skyguide

ENSG Géomatique

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 formelles**
- 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 constitué d'un ensemble fini de règles de production de forme A → 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 :

  $S \rightarrow AB$

  $S \rightarrow \varepsilon$

  $A \rightarrow aS$

  $B \rightarrow b$

Questions ?

- Pourquoi plusieurs règles S
- Peut-on réduire ?
- Quel est les symbole de départ ?
- Quel mots peu générer cette grammaire ?
- Et de quelle forme en générale ?

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 symbole de départ est S
- Peut générer : rien, ab, aabb, aaabbb …
- Les mots de la forme $\mathbf{a^n b^n}$

  (n lettre(s) ‘a’ suivit de n lettre(s) ‘b’ ou rien)

Exemple

- Éléments terminaux :

  { generate, hate, great, green, ideas, linguists }

- Éléments non terminaux :

  { SENTENCE, NOUNPHRASE, VERBPHRASE, NOUN, VERB, ADJ }

Exemple

- 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

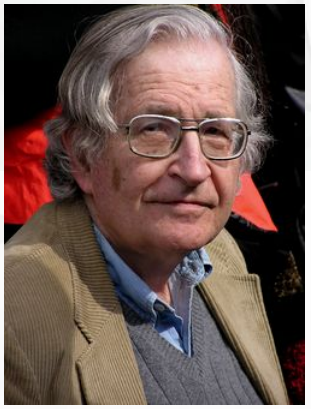
Exemple

- Peut générer des phrases tels que :
  - “ideas hate great linguists”
  - “ideas generate”

- Mais ne peut pas générer des phrases tels que :
  - “ideas ideas great hate”

Hierarchie de Chomsky

Chomsky

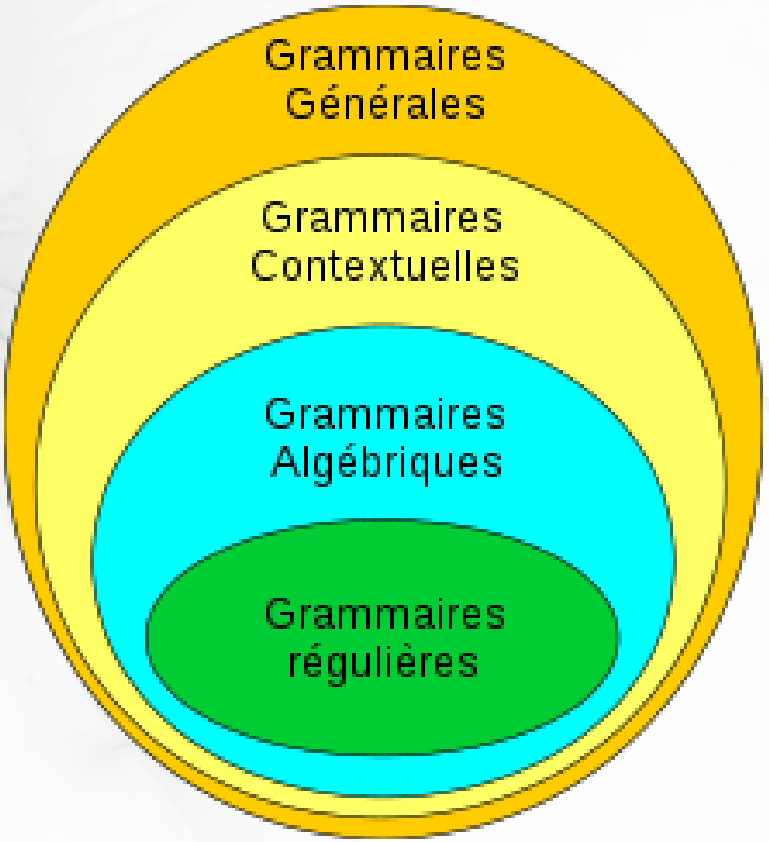


Noam Chomsky
1928

- Classification inventé 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

Hiérarchie de Chomsky

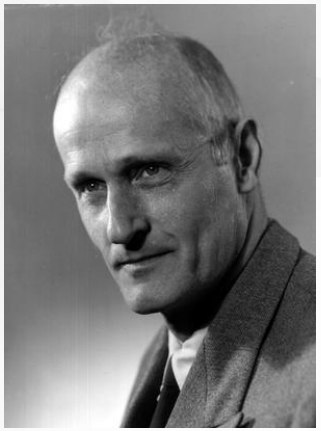
- Quatre classes de grammaires et de langages formels
- Tel que :
- $L3 \subset L2 \subset L1 \subset L0 \subset U$
- Ou U est l'univers des langages



