



Théorie des langages

Présenté par Yann Caron
skyguide

ENSG Géomatique



Plan du cours

Grammaire formelle

Hierarchie de Chomsky

Automates



Grammaire formelle



Introduction

- ✓ Un langage de programmation est issue de la théorie des **langages formels**
- ✓ Discipline entre les mathématiques et la linguistique
- ✓ La syntaxe est la représentation (la forme)
- ✓ La sémantique en est le sens
- ✓ On distinguera donc la syntaxe du langage (C# vs Java) et leurs paradigmes (impératif, structuré, fonctionnel, objet, concurrentiel)

Grammaire formelle

- ✓ Définition :
- ✓ Une grammaire formelle est constituée d'un ensemble fini de règles de production de forme $A \rightarrow B$
- ✓ Où A et B sont constitués d'un ensemble fini de symboles
 - ✓ Des symboles non terminaux (nœuds)
 - ✓ Des symboles terminaux (feuilles)
 - ✓ Un symbole de départ (root)

Conventions

- ✓ Les symboles non terminaux sont en majuscule
- ✓ Les symboles terminaux en minuscule
- ✓ Epsilon « ϵ » représente une chaîne vide
- ✓ Exemple de grammaire :

$$S \rightarrow AB$$

$$S \rightarrow \epsilon$$

$$A \rightarrow aS$$

$$B \rightarrow b$$

Questions ?

- ✓ Pourquoi plusieurs règles S
- ✓ Peut-on réduire ?
- ✓ Quel est le symbole de départ ?
- ✓ Quel mots peut générer cette grammaire ?
- ✓ Et de quelle forme en général ?

Réponses

- ✓ Définir plusieurs fois la même règle revient à appliquer une opération "ou" : $S \rightarrow AB \mid \varepsilon$
- ✓ L'expression peut être réduite sous la forme :
$$S \rightarrow aSb$$
$$S \rightarrow \varepsilon$$
- ✓ Les symboles de départ est S
- ✓ Peut générer : rien, ab, aabb, aaabbbb ...
- ✓ Les mots de la forme **$a^n b^n$**
(n lettre(s) 'a' suivit de n lettre(s) 'b' ou rien)

Example

- ✓ Éléments terminaux :
{ generate, hate, great, green, ideas, linguists }
- ✓ Éléments non terminaux :
{ SENTENCE, NOUNPHRASE, VERBPHRASE, NOUN, VERB, ADJ }

Example

- ✓ Règles de production

SENTENCE → NOUNPHRASE VERBPHRASE

NOUNPHRASE → ADJ NOUNPHRASE

NOUNPHRASE → NOUN

VERBPHRASE → VERB NOUNPHRASE

VERBPHRASE → VERB

NOUN → ideas

NOUN → linguists

VERB → generate

VERB → hate

ADJ → great

ADJ → green

Exemple

- ✓ Peut générer des phrases telles que :
 - ✓ “ideas hate great linguists”
 - ✓ “ideas generate”
- ✓ Mais ne peut pas générer des phrases telles que :
 - ✓ “ideas ideas great hate”

Hierarchie de Chomsky



Chomsky

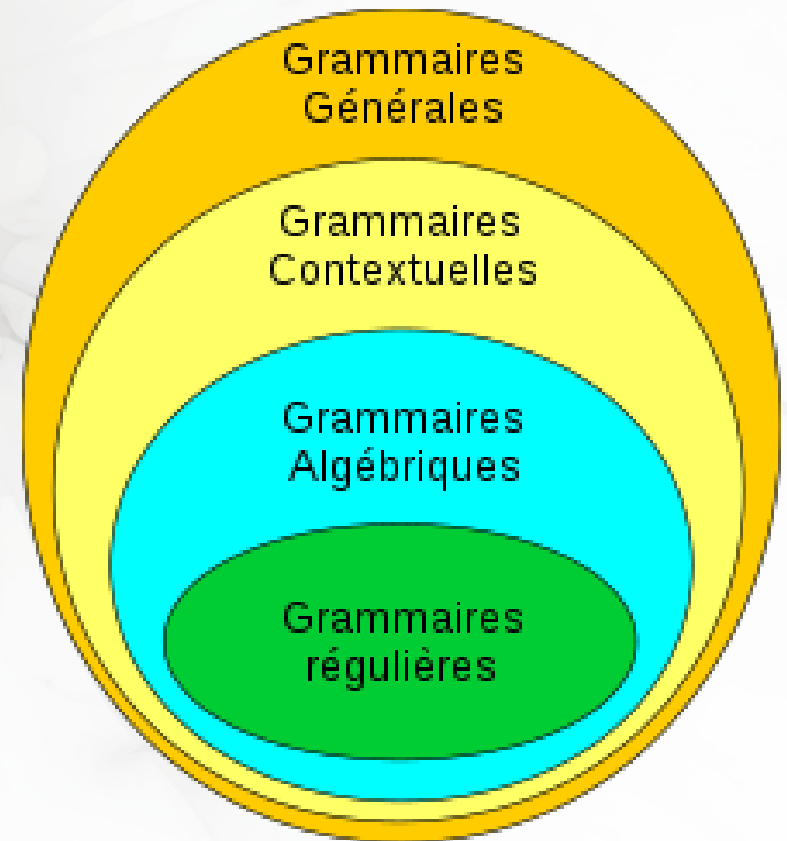
- ✓ Classification inventée par Noam Chomsky
- ✓ Né en 1928 à Philadelphie
- ✓ Linguiste théoricien
- ✓ Fondateur de la **grammaire générative et transformationnelle** dans l'étude des langages naturels
- ✓ Qualifié de "révolution Chomskienne"
- ✓ Anarchiste / Militant contre la guerre du Vietnam



Noam Chomsky
1928

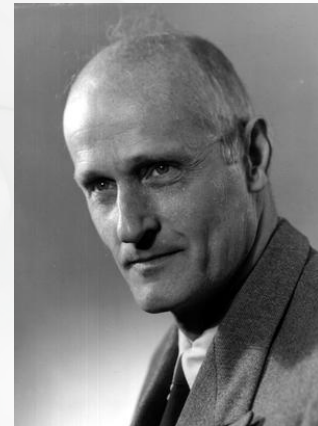
Hiérarchie de Chomsky

- ✓ Quatre classes de grammaires et de langages formels
- ✓ Tel que :
- ✓ $L_3 \subset L_2 \subset L_1 \subset L_0 \subset U$
- ✓ Ou U est l'univers des langages



