Analyse lexicale

Cyril Rabat
cyril.rabat@univ-reims.fr

Licence 3 Informatique - Info0602 - Langages et compilation

2021-2022

UFR SEN
Sciences Exactes et Naturelles

Université de Reims Champagne-Ardenne

Cours n°2

Qu'est-ce qu'un analyseur lexical ?

Langages, expressions régulières, automates finis

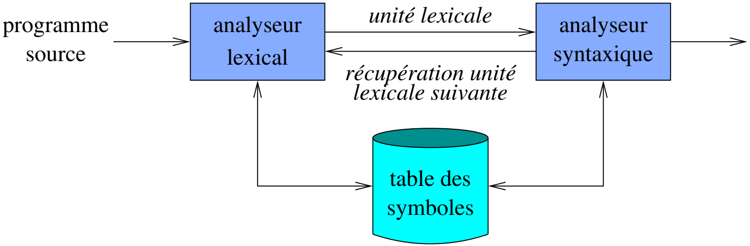
Version 10 janvier 2022

Table des matières

2. L'analyse lexicale
   - Un analyseur lexical
   - Les langages
   - Les expressions régulières
   - Les automates finis
   - De l'expression régulière à un AFN
   - Transformation d'un AFN en AFD
   - Partitionnement
   - De l'automate à l'expression régulière
   - Construction d'un analyseur lexical
   - Automates à pile

Un analyseur lexical

- Lecture des caractères en entrée
- Production d'unités lexicales
  $\hookrightarrow$ Analysées par l'analyseur syntaxique
- Interactions entre les deux analyseurs (lexical et syntaxique) :



Tâches secondaires de l'analyseur lexical

- Éliminer :
  - Les commentaires
  - Les caractères "inutiles" (espaces, tabulations, lignes vides...)
- Faciliter la gestion des erreurs :
  - Conservation/calcul du numéro de ligne
  - Associer les messages d'erreur à une ligne
- Le principal intérêt de l'analyseur lexical est de simplifier l'analyseur syntaxique

Modèle et unité lexicale

Définition : modèle

Règle qui décrit un ensemble de chaînes

Exemples

- $[0-9]^*$
- a^*b

Définition : unité lexicale

Éléments produit par l'ensemble des chaînes du modèle

Exemples

- mots-clés, opérateurs, identificateurs, constantes, chaînes littérales...

Lexème et attribut

Définition : lexème

Suite de caractères du programme source qui correspond au modèle

Exemples

- Modèle : $[0-9]^*$
- Lexème : 100, 001, 123

Définition : attribut

Données liées aux unités lexicales

Exemple

- L'entrée dans la table des symboles pour un identificateur

Alphabet et mots

Définition : alphabet

*Un **alphabet** est un ensemble fini de symboles appelés caractères. Il est noté $\mathcal{A}$.*

- Exemples de symboles : lettres et caractères
- Exemples d'alphabets : $\{0, 1\}$ (l'alphabet binaire), l'ASCII

Définition : mot (ou chaîne)

*Un **mot** sur un alphabet est une séquence finie de symboles de cet alphabet. La **longueur du mot** w (notée $|w|$) est le nombre de symboles dans ce mot. Le **mot vide**, noté ϵ, est un mot de longueur 0.*

- Exemples de mots sur l'alphabet $\{a, b, c\}$: a, $baba$

Partie de mots

- **Préfixe** de w : mot obtenu en supprimant un nombre quelconque de symboles en fin de w (voire aucun)
- **Suffixe** de w : mot obtenu en supprimant un nombre quelconque de symboles en début de w (voire aucun)
- **Sous-mot** de w : mot obtenu en supprimant un préfixe et un suffixe de w
- **Préfixe propre** de w : tout mot non vide x, préfixe de w tel que $x \neq w$
- Idem pour **suffixe propre** et **sous-mot propre** de w
- **Sous-suite** de w : tout mot obtenu en supprimant un nombre quelconque de symboles de w, éventuellement aucun, pas nécessairement consécutifs

Opérations sur les mots

- **Concaténation de mots** : si x et y sont des mots, la concaténation xy est la chaîne formée en joignant x et y
  $\hookrightarrow$ Exemple : pour $\mathcal{A} = \{a, b\}$, si $x = aa$ et y = bb, alors $xy = aabb$
- **Exponentiation** : $s^0 = \epsilon$; $s^i = s^{i-1}s$
  $\hookrightarrow$ Exemple : pour $\mathcal{A} = \{a, b\}$, si $x = ba$ alors $x^3 = bababa$

Langage

Définition : langage

Un langage est un ensemble de mots définis sur un même alphabet.

- Soit $\mathcal{A} = \{1, 2, 3\}$, l'ensemble $\{1, 11, 12, 21\}$ est un langage sur $\mathcal{A}$
- Le langage vide est noté $\emptyset$
- Le langage $\{\epsilon\}$ ne contient que le mot vide

$$\emptyset \neq \{\epsilon\}$$

- Un langage peut être défini de plusieurs manières :
  - Expression régulière, automate, grammaire, expression mathématique
  - Ensemble de mots

Opérations sur les langages (1/2)

Soit deux langages L_1 et L_2 définis respectivement sur les alphabets $\mathcal{A}_1$ et $\mathcal{A}_2$.

Définition : union de deux langages

L'union de L_1 et L_2 définie sur $\mathcal{A}_1 \cup \mathcal{A}_2$ est le langage contenant tous les mots de L_1 et L_2 :

$$L_1 \cup L_2 = \{w \mid w \in L_1 \lor w \in L_2\}$$

Définition : intersection de deux langages

L'intersection de L_1 et L_2 définie sur $\mathcal{A}_1 \cap \mathcal{A}_2$ est le langage contenant tous les mots qui sont à la fois dans L_1 et L_2 :

$$L_1 \cap L_2 = \{w \mid w \in L_1 \land w \in L_2\}$$

Opérations sur les langages (2/2)

Définition : complément d'un langage

Le complément de L est le langage défini sur $\mathcal{A}$ contenant tous les mots qui ne sont pas dans L :

$$\mathcal{C}(L) = \{w \mid w \in \mathcal{A} \land w \notin L\}$$

Définition : différence de deux langages

La différence de L_1 et L_2 est le langage défini sur $\mathcal{A}$ contenant tous les mots de L_1 qui ne sont pas dans L_2 :

$$L_1 - L_2 = \{w \mid w \in L_1 \land w \notin L_2\}$$

Produit et puissances

Définition : produit de deux langages

Le produit ou ***concaténation*** *de* L_1 *et* L_2 *est le langage défini sur* $\mathcal{A}_1 \cup \mathcal{A}_2$ *contenant tous les mots formés d'un mot de* L_1 *suivi d'un mot de* L_2 *:*

$$L_1.L_2 = \{w_1 w_2 \mid w_1 \in L_1 \land w_2 \in L_2\}$$

Définition : puissances d'un langage

Les puissances successives de L *définies sur* $\mathcal{A}$ *sont définies récursivement :*

