIER}(X)$
3. **Si**  $X$  est un non-terminal et  $X \rightarrow Y_1 Y_2 \dots Y_n$  est une production (avec  $Y_i$  symbole terminal ou non-terminal) **alors**  
ajouter les éléments de  $\text{PREMIER}(Y_1)$  **sauf**  $\varepsilon$  dans  $\text{PREMIER}(X)$   
**si** il existe  $j$  dans  $[2, n]$  tel que pour tout  $i=1, \dots, j-1$ ,  $\varepsilon$  appartient  
à  $\text{PREMIER}(Y_i)$  **alors**  
ajouter les éléments de  $\text{PREMIER}(Y_j)$  **sauf**  $\varepsilon$  dans  
 $\text{PREMIER}(X)$   
**si** pour tout  $i=1, \dots, n$   $\varepsilon$  appartient à  $\text{PREMIER}(Y_i)$  **alors**  
ajouter  $\varepsilon$  dans  $\text{PREMIER}(X)$
4. Recommencer jusqu'à ce qu'on n'ajoute **rien de nouveau** dans les ensembles PREMIER

## Algorithme de construction de PREMIER ( $\alpha$ )

Soit  $\alpha$  une suite de terminaux (de  $V_T$ ) et non-terminaux (de  $V_N$ )

$$\alpha = X_1 X_2 \dots X_n$$

**Pour**  $i = 1, \dots, n$

calculer PREMIER (  $X_i$  )

**fin pour**

**Si** pour tout  $i = 1, \dots, n$   $\varepsilon$  appartient à PREMIER (  $X_i$  )

**Alors**

ajouter  $\varepsilon$  dans PREMIER (  $\alpha$  )

Ajouter les éléments de PREMIER (  $X_1$  ) **sauf**  $\varepsilon$  dans PREMIER (  $\alpha$  )

**Si** il existe  $j$  dans  $[ 2, n ]$  tel que pour tout  $i = 1, \dots, j - 1$ ,  
 $\varepsilon$  appartient à PREMIER (  $X_i$  )

**Alors**

ajouter les éléments de PREMIER (  $X_j$  ) **sauf**  $\varepsilon$  dans PREMIER (  $\alpha$  )

### Exemple 1 :

$G_2 = (V_N, V_T, S, R)$

$V_N = (E, E', T, T', F)$

$V_T = (+, *, (, ), nb)$

Racine = E

$R = \{$   
1 E  $\rightarrow T E'$   
2 3 E'  $\rightarrow + T E' \quad | \varepsilon$   
4 T  $\rightarrow F T'$   
5 6 T'  $\rightarrow * F T' \quad | \varepsilon$   
7 8 F  $\rightarrow ( E ) \quad | nb$   
 $\}$

PREMIER (E) = PREMIER (T) = { (, nb }

PREMIER (E') = { +,  $\varepsilon$  }

PREMIER (T) = PREMIER (F) = { (, nb }

PREMIER (T') = { \*,  $\varepsilon$  }

PREMIER (F) = { (, nb }

	PREMIER
E	(, nb 7 8
E'	+, $\varepsilon$ 2 3
T	(, nb 7 8
T'	*, $\varepsilon$ 5 6
F	(, nb 7 8

## Exemple 2 :

$G_3 = \{ V_N, V_T, S, R \}$

$V_N = \{ S, A, B, C \}$

$V_T = \{ a, b, c, d \}$

Racine = S

$R = \{$

1 S  $\rightarrow$  A B C

2 3 A  $\rightarrow$  a A |  $\varepsilon$

4 5 6 B  $\rightarrow$  b B | c B |  $\varepsilon$

7,8,9 C  $\rightarrow$  d e | d a | d A

$\}$

PREMIER (S) = { a, b, c, d }

PREMIER (A) = { a,  $\varepsilon$  }

PREMIER (B) = { b, c,  $\varepsilon$  }

PREMIER (C) = { d }

	PREMIER
S	a, b, c, d 1 2 3 4 5 6 7 8 9
A	a, $\varepsilon$ 2 3
B	b, c, $\varepsilon$ 4 5 6
C	d 7 8 9