Conventions mathématiques

- ✓ N est l'ensemble des éléments non terminaux de la grammaire
- ✓ T est l'ensemble des éléments terminaux de la grammaire
- ✓ $V \in (N \cup T)$
- ✓ ϵ est le mot vide
- ✓ $*$ est l'étoile de Kleene
tel que $V^* = V^n : n \geq 0$



Stephen Cole Kleene
1978 - 1994

Grammaire régulière L3

- ✓ Règle grammaire linéaire à gauche :
 $A \rightarrow Ba, A \rightarrow a \quad (A, B \in T : a \in T)$
- ✓ Règle grammaire linéaire à droite :
 $A \rightarrow aB, A \rightarrow a \quad (A, B \in T : a \in T)$
- ✓ Engendre des langages rationnels ou réguliers
- ✓ Fonctionne avec des **automates finis**
- ✓ Exemples :
 $a, b, a^*b^*, (aaab)^*$

Grammaire régulière L3

- ✓ Les règles à gauche et à droite ne peuvent pas être autorisées simultanément :

$$A \rightarrow aBb, A \rightarrow a \quad (A, B \in N : a, b \in T \cup \varepsilon)$$

- ✓ Sinon on obtient une grammaire linéaire qui est intermédiaire entre L3 et L2

Grammaire non contextuelle

L2

- ✓ Les règles sont de la forme :

$$A \rightarrow \gamma \quad (A \in N : \gamma \in V^*)$$

- ✓ Engendre des langages algébriques
- ✓ Fonctionne avec des **automates à pile**

Grammaire non contextuelle

L2

- ✓ Sont des langages algébriques
- ✓ $\{ a^n b^n : n \geq 0 \}$
- ✓ Les langages de Dyck
langages bien parenthésés comme :
 $((\))$ ou $([\]\{\}\([\]))$
couples de parenthèses (ouvrante
fermante)
- ✓ Les palindromes
- ✓ Plus généralement, il y a des règles qui
régissent b en fonction de a

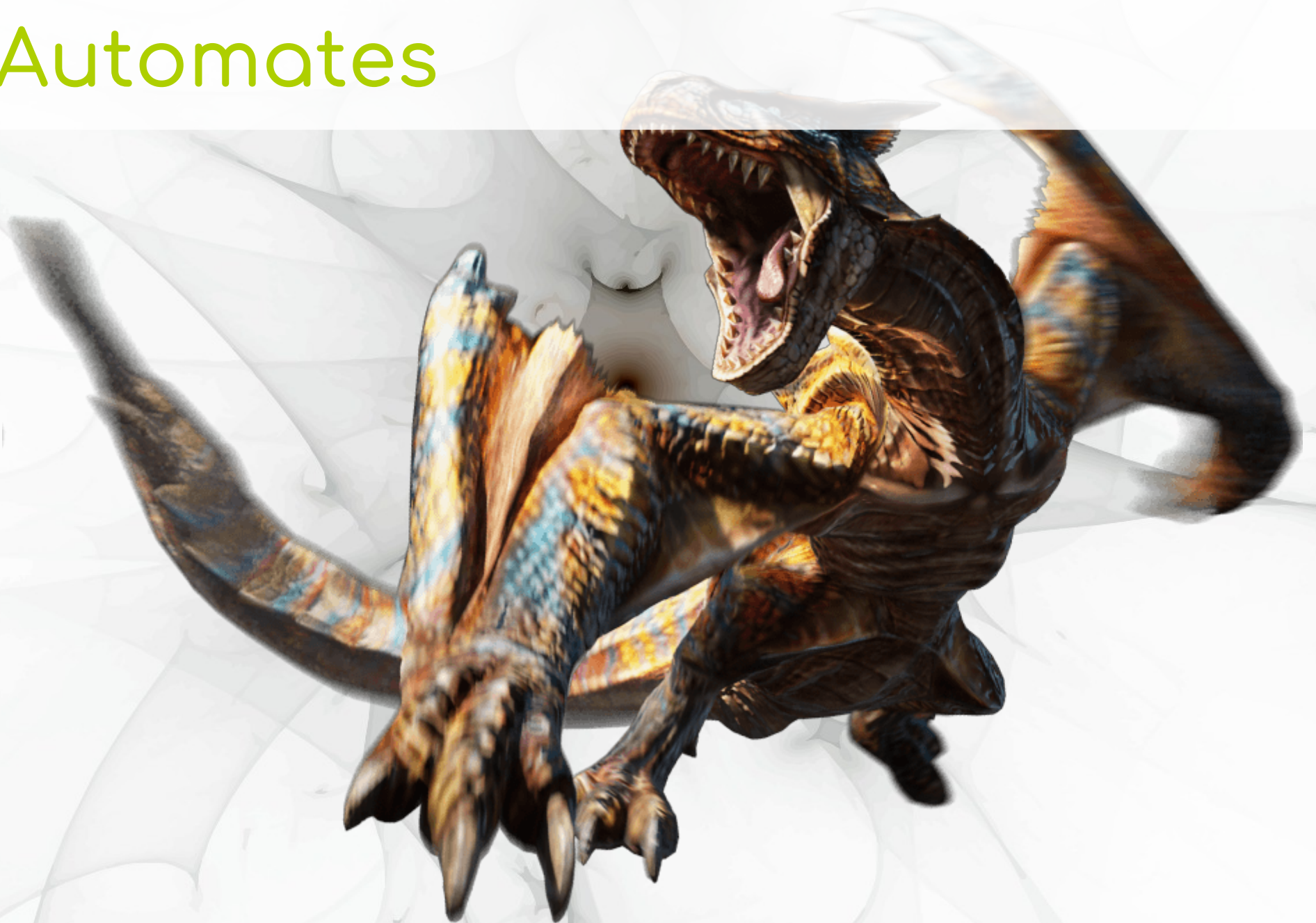
Grammaire contextuelle L1

- ✓ Les règles sont de la forme :
 $\alpha A \beta \rightarrow \alpha \gamma \beta \quad (A \in N : \alpha, \beta, \gamma \in V^* : \gamma \neq \varepsilon)$
- ✓ Toute règle comprenant un **non terminal** entouré de deux mots (son contexte)
- ✓ Engendre des langages contextuels
- ✓ Fonctionne avec des **automates linéaires bornés** (machine de Turing non déterministe à mémoire linéaire)
- ✓ Exemples : $(a^n b^n c^n : n > 0)$
- ✓ $(a^n b^m c^n d^m : n > 0 : m > 0)$