- $L^0 = \{\epsilon\}$
- $L^n = L.L^{n-1}$ *pour* $n \geq 1$

Fermeture itérative

Définition : fermeture de Kleene de deux langages

La fermeture de Kleene de L, appelée également la ***fermeture itérative****, définie sur $\mathcal{A}$, est l'ensemble des mots formés par une concaténation finie des mots de L_1 :*

$$L^* = \{w \mid \exists k \geq 0 \wedge w_1 \ldots w_k \in L \textit{ tels que } w = w_1 w_2 \ldots w_k\}$$

- On définit également L^+ :
  $L^+ = \{w \mid \exists k > 0 \wedge w_1 \ldots w_k \in L \text{ tels que } w = w_1 w_2 \ldots w_k\}$

Langage fini et infini

Définition : langage fini

Un langage fini peut être décrit par l'énumération des mots qui le compose. Ce qui n'est pas le cas pour un langage infini.

- Certains langages infinis peuvent être décrits à l'aide d'opérations sur des langages simples
- Certains langages infinis peuvent être décrits à l'aide de règles (grammaires)
- Les langages qui ne peuvent être décrits ni par des opérations, ni par des grammaires sont des langages **indécidables**.

Expressions régulières

Définition : expression régulière

Les expressions régulières pour un alphabet $\mathcal{A}$ sont les expressions formées par les règles suivantes :

- *$\emptyset$, ϵ et les symboles de $\mathcal{A}$ sont des expressions régulières*
- *Si α et β sont des expressions régulières sur $\mathcal{A}$, $(\alpha|\beta)$, $(\alpha.\beta)$ et $(\alpha)^*$ sont des expressions régulières*

- On note indifféremment $\alpha.\beta$ et $\alpha\beta$
- On définit une priorité décroissante sur les opérateurs : $*$, . et $|$

Langage décrit par une expression régulière

Définition : langage décrit par une expression régulière

Le langage $L(E)$ où E est une expression régulière définie sur $\mathcal{A}$, est défini comme suit :

- $L(E) = \emptyset$ *si* $E = \emptyset$
- $L(E) = \{\epsilon\}$ *si* $E = \epsilon$
- $L(E) = \{a\}$ *si* $E = a$ *pour tout* $a \in \mathcal{A}$
- $L(E) = L(E_1) \cup L(E_2)$ *si* $E = E_1 | E_2$
- $L(E) = L(E_1).L(E_2)$ *si* $E = E_1.E_2$
- $L(E) = L(E_1)^*$ *si* $E = E_1^*$

Introduction

- Expression régulière : définie un ensemble de mots
- Nécessité de les compiler pour créer un programme qui la reconnait
- But du programme :
    - Entrée : le mot à reconnaître
    - Sortie : oui ou non suivant si le mot est reconnu par l'expression régulière ou non
- Utilisation d'**automates**

Présentation des automates

Un automate fini se compose :

- D'un **ruban d'entrée** :
  - Constitué d'un ensemble de cases, chacune contenant un caractère
  - Le mot à traiter est placé dans ces cases
  - Une tête de lecture permet de connaître le caractère suivant
- D'un **ensemble d'états** :
  - L'automate passe d'un état à l'autre en cours d'exécution
  - L'**état initial** : état en début d'exécution
  - Les **états d'acceptation** (ou états finals) : états atteints lorsque le mot est accepté
- D'une **fonction de transition** :
  - Indique pour chaque état et symbole lu, le prochain état

Automates finis déterministes *vs* non déterministes

- Dans un état donné et pour un symbole donné
  $\hookrightarrow$ La fonction de transition indique le prochain état
- Deux possibilités lorsqu'un symbole est rencontré :
  - Une seule transition possible :
    $\hookrightarrow$ L'automate est **déterministe**
  - Plusieurs transitions possibles :
    $\hookrightarrow$ L'automate est **non déterministe**

Remarque

Des définitions plus formelles sont présentées dans la suite !

Automate fini déterministe

Définition : automate fini déterministe

Un automate fini déterministe (noté AFD) est défini formellement par le quintuplet $M = \{Q, \mathcal{A}, \delta, s, F\}$, où :

- *Q est un ensemble fini d'états*
- *$\mathcal{A}$ est un alphabet*
- *δ est la fonction de transition de $Q \times \mathcal{A}$ dans Q*
- *$s \in Q$ est l'état initial*
- *$F \subseteq Q$ est l'ensemble des états d'acceptation*

Représentation d'un automate fini par un graphe

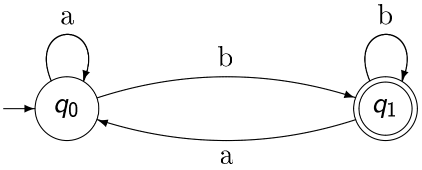
- Chaque état de l'automate est représenté par un nœud du graphe
- Relation de transition représentée par des arcs valués :
  $\hookrightarrow$ Si $\delta(q, a) = q'$, $(q, q') \in Q^2 \wedge a \in \mathcal{A}$ alors il existe un arc a entre les sommets q et q'
- L'état initial est signalé par une flèche
- Les états d'acceptation sont signalés par des doubles cercles

Exemple de représentation

Soit l'automate $M = \{Q, \mathcal{A}, \delta, s, F\}$ suivant :

$\mathcal{A} = \{a, b\}$, $Q = \{q_0, q_1\}$, $s = q_0$, $F = \{q_1\}$ et $\delta =$

q	a	$\delta(q, a)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_0
q_1	b	q_1



Exécution d'un automate

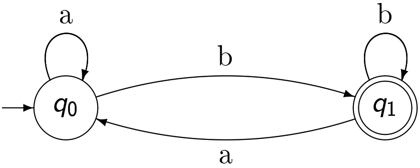
Définition : configuration d'un AFD

Une configuration d'un AFD est une paire composée de l'état de l'automate, ainsi que la partie du mot restant à traiter :

$$(q, w) \in Q \times \mathcal{A}^*$$

- **Exécution d'un automate** : déterminer les configurations successives de l'automate en fonction du mot d'entrée
- **Dérivation** : le passage d'une configuration à une autre

Exemple d'exécution



Soit le mot *aabab* à reconnaître, nous avons les configurations suivantes :

1. $(q_0, aabab)$
2. $(q_0, abab)$
3. (q_0, bab)
4. (q_1, ab)
5. (q_0, b)
6. (q_1, ϵ)

L'exécution comporte 6 configurations et 5 dérivations.

