Automates et Langages
Partie 2

Emmanuelle Grislin

Université Polytechnique HAUTS-DE-FRANCE

INSA 3 FISE et FISA Informatique

Partie 2

Mars 2021

Objectifs du cours 3

Savoirs :

- connaissance de la représentation par *automate à pile* pour les langages algébriques

Savoir-faire (Cf. TD 3) :

- établir quel est le langage reconnu par un automate à pile donné
- construire un automate à pile reconnaissant un langage algébrique donné

Conclusion :

- introduction à l'analyse syntaxique
- récapitulatif de cette partie du module

Grammaire de type 2 : algébrique

Grammaire algébrique ou "hors contexte" (context-free)

Grammaire définie par un quadruplet $G =< \Sigma, V, S, R >$ avec

- Σ : ensemble fini de symboles terminaux
- V : ensemble fini de variables (symboles non terminaux, $\notin \Sigma$)
- S : symbole de V particulier appelé "axiome" ou "racine"
- R : ensemble fini de règles de production de la forme :

$$T \rightarrow u$$
$$\text{avec } T \in V \text{ et } u \in (\Sigma \cup V)^*$$

- le type de la plupart des langages de programmation
- besoin d'automates à pile pour les reconnaître

Automates à pile

- Langages algébriques reconnus par des **automates à pile**
- **Pile** = mémoire additionnelle qui permet de "compter"
- Pile initialisée à vide et de capacité illimitée

Automate à pile non déterministe

APN défini par $A =< \Sigma, Q, \Gamma, Z, q_0, F, \Delta >$ avec

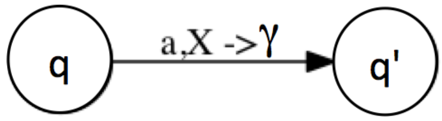
- Σ : alphabet de l'automate
- Q : ensemble fini d'états
- Γ : alphabet de la pile
- Z : symbol initial de la pile= symbole de pile vide $\notin (\Gamma \cup \Sigma)$
- q_0 : état initial
- F : ensemble d'états finaux
- Δ : relation de transition $\subset ((Q \times \Sigma \times \Gamma) \times (Q \times \Gamma^*))$

Automate à pile : représentation graphique

Un automate à pile se représente comme les automates vus en Partie 1 du cours avec,
sur chaque arc : symbole de transition + opération sur la pile

$$(q, a, X) \rightarrow (q', \gamma)$$

(état de l'automate, symbole de Σ, symbole en sommet de pile)
$\rightarrow$ (nouvel état, séquence de symboles qui remplace X dans la pile)



Langage accepté par APN

Langage accepté par automate à pile A ?

2 modes équivalents :

- reconnaissance sur état final :

$$L(A) = \{u \in A^* | \text{ la pile étant vide au départ, il existe un chemin de trace } u \text{ de } q_0 \text{ à } f, f \in F\}$$

- reconnaissance par pile vide :

$$L(A) = \{u \in A^* | \text{ la pile étant vide au départ, il existe un chemin de trace } u \text{ à partir de } q_0 \text{ qui compte au moins une transition et se termine avec la pile vide}\}$$

Si le mode de reconnaissance n'est pas spécifié, alors on suppose que les 2 modes sont valides : reconnaissance sur état final **et** *pile vide*

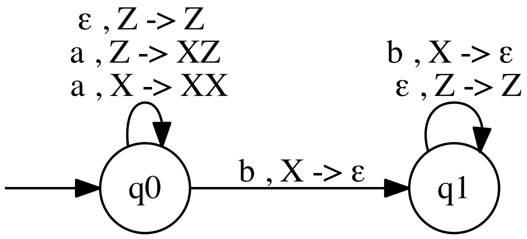
Reconnaissance par pile vide : exemple

Automate à pile $A_1 =< \Sigma, Q_1, \Gamma_1, Z, q_0, F_1, \Delta_1 >$

$\Sigma = \{a, b\}$, $Q_1 = \{q_0, q_1\}$, $\Gamma_1 = \{Z, X\}$, $F_1 = \emptyset$

$$\begin{aligned} \Delta_1 = \{ \quad & ((q_0, \epsilon, Z), (q_0, Z)) \\ & ((q_0, a, Z), (q_0, XZ)) \\ & ((q_0, a, X), (q_0, XX)) \\ & ((q_0, b, X), (q_1, \epsilon)) \\ & ((q_1, b, X), (q_1, \epsilon)) \\ & ((q_1, \epsilon, Z), (q_1, Z))\} \end{aligned}$$

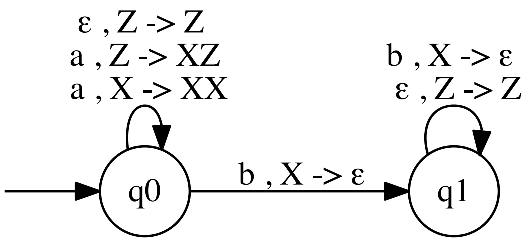
Mode de reconnaissance : pile vide.