Grammaire générale L0

- ✓ Les règles sont de la forme :
 $\alpha \rightarrow \beta \quad (\alpha \in V^*NV^* : \beta \in V^*)$
- ✓ Aucunes restrictions
- ✓ Engendre des langages récursivement énumérables
- ✓ Fonctionnent sur des machines de Turing
- ✓ Problème de l'appartenance d'un mot à un langage de cette classe est indécidable

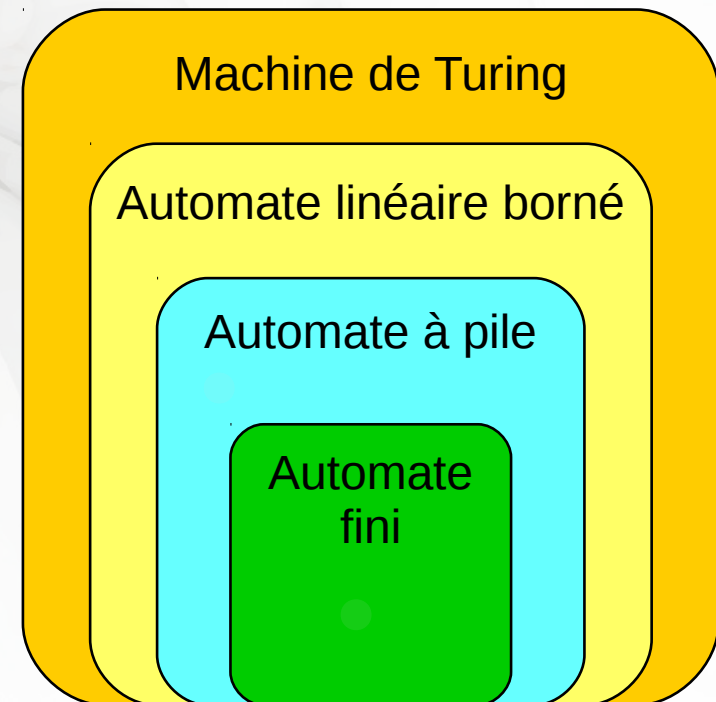
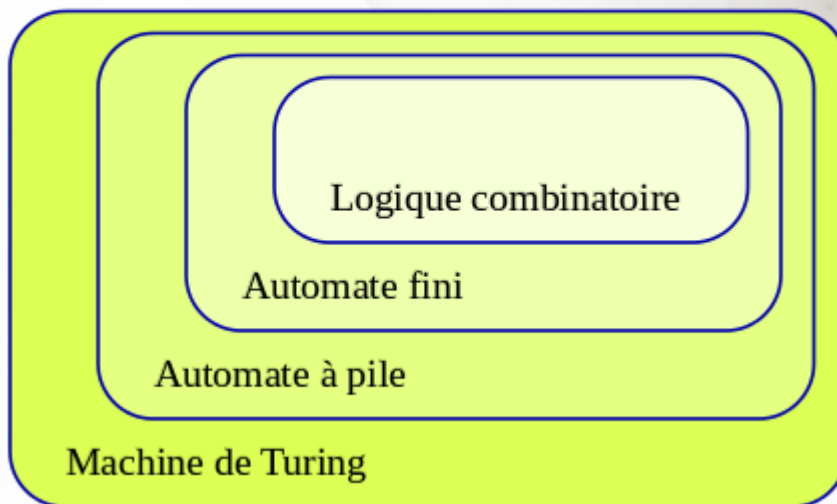
Automates



Automates

- ✓ Modèles mathématiques de calcul
- ✓ Modèle théorique d'ordinateur

Hiérarchie d'automates



Automate fini

- ✓ Possède un nombre fini d'état à un instant donné
- ✓ Les transitions s'effectuent sous une condition donnée
- ✓ Peu résoudre une sous classe de problèmes en regards de la machine de Turing
- ✓ Permet de reconnaître les langages réguliers L3 - **Lexer**

Automate fini

- ✓ Possède un nombre fini d'état à un instant donné
- ✓ Les transitions s'effectuent sous une condition donnée
- ✓ Peu résoudre une sous classe de problèmes en regards de la machine de Turing
- ✓ Permet de résoudre les grammaires formelles (langages réguliers L3)

Automate fini

- ✓ Représenté par un quintuplet $(Q, \Sigma, \delta, q_0, T)$

Où :

- ✓ Q est l'ensemble des états
- ✓ Σ est l'ensemble des symboles de l'alphabet
- ✓ δ est l'ensemble des transitions
- ✓ $q_0 \in Q$ l'état initial
- ✓ $T \subset Q$ l'ensemble des états terminaux

Exemple

- ✓ On veut reconnaître tous les mots qui commencent par **a**, finissent par **b** et dont le milieu est un nombre indéterminé de **a** ou de **b**
- ✓ Soit l'expression suivante : **$a(a, b)^*b$**
- ✓ Soit la grammaire suivante :