## 5.2 Calcul de SUIVANT

Pour tout non-terminal  $A$ , on cherche  $\text{SUIVANT}(A)$  : l'ensemble de tous les symboles terminaux  $a$  qui peuvent apparaître immédiatement à droite de  $A$  dans une dérivation

$$S \rightarrow^* \alpha A a \beta$$

### Exemple

$$G3 = \{ V_N, V_T, S, R \}$$

$$V_N = \{ S, A, B, C \}$$

$$V_T = \{ a, b, c, d, e \}$$

$$R = \left\{ \begin{array}{lcl} S & \rightarrow & A B C \\ A & \rightarrow & a A \mid \varepsilon \\ B & \rightarrow & b B \mid c B \mid \varepsilon \\ C & \rightarrow & d e \mid d a \mid d A \end{array} \right\}$$

$b, c, d$  appartiennent à  $\text{SUIVANT}(A)$  à cause des dérivations

$$B \rightarrow b B \mid c B \mid \varepsilon$$

La 3è met en jeu  $C$  et ses PREMIER

$$C \rightarrow d e \mid d a \mid d A$$

## Algorithme de construction de SUIVANT ( X )

Ajouter un **marqueur de fin de chaîne** (symbole \$ par exemple) au texte à analyser

$SUIVANT ( S ) = \{ \$ \}$  où **S est l'axiome** de la grammaire)

### Repeter

**Si**  $A \rightarrow \alpha B \beta$  est une production, où B est un non-terminal, **alors**  
ajouter le contenu de  $PREMIER ( \beta )$  à  $SUIVANT ( B )$ , **sauf**  $\epsilon$

**Si**  $A \rightarrow \alpha B$  est une production **alors**  
ajouter  $SUIVANT ( A )$  à  $SUIVANT ( B )$

**Si**  $A \rightarrow \alpha B \beta$  est une production avec  $\epsilon$  dans  $PREMIER ( \beta )$  **alors**  
ajouter  $SUIVANT ( A )$  à  $SUIVANT ( B )$

**Jusqu'à** ce qu'on n'ajoute rien de nouveau dans les ensembles SUIVANT.

## Exemple 1 :

$G2 = (V_N, V_T, S, R)$

$V_N = (E, E', T, T', F)$

Racine = E

$V_T = (+, *, (, ), \text{nb})$

$R = \{$   
     1    E        $\rightarrow T E'$   
     2 3   E'       $\rightarrow + T E'$   
     4    T        $\rightarrow F T'$   
     5 6   T'       $\rightarrow * F T'$   
     7 8   F        $\rightarrow ( E )$   
      $\}$

|  $\varepsilon$

|  $\varepsilon$

| nb

	PREMIER	SUIVANT
E	(, nb	\$, ) 7
E'	+, $\varepsilon$	\$, ) 1
T	(, nb	\$, ), + 1,2
T'	*, $\varepsilon$	\$, ), + 4
F	(, nb	\$, ), +, * 4,5

## Exemple 2 :

$G4 = \{ V_N, V_T, S, R \}$

$V_N = \{ S, A, B \}$

$V_T = \{ a, b, c, d, e \}$

Racine = S

$R = \{$

1 2 3 S  $\rightarrow$  a S b

4 5 A  $\rightarrow$  a A d B

6 B  $\rightarrow$  b b

$\}$

| c d | S A e

|  $\varepsilon$

	PREMIER	SUIVANT
S	A, c	<span style="color: red;">\$,b,a,e</span> <span style="color: blue;">2, 5, 3</span>
A	A, $\varepsilon$	e,d <span style="color: blue;">3, 4</span>
B	b	e,d <span style="color: blue;">4</span>

## 5.3 Construction de la table d'analyse

La table d'analyse est une matrice  $M$  à deux dimensions qui indique pour chaque **symbole non-terminal**  $A$  et chaque **symbole terminal**  $a$  ou symbole  $\$$  la **règle** de production à placer dans une des cases  $M ( A, x )$ ..