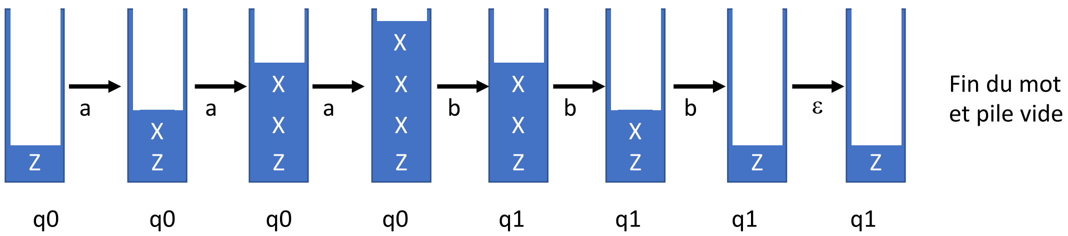
A_1 accepte les mots de $\{a^n b^n | n \geq 0\}$

Reconnaissance par pile vide : exemple

Mode de reconnaissance : pile vide.



Le mot *aaabbb* est-il reconnu ?



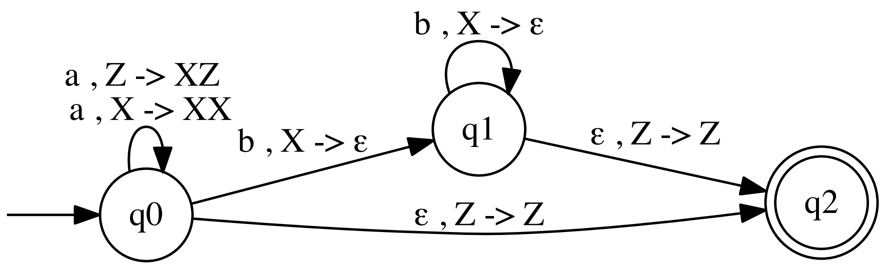
A_1 accepte les mots de $\{a^n b^n | n \geq 0\}$

Reconnaissance par état final : exemple

Ex. : Automate à pile $A_2 =< \Sigma, Q_2, \Gamma_2, Z, q_0, F_2, \Delta_2 >$
$\Sigma = \{a, b\}$, $Q_2 = \{q_0, q_1, q_2\}$, $\Gamma_2 = \{Z, X\}$, $F_2 = \{q_2\}$

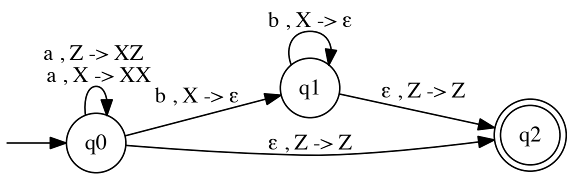
$$\begin{array}{ll} \Delta_2 = \{ & ((q_0, \epsilon, Z), (q_2, Z)) \\ & ((q_0, a, Z), (q_0, XZ)) \\ & ((q_0, a, X), (q_0, XX)) \\ & ((q_0, b, X), (q_1, \epsilon)) \\ & ((q_1, b, X), (q_1, \epsilon)) \\ & ((q_1, \epsilon, Z), (q_2, Z))\} \end{array}$$

Mode de reconnaissance : par état final.

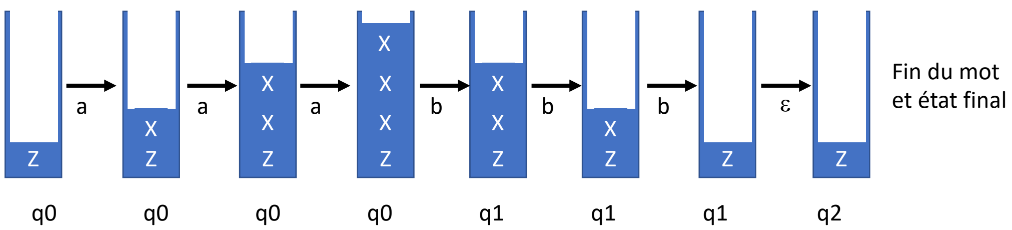


Reconnaissance par état final : exemple

Mode de reconnaissance : état final.