$A \rightarrow aB$

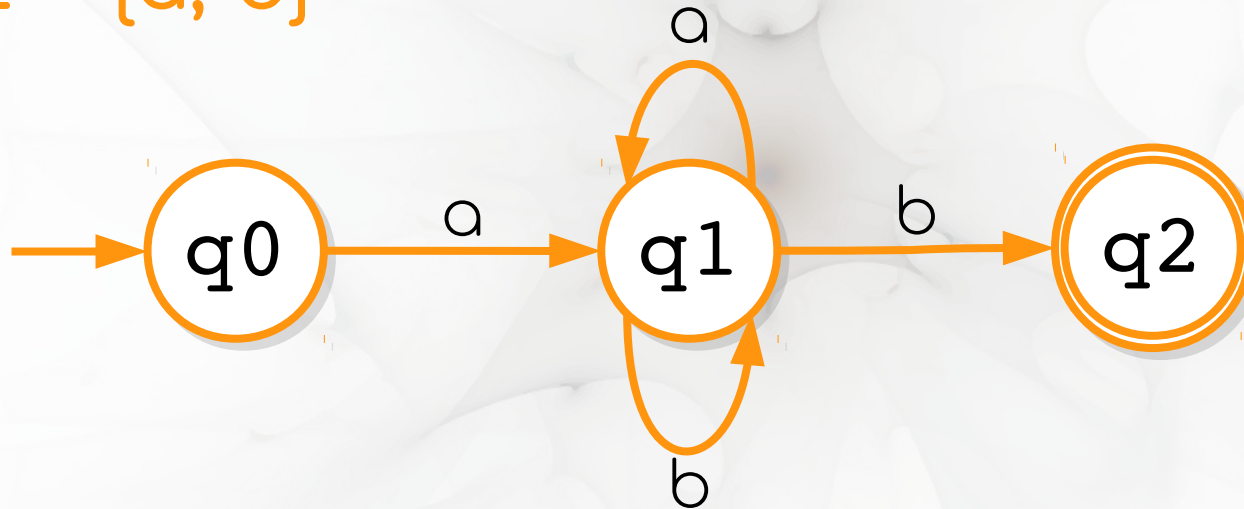
$B \rightarrow aB$

$B \rightarrow bB$

$B \rightarrow b$

Exemple

- ✓ Expression : $\{ a (a, b)^n b : n \geq 0 \}$
- ✓ $\Sigma = \{a, b\}$



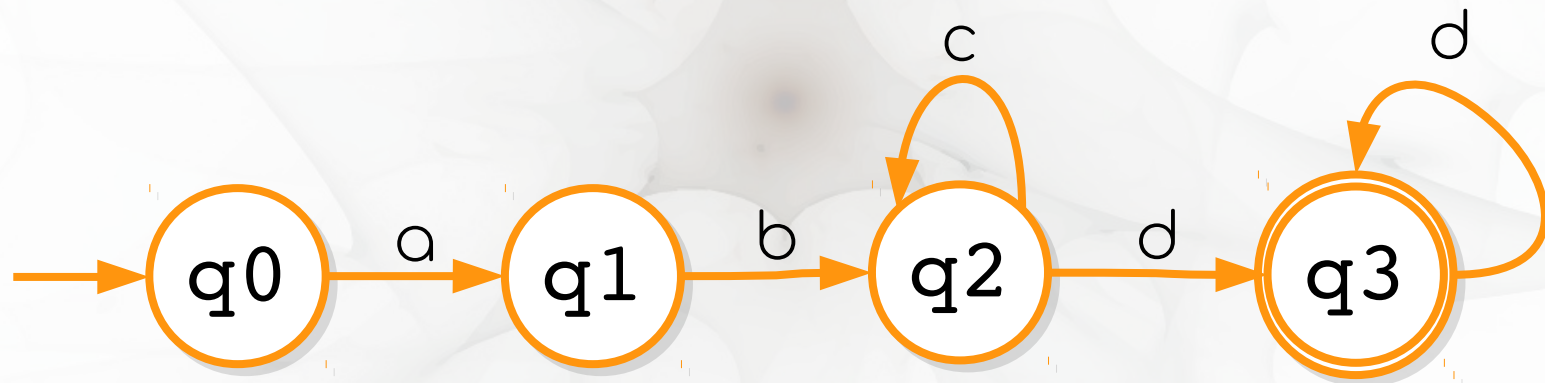
- ✓ Accepte : ab, aabab, abbb, aab
- ✓ N'accepte pas : a, b, abba, baba, bbbb

Exercice

- ✓ Existe-t-il un automate fini pour l'expression : $\{ a^n b^n : n \geq 1 \}$?
- ✓ Définir un automate qui accepte l'expression : $\{ a b c^m d^n : m \geq 0, n \geq 1 \}$
- ✓ Quels mots accepte-t-il ?
- ✓ Quels mots n'accepte-t-il pas ?

Réponse

- ✓ Non il faut un automate à pile !
- ✓ DFA : $\{ a b c^m d^n : m \geq 0, n \geq 1 \}$



- ✓ Accepte : abd, abdd, abcd, abccd, abccdd
- ✓ N'accepte pas : bd, ab, abc, abcc, abdb

Exemple concret

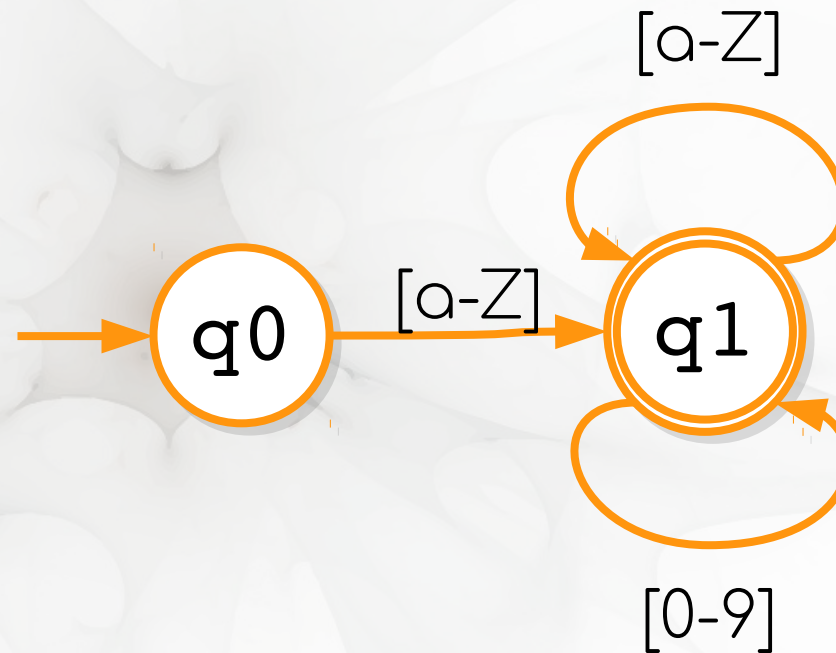
- ✓ Grammaire :

