



# Compilation

---

S. Gire  
F. Monin  
V.-A. Nicolas

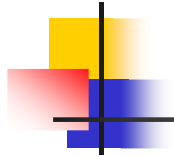
gire@univ-brest.fr  
monin@univ-brest.fr  
vnicolas@univ-brest.fr



## Organisation de l'UE

---

- Méthodes d'enseignement
  - CM, TD, TP
  - Semaines 2 et 3 : cours à distance COMP1 sur <https://moodlescience.univ-brest.fr/moodle/>
  - Projet (en TP)
- Évaluation (première session)
  - Écrit (2h, coefficient 3)
  - Projet + exercices notés (coefficient 1)



# Plan de l'UE

---

Introduction

1. Analyse lexicale
2. Grammaires algébriques
3. Analyse syntaxique
4. Analyse sémantique
5. Production de code

Conclusion - bilan

3



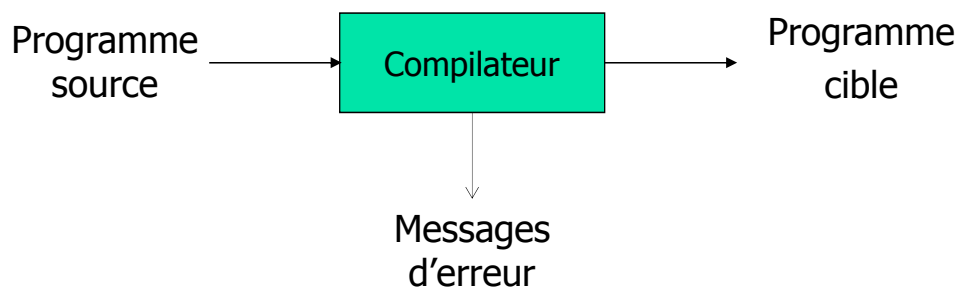
# Introduction

---

4

# Compilation ?

- Traduction d'un programme dans un langage *source* en un programme dans un langage *cible*
- Plus diagnostic de correction (messages d'erreur)



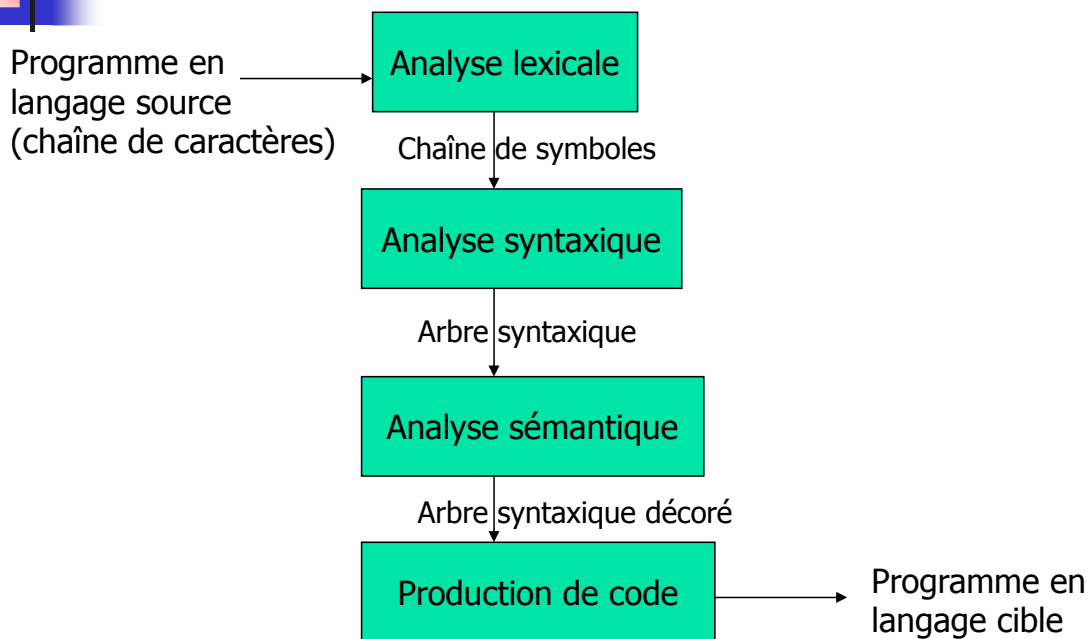
5

# Phases de la compilation

- Phases d'analyses :
  - lexicale
  - syntaxique
  - sémantique
- Phases de synthèse :
  - production de code intermédiaire
  - optimisations (dépendantes ou non du code)
  - production du code
- Traitements parallèles :
  - Table des symboles
  - erreurs