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 gauche :
  $A \rightarrow aB,\ A \rightarrow a \quad (A, B \in T : a \in T)$
- 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 autorisés simultanément :

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

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

Grammaire non contextuelle L2

- Les règles sont de la forme :
  $A \rightarrow \gamma \qquad (A \in N : \gamma \in V^*)$
- 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 :

  $\alpha A \beta \rightarrow \alpha \gamma \beta$ ($A \in N : \alpha, \beta, \gamma \in V^* : \gamma \neq \varepsilon$)
- Toute règle comprenant un non terminal entouré de deux mots (son contexte)
- Engendre des langages contextuels
- Fonctionne avec des **automates linéaire borné** (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

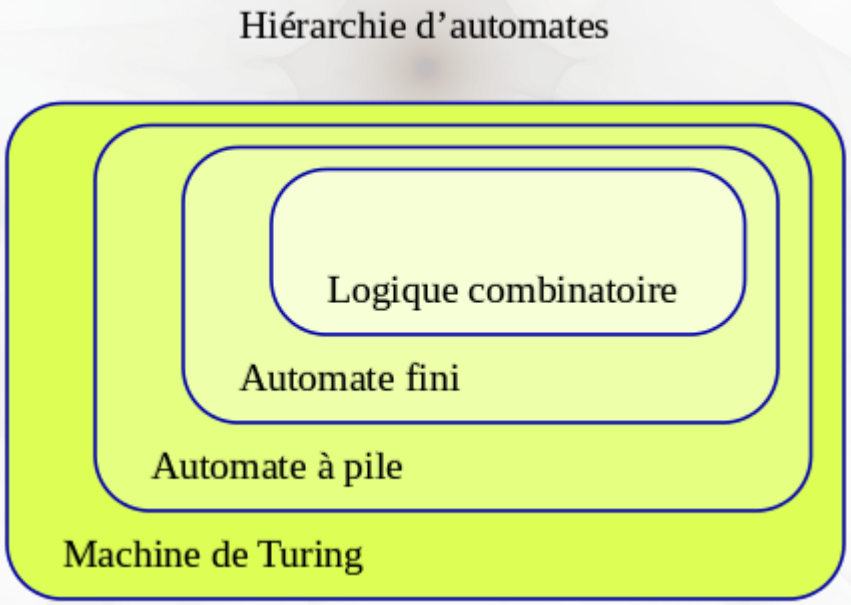
- Les règles sont de la forme :

  $\alpha \rightarrow \beta \qquad (\alpha \in V^*NV^* : \beta \in V^*)$
- Aucunes restrictions
- Engendre des langage 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



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 formels (langages réguliers L3)

Automate fini

- Représenté par un quintuplet (Q, Σ, δ, q0, T)
  Où :
  - Q est l'ensemble des états
  - Σ est l'ensemble des symboles de l'alphabet
  - δ est l'ensemble des transitions
  - $q0 \in Q$ 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 milieux est un nombre indéterminé de a ou de b
- Soit l'expression suivante : a (a, b)* b
- Soit la grammaire suivante :