$V \rightarrow A A N^*$

$A N = A \mid N$

$A \rightarrow [a-Z]$

$N \rightarrow [0-9]$



- ✓ Expression :

$[a-Z] ([a-Z] \mid [0-9])^n : n \geq 0$

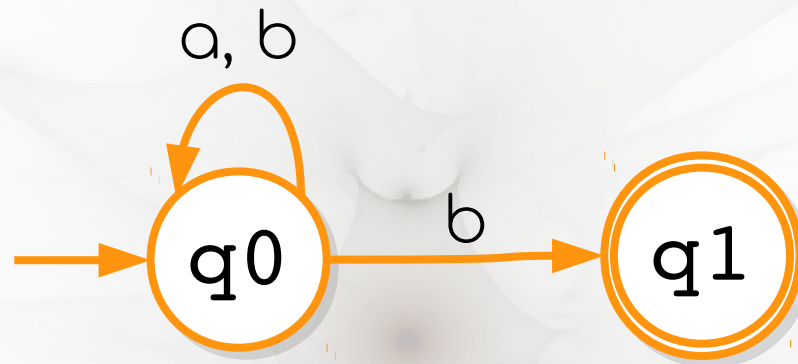
DFA, NFA, NFA- ϵ

- ✓ Un automate fini est déterministe SSI :
 - ✓ Il ne possède qu'un état initial
 - ✓ Pour tout état q et alphabet a , il n'existe qu'une seule transition de q par l'alphabet a
- ✓ Par opposition un automate fini est non-déterministe quand une transition peut mener à des états différents

DFA, NFA, NFA- ϵ

- ✓ Un automate fini est déterministe SSI :
 - ✓ Il ne possède qu'un état initiale q_0
 - ✓ Pour tout état q et alphabet a , il n'existe qu'une seule transition de q vers $q+1$ par mot de a
- ✓ Par opposition un automate fini est non-déterministe quand une transition peut mener à des états différents
- ✓ Le NFA supporte les transition ϵ (mot vide)

DFA, NFA, NFA- ϵ



- ✓ Est non déterministe car la transition b peut mener vers **q0** ou **q1**
- ✓ NFA et DFA supporte les mêmes langages
- ✓ NFA est une généralisation de DFA
- ✓ Une algorithme permet de transformer un NFA en NFA- ϵ puis en DFA

Lemme de la pompe

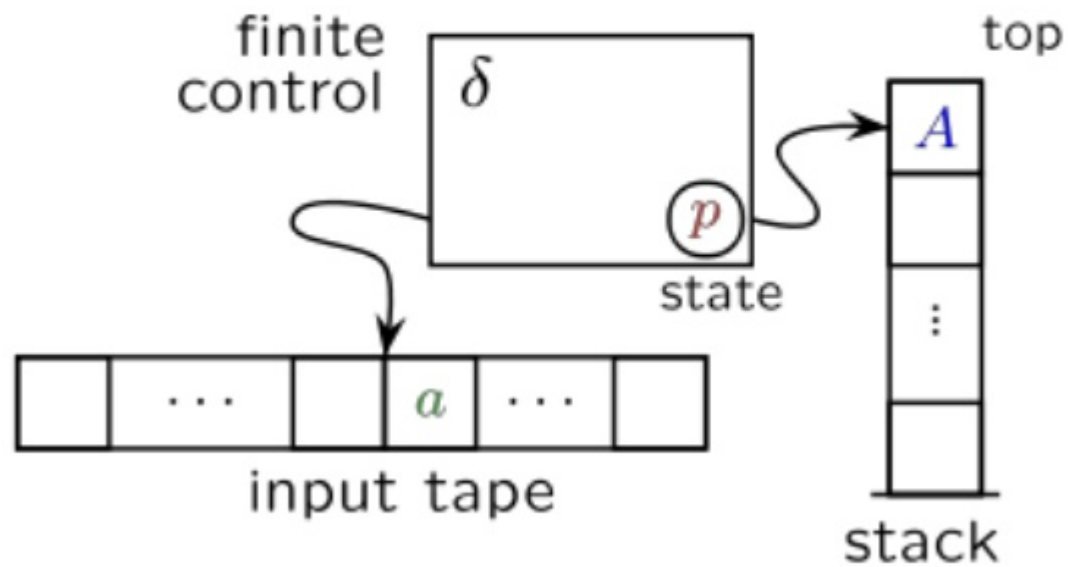
- ✓ Certains langages ne sont pas reconnaissables par un automate fini
 $\{ a^n b^n : n \geq 0 \}$
- ✓ Le Lemme de la pompe permet de le démontrer mathématiquement (par l'absurde) si un langage est rationnel ou non

Automate à pile

- ✓ Généralisation de l'automate fini mais dispose d'une mémoire infinie sous forme d'une pile
- ✓ Les transitions s'effectuent sous une condition donnée en input et selon l'état de la pile (vide, non vide, dernier état = x)
- ✓ Chaque transition peut empiler un état
- ✓ Permet de reconnaître les langages non contextuels L2 - **Parser**