6

# Phases de la compilation



7

## Pourquoi étudier la compilation ?

- Permet d'écrire soi-même un traducteur
- Permet de mieux comprendre le pourquoi des syntaxes des langages de programmation existants
- Les outils utilisés sont
  - formateurs
  - utilisés dans d'autres domaines

8



# Outils de la compilation

---

- Phases d'analyses :
  - lexicale expressions régulières, AEF
  - syntaxique grammaires, automates à pile
  - sémantique traduction dirigée par la syntaxe
- Phases de synthèse :
  - code intermédiaire traduction dirigée par la syntaxe
  - optimisations
  - production du code
- Traitements parallèles :
  - Table des symboles
  - erreurs

9



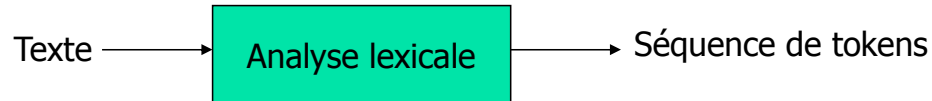
## 1. Analyse lexicale

---



# 1. Analyse lexicale

---



- Transformation d'un texte (une chaîne de caractères) en une séquence de *tokens* (ou d'*unités lexicales*)
- Reconnaissance de l'alphabet du programme écrit en langage source

11



## Analyse lexicale : plan

---

- 1.1. Analyseur lexical
- 1.2. Analyse lexicale et théorie des langages
- 1.3. Génération d'analyseurs lexicaux

12

## Exemple

```
var  a,bif: integer;  <NL>  
a:=2;                <NL>  
bif:=a*a+1;          <NL>
```



```
motclé  ident virg ident deuxpt type ptvirg  
ident affect entier ptvirg  
ident affect ident mult ident add entier ptvirg
```

13

## 1.1. Analyse lexicale

Chaînes de  
caractères



Analyseur lexical



Séquence de tokens

### ■ *Token (unité lexicale)* :

mots avec lesquels les phrases sont  
formées

Exemples : identificateur, motcléIF, entier,  
opérateur, etc

14

# Analyse lexicale

- Reconnait les mots clés du langage (begin, end, if, else...)
- Reconnait les identificateurs, les constantes, les séparateurs, etc
- Reconnait et ignore les symboles non pertinents pour la suite de l'analyse (espaces, line feeds, commentaires...)
- Reconnait les directives pour le compilateur
- + Réalise souvent l'encodage de certains symboles (identificateurs...) avec une table des symboles

15

## Exemple (suite)

```
var  a,bif: integer;  <NL>  
a:=2;                <NL>  
bif:=a*a+1;          <NL>
```



```
motclé  ident virg ident deuxpt type ptvirg  
ident affect entier ptvirg  
ident affect ident mult ident add entier ptvirg
```

+

Table des  
symboles

```
« a », entier, ...  
« bif »,entier, ...
```

16



## 1.2. Analyse lexicale et théorie des langages

- Unités lexicales (tokens)  
= expressions régulières  
(cf langages réguliers/rationnels)  
(En pratique, descriptions régulières au lieu d'expressions régulières)
- Outil de reconnaissance associé : automate à états fini (AEF)

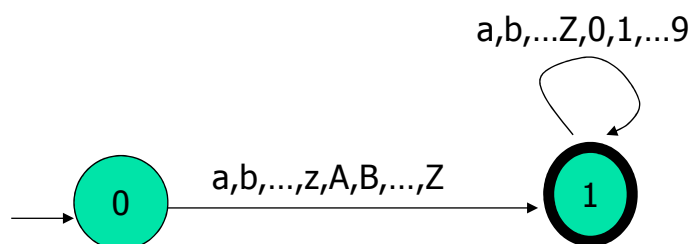
17

### Exemple : les identificateurs

Token : ident

Définition régulière :  $[a-zA-Z][a-zA-Z0-9]^*$

AFD :



18

## 1.3. Génération d'analyseurs lexicaux

Expressions régulières

Générateur d'analyseur lexical

Automate à états fini déterministe

- À la main :
  - facile à faire,
  - mais modifications difficiles
- Automatiquement avec la famille des outils Lex : Lex et Flex pour C, Jflex et JFlex pour Java