Le mot *aaabbb* est-il reconnu ?



A_2 accepte les mots de $\{a^n b^n | n \geq 0\}$

Les 2 automates A_1 et A_2 sont équivalents : ils acceptent le même langage.
Dans les 2 cas, on empile un X à chaque 'a' et on dépile un X à chaque 'b'.

Langage algébrique et AP

Un langage est algébrique ssi il est reconnu par un automate à pile

Mais

Tout langage algébrique n'est pas reconnu par un automate à pile **déterministe**

Automate à pile déterministe

Automate à pile déterministe

AP est déterministe si, pour chaque configuration (état de l'automate, symbole lu, état de la pile), il existe au plus une configuration dérivable.

Conditions :

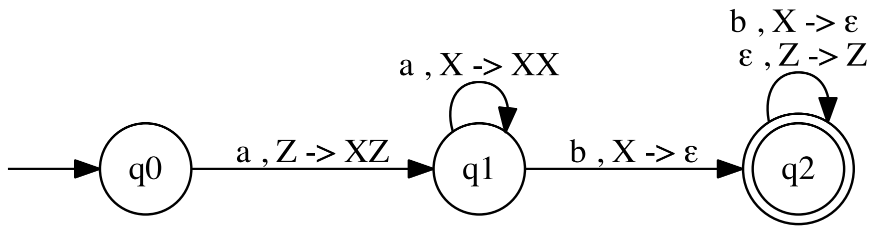
1. pour un état q donné, un symbole a donné, un sommet de pile X donné,
   il existe au plus une transition partant de (q, a, X)
2. pour un état q donné et un sommet de pile X donné,
   s'il existe une transition (q, ϵ, X),
   - elle est unique
   - il n'y a pas d'autre symbole a tel que $(q, a, X) \in \Delta$

Automate à pile déterministe

Ex. : Automate à pile $A_3 =< \Sigma, Q_3, \Gamma_3, Z, q_0, F_3, \delta_3 >$
$\Sigma = \{a, b\}$, $Q_3 = \{q_0, q_1, q_2\}$, $\Gamma_3 = \{Z, X\}$, $F_3 = \{q_2\}$

$$\begin{array}{ll} \delta_3(q_0, a, Z) & = (q_1, XZ) \\ \delta_3(q_1, a, X) & = (q_1, XX)) \\ \delta_3(q_1, b, X) & = (q_2, \epsilon) \\ \delta_3(q_2, b, X) & = (q_2, \epsilon) \\ \delta_3(q_2, \epsilon, Z) & = (q_2, Z)) \end{array}$$

Reconnaissance : sur état final et pile vide



Automate déterministe. A_3 accepte les mots de $\{a^n b^n | n > 0\}$

Passage du langage algébrique à l'automate à pile

Langage algébrique reconnu par AP

Le langage engendré par une grammaire algébrique est reconnu par un automate à pile, car :

- langage algébrique $\rightarrow$ grammaire propre :
  Tout langage algébrique est engendré par une grammaire *propre*
- grammaire propre $\rightarrow$ FNG :
  Toute grammaire propre ne contenant pas de production vide peut être mise sous *forme normale de Greibach*
- FNG $\rightarrow$ automate à pile :
  Toute grammaire sous forme normale de Greibach peut être représentée par un *automate à pile*

Grammaire propre

- *Tout langage algébrique ne contenant pas le mot vide est engendré par une grammaire propre ne contenant pas de production vide*
- *Tout langage algébrique contenant le mot vide est engendré par une grammaire propre telle que :*
  - *la seule production vide est :* $S \to \epsilon$
  - S *est absent des parties droites des règles de production*

Grammaire propre

Une grammaire $G =< \Sigma, V, S, R >$ est dite "propre" si :

- chaque symbole de Σ et chaque symbole de V apparaît dans la dérivation d'au moins un mot du langage L(G)
- R ne contient pas de règle de la forme :
  $X \to Y$ avec $X, Y \in V$
- G ne contient pas de cycle : il n'existe pas de $X \in V$ tel que
  $X \to^{+} X$

Forme normale de Greibach

- *Toute grammaire propre ne contenant pas de production vide peut être mise sous forme normale de Greibach*

Forme normale de Greibach (FNG)

Une grammaire $G =< \Sigma, V, S, R >$ est sous *forme normale de Greibach* si l'ensemble de règles R est tel que :

$$\begin{array}{lll} X & \to aU & \text{avec } X \in V, a \in \Sigma, U \in (V \setminus \{S\})^* \\ S & \to \epsilon & \text{si } \epsilon \in L(G) \end{array}$$

Grammaire récursive gauche

Une grammaire FNG n'a pas de récursivité gauche.