**Pour** chaque production  $A \rightarrow v$

1. **Pour** tout  $a$  de  $\text{PREMIER}(v)$  **différent de  $\varepsilon$ , faire**  
rajouter la production  $A \rightarrow v$  dans la case  $M [ A, a ]$

2. **Si**  $\varepsilon$  est dans  $\text{PREMIER}(v)$



**Alors**

**Pour** chaque  $b$  de  $\text{SUIVANT}(A)$   
ajouter  $A \rightarrow v$  dans  $M [ A, b ]$

**Fin pour**

**Fin pour**

-- Chaque case  $M [ A, a ]$  vide est une erreur de syntaxe .

### Exemple :

Avec G2

1 E → T E'  
2 3 E' → + T E' | ε  
4 T → F T'  
5 6 T' → \* F T' | ε  
7 8 F → ( E ) | nb

### Table d'analyse

<b>VN</b> <b>VT</b>	<b>nb</b>	<b>+</b>	<b>*</b>	<b>(</b>	<b>)</b>	<b>\$</b>
<b>E</b>	$E \rightarrow T E'$			$E \rightarrow T E'$		
<b>E'</b>		$E' \rightarrow +T E'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
<b>T</b>	$T \rightarrow F T'$			$T \rightarrow F T'$		
<b>T'</b>		$T' \rightarrow \varepsilon$	$T' \rightarrow *F T'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
<b>F</b>	$F \rightarrow nb$			$F \rightarrow ( E )$		

## 5.4 Analyseur

Maintenant qu'on a la table, comment l'utiliser pour déterminer si un mot  $w$  donné est tel que :

$$S \rightarrow^* w$$

On utilise une **pile**.

### Algorithme :

données : mot  $w$  à reconnaître, table d'analyse  $M$

### Initialisation

- placer \$ comme dernière lettre du mot  $w$
- placer \$ puis  $S$  dans la pile
- lire la 1ère lettre "a" du mot  $w$