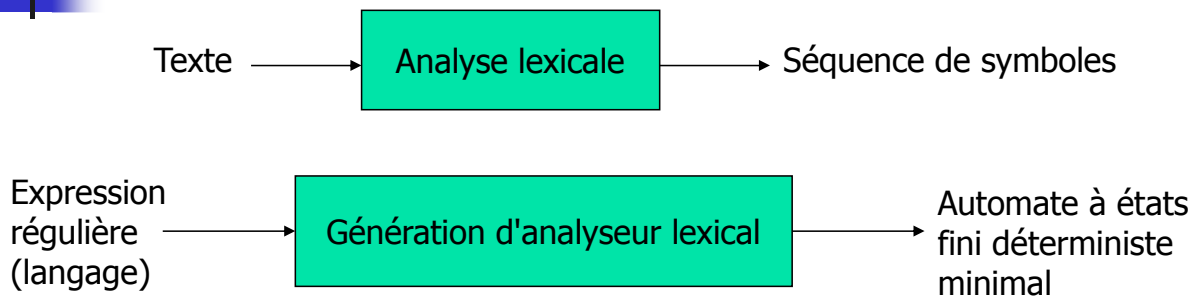
19

## Principe des générateurs

- Générer un automate à états fini déterministe (AFD) à partir d'une expression régulière (ER)
- En plusieurs étapes :
  - ER -> Automate à états fini non-déterministe (AFN)
  - AFN -> AFD (déterminisation)
  - Minimisation de l'AFD

20

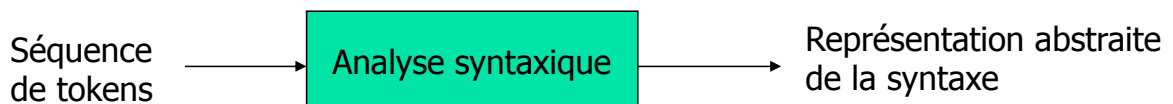
# Conclusion sur l'analyse lexicale



- Reconnaissance automatisée d'un mot d'un langage source (un texte) sous forme de séquence de tokens (d'unités lexicales)

21

## Après l'analyse lexicale : l'analyse syntaxique



- Reconnaissance de la structure syntaxique d'un programme/langage (déclarations, expressions, instructions, parenthésage ...)  
→ notion de **grammaire**
- Détection et traitement des erreurs syntaxiques

22



## 2. Grammaires algébriques

---

23



## Grammaires algébriques : plan

---

- 2.1. Grammaire algébrique et arbres de dérivation
- 2.2. Propriétés des grammaires algébriques
  - Réduite, récursive, ambiguë, propre
- 2.3. Reconnaisseurs associés
  - Automates à pile
- 2.4. Classification des grammaires (Chomsky)

24



## 2.1. Grammaire algébrique

**Définition :** une *grammaire algébrique* (ou *context-free*) est un quadruplet  $G = \langle T, N, P, S \rangle$  où :

- $T$  est l'ensemble des symboles *terminaux*,
- $N$  est l'ensemble des symboles *non-terminaux*,
- $S \in N$  est l'*axiome* (symbole initial),
- $P \subseteq N \times (N \cup T)^*$  est l'ensemble des *règles de production*.

**Définition :** une règle, ou encore une production, de la forme  $X \rightarrow \omega$  avec  $X \in N$  et  $\omega \in (N \cup T)^*$  permet de réécrire  $X$  en  $\omega$ .

$X$  est appelé *partie gauche* et  $\omega$  *partie droite* de la production.

25



## Exemple

$G = \langle \{a, b, c, d, e\}, \{S, T\}, P, S \rangle$

avec  $P = \{S \rightarrow aSbT \mid c$

$T \rightarrow dT \mid e\}$

On peut **dériver** à partir de l'axiome :

$S \rightarrow aSbT \rightarrow aaSbTbT \rightarrow aaSbTbdT$

$\rightarrow aaSbebdT \rightarrow \text{etc.}$

jusqu'à n'avoir que des terminaux

26



## Dérivation

**Définition :**  $m \rightarrow m'$  est une *dérivation simple* si

- $m$  contient un non-terminal  $X$
- La grammaire contient une règle  $X \rightarrow x_1 \dots x_n$
- $m'$  est obtenu en remplaçant dans  $m$  une occurrence de  $X$  par  $x_1 \dots x_n$

On parle de dérivation gauche (resp. droite) lorsqu'il y a remplacement du non-terminal le plus à gauche (resp. droite).