Dérivation (1/2)

Définition : dérivation (en une étape)

La configuration (q', w') est dérivable en une étape de (q, w) par M si :

- $w = aw'$ *($a \in \mathcal{A}$, a est le premier caractère de w et w' est égal à w privé de a)*
- $q' = \delta(q, a)$ *(q' est le prochain état déterminé par la fonction de transition δ pour q et a)*

On note la dérivation de deux configurations $(q, w) \vdash (q', w')$.
On peut également noter $(q, w) \vdash_M (q', w')$.

Dérivation (2/2)

Définition : dérivation (en plusieurs étapes)

La configuration (q', w') est dérivable (en plusieurs étapes) de (q, w) s'il existe $k \geq 0$ et des configurations $(q_i, w_i), 0 \leq i \leq k$ telles que :

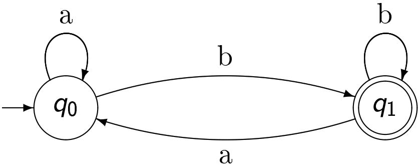
- $(q_0, w_0) = (q, w)$, $(q_k, w_k) = (q', w')$
- $\forall i \in [0, k], (q_i, w_i) \vdash (q_{i+1}, w_{i+1})$

On note $(q, w) \vdash^* (q', w')$.

- Soit les dérivations suivantes (on parle aussi d'exécution) :
$$(s, w) \vdash (q_1, w_1) \vdash \ldots \vdash (q_n, \epsilon)$$
- s est l'état initial, ϵ le mot vide, $n = |w|$

Une seule exécution possible pour chaque mot

Exemple de dérivations



1. (q_0, abb) est dérivable en une étape de $(q_0, aabb)$
   $\hookrightarrow$ Il existe une transition $(q_0, a) \rightarrow q_0$
2. (q_1, ϵ) est dérivable en 4 étapes de $(q_0, aabb)$
   $\hookrightarrow$ $(q_0, aabb) \vdash (q_0, abb) \vdash (q_0, bb) \vdash (q_1, b) \vdash (q_1, \epsilon)$

Acceptation

Définition : acceptation

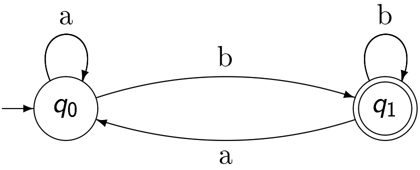
Un mot est accepté par un automate M si le dernier état est un état d'acceptation :

$$w \text{ est accepté par } M \text{ si } (s, w) \vdash_M^* (q, \epsilon) \wedge q \in F$$

Le langage accepté par M (noté L(M)) est défini par l'ensemble des mots acceptés par M :

$$L(M) = \{w \in \mathcal{A}^* \mid (s, w) \vdash_M^* (q, \epsilon) \wedge q \in F\}$$

Exemple d'acceptation



Les mots suivants sont-ils acceptés ?

1. *aabab* : oui
   $\hookrightarrow (q_0, aabab) \vdash (q_0, abab) \vdash (q_0, bab) \vdash (q_1, ab) \vdash (q_0, b) \vdash (q_1, \epsilon)$
2. *aababa* : non
   $\hookrightarrow (q_0, aababa) \vdash (q_0, ababa) \vdash (q_0, baba) \vdash (q_1, aba) \vdash (q_0, ba) \vdash (q_1, a) \vdash (q_0, \epsilon)$ et $q_0 \notin F$

Simulation du comportement d'un AFD

Fonction **simulationAFD(**Q**,** $\mathcal{A}$**,** δ**,** s**,** F**,** w**)** : booléen

etat $\leftarrow$ s
position $\leftarrow$ 0
erreur $\leftarrow$ *faux*
Tant que !erreur $\wedge$ position $<$ $|w|$ **Faire**
  Si w[position] $\notin$ $\mathcal{A}$ **Alors**
    erreur $\leftarrow$ *vrai*
  Sinon
    Si $\exists$ $\delta(etat, w[position])$ **Alors**
      etat $\leftarrow$ $\delta(etat, w[position])$
      position $\leftarrow$ position + 1
    Sinon
      erreur $\leftarrow$ *vrai*
retourner $\leftarrow$!erreur $\wedge$ etat $\in$ F

Les automates finis non déterministes

Automates où les constructions suivantes sont autorisées :

- Plusieurs transitions sur le même symbole partant d'un même état
- Transitions sur le mot vide acceptées
- Transitions sur des mots de longueur supérieure à 1 sont possibles
  $\hookrightarrow$ Regroupement de transitions

Remarque

Les automates finis non déterministes sont généralement plus faciles à écrire que les AFD mais moins rapides à simuler.

Automate fini non déterministe

Définition : automate fini non déterministe

Un automate fini non déterministe (noté AFN) est défini formellement par le quintuplet $M = \{Q, \mathcal{A}, \Delta, s, F\}$, où :

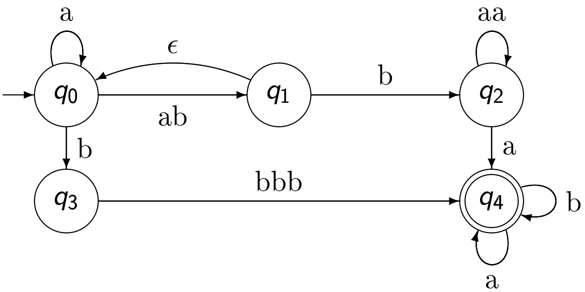
- *Q est un ensemble fini d'états*
- *$\mathcal{A}$ est un alphabet*
- *Δ est la fonction de transition de $Q \times \mathcal{A}^*$ dans Q*
- *$s \in Q$ est l'état initial*
- *$F \subseteq Q$ est l'ensemble des états d'acceptation*

Exemple d'AFN

Soit l'automate $M = \{Q, \mathcal{A}, \Delta, s, F\}$ suivant :
$\mathcal{A} = \{a, b\}$, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $s = q_0$, $F = \{q_4\}$

$\Delta =$

q	u	$\Delta(q,u)$	q	u	$\Delta(q,u)$	q	u	$\Delta(q,u)$
q_0	a	q_0	q_0	ab	q_1	q_0	b	q_3
q_1	ϵ	q_0	q_1	b	q_2	q_2	aa	q_2
q_2	a	q_4	q_3	bbb	q_4	q_4	a	q_4
q_4	b	q_4						



Dérivation d'un AFN

Définition : dérivation (en une étape)

La configuration (q', w') est dérivable en une étape de (q, w) par M si :

- $w = uw'$ $(u \in \mathcal{A}^*)$
- $(q, u, q') \in \Delta$

- Acceptation d'un mot par un AFN : il existe au moins une dérivation qui accepte ce mot

  Des dérivations possibles peuvent le refuser !