  A → aB

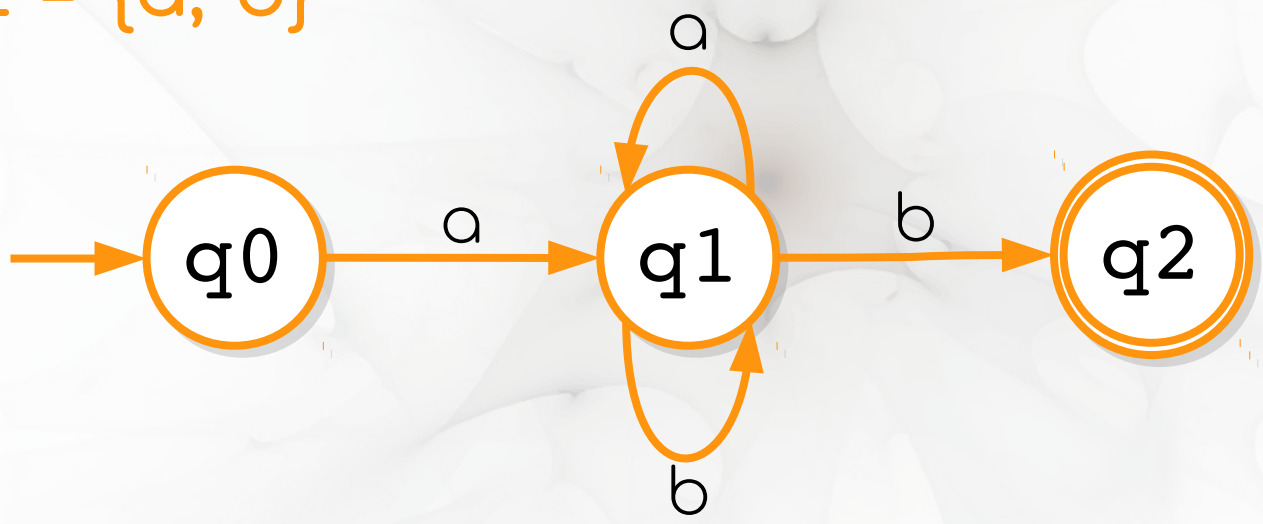
  B → aB

  B → bB

  B → b

Exemple

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



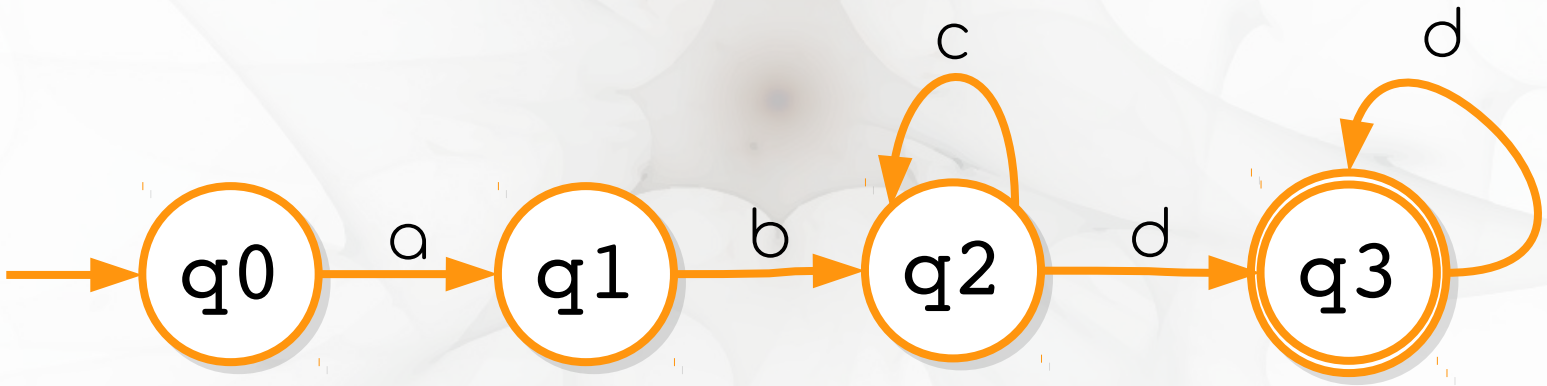
- Accepte : ab, aabab, abbb, aab
- N'accepte pas : a, b, abba, baba, bbbb

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

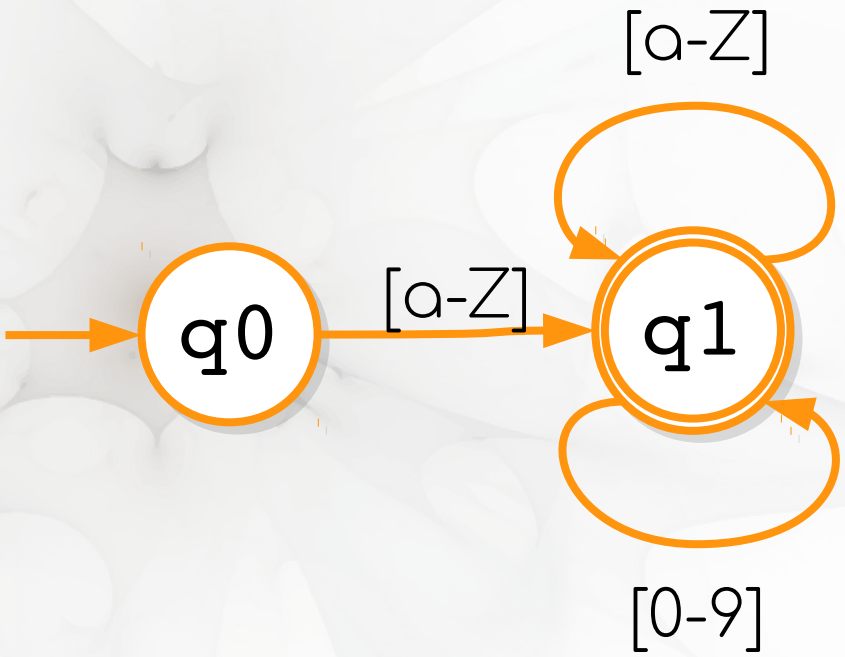
- Non il faut un automate à pile !
- DFA : $\{ a\ b\ c^m\ d^n : m \geq 0, n \geq 1 \}$