27



## Langage algébrique

Lorsqu'il y a 0 ou plusieurs dérivations simples à suivre, on parle d'une *dérivation* et on la note :

$$A \xrightarrow{*} \omega$$

La notation :  $A \xrightarrow{+} \omega$  indique que la dérivation admet au moins une dérivation simple.

**Définition :** le langage  $L(G)$  généré par une grammaire algébrique  $G$  est défini par :

$$\{w \in T^* \mid S \xrightarrow{+} w\}$$

28

# Arbre de dérivation

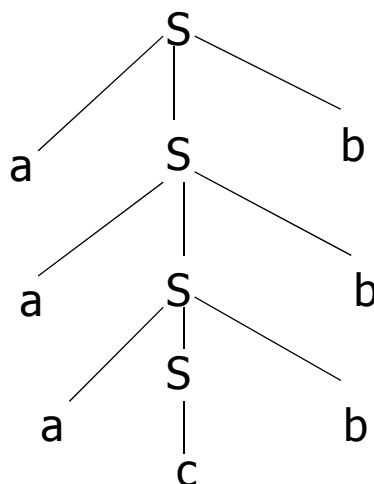
**Définition :** un *arbre de dérivation* (ou *arbre syntaxique*) pour un mot  $w$  de  $L(G)$  selon une grammaire  $G$  est un arbre ordonné tel que :

- Sa racine est l'axiome de  $G$  ;
- Ses feuilles sont des terminaux de  $T$  ou  $\varepsilon$  ;
- La séquence ordonnée de toutes les feuilles est  $w$  ;
- Ses nœuds internes sont des non-terminaux de  $N$  ;
- Les fils d'un nœud  $X$  sont  $Y_1, Y_2, \dots, Y_k$  si et seulement si  $Y_i$  est dans  $(T \cup N)$  et  $X \rightarrow Y_1 Y_2 \dots Y_k$  est une règle de  $G$

29

## Exemple

- $G = \langle \{a,b,c\}, \{S\}, \{S \rightarrow aSb \mid c\}, S \rangle$
- $L(G) = \{c, acb, aacbb, \dots, a^n c b^n, \dots\}$
- Arbre de dérivation de  $aaacbbb$  :



30

## 2.2. Propriétés des grammaires algébriques

- Grammaire réduite
  - Non-terminaux accessibles
  - Non-terminaux productifs
- Récursivité à gauche
- Récursivité à gauche immédiate
- Factorisée à gauche
- Ambiguïté
- Grammaire propre

31

## Grammaire réduite

**Définition :** un non-terminal  $A$  est *inaccessible* s'il n'existe pas de mots quelconques  $\beta, \delta$  tels que :  $S \rightarrow^* \beta A \delta$

**Définition :** *Accessible* = le contraire de *inaccessible*

**Définition :** un non-terminal  $A$  est *improductif* s'il n'existe pas de mot  $w$  sur  $T$  tel que :

$$A \rightarrow^* w$$

**Définition :** *Productif* = le contraire de *improductif*.

**Définition :** une grammaire algébrique  $G$  est *réduite* si elle ne contient que des non-terminaux accessibles et productifs.

32





## Exemple

---

$S \rightarrow aXb \mid cY$

$X \rightarrow aX \mid d$

$Y \rightarrow dY \mid eY$

$Z \rightarrow aS \mid f$

33



## Exemple

---

$S \rightarrow aXb \mid cY$

$X \rightarrow aX \mid d$

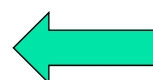
$Y \rightarrow dY \mid eY$

$Z \rightarrow aS \mid f$

- Y est improductif
- Z est inaccessible

$S \rightarrow aXb$

$X \rightarrow aX \mid d$



Grammaire réduite

34



# Réversivité à gauche

**Définition :** une grammaire est *réversivité à gauche* si elle contient au moins un non-terminal  $A$  tel qu'il existe un mot  $\beta$  quelconque et une dérivation  $A \xrightarrow{+} A\beta$ .

**Définition :** une grammaire est plus précisément *immédiatement réversivité à gauche* si elle contient au moins un non-terminal  $A$  tel qu'il existe une production  $A \rightarrow A\beta$  où  $\beta$  est un mot quelconque.