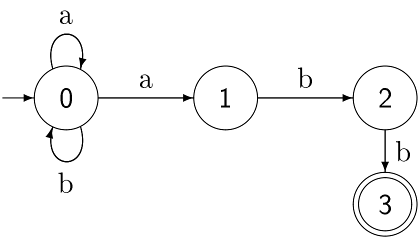
- Construction plus simple que les AFD mais sans forcément reconnaître plus de langages

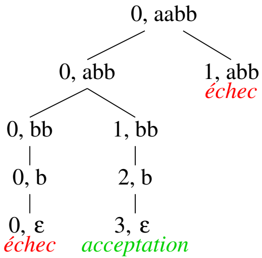
Simulation d'un AFN (1/2)

Problème : plusieurs dérivations possibles pour un même mot !

- Vérifier si l'une d'elles permet d'accepter le mot
- Être en mesure de revenir en arrière (*backtrack*)

$\hookrightarrow$ Utilisation de la récursivité





Simulation d'un AFN (2/2)

Fonction **simulationAFN(M, e, w)** : booléen

Si $|w| = 0$ **Alors**

retourner $e \in M(F)$

Sinon

accepte $\leftarrow$ *faux*

couples $\leftarrow \{(e', w_1)/\exists(e, w_2, e') \in M(\Delta) \wedge w = w_2 w_1\}$

Tant que couples $\neq \emptyset \wedge$!accepte **Faire**

choisir $(e', w_1) \in$ couples

couples $\leftarrow$ couples $\setminus (e', w_1)$

accepte $\leftarrow$ simulationAFN(M, e', w_1)

retourner accepte

M est l'automate, e un état et w le mot à reconnaître

Fonction exécutée avec $e = s$

Il est possible d'avoir une exécution infinie ! ! !

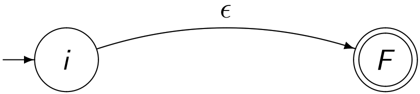
Construction de Thomson

- Objectif : construire de manière automatique un AFN à partir de toute expression régulière
- Idée :
  - Décomposition de l'expression régulière en sous-expressions
  - Construction d'un AFN pour chaque sous-expression
  - Combinaison des AFN

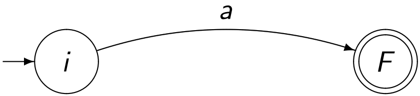
Il existe d'autres méthodes, mais c'est celle que nous utiliserons en INFO0602

Règles de base

- **Règle n°1** : pour l'expression ϵ, nous construisons l'AFN suivant :



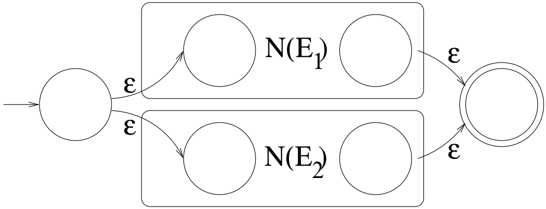
- **Règle n°2** : pour l'expression $a \in \mathcal{A}$, nous construisons l'AFN suivant :



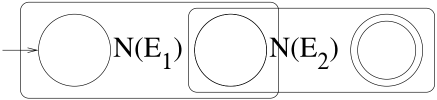
Combinaisons (1/2)

Supposons que $N(E_1)$ et $N(E_2)$ sont des AFN pour les expressions régulières E_1 et E_2.

- **Combinaison n°1** : pour l'expression $E_1|E_2$

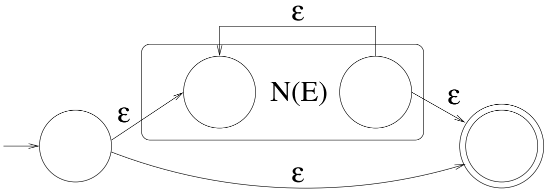


- **Combinaison n°2** : pour l'expression $E_1.E_2$



Combinaisons (2/2)

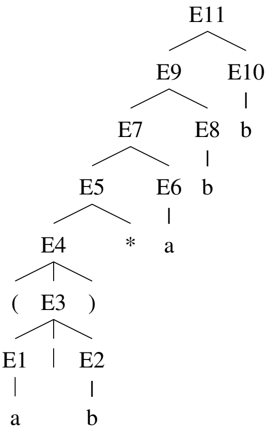
- **Combinaison n°3** : pour l'expression E^*



- **Combinaison n°4** : pour l'expression (E), c'est N(E) lui-même

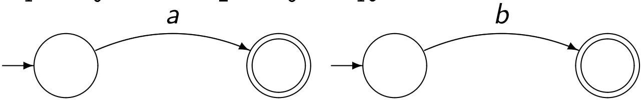
Exemple (1/4)

- Supposons l'expression régulière $(a|b)^*abb$
- Première étape : construire l'arbre syntaxique
  $\hookrightarrow$ Construction des AFN pour chaque expression de l'arbre

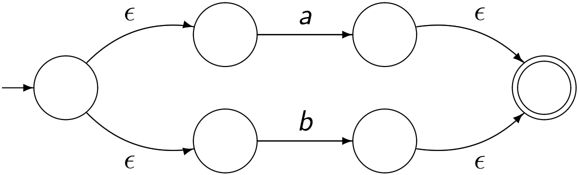


Exemple (2/4)

- Pour $E_1 = E_6 = a$ et $E_2 = E_8 = E_{10} = b$:

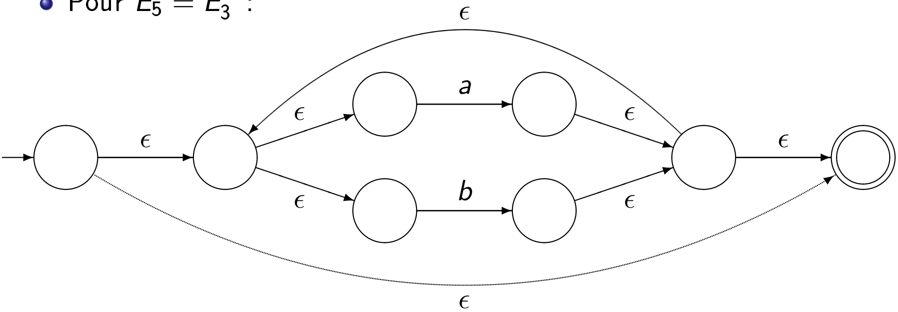


- Pour $E_3 = E_1 | E_2$:



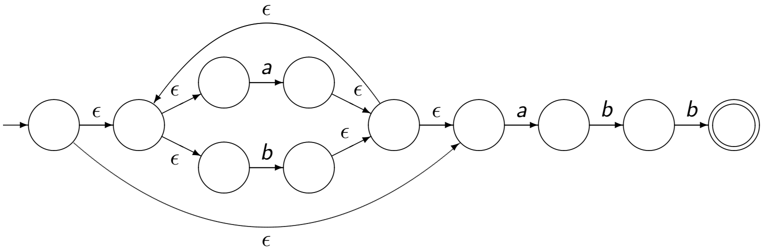
Exemple (3/4)

- Pour $E_4 = (E_3)$, c'est E_3 lui-même
- Pour $E_5 = E_3^*$:



Exemple (4/4)

- Pour finir en combinant avec E_7, E_9 et E_{11}

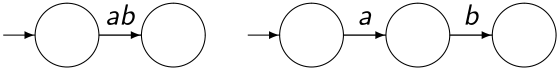


Équivalence entre un AFN et un AFD

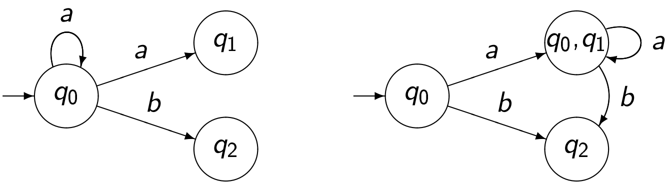
Définition : équivalence entre un AFN et un AFD

Un AFD est équivalent à un AFN s'il accepte le même langage.

- Exemple 1 :



- Exemple 2 :



Définition d'opérations sur un AFN

Définition : ϵ-fermeture d'un état

Soit l'AFN $M = \{Q, \mathcal{A}, \Delta, s, F\}$ et $e \in Q$, ϵ-fermeture(e) est l'ensemble des états de M accessibles depuis e par des ϵ-transitions uniquement. $e \in \epsilon$-fermeture(e).

Définition : ϵ-fermeture d'un ensemble d'états

Soit l'AFN $M = \{Q, \mathcal{A}, \Delta, s, F\}$ et $T \subseteq Q$, ϵ-fermeture(T) est l'ensemble des états de M accessibles depuis tout $e \in T$ par des ϵ-transitions uniquement. $T \subseteq \epsilon$-fermeture(T).

Définition : transiter

Soit l'AFN $M = \{Q, \mathcal{A}, \Delta, s, F\}$, $T \subseteq Q$ et $a \in \mathcal{A}$, transiter(T,a) est l'ensemble des états de M tels qu'il existe une transition sur a à partir d'un $e \in T$. Si transiter(T,a)=T' alors on note $T \xrightarrow{a} T'$

Principes de la transformation

- Soit l'AFN $M = \{Q, \mathcal{A}, \Delta, s, F\}$
- Construction de l'AFD $M' = \{Q', \mathcal{A}, \delta, s', F'\}$
- Chaque état de l'AFD = ensemble d'états de l'AFN
- Construction d'un ensemble V (états à visiter) :
  $\hookrightarrow$ Au départ, $V = \{$ ϵ-fermeture(s) $\}$
- Pour chaque état q de V :
  $\hookrightarrow$ Ajout de q à Q' et retrait de q de V
  $\hookrightarrow$ Pour chaque $a \in \mathcal{A}$, calcul de $q' = \epsilon$-fermeture(transiter(q,a))
  $\hookrightarrow$ Ajout de q' dans V si $q' \notin Q'$
- F est formé par tous les états de M' contenant un $q \in F$

Algorithme de transformation AFN en AFD

Fonction **AFNversAFD(M)** : M'

  $s' \leftarrow \epsilon$-fermeture(s)

  $Q' \leftarrow \{s'\}; \delta = \emptyset; V \leftarrow \{s'\}$

  Tant que $\exists x \in V$ **Faire**

    $V \leftarrow V \backslash x$

    Pour tout $a \in \mathcal{A}$ **Faire**

      $y \leftarrow \epsilon$-fermeture(transiter(x, a))

      Si $y \notin Q'$ **Alors**

        $Q' \leftarrow Q' \cup y$

        $V \leftarrow V \cup y$

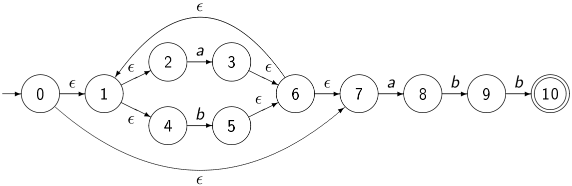
      $\delta \leftarrow \delta \cup \{(x, y, a)\}$

    Fin Pour

  $F' \leftarrow \{x \in Q' \mid \exists f \in x \wedge f \in F\}$

Exemple (1/2)

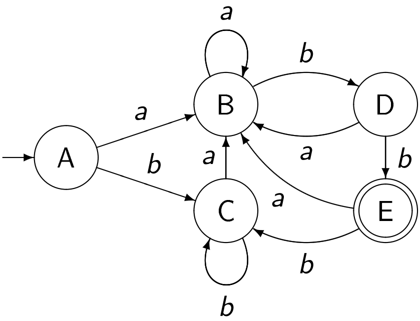
$M((a|b)^*abb)$:



x	transiter	y	δ
s'=A={0,1,2,4,7}	$A \xrightarrow{a} \{3,8\}$	{3,8,6,1,7,2,4} = B	$A \xrightarrow{a} B$
	$A \xrightarrow{b} \{5\}$	{5,6,7,1,2,4} = C	$A \xrightarrow{b} C$
B={1,2,3,4,6,7,8}	$B \xrightarrow{a} \{3,8\}$		$B \xrightarrow{a} B$
	$B \xrightarrow{b} \{5,9\}$	{5,9,6,7,1,2,4}=D	$B \xrightarrow{b} D$
C={1,2,4,5,6,7}	$C \xrightarrow{a} \{3,8\}$		$C \xrightarrow{a} B$
	$C \xrightarrow{b} \{5\}$		$C \xrightarrow{b} C$
D={1,2,4,5,6,7,9}	$D \xrightarrow{a} \{3,8\}$		$D \xrightarrow{a} B$
	$D \xrightarrow{b} \{5,10\}$	{5,6,1,2,4,7,10}=E	$D \xrightarrow{b} E$
E={1,2,4,5,6,7,10}	$E \xrightarrow{a} \{3,8\}$		$E \xrightarrow{a} B$
	$E \xrightarrow{b} \{5\}$		$E \xrightarrow{b} C$