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

Exemple concret

- Grammaire :
  - $V \rightarrow A\ AN^*$
  - $AN = A \mid N$
  - $A \rightarrow [a\text{-}Z]$
  - $N \rightarrow [0\text{-}9]$



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

DFA, NFA, NFA-ε

- Un automate fini est déterministe SSI :
  - Il ne possède qu'un état initiale
  - Pour tout état q et alphabet a, il n'existe qu'une seul transition de q par l'alphabet a

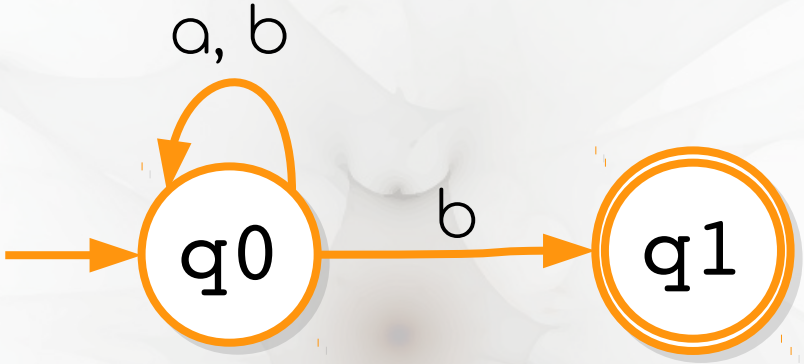
- Par opposition un automate fini est non-déterministe quand une transition peut mener à des états différents

DFA, NFA, NFA-ε

- Un automate fini est déterministe SSI :
  - Il ne possède qu'un état initiale q0
  - Pour tout état q et alphabet a, il n'existe qu'une seul 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 ver 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