Automate à pile

PDA MODEL



Automate à pile

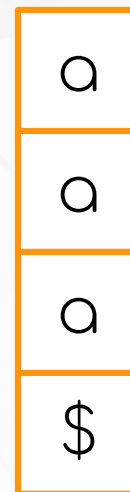
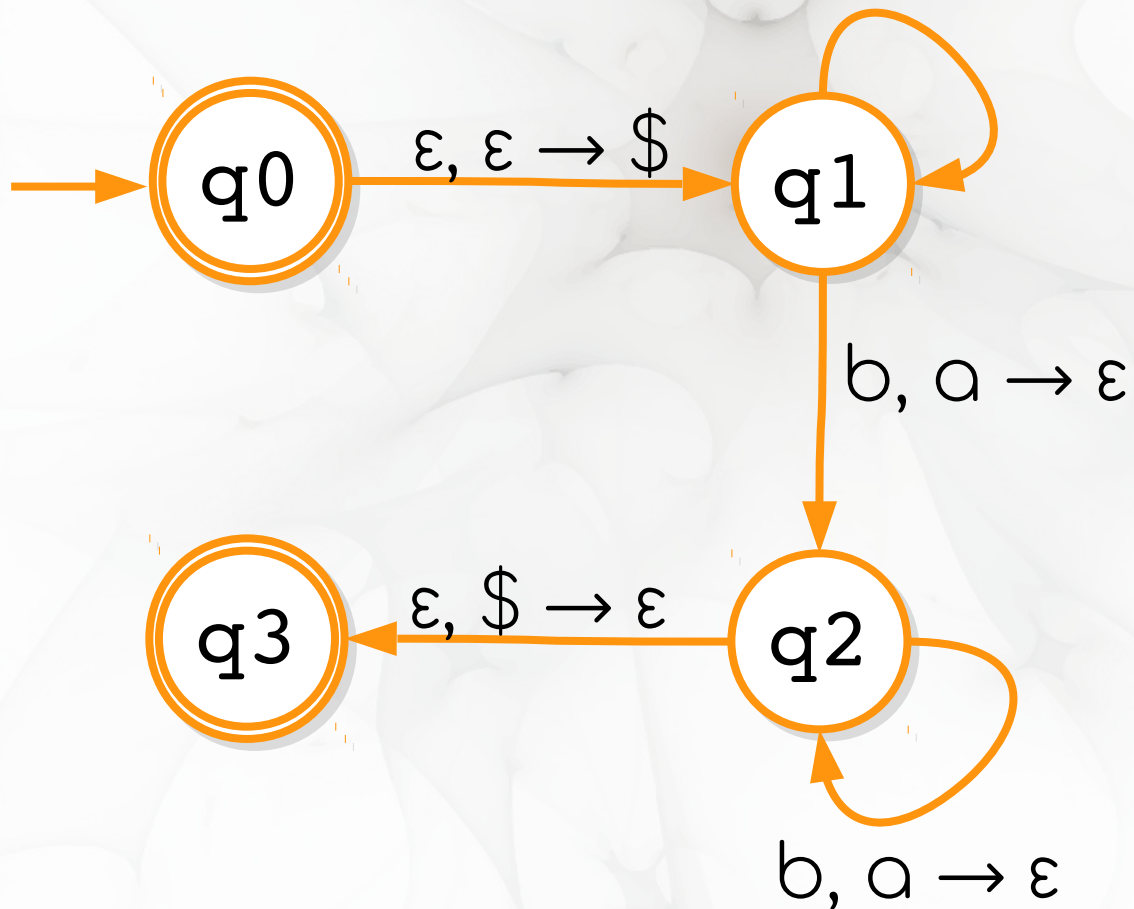
- ✓ Un septuplet $(Q, A, Z, \delta, z_0, q_0, T)$
Où :
 - ✓ Q est l'ensemble des états
 - ✓ A l'alphabet d'entré
 - ✓ Z l'alphabet de pile
 - ✓ δ est l'ensemble des transitions
 - ✓ $z_0 \in Z$ est le symbole de fond de pile
 - ✓ $q_0 \in Q$ l'état initial
 - ✓ $T \subset Q$ l'ensemble des états terminaux

Example

✓ Expression : $\{ a^n b^n : n \geq 0 \}$

✓ aaabbbb

$a, \varepsilon \rightarrow a$



Exemple

- ✓ Accepte :
ab, aabb, aaabbbb
- ✓ N'accepte pas :
a, b, abba, baba

Questions

- ✓ Comment transformer l'automate pour l'expression : $\{ a^n b^n : n \geq 1 \}$?
- ✓ Comment transformer l'automate pour l'expression : $\{ a^n b^n c^n : n \geq 0 \}$?

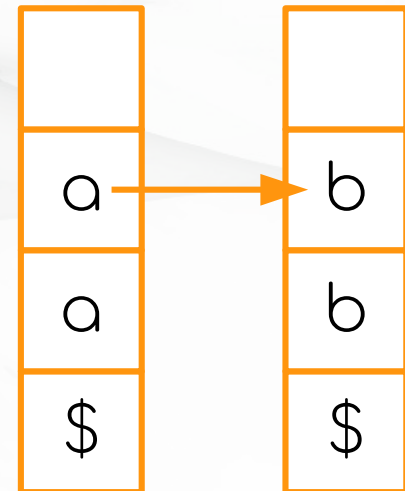
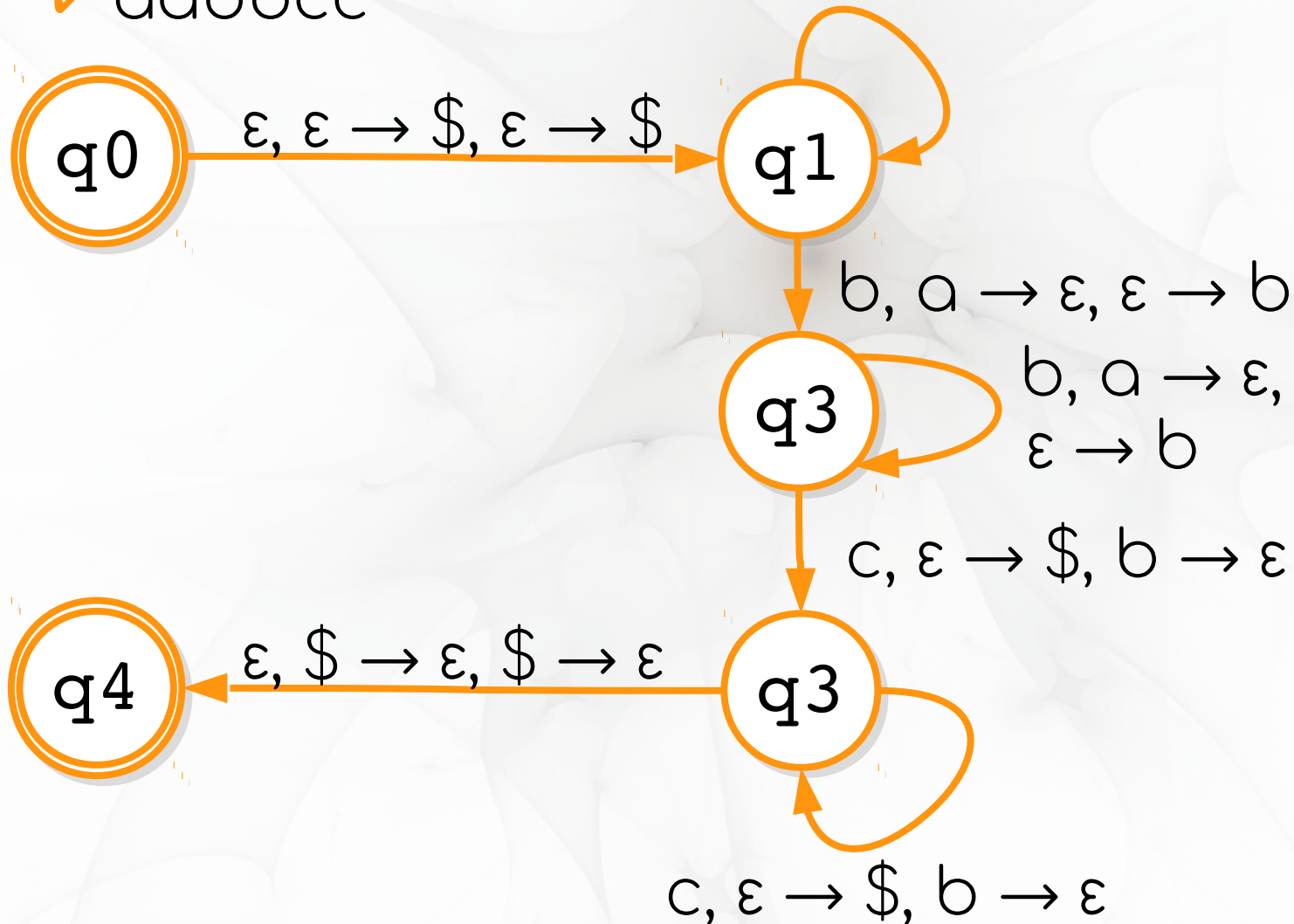
Automate linéaire borné