Exemple (2/2)

Sachant que seul E contient un état d'acceptation, $E \in Q'$. On obtient :



Partitionnement

- Objectif : réduire le nombre d'états d'un AFD
- Idée : décomposition de Q en sous-ensembles d'état équivalents
  $\hookrightarrow$ Partitionnement
- Les états sont équivalents si $\forall a \in \mathcal{A}$, les états ont des transitions vers des états des mêmes sous-ensembles
- Algorithme :
  1. Au départ, on sépare en deux sous-ensembles : les états d'acceptation et les autres
  2. Puis séparation des sous-ensembles en sous-ensembles d'états équivalents
  3. On recommence à l'étape 2
  4. Dès qu'il n'y a plus de création de nouveaux sous-ensembles, la procédure s'arrête

Définitions

Définition : partitionnement

$\pi = \{B_1, \ldots, B_k\}$ avec B_i un ensemble d'états de Q, est une partition de Q si :

- $\forall i, B_i \subseteq Q$
- $\forall i, B_i \neq \emptyset \wedge \forall i, j, i \neq j, B_i \cap B_j = \emptyset$
- $\cup_{i=1}^{k} B_i = Q$

Définition : états équivalents

Avec $(s, t) \in Q^2$, s et t sont équivalents dans π, si et seulement si :
$\forall a \in \mathcal{A}, (s', t') \in Q^2, (s, s', a) \in \delta \wedge (t, t', a) \in \delta$
$\Rightarrow \exists B \in \pi, s' \in B \wedge t' \in B$
On note $s \Leftrightarrow_\pi t$.

Algorithme de minimisation des états

Procédure **minimisation**(M)

$\pi = \{Q \backslash F, F\}$; $change \leftarrow vrai$

Tant que change = vrai **Faire**

$change \leftarrow faux$; $\pi' \leftarrow \emptyset$

Pour tout $B \in \pi$ **Faire**

Tant que $\exists s \in B$ **Faire**

$B \leftarrow B \backslash \{s\}$; $B' \leftarrow \{s\}$

$\pi' \leftarrow \pi' \cup B'$

Tant que $\exists s' \in B | s' \Leftrightarrow_\pi s$ **Faire**

$B \leftarrow B \backslash \{s'\}$; $B' \leftarrow B' \cup \{s'\}$

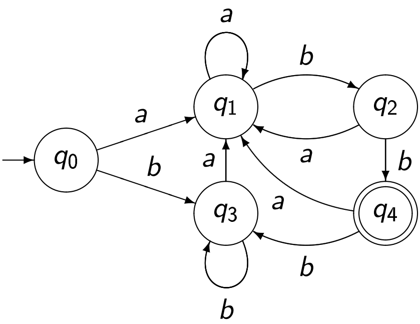
Si $B \neq \emptyset$ **Alors**

$change \leftarrow vrai$

Fin Pour

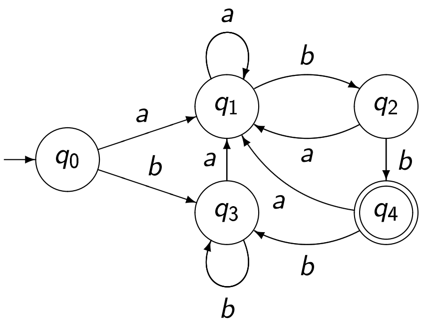
$\pi \leftarrow \pi'$

Exemple (1/4)



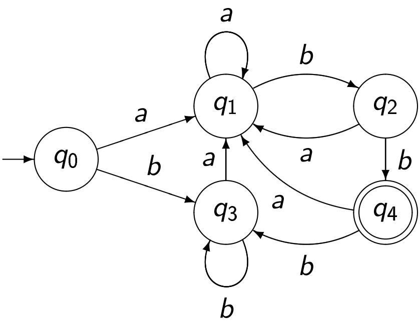
- Étape 1 : $\pi = \{\{q_0, q_1, q_2, q_3\}, \{q_4\}\}$
  - $B = \{q_0, q_1, q_2, q_3\}$
    - $q_0 \xrightarrow{a} q_1$ $q_1 \xrightarrow{a} q_1$ $q_2 \xrightarrow{a} q_1$ $q_3 \xrightarrow{a} q_1$
      $q_0 \xrightarrow{b} q_3$ $q_1 \xrightarrow{b} q_2$ $q_2 \xrightarrow{b} q_4$ $q_3 \xrightarrow{b} q_3$
      $\hookrightarrow \{q_0, q_1, q_3\}$ et $\{q_2\}$
  - $B = \{q_4\}$
    $\hookrightarrow \pi' = \{\{q_0, q_1, q_3\}, \{q_2\}, \{q_4\}\}$: changements donc on continue

Exemple (2/4)



- Étape 2 : $\pi = \{\{q_0, q_1, q_3\}, \{q_2\}, \{q_4\}\}$
  - $B = \{q_0, q_1, q_3\}$
    - $q_0 \xrightarrow{a} q_1$ $q_1 \xrightarrow{a} q_1$ $q_3 \xrightarrow{a} q_1$
      $q_0 \xrightarrow{b} q_3$ $q_1 \xrightarrow{b} q_2$ $q_3 \xrightarrow{b} q_3$
      $\hookrightarrow \{q_0, q_3\}$ et $\{q_1\}$
  - $B = \{q_2\}$ puis $B = \{q_4\}$
    $\hookrightarrow \pi' = \{\{q_0, q_3\}, \{q_1\}, \{q_2\}, \{q_4\}\}$: changements donc on continue

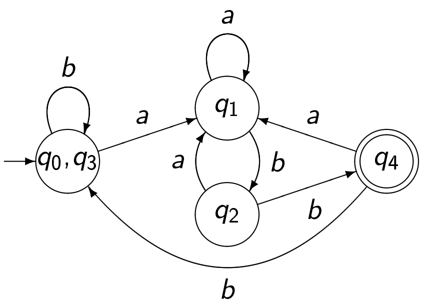
Exemple (3/4)