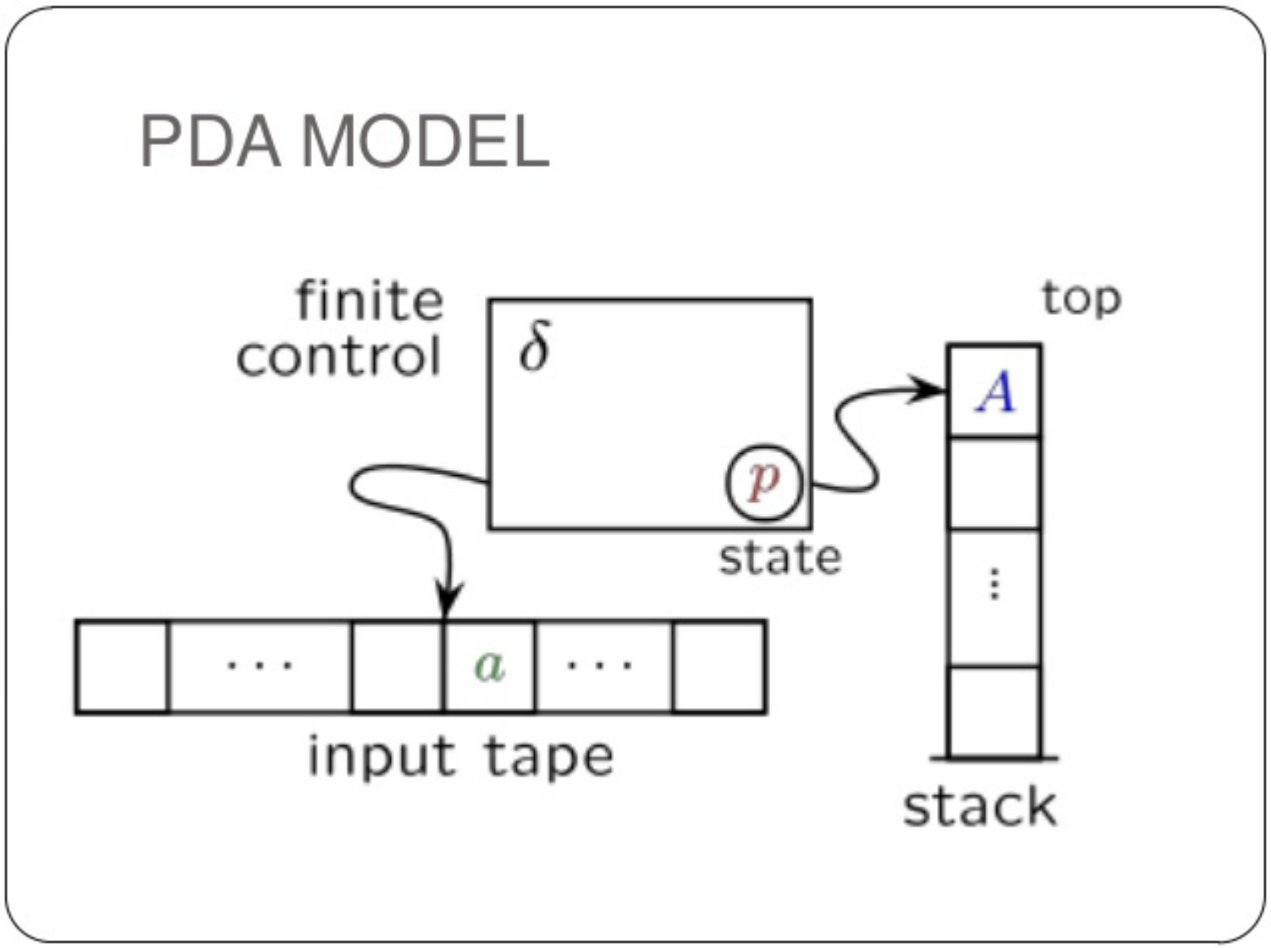
  $\{ a^n b^n : n \geq 0 \}$

- 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 peu empiler un état
- Permet de reconnaître les langages non contextuels L2 - Parser