- ✓ Solution :
- ✓ Comment transformer l'automate pour l'expression : $\{ a^n b^n c^n : n \geq 0 \}$?
- ✓ Impossible avec un automate à pile
- ✓ Il faut utiliser deux piles (automate linéaire borné)
- ✓ Lien YouTube Prof Bill Byrne :
- ✓ <https://www.youtube.com/watch?v=p1peR1Qbp0s>

Automate linéaire borné

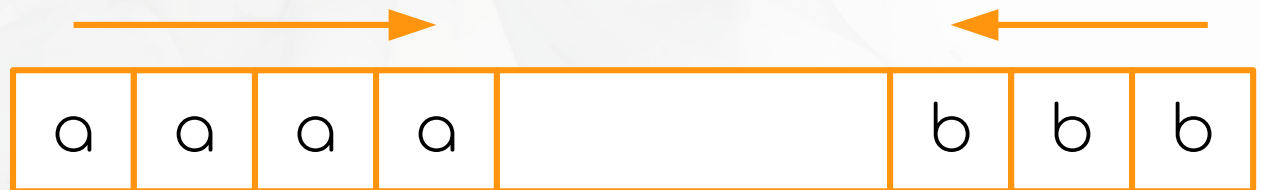
✓ aabbcc

$a, \varepsilon \rightarrow a, \$ \rightarrow \varepsilon$



Automate linéaire borné

- ✓ Deux piles peuvent être contenues dans le même espace
- ✓ Il suffit de les mettre en vis-à-vis
- ✓ Théoriquement, on rejoint le fonctionnement de la mémoire d'un ordinateur (cf fonctionnement d'un programme)



- ✓ Si la mémoire est infinie alors c'est une machine de Turing

Automate linéaire borné

- ✓ Reconnaît des langages de type **Context Sensitive L1**
- ✓ **Langages Rékursifs** (capable de se définir eux même ?)
- ✓ Ne pas confondre avec les langages récursivement énumérables (machine de Turing)
- ✓ Justement, un automate dont la mémoire est infinie est une machine de Turing

Machine de Turing

- ✓ Machine abstraite (mathématiques) qui fonde la science informatique
- ✓ Permet de vérifier les langages récursivement énumérables L_0
- ✓ Première définition systématique des programmes informatique
- ✓ Porte en son fondement que les données et le programme sont de même nature :
- ✓ Le programme est contenu sur une seconde bande infinie

Machine de Turing

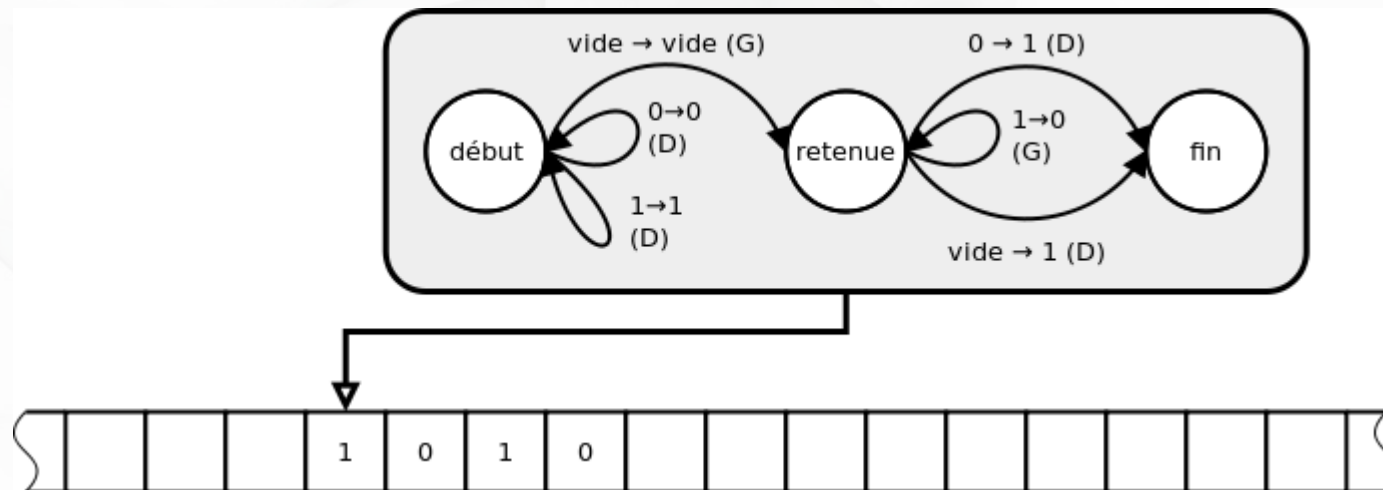
- ✓ Composée de :
 - ✓ Une tête de lecture qui se déplace, lit et écrit sur une bande infinie
 - ✓ Un registre d'état qui mémorise l'état de la machine
 - ✓ Une table d'actions (programme)