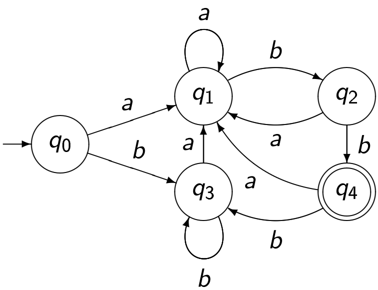
- Étape 3 : $\pi = \{\{q_0, q_3\}, \{q_1\}, \{q_2\}, \{q_4\}\}$
    - $B = \{q_0, q_3\}$
        - $q_0 \xrightarrow{a} q_1 \;\; q_3 \xrightarrow{a} q_1$
          $q_0 \xrightarrow{b} q_3 \;\; q_3 \xrightarrow{b} q_3$
          $\hookrightarrow \{q_0, q_3\}$
    - $B = \{q_1\}$ puis $B = \{q_2\}$ puis $B = \{q_4\}$
      $\hookrightarrow \pi' = \{\{q_0, q_3\}, \{q_1\}, \{q_2\}, \{q_4\}\}$: pas de changement donc on arrête

Exemple (4/4)

- Avec $\pi = \{\{q_0, q_3\}, \{q_1\}, \{q_2\}, \{q_4\}\}$ et
  $q_0 \xrightarrow{a} q_1$ $q_1 \xrightarrow{a} q_1$ $q_2 \xrightarrow{a} q_1$ $q_3 \xrightarrow{a} q_1$ $q_4 \xrightarrow{a} q_1$
  $q_0 \xrightarrow{b} q_3$ $q_1 \xrightarrow{b} q_2$ $q_2 \xrightarrow{b} q_4$ $q_3 \xrightarrow{b} q_3$ $q_4 \xrightarrow{b} q_3$
- On obtient l'automate suivant :



Représentation avec un tableau



	q_0	q_1	q_2	q_3	q_4
π'	A	A	A	A	B
$\xrightarrow{a}$	A	A	A	A	A
$\xrightarrow{b}$	A	A	B	A	A
π'	A	A	B	A	C
$\xrightarrow{a}$	A	A	A	A	A
$\xrightarrow{b}$	A	B	C	A	A
π'	A	B	C	A	D
$\xrightarrow{a}$	B	B	B	B	B
$\xrightarrow{b}$	A	C	D	A	A
π'	A	B	C	A	D

De l'expression régulière à l'automate

- Objectif : à partir d'un automate, retrouver l'expression régulière associée
- Plusieurs techniques existantes :
  - Technique basée sur les systèmes d'équation
    $\hookrightarrow$ L'automate est représenté sous la forme d'un système linéaire
  - Technique basée sur l'élimination d'états
    $\hookrightarrow$ Les états sont supprimés au fur-et-à-mesure, remplacés par de nouvelles transitions équivalentes
- Nous allons utiliser la deuxième technique qui est plus intuitive
  $\hookrightarrow$ Les expressions régulières générées sont cependant plus longues

Automate normalisé

Définition : automate normalisé

Un automate $M = \{Q, \mathcal{A}, \Delta, s, F\}$ *est normalisé si :*

- $\Delta \cap (Q \times \mathcal{A} \times s) = \emptyset$
- $|F| = 1 \wedge (\Delta \cap (F \times \mathcal{A} \times Q)) = \emptyset$

On dit également que M *est généralisé.*

- Un automate normalisé ne possède pas de transition entrante sur son état initial
- Un automate normalisé possède un seul état final sans transition sortante

Normalisation d'un automate (1/2)

Définition : normalisation pour l'initialisation

Soit l'automate $M = \{Q, \mathcal{A}, \Delta, s, F\}$.
Si $\exists q \in Q \wedge a \in \mathcal{A} / (q, a, s) \in \Delta$,
$M' = \{Q \cup \{s'\}, \mathcal{A}, \Delta \cup \{(s', \epsilon, s)\}, s', F\}$.
Sinon $M' = M$.
M' *est la version normalisée pour l'initialisation de* M.

Définition : normalisation pour la terminaison

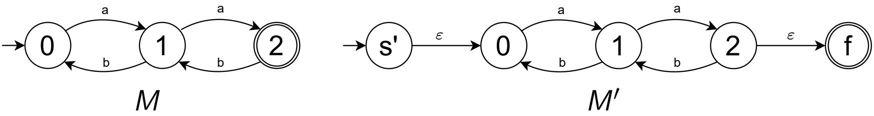
Soit l'automate $M = \{Q, \mathcal{A}, \Delta, s, F\}$.
Si $|F| > 1$ *ou si* $\exists q \in F \wedge q' \in Q \wedge a \in \mathcal{A} / (q, a, q') \in \Delta$,
$M' = \{Q \cup \{f\}, \mathcal{A}, \Delta \cup \{(q, \epsilon, f) / q \in F\}, s, \{f\}\}$.
Sinon $M' = M$.
M' *est la version normalisée pour la terminaison de* M.

Normalisation d'un automate (2/2)

Définition : normalisation d'un automate

La normalisation d'un automate M consiste à lui appliquer successivement les normalisations pour l'initialisation et pour la terminaison. Le résultat M' est la version normalisée de l'automate.

Exemple



Prédécesseur et successeur

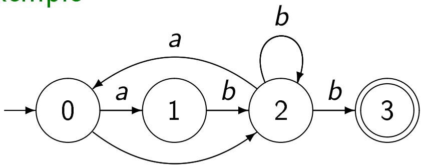
Définition : prédécesseur d'un état

Un prédécesseur d'un état q est un état $q' \in Q \backslash \{q\}$ qui possède une transition vers q. pred(q) est l'ensemble des prédécesseurs de q.

Définition : successeur d'un état

Un successeur d'un état q est un état $q' \in Q \backslash \{q\}$ tel qu'il existe une transition de q vers cet état q'. succ(q) est l'ensemble des successeurs de q.