Automate à pile

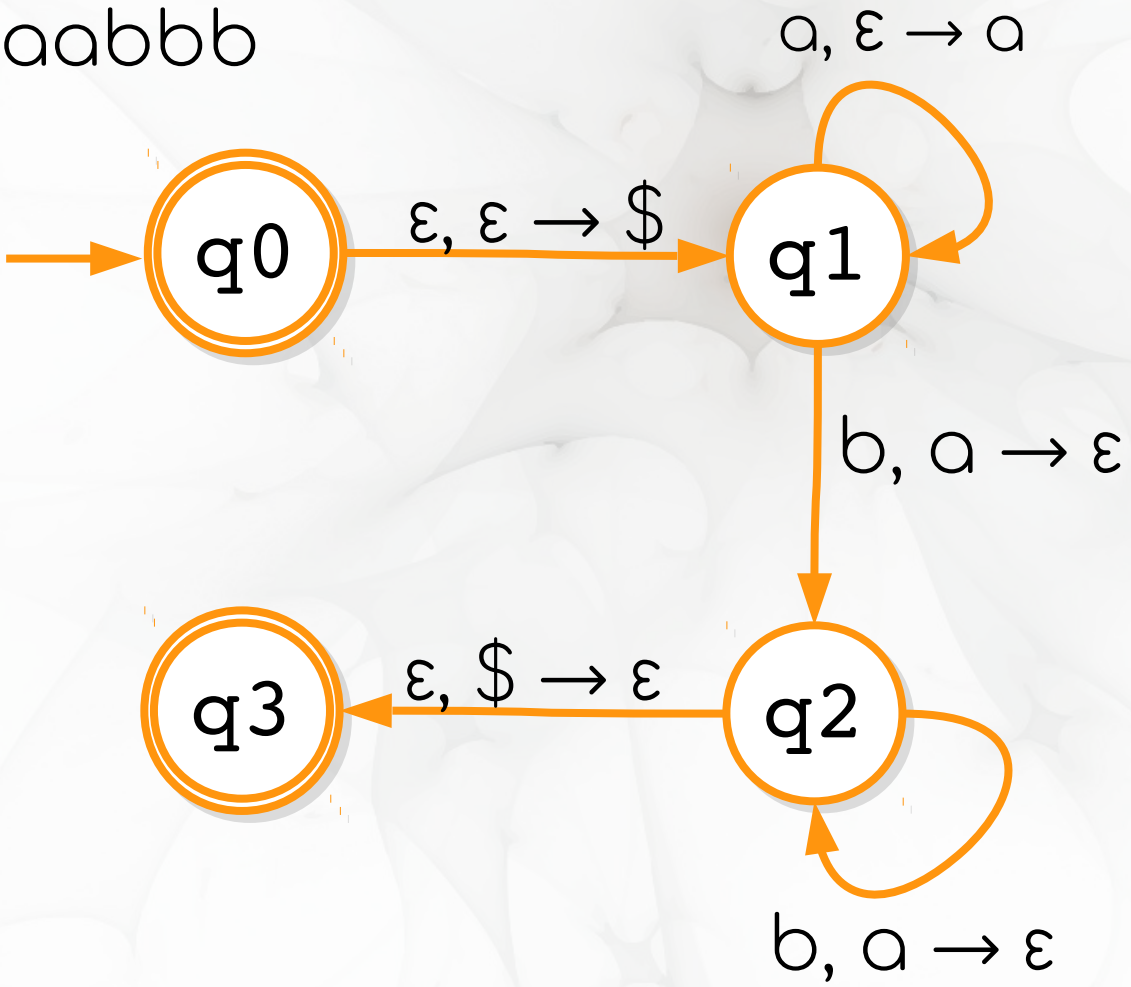


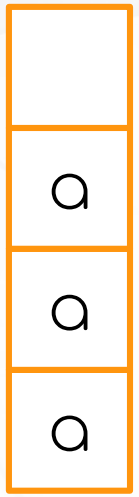
Automate à pile

- Un septuplet (Q, A, Z, δ, z0, q0, T)
  Où :
  - Q est l'ensemble des états
  - A l'alphabet d'entré
  - Z l'alphabet de pile
  - δ est l'ensemble des transitions
  - z0 ∈ Z est le symbole de fond de pile
  - q0 ∈ Q l'état initial
  - T ⊂ Q l'ensemble des états terminaux

Exemple

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





Exemple

- Accepte :
  ab, aabb, aaabbb
- 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 : $\{ a^n b^n c^n : n \geq 0 \}$?