Pile

S
\$

## Repeter

Soit "X" le symbole en **sommet de pile** et "a" la lettre du mot lue

**Si** X est un non terminal **alors**

**Si**  $M[X, a] = X \rightarrow Y_1 \dots Y_n$  **Alors**

enlever X de la pile

mettre  $Y_n$  puis  $Y_{n-1}$  puis ... puis  $Y_1$  dans la pile

émettre en sortie la production  $X \rightarrow Y_1 \dots Y_n$

**Sinon**

**ERREUR**

**Finsi**

**Sinon**

**Si**  $X = \$$  **Alors**

**Si**  $a = \$$  **Alors** **ACCEPTER** w

**Sinon** **ERREUR**

**Finsi**

**Sinon**

**Si**  $X = a$  **Alors**

enlever X de la pile

avancer *la lecture d'un caractère*

**Sinon**

**ERREUR**

**Finsi**

**Finsi**

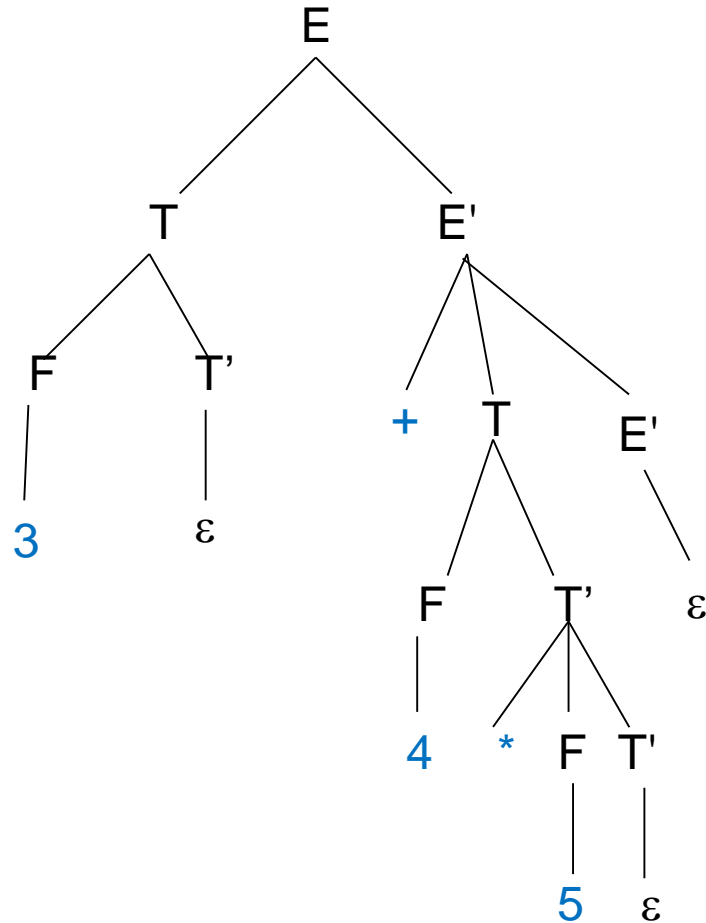
**Finsi**

**Jusqu'à** **ERREUR** ou **ACCEPTER** w



# Analyse du mot

$w = 3 + 4 * 5$



PILE

Entrée

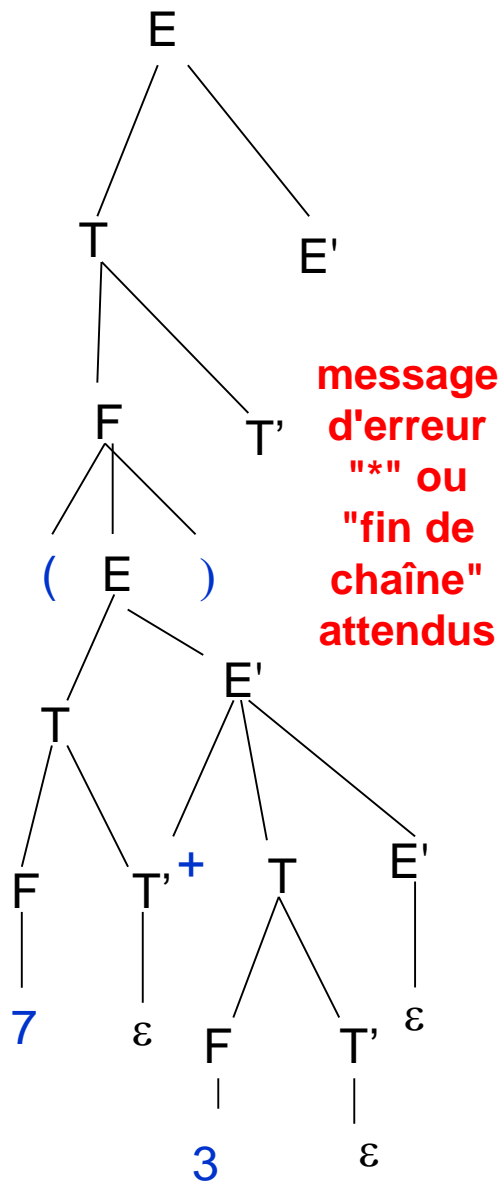
Sortie

\$ E	3 + 4 * 5 \$	$E \rightarrow T E'$
\$ E' T	3 + 4 * 5 \$	$T \rightarrow F T'$
\$ E' T' F	3 + 4 * 5 \$	$F \rightarrow \text{nb}$
\$ E' T' 3	3 + 4 * 5 \$	
\$ E' T'	+ 4 * 5 \$	$T' \rightarrow \varepsilon$
\$ E'	+ 4 * 5 \$	$E' \rightarrow + T E'$
\$ E' T +	+ 4 * 5 \$	
\$ E' T	4 * 5 \$	$T \rightarrow F T'$
\$ E' T' F	4 * 5 \$	$F \rightarrow \text{nb}$
\$ E' T' 4	4 * 5 \$	
\$ E' T'	* 5 \$	$T' \rightarrow * F T'$
\$ E' T' F *	* 5 \$	
\$ E' T' F	5 \$	$F \rightarrow \text{nb}$
\$ E' T' 5	5 \$	
\$ E' T'	\$	$T' \rightarrow \varepsilon$
\$ E'	\$	$E' \rightarrow \varepsilon$
\$	\$	