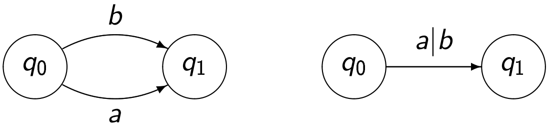
Exemple



- $pred(2) = \{0, 1\}$
- $succ(2) = \{0, 3\}$

Réduction de transitions

- Soit deux transitions (p, r, q) et (p, s, q)
- On peut les remplacer par la transition $(p, r|s, q)$



Méthode d'élimination des états

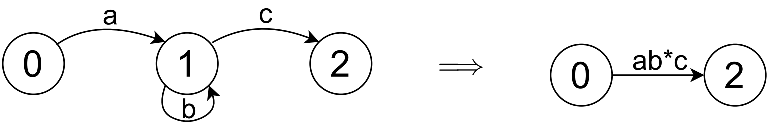
- Appelée la méthode de Brzozowski et McCluskey
- L'automate doit être normalisé
- Pour supprimer un état q et pour chaque couple (q_1, q_2) avec $q_1 \in pred(q) \wedge q_2 \in succ(p)$, on remplace les transitions (q_1, r, q), (q, s, q) et (q, t, q_2) par :
  - (q_1, rs^*t, q_2) si $\exists (q, s, q) \in \Delta$
  - (q_1, rt, q_2) sinon
- On réduit les transitions
- On procède de même pour tous les états de $Q \backslash \{s, f\}$
  $\hookrightarrow$ À la fin, il n'y a plus que deux états et une seule transition
  $\hookrightarrow$ L'étiquette de la transition est l'expression régulière correspondant à l'automate

Le changement de l'ordre d'élimination des états peut donner des expressions régulières différentes, mais équivalentes

Exemples

- Suppression de l'état 1 qui ne possède pas de transition sur lui-même :

- Suppression de l'état 1 qui possède une transition sur lui-même :



Construction d'un analyseur lexical

- Soit un analyseur lexical qui accepte n unités lexicales :
  - Représenter chaque unité lexicale par une expression régulière
  - Construire un AFN pour chaque expression
  - Transformation des AFN en AFD
  - Réduction des états
- Algorithme général de l'analyseur :
  - Test sur le premier AFD
  - Si non accepté, alors on teste le second
  - etc.
- En pratique, recherche du plus grand lexème reconnu

L'ordre de définition des unités lexicales est important ; si un lexème est reconnu par deux lexèmes, le lexème défini en premier est choisi.

Introduction aux automates à pile

- Tout langage ne peut être défini par une expression régulière
  $\hookrightarrow$ *Exemple* : $L = \{w \mid w = a^n b^n, n \geq 0\}$
- Possible d'utiliser un automate à pile qui est constitué des mêmes éléments qu'un AFN :
  - Un ruban d'entrée et une tête de lecture
  - Un ensemble d'états avec un état initial et un ensemble d'état d'acceptation
  - Une relation de transition
- Ajout d'une pile de capacité infinie, initialement vide
- À chaque étape, le sommet de la pile est consulté et remplacé par une suite de symboles
  $\hookrightarrow$ La fonction de transition dépend également du sommet de la pile

Automate fini non déterministe à pile

Définition : automate fini non déterministe à pile (AFNP)

Un automate fini non déterministe à pile est défini par :

$$M = \{Q, \mathcal{A}, \mathcal{B}, \Delta, z, s, F\}$$

Où :

- Q *est un ensemble fini d'états*
- $\mathcal{A}$ *est l'alphabet*
- $\mathcal{B}$ *est l'alphabet de la pile*
- Δ *est la relation de transition :*
  $\Delta \subset ((Q \times \mathcal{A}^* \times \mathcal{B}^*) \times (Q \times \mathcal{B}^*))$
- $z \in \mathcal{B}$ *est le symbole initial de la pile*
- $s \in Q$ *est l'état initial*
- $F \subseteq Q$ *est l'ensemble des états d'acceptation*

La fonction de transition

Soit la transition $((p, u, v), (q, v')) \in \Delta \subset ((Q \times \mathcal{A}^* \times \mathcal{B}^*) \times (Q \times \mathcal{B}^*))$

- L'automate passe de p à q sur le mot u si la chaîne v est au sommet de la pile
- À l'état q, le sommet de la pile v est remplacé par v'

Configuration

Définition : configuration

La configuration d'un AFNP est définie par un triplet :

$$(q, a, a') \in Q \times \mathcal{A}^* \times \mathcal{B}^*$$

- *q est l'état courant*
- *a est le mot en entrée*
- *a' est le contenu de la pile*

Dérivation en une étape

Définition : dérivation (en une étape)

La configuration $(q', b, b') \in Q \times \mathcal{A}^* \times \mathcal{B}^*$ *est dérivable en une étape de* (q, a, a') *par* $(q, a, a') \vdash_M (q', b, b')$ *si :*

- $((q, u, v), (q', v')) \in \Delta$
- $a = ub$ *: le mot* a *commence par le préfixe* u
- $a' = vw$ *: avant la transition, le sommet de la pile contient* $v \in \mathcal{B}^*$
- $b' = v'w$ *: après la transition,* v *est remplacé par* v'

Dérivation en plusieurs étapes

Définition : dérivation (en plusieurs étapes)

La configuration (q', b, b') *est dérivable en plusieurs étapes de la configuration* (q, a, a') *s'il existe des configurations intermédiaires* $C_0, \ldots, C_k$ *avec* $k \geq 0$ *telles que :*

- $C_0 = (q, a, a')$
- $C_k = (q', b, b')$
- $C_i \vdash_M C_{i+1}$ *pour* $0 \leq i \leq k$

- Une exécution d'un AFNP sur un mot w est une suite de configurations :

$$(s, w, z) \vdash (q_1, w_1, \alpha_1) \vdash \ldots \vdash (q_n, \epsilon, \alpha_n)$$

Acceptation d'un mot par un automate à pile

Soit un AFNP $M = \{Q, \mathcal{A}, \mathcal{B}, \Delta, z, s, F\}$

- **Acceptation sur état d'acceptation**. w est accepté par M si

$$(s, w, z) \vdash_M^* (q, \epsilon, \alpha) \text{ avec } q \in F$$

- **Acceptation sur pile vide**. w est accepté par M si :

$$(s, w, z) \vdash_M^* (q, \epsilon, \epsilon)$$

Remarques

- Si l'automate accepte sur pile vide, F est inutile
- Les deux définitions sont équivalentes ; les deux types d'automates reconnaissent les mêmes langages.

Exemple

- Soit le langage $L = \{a^n b^n, n \geq 0\}$. Définition de M qui reconnaît L sur pile vide :
  - $Q = \{q_0, q_1\}$
  - $\mathcal{A} = \{a, b\}$, $\mathcal{B} = \{a\}$
  - $z = \epsilon$, $s = q_0$
  - Δ : $\begin{array}{c} (q_0, a, \epsilon), (q_0, a) \\ (q_0, a, a), (q_0, aa) \\ (q_0, b, a), (q_1, \epsilon) \\ (q_1, b, a), (q_1, \epsilon) \end{array}$
- Exécution pour $w = aaabbb$:
  $(q_0, aaabbb, \epsilon) \vdash (q_0, aabbb, a) \vdash (q_0, abbb, aa) \vdash (q_0, bbb, aaa) \vdash (q_1, bb, aa) \vdash (q_1, b, a) \vdash (q_1, \epsilon, \epsilon)$
  $\hookrightarrow$ Le mot w est accepté par M