35



## Exemple

$S \rightarrow A \mid B$

$A \rightarrow Axuv \mid C$

$B \rightarrow Cyz$

$C \rightarrow Bc \mid d$

- Réversivité immédiate à gauche à cause de la production  $A \rightarrow Axuv$
- Réversivité (non immédiate) à gauche car il existe une dérivation  $B \rightarrow Cyz \rightarrow Bcyz$

36



## Factorisée à gauche

- **Définition** : Une grammaire est dite **factorisée à gauche**, si les parties droites de deux règles distinctes quelconques, ayant la même partie gauche, n'ont pas de préfixe commun propre.

**Autrement dit** : Une grammaire **factorisée à gauche** n'a pas deux règles de la forme :

$$A \rightarrow \alpha\beta_1 \text{ et } A \rightarrow \alpha\beta_2 \text{ avec } \alpha \neq \varepsilon.$$

37



## Grammaire ambiguë

**Définition** : une grammaire  $G$  est *ambiguë* si  $L(G)$  contient au moins un mot ayant plusieurs arbres de dérivation possibles.

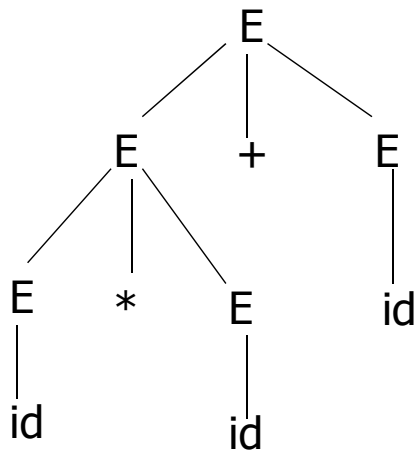
**Exemple** :

$$E \rightarrow E + E \mid E * E \mid (E) \mid id$$

38

## Exemple

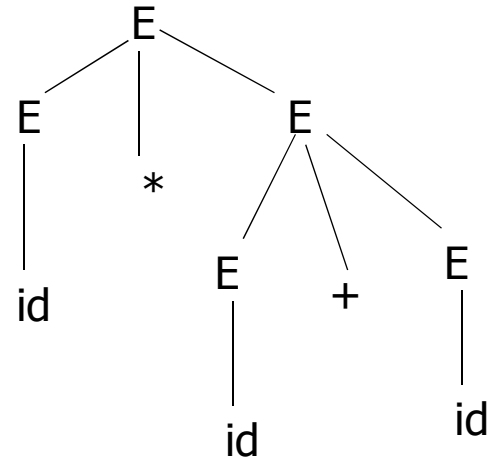
$E \rightarrow E + E \mid E * E \mid (E) \mid id$



Calcul de  $[id * id] + id$

ambiguïté

Deux arbres de dérivation pour le mot :  $id * id + id$



Calcul de  $id * [id + id]$

39

## Grammaire propre

**Définition :** une grammaire est dite *propre* si

- elle ne contient aucune production de la forme  $A \rightarrow \varepsilon$
- et elle ne contient aucune production de la forme  $A \rightarrow B$  (où  $A$  et  $B$  sont des non-terminaux)

## 2.3. Reconnaisseurs pour les grammaires algébriques

**Théorème :** soit une grammaire algébrique  $G$ , alors il existe *un automate à pile* reconnaissant  $L(G)$ .

**En pratique :** les étapes de la reconnaissance sont caractérisées par :

- La position de la tête de lecture sur le mot en entrée
- L'état de l'automate
- L'état de la pile associée

41

## Automate à pile (AP)

**Définition :** Un *automate à pile*  $A = \langle \Sigma, \Gamma, \$, E, e_0, F, \Delta \rangle$  est défini par :

- un alphabet fini  $\Sigma$  des *symboles d'entrée*,
- un alphabet fini  $\Gamma$  des *symboles de pile*,
- un symbole de fond de pile  $\$ \in \Gamma$ ,
- un ensemble  $E$  fini d'*états*,
- un *état initial*  $e_0 \in E$ ,
- un ensemble fini  $F$  d'*états finaux* (ou *états terminaux*),
- une *relation de transition*  $\Delta$  qui à tout triplet formé d'un état, du contenu de la pile et d'un symbole de  $\Sigma \cup \{\$\}$  fait correspondre un ensemble (éventuellement vide) de couples (état, contenu de la pile) :