**arbre complet = analyse syntaxique réussie**

# Analyse du mot

$w = ( 7 + 3 ) 5$



## PILE

\$ E  
\$ E' T  
\$ E' T' F  
\$ E' T' ) E (  
\$ E' T' ) E  
\$ E' T' ) E' T  
\$ E' T' ) E' T' F  
\$ E' T' ) E' T' 7  
\$ E' T' ) E' T'  
\$ E' T' ) E'  
\$ E' T' ) E' T +  
\$ E' T' ) E' T  
\$ E' T' ) E' T' F  
\$ E' T' ) E' T' 3  
\$ E' T' ) E' T'  
\$ E' T' ) E'  
\$ E' T' )  
\$ E' T'

## Entrée

( 7 + 3 ) 5 \$  
( 7 + 3 ) 5 \$  
( 7 + 3 ) 5 \$  
( 7 + 3 ) 5 \$  
7 + 3 ) 5 \$  
7 + 3 ) 5 \$  
7 + 3 ) 5 \$  
7 + 3 ) 5 \$  
+ 3 ) 5 \$  
+ 3 ) 5 \$  
+ 3 ) 5 \$  
3 ) 5 \$  
3 ) 5 \$  
3 ) 5 \$  
) 5 \$  
) 5 \$  
) 5 \$  
5 \$

## Sortie

$E \rightarrow T E'$   
 $T \rightarrow F T'$   
 $F \rightarrow ( E )$   
 $E \rightarrow T E'$   
 $T \rightarrow F T'$   
 $F \rightarrow \text{nb}$   
 $T' \rightarrow \epsilon$   
 $E' \rightarrow + T E'$   
 $T \rightarrow F T'$   
 $F \rightarrow \text{nb}$   
 $T' \rightarrow \epsilon$   
 $E' \rightarrow \epsilon$

arbre incomplet = **ERREUR** de  
syntaxe :  $M[T, \text{nb}]$  est vide  $\leftrightarrow$  5 n'est pas le 1<sup>er</sup>  
caractère de la partie gauche des règles 5 et 6: \* ou  $\epsilon$

# Sommaire

1. Expressions régulières	3
2. Grammaires	16
3. Principes de l'analyse descendante	50
4. Automates	66
5. Mise en œuvre d'une analyse descendante	145
<b>6. Conclusion</b>	<b>164</b>
<b>Bibliographie</b>	<b>177</b>

## 6. Conclusion

Faire exécuter

```
For i := 1 to vmax do  
  a := a + i ;
```

par un ordinateur qui ne reconnaît que du binaire

**Moyen : Traduire** le texte en binaire

Comment traduire ?

A l'aide d'un **compilateur** ou traducteur de langages

En 1957

1er compilateur :

langage Fortran (Backus et al.)

18 hommes années de développement

Depuis 2000

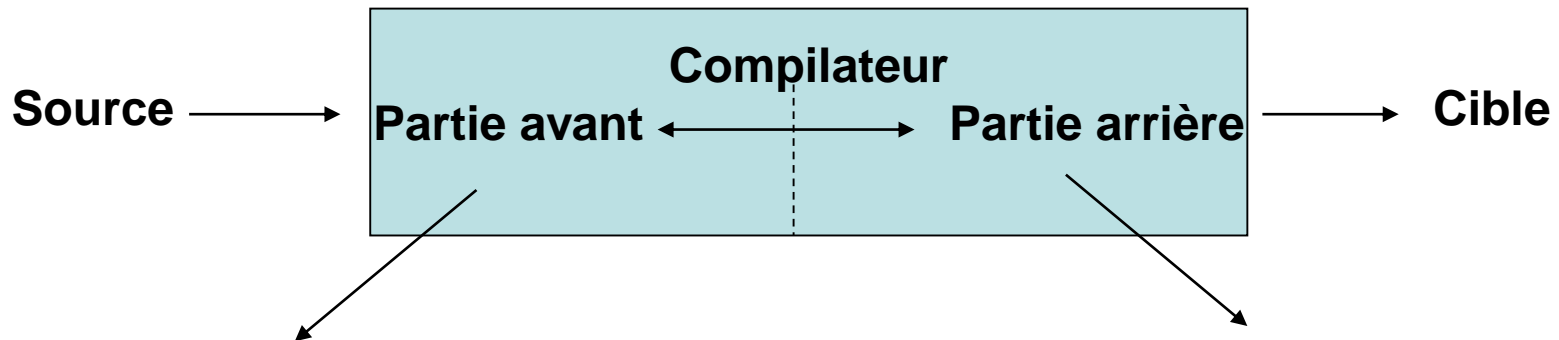
compilateur standard :