Elimination de la récursivité à gauche

- Une variable X est récursive gauche si $\exists w \in (\Sigma \cup V)^*, \exists n \geq 1, X \xrightarrow{n} Xw$
- une grammaire est récursive gauche si elle comporte une variable récursive gauche
- tout langage algébrique peut être engendré par une grammaire algébrique non récursive gauche :
  - toute règle de la forme $X \rightarrow Xw_1|...|Xw_n|u_1|...|u_p$ est remplacée par :
    $$\begin{array}{ll} X & \rightarrow u_1Y|...|u_pY \\ Y & \rightarrow w_1Y|...|w_nY|\epsilon \end{array}$$

ex. : Soit $G =< \{a, b\}, \{S\}, S, \{S \rightarrow Sa|b\} >$

remplacé par : $\begin{array}{ll} S & \rightarrow bS' \\ S' & \rightarrow aS'|\epsilon \end{array}$

Automate à pile

- *Toute grammaire sous forme normale de Greibach peut être représentée par un automate à pile*

Passage de la FNG à l'automate à pile

Si G est sous forme normale de Greibach, on peut construire l'automate à pile $A =< \Sigma, \{q_0, q_1\}, V, Z, q_0, \emptyset, \delta >$:

1. Pout toute règle $X \rightarrow aX_1 \dots X_n$, on crée $\delta(q_1, a, X) = (q_1, X_1 \dots X_n)$
2. Si $S \rightarrow \epsilon \in R$, on ajoute $\delta(q_0, \epsilon, S) = (q_0, \epsilon)$
3. On ajoute $\delta(q_0, \epsilon, Z) = (q_1, SZ)$ pour démarrer

Langage algébrique et AP

Langage algébrique reconnu par AP

Le langage engendré par une grammaire algébrique est reconnu par un automate à pile, car :

- langage algébrique $\rightarrow$ grammaire propre :
  - Tout langage algébrique ne contenant pas le mot vide est engendré par une grammaire propre ne contenant pas de production vide
  - Tout langage algébrique contenant le mot vide est engendré par une grammaire propre telle que :
    - la seule production vide est : $S \rightarrow \epsilon$
    - S est absent des parties droites des règles de production
- grammaire propre $\rightarrow$ FNG :
  Toute grammaire propre ne contenant pas de production vide peut être mise sous forme normale de Greibach
- FNG $\rightarrow$ automate à pile :
  Toute grammaire sous forme normale de Greibach peut être représentée par un automate à pile

Analyse syntaxique

Problématique initiale :

Etant donnée une grammaire G, déterminer si un mot (ou un texte) donné $u \in \Sigma^*$ appartient au langage $L(G)$.

Analyse syntaxique

- Processus qui :
  - vérifie l'appartenance d'un mot (ou un texte) au langage
  - donne la structure logique du mot par un arbre de dérivation
  - renvoie VRAI ssi le texte respecte les spécifications du langage décrites par la grammaire
- Deux stratégies possibles :
  - **Analyse descendante** : de l'axiome (la racine) vers le mot (les feuilles). Analyses de type LL(1), LL(k)
  - **Analyse ascendante** : du mot vers l'axiome. Analyses de type LR(0), LR(1), SLR(1), LALR(1),...

Analyse LL(1)

Analyse descendante : parcours *en profondeur d'abord* de l'arbre des possibilités de dérivation

1. texte parcouru de gauche à droite par une "tête de lecture" (pas de retour arrière) à partir d'un tampon d'entrée contenant la chaîne à analyser, suivie du symbole $ comme marque de fin de chaîne
2. dérivations à gauche
3. utilisation de :
   - une **pile** contenant une séquence de symboles avec $ en marqueur de fond de pile
   - une **table d'analyse** M[A,a] avec $A \in V$ et $a \in \Sigma$, qui donne règle applicable étant donnés le symbole sous la tête de lecture et le symbole en sommet de pile
4. arrêt si texte complet trouvé (renvoie VRAI) ou si aucune règle applicable (FAUX)

Analyse LL(1)