$$\Delta(e_i, w, a) = \{(e_{i1}, w_1), \dots, (e_{in}, w_n)\}$$

42

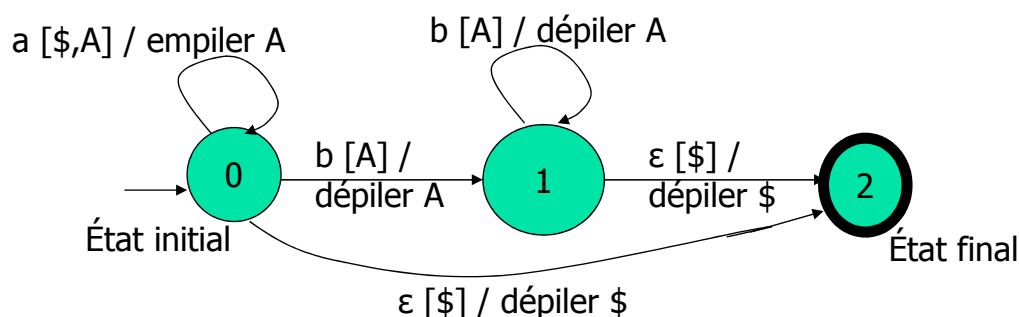
# Automate à pile (AP)

**Définition :** Le *langage*  $L(A)$  reconnu par un *automate à pile*  $A$  est l'ensemble des chaînes qui, en partant d'une pile initialisée avec le symbole de fond de pile,

- permettent de passer de l'état initial à un état final
- et terminent avec une pile vide.

43

## Exemple : automate à pile reconnaissant le langage $\{a^n b^n, n \geq 0\}$



$A1 = \langle \{a,b\}, \{\$,A\}, \$, \{0,1,2\}, 0, \{2\}, \Delta \rangle$  avec :

$\Delta(0, \$, a) = \{(0, \$A)\}$

$\Delta(1, A, b) = \{(1, \epsilon)\}$

$\Delta(0, A, a) = \{(0, AA)\}$

$\Delta(1, \$, \$) = \{(2, \epsilon)\}$

$\Delta(0, A, b) = \{(1, \epsilon)\}$

$\Delta(0, \$, \$) = \{(2, \epsilon)\}$

44

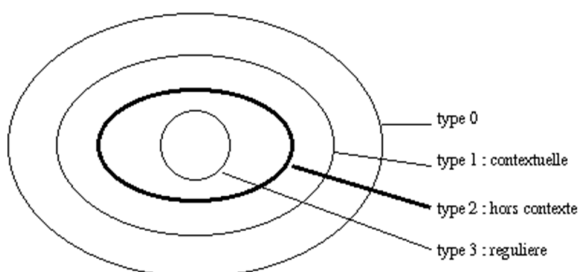
## A-t-on aaabbb dans $L(A1)$ ?

Pile	Entrée	Etat	Transition exécutée
\$	aaabbb\$	0	$(0, \$, a) \rightarrow (0, \$A)$
A\$	aabbb\$	0	$(0, A, a) \rightarrow (0, AA)$
AA\$	abbb\$	0	$(0, A, a) \rightarrow (0, AA)$
AAA\$	bbb\$	0	$(0, A, b) \rightarrow (1, \epsilon)$
AA\$	bb\$	1	$(1, A, b) \rightarrow (1, \epsilon)$
A\$	b\$	1	$(1, A, b) \rightarrow (1, \epsilon)$
\$	\$	1	$(1, \$, \$) \rightarrow (2, \epsilon)$
		2	Mot accepté !

45

## 2.4. Hiérarchie de Chomsky

4 types de grammaires / productions :



*Grammaire de type 1 (contextuelle) :*

$\alpha \rightarrow \beta$   
avec  $\alpha \in (N \cup T)^+$  et  $\beta \in (N \cup T)^*$   
et  $|\alpha| \leq |\beta|$