Machine de Turing

- ✓ Un septuplet $(Q, A, a, Z, q_0, \delta, T)$ ou :
 - ✓ Q est un ensemble fini d'états
 - ✓ A l'alphabet de travail
 - ✓ $a \in A$ symbole blanc (case vide)
 - ✓ $Z \in A \setminus a$ est l'alphabet d'entrée
 - ✓ $q_0 \in Q$ l'état initial
 - ✓ $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ est la fonction de transition
 - ✓ $T \subset Q$ l'ensemble des états terminaux

Opération + 1



Choisir un programme :
Ajouter 1 ▼

Entrer une valeur binaire :
1010 OK

Lancer le programme :
Pas à pas — Commencer

Etat	Lit	Ecrit	Déplace	Suivant
e1	VIDE	VIDE	gauche	e2
e2	0	0	gauche	e2
	1	1	gauche	e2
	VIDE	VIDE	droite	e3
e3	0	1	droite	fin
	1	0	droite	e3
	VIDE	1	droite	fin

1

0

1

0

Etat de départ : e2

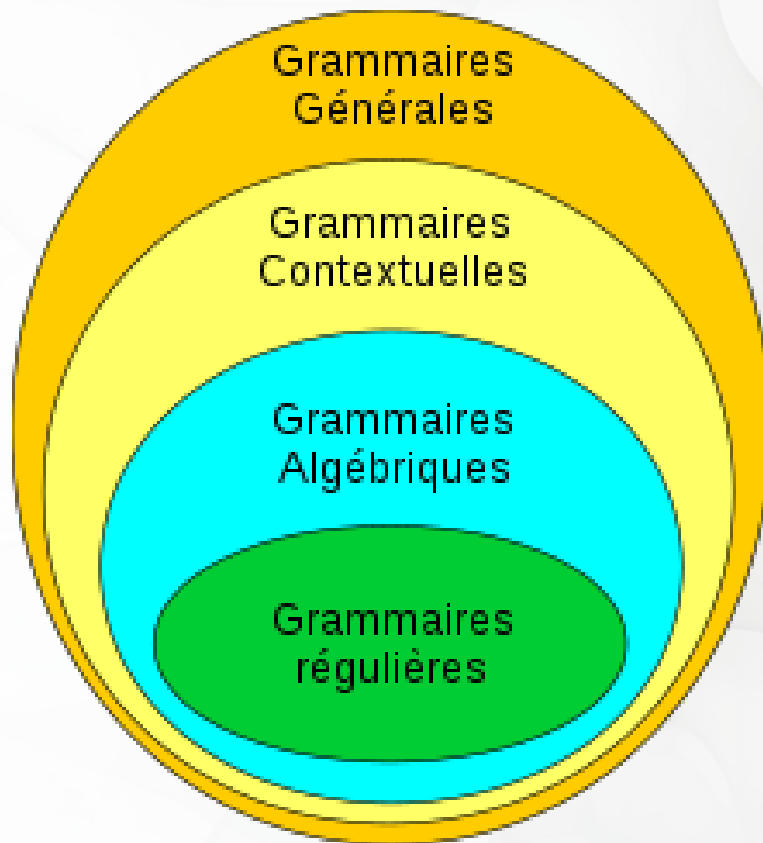
Démonstration

- ✓ Online :
- ✓ https://interstices.info/jcms/nn_72391/comment-fonctionne-une-machine-de-turing

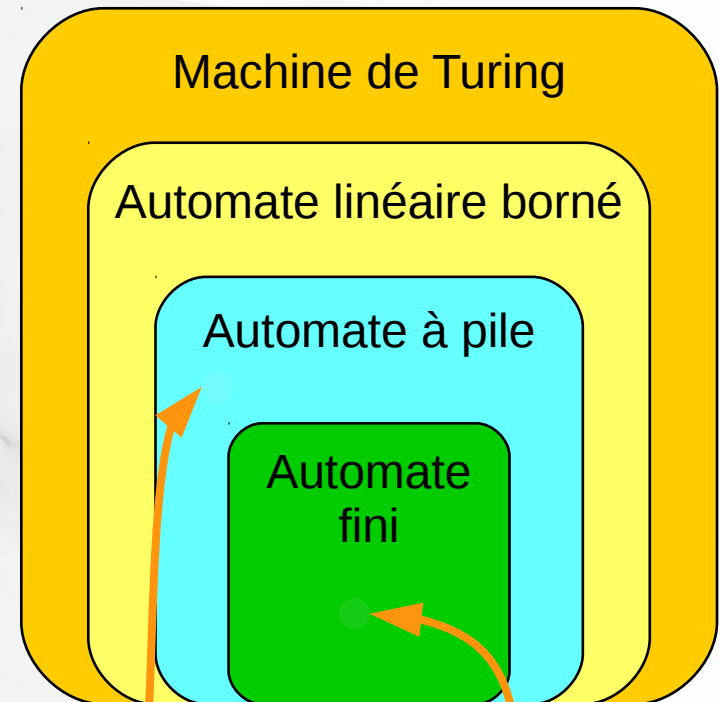
Conclusion

Grammaire	Règles de production	Langage	Machine abstraite
L0	$\alpha \rightarrow \beta$	Récurivement énumérable Sans restriction	Machine de Turing (bande infinie)
L1	$\alpha A \beta \rightarrow \alpha \gamma \beta$	Contextuel	Automate linéairement borné (2 piles) (récursion + 1 pile)
L2	$A \rightarrow \gamma$	Algébrique Non Contextel Context free language	Automate à pile (1 pile) (ou récursion)
L3	$A \rightarrow aB,$ $A \rightarrow a$	Rationnel, Langages réguliers	Automate fini (1 seul état courant)

Conclusion



Analyseur Syntaxique
Grammaire non contextuelle (CFG)



Analyseur Lexical
Grammaire régulière

Merci de votre attention