1 semestre étudiant

De 1957 à 2022 :

- compréhension des tâches à réaliser
- définitions formelles
- découverte de fondements théoriques qui apportent des résultats pratiques
- avancées méthodologiques dans la production de logiciels de grande taille
- développement d'outils logiciels puissants et fiables

# Modèle de compilateur



**Analyse** le **texte source** (programme)  
conformément à la  
définition du **langage source**  
**indépendamment**  
du langage **cible**

**Produit** le **texte cible** (programme)  
conformément à la  
définition du **langage cible**  
**indépendamment**  
du langage **source**

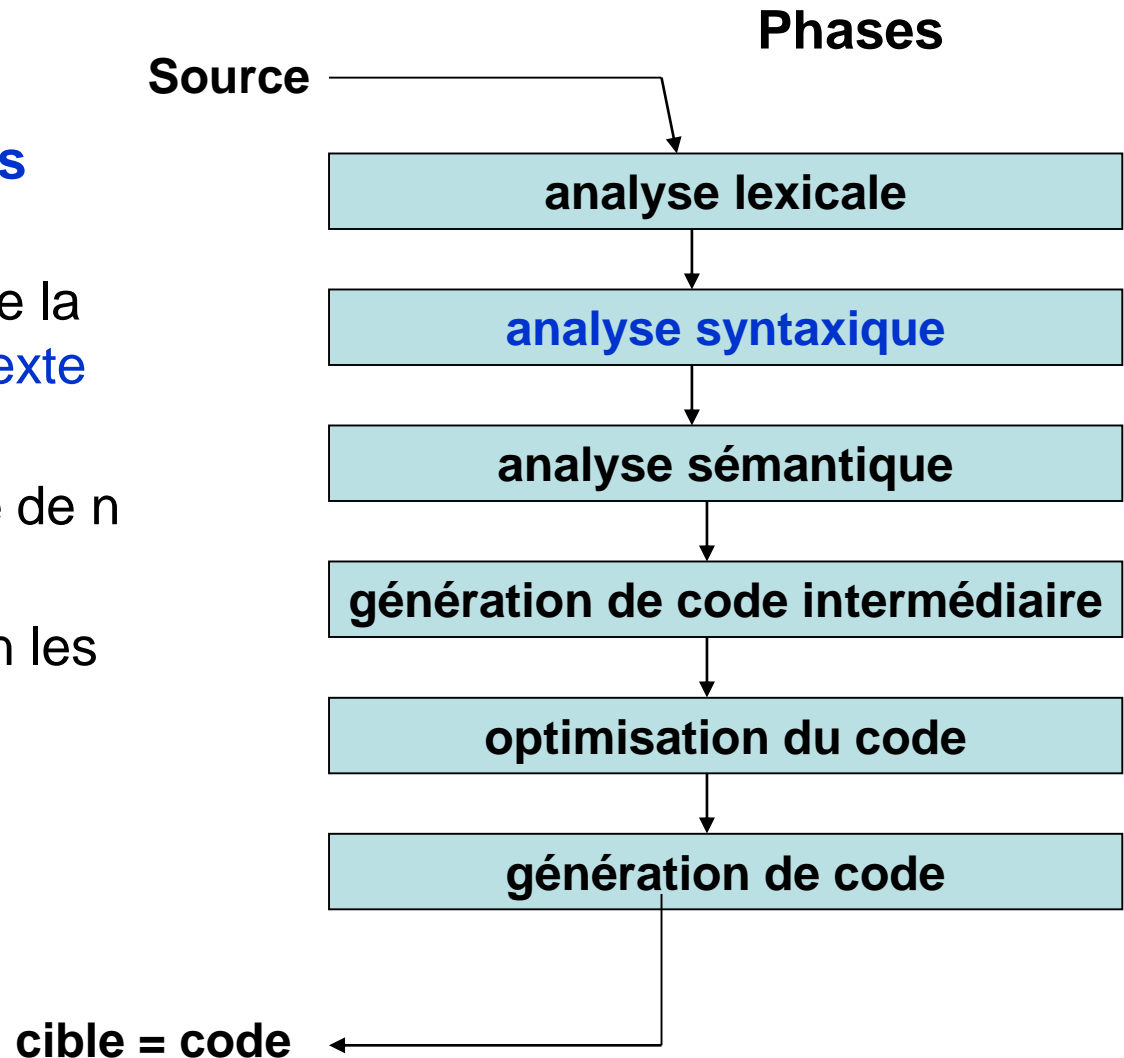
Les parties avant et arrière communiquent grâce à des structures de données intermédiaires : tables, graphes