*Grammaire de type 3 (régulière) :*

$X \rightarrow wY$  ou  $X \rightarrow w$   
avec  $X, Y \in N$  et  $w \in T^*$

*Grammaire de type 2  
(hors contexte, algébrique) :*

$X \rightarrow \omega$   
avec  $X \in N$  et  $\omega \in (N \cup T)^*$

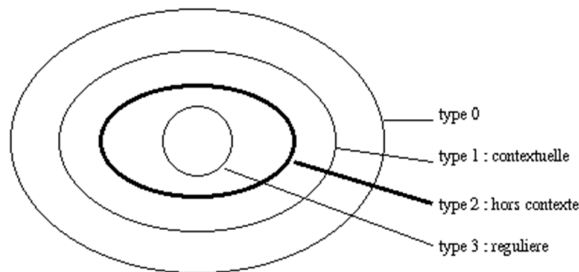
*Grammaire de type 0 (générale) :*

$\alpha \rightarrow \beta$   
avec  $\alpha \in (N \cup T)^+$  et  $\beta \in (N \cup T)^*$

46

## 2.4. Hiérarchie de Chomsky

4 types de langages :



*Définition* : Un langage  $L$  est dit de type  $x$  s'il existe une grammaire  $G$  de type  $x$  telle que  $L(G)=L$ .

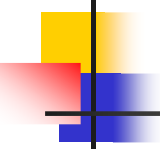
47

### Exemple de grammaire pour $(a|b)^*ab^+(bb|ab)^+$

- Une grammaire possible :  
 $S \rightarrow AaBC$   
 $A \rightarrow aA \mid bA \mid \varepsilon$   
 $B \rightarrow bB \mid b$   
 $C \rightarrow bbC \mid abC \mid bb \mid ab$
- Cette grammaire n'est pas régulière

48





## Exemple de grammaire régulière pour $(a|b)^*ab^+(bb|ab)^+$

- Une grammaire **régulière** équivalente :  
     $S \rightarrow aS \mid bS \mid aT$   
     $T \rightarrow bT \mid bU$   
     $U \rightarrow bbU \mid abU \mid bb \mid ab$
- Donc le langage est régulier

49



## Exemple de grammaire de type 0 : $\{ucu, u \text{ sur } \{a,b\}\}$

- Mots : c, aca, bcb, ..., **abbbabcabbbab**,  
etc.
- Il n'existe pas de grammaire **algébrique**  
généralisant ce langage

50

## Exemple de grammaire de type 0

:  $\{ucu, u \text{ sur } \{a,b\}\}$

- Mots : c, aca, bcb, ..., **abbbabcabbbab**, etc.

- Il n'existe pas de grammaire **algébrique** générant ce langage

La plus proche :  $S \rightarrow aSa \mid bSb \mid c$

génère  $\{ucv, v \text{ est l'image miroir de } u\}$

51

## Exemple de grammaire de type 0

:  $\{ucu, u \text{ sur } \{a,b\}\}$

- On part de :  $S \rightarrow aSa \mid bSb \mid c$
- On oblige à continuer :  $S \rightarrow aSA \mid bSB \mid c$
- On prépare le A/B le plus à gauche à se déplacer :  
 $cA \rightarrow cA'$  et  $cB \rightarrow cB'$
- On déplace les A'/B' vers la droite :  
 $A'A \rightarrow AA'$   $A'B \rightarrow BA'$   $B'A \rightarrow AB'$   $B'B \rightarrow BB'$
- Besoin d'un marqueur de fin : axiome  $S' \rightarrow SF$
- Fin quand les A'/B' sont complètement à droite :  
 $A'F \rightarrow Fa$   $B'F \rightarrow Fb$
- Fin quand F touche le c :  $cF \rightarrow c$

52

# Exemple de grammaire de type 0

## : $\{ucu, u \text{ sur } \{a,b\}\}$

- Conclusion :

$S' \rightarrow SF$

$S \rightarrow aSA \mid bSB \mid c$

$cA \rightarrow cA' \quad cB \rightarrow cB'$

$A'A \rightarrow AA' \quad A'B \rightarrow BA' \quad B'A \rightarrow AB' \quad B'B \rightarrow BB'$

$A'F \rightarrow Fa \quad B'F \rightarrow Fb \quad cF \rightarrow c$

- Exemple : générer le mot abbcabb ?

$S' \rightarrow SF \rightarrow aSAF \rightarrow abSBAF \rightarrow abbSBBAF$

$\rightarrow abbcBBAF \rightarrow abbcB'BAF \rightarrow abbcBB'AF \rightarrow abbcBAB'F$

$\rightarrow abbcBAFb \rightarrow abbcB'AFb \rightarrow abbcAB'Fb \rightarrow abbcAFbb$

$\rightarrow abbcA'Fbb \rightarrow abbcFabb \rightarrow abbcabb$