Principe

L'analyseur agit en fonction de X le sommet de pile et a le symbole courant lu en entrée :

1. si $X = a = \$$: arrêt sur réussite de l'analyse (renvoie VRAI)
2. si $X = a$ et $a \neq \$$: l'analyseur dépile X et avance au symbole suivant
3. si $X \in V$,
   - si M[X,a] est vide, aucune règle applicable, il y a une erreur (renvoie FAUX)
   - sinon, dépile X et empile les symboles produits par la règle qui est dans M[X,a].
     - Ex. :
       si $M[X, a] = \{X \rightarrow UVW\}$, alors dépile X et empile W puis V puis U

Analyse LL(1)

- Conditions de mise en oeuvre d'une analyse LL(1) :
  - grammaire non récursive à gauche
  - grammaire non ambiguë
  - une seule règle au plus par case de la table d'analyse
- Le 1 de LL(1) signifie que 1 un seul symbole d'entrée est suffisant pour prévoir la suite de l'analyse. Il existe des analyseurs LL(k) avec k>1

Analyse ascendante

- **L'analyse ascendante construit l'arbre syntaxique de bas en haut** : part du mot (les feuilles) et remonte vers le symbole axiome (la racine)
- Le mot w est "réduit" progressivement jusqu'à l'axiome : on repère des dérivations droites dans w et on remplace les symboles dérivés par les parties gauches des règles associées
  En simplifié :
  Si $w = \alpha\beta u$ avec $u \in \Sigma^*$ et s'il existe $A \rightarrow \beta$, on remplace β par A. β est un "manche" de w
- Ce type d'analyse peut être appliqué à un ensemble plus important de grammaires que l'analyse descendante
- Principales méthodes : LR(0), SLR(1), LR(1), LALR(1)

Analyse ascendante LR(k)

Principe

- utilisation d'un tampon d'entrée avec la chaîne de symboles à analyser, une pile et une table d'analyse divisée en parties : les Actions et les Successeurs
- le symbole $ = fond de pile et extrêmité droite de la chaîne d'entrée (comme pour LL)
- init. sur pile vide et chaîne dans le tampon d'entrée
- tant que ((pas d'erreur) et (axiome non empilé) et (tampon d'entrée non vide)) faire :
  1. chaque symbole lu est empilé jusqu'à ce qu'un "manche" β soit en sommet de pile
  2. β est réduit : il est remplacé par la partie gauche de la règle associée

Analyse ascendante

Exemple

Soit la grammaire $G =< \{a, b, c, d, e\}, \{S, A, B\}, S, R >$ avec R :

$$S \rightarrow aABe$$
$$A \rightarrow Abc|b$$
$$B \rightarrow d$$

Le mot *abbcde* est réduit ainsi :

1. *abbcde*
2. *aAbcde*
3. *aAde*
4. *aABe*
5. S

ce qui correspond à retrouver, en sens inverse, les dérivations à droite menant de S au mot *abbcde* :

$S \rightarrow_d aABe \rightarrow_d aAde \rightarrow_d aAbcde \rightarrow_d abbcde$

Analyse ascendante

Exemple de l'algorithme CYK (Cocke, Yunger et Kasami)

Soit $G =< \Sigma, V, S, R >$ une grammaire algébrique sous forme normale de Chomsky et m un mot sur l'alphabet Σ.

Comment déterminer si $m \in L(G)$?

- On appelle $m_{i,j}$ le facteur de m débutant à la lettre $m(i)$ et de longueur j.
- On calcule les non terminaux $M[i,j] = \{X \in V | X \rightarrow_* m_{i,j}\}$
- $m \in L(G) \Leftrightarrow S \in M[1, |m|]$

Récapitulatif de cette partie du module "Automates et langages"

- Savoirs :
  - connaissance de la *hiérarchie de Chomsky* : langages et grammaires réguliers, algébriques, contextuels et généraux
  - connaissance du *théorème de Kleene* : équivalence entre expression régulière $\leftrightarrow$ grammaire régulière $\leftrightarrow$ AFN
  - connaissance de la représentation par *automate à pile* pour les langages algébriques
- Savoir-faire :
  - *déterminer le type d'un langage au sens de la hiérarchie de Chomsky*
  - *donner le langage reconnu par une grammaire de type 3 ou 2*
  - *donner une grammaire engendrant un langage de type 3 ou 2*
  - *construire un automate à pile reconnaissant un langage algébrique donné*
- En conclusion : petite introduction à l'*analyse syntaxique*