## 1 ou plusieurs **passes**

**Passe** ; processus de  
parcours complet de la  
représentation du **texte**  
**source**

1 passe est composée de n  
**phases**

Plusieurs passes selon les  
langages





# Phase d'analyse lexicale

Traite le texte source

- 1- découpe le texte en unités lexicales : lexèmes ou "tokens"
  - mots clés
  - mots réservés
  - constantes entières
  - identificateurs
  - séparateurs, opérateurs
  - espaces, caractères spéciaux
  - commentaires
- 2 - élimine les unités lexicales inutiles pour la compilation
  - espaces, caractères spéciaux
  - commentaires

# Phase d'analyse lexicale

```
for i := 1 to vmax do  
  a := a + i ;
```

for	mot clé
i	identificateur
:=	affectation
1	entier
to	mot clé
vmax	identificateur
do	mot clé
a	identificateur
:=	affectation
a	identificateur
+	opérateur
i	identificateur
;	séparateur

# Phase d'analyse lexicale

3 – Interagit avec

- Gestionnaires de la **table des symboles**
- Analyseur syntaxique  
(entrée = unités lexicales)

4- Mémorise le source pour  
l'environnement de  
développement :

éditeurs, messages d'erreurs,  
débugueurs,...

**Table des symboles**

n° symbole	token	type
10	for	mot clé
11	to	mot clé
12	do	mot clé
13	;	séparateur
...	...	...
100	:=	affectation
101	+	opérateur
...	...	...
1000	i	identificateur
1001	vmax	identificateur
...	...	...
5001	1	entier