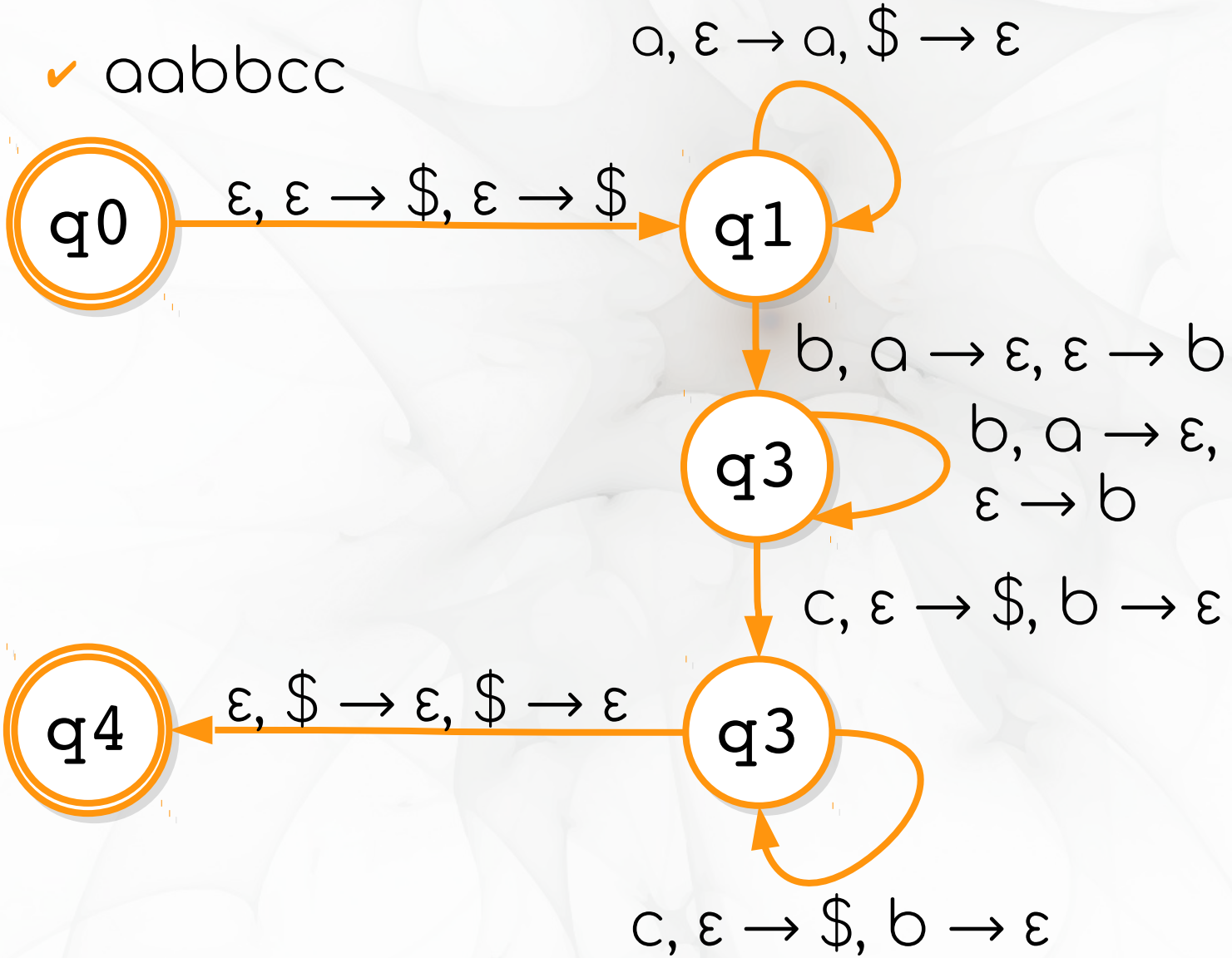
Automate linéaire borné

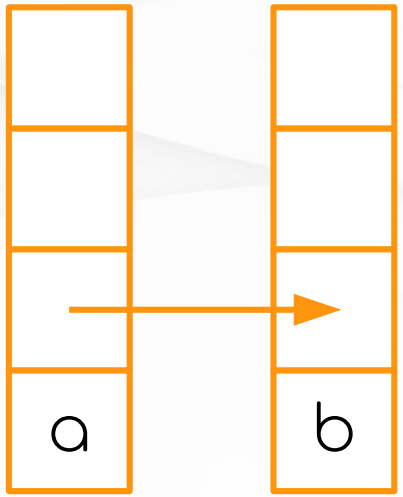
- Solution :
- Comment transformer l'automate pour l'expression : $\{ a^n b^n c^n : n \geq 0 \}$?
- Impossible avec un automate à pile
- Il faut utiliser deux piles (automate linéaire borné)

- Lien YouTube Prof Bill Byrne :
- https://www.youtube.com/watch?v=p1peR1Qbp0s

Automate linéaire borné

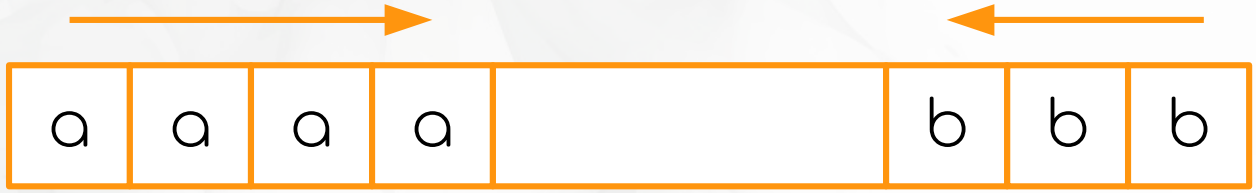
- ✔ aabbcc





Automate linéaire borné

- Deux piles peuvent être contenue fans 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 Récursifs (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 (mathématique) qui fonde la science informatique
- Permet de vérifier les langages récursivement énumérables L0
- Première définition systématique des programmes informatiques
- Porte en son fondement que les données et le programme sont de même nature :
- Le programme est contenu sur une seconde bande infinie