# Phase d'analyse lexicale

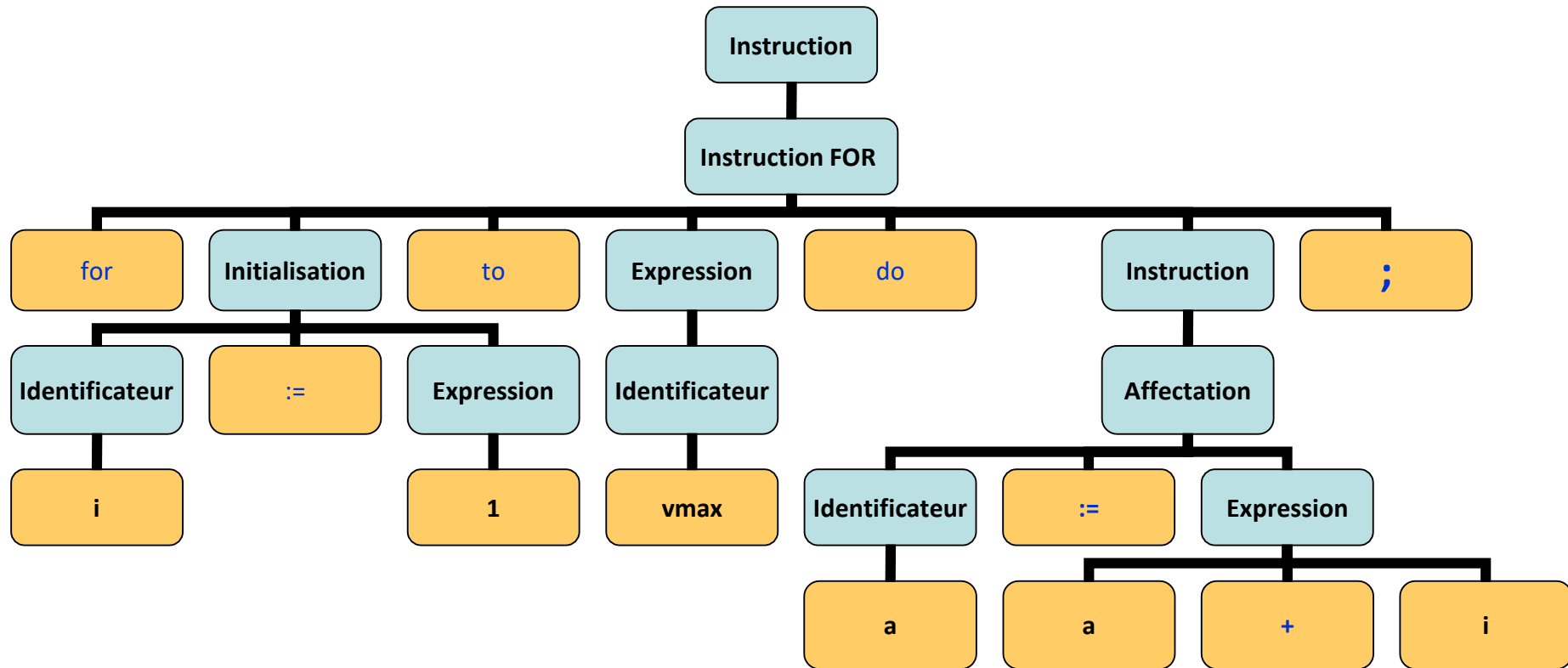
5- Produit la sortie

**10 1000 100 5001 11 1001 12 1002 100 1002 101 1000 13**

# Phase d'analyse syntaxique

- 1 - Analyse le flot d'unités lexicales conformément à la **syntaxe du langage source**, à sa **grammaire**
- 2 - Produit
  - un **arbre de dérivation** : donnée codant la structure syntaxique reconnue
  - d'éventuels **messages d'erreurs** (syntaxiques)
- 3- Interagit avec  
Analyseur sémantique

# Phase d'analyse syntaxique : arbre de dérivation / arbre syntaxique



$V_N$  = Instruction, Instruction FOR, Initialisation, Expression, Identificateur, Affectation

$V_T$  = for, to, do, ";", :=, +, i, a, 1

$R$  = ... Instruction FOR  $\rightarrow$  for Initialisation to Expression do Instruction ; ...

# Phase d'analyse sémantique

Vérification

- du type des variables

**Exemple :**    i de type entier  
                  a est un nombre

- des opérations à tous les niveaux de l'arbre syntaxique

# Génération de code

Code généré pour du langage machine ou d'assemblage

## Exemple

var\_a           A000

var\_i           A001     ➔ étiquettes des variables

var\_vmax        A002

loop :

mov A0, (var\_i)

jge A0, (var\_vmax, fin\_for)     ➔ si i > v\_max alors fin\_for

mov A0, (var\_a)

add A0, A0, (var\_i)     ➔ a := a + i

mov var\_a, A0

mov A0, (var\_i)

add A0, A0, 1           ➔ i := i + 1

fin\_for



# Bibliographie

- Aho Alfred, Ullman Jeffrey, Concepts fondamentaux de l'Informatique, Dunod, 1993 (version française)
- Wolper Pierre, Introduction à la calculabilité, Interéditions, 1991