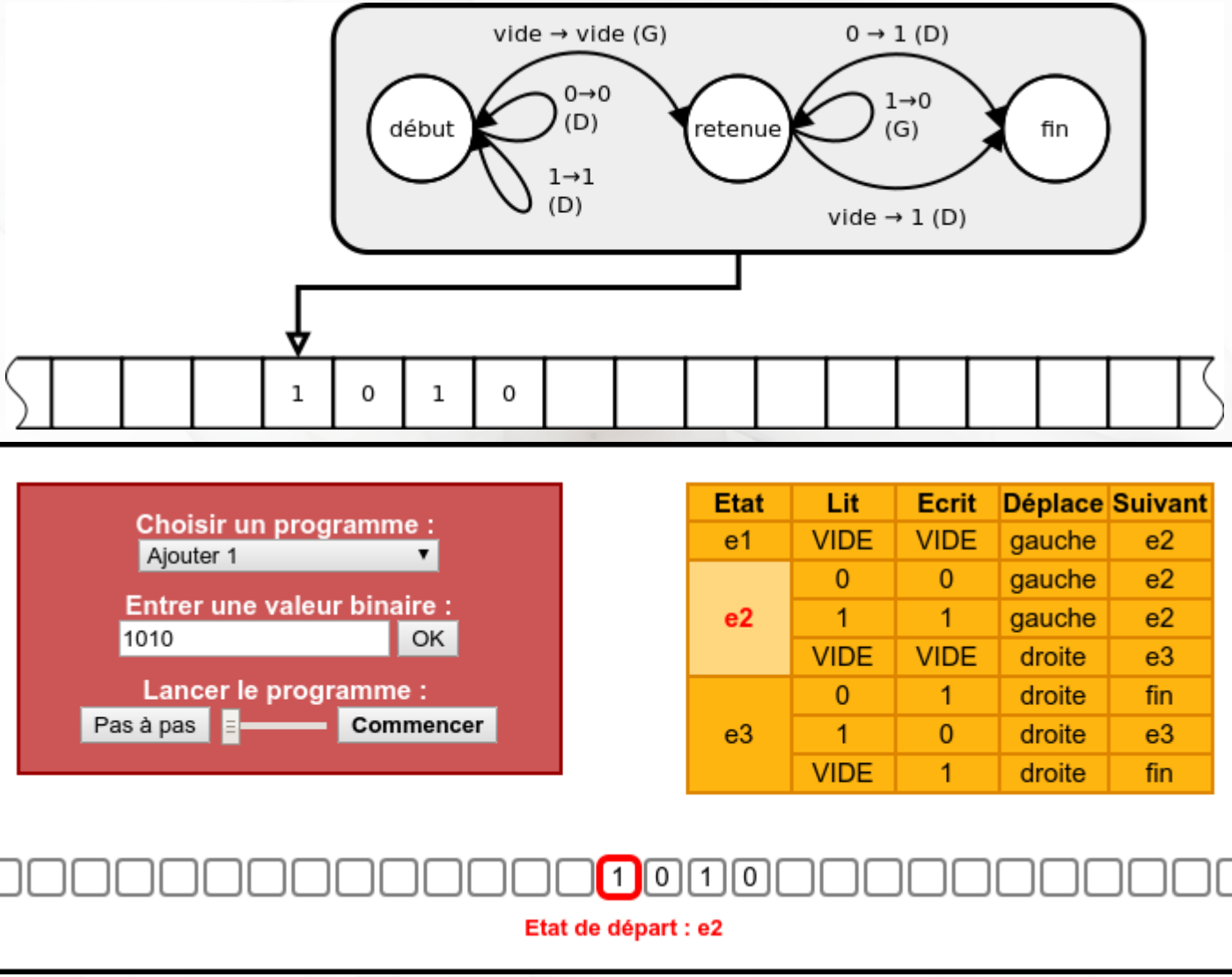
Machine de Turing

- 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, q0, δ, T) ou :
  - Q est un ensemble fini d'états
  - A l'alphabet de travail
  - a ∈ A symbole blanc (case vide)
  - Z ∈ A \ a est l'alphabet d'entrée
  - q0 ∈ Q l'état initial
  - δ : Q x Γ → Q x Γ x {←, →} est la fonction de transition
  - T ⊂ Q l'ensemble des états terminaux

Opération + 1



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

Démonstration

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

Conclusion

Grammaire	Règles de production	Langage	Machine abstraite
Type 0	$\alpha \rightarrow \beta$	Récursivement énumérable	Machine de Turing
Type 1	$\alpha A \beta \rightarrow \alpha \gamma \beta$	Contextuel	Automate linéairement borné (2 piles)
Type 2	$A \rightarrow \gamma$	Algébrique, Non Contextel	Automate à pile (1 pile)
Type 3	$A \rightarrow aB$, $A \rightarrow a$	Rationnel	Automate fini (1 seul état courant)

Merci de votre attention