

Transparents du cours

Langages formels

HLIN502

Licence 3e année - premier semestre

2017 – 2018

# Langages formels

HLIN502

Licence 3e année - premier semestre

Cours : Rémi Legrand

TD : Hervé Dicky

Michel Meynard

Rémi Legrand

## Organisation du cours

- 11 cours
  - Le mardi de 15h à 16h30 en A6.03
  - Distribution transparents de cours, feuilles de TD
- 22 TD , pas de TP accompagnés
  - Le jeudi de 9h45 à 13h (deux TD)
  - Début le 21 septembre
  - Répartition dans les 3 groupes de TD : A, B et C
- Site : l'ENT - moodle
  - HLIN502 (mot de passe : lang17)
    - Polycopié de cours, feuilles de TP

## Contrôle des connaissances

- 1 examen terminal avec une session de rattrapage
  - Durée 2 heures
  - Documents autorisés :
    - juste une feuille A4 recto-verso personnelle
  - Annales corrigées sur le site ENT
- Pas de contrôle continu

## Objectifs

- **Fondements** de l'informatique : les **langages**
  - Langages permettant de spécifier les problèmes de l'informatique, afin de les résoudre « scientifiquement »
  - Description des langages et traitement des données
    - Grammaires, Automates, Expressions rationnelles
- Applications :
  - Savoir **modéliser** :
    - **Abstraire** = éliminer le bruitage et lever les imprécisions
    - **Formaliser** = afin de pouvoir établir des propriétés
  - Maîtriser les outils et propriétés de ces langages
    - Savoir faire des **raisonnements par induction**

# Plan du cours

- Mots et langages
- Grammaires
- Automates
  - Déterministe, indéterministe, avec  $\varepsilon$ -transitions
- Transformation d'automates
  - Élimination des  $\varepsilon$ -transitions, déterminisation, Minimisation
- Expressions rationnelles
- Classification des langages
  - Grammaires régulières, lemme de la Pompe, automates à pile

## Mots et langages

## Définitions

Analogie avec les langues naturelles

- **Alphabet** : un ensemble fini, souvent noté  $\Sigma$  (sigma) d'éléments appelés des lettres.
  - Exemples :  $\{a,b,c,\dots,z\}$ ,  $\{a,b\}$ ,  $\{0,1\}$
- **Mot** : une «suite» ordonnée et finie de lettres
  - Exemples : maison, aabbaba, 11100
- **Langage** : un ensemble de mots
  - Exemples : Les-mots-français  $\{\underbrace{a\dots a}_{n \text{ fois}}\}$   $\{11,01\}$

## Que faire des langages ?

- Langages informatiques : Programme = mot
  - Compilation, analyse lexicale et syntaxique
  - Transformation de programmes
- Algorithmes
  - Recherche d'un motif dans un texte
  - Analyse  $\pm$  intelligente de textes
- Démarches générales
  - Classement des langages
  - Étude des propriétés des langages
  - Travail sur des langages abstraits comme  $\{a, b\}^*$

## Les mots

- Mot = suite finie de lettres (élément de  $\Sigma$ )
  - Cas particulier du mot vide : noté  $\varepsilon$  (epsilon)
    - Suite vide de lettres
    - $\varepsilon$  n'est pas une lettre de l'alphabet  $\Sigma$
  - Longueur d'un mot = nombre de lettres du mot
    - Notation :  $|m|$  nombre de lettres du mot  $m$   
 $|m|_a$  nombre d'occurrences de  $a$  dans  $m$
- Exemples :  $|bbababba|_a = 3$
- $|m| = 0$  si et seulement si  $m = \varepsilon$   
 $|m| = 1$  si et seulement si  $m$  est une lettre
- La lettre  $\alpha$  est confondu avec le mot constitué d'une lettre  $\alpha$

## Composants d'un mot

- Notation :  
 $m[i..j]$  est le mot extrait de  $m$  en ne conservant que les lettres des positions  $i$  à  $j$  (inclus)  
 Exemple :  $m = abcdefgh$   
 $m[2..4] = bcd$        $m[1..|m|] = m$
- $m[i..j]$  est appelé un **facteur** de  $m$
- Notation :  
 $m[i]$  est la  $i$ -ième lettre extraite du mot  $m$   
 Exemple :  $m = abcdefgh$        $m[3] = c$

## Langages

- Un langage sur l'alphabet  $\Sigma$  est un ensemble de mots construit avec l'alphabet  $\Sigma$
  - Le plus petit langage sur  $\Sigma$  :  $\{\}$  (aucun mot)
  - Le plus grand langage sur  $\Sigma$  :  $\Sigma^*$  (tous les mots)
- $\Sigma^*$  est appelé le monoïde libre engendré par  $\Sigma$
- Un langage intermédiaire :  
 $\{m \in \Sigma^*, |m|=1\}$  les mots n'ayant qu'une lettre

## Concaténation de mots

- **Concaténation** de deux mots  $u$  et  $v$  :  
 C'est le mot noté  $u.v$  (voire  $uv$ ) tel que :  
 a)  $|u.v| = |u| + |v|$   
 b)  $\forall i \in [1, |u|], u.v[i] = u[i]$   
 $\forall i \in [1, |v|], u.v[i+|u|] = v[i]$
- $u = \text{para}$      $v = \text{chute}$      $\Rightarrow u.v = \text{parachute}$   
 $\varepsilon . \text{chute} = \text{chute}$     et     $\text{para} . \varepsilon = \text{para}$
- $m . \varepsilon = \varepsilon . m = m$  et  $|m|=0 \Leftrightarrow m = \varepsilon$
- $|m_1 . m_2|_\alpha = |m_1|_\alpha + |m_2|_\alpha$      $|m_1 . m_2| = |m_1| + |m_2|$
- Notation puissance :  $m^n = \underbrace{m.m...m}_{n \text{ fois}}$      $m^0 = \varepsilon$

## Vive le monoïde libre

- $(\Sigma^*, .)$  est un monoïde
  - Interne
  - Associatif
  - Élément neutre :  $\varepsilon$
- Définition inductive de  $\Sigma^*$  :
  - a)  $\varepsilon \in \Sigma^*$
  - b) Si  $m \in \Sigma^*$  et  $a \in \Sigma$  alors  $a.m \in \Sigma^*$   
(« a » est le mot d'une lettre « a »)
- Notation :  $\Sigma^+ = \Sigma^* - \{\varepsilon\}$

Pas d'inverse :

$$m.m' = \varepsilon \Rightarrow m = m' = \varepsilon$$

## Langages non triviaux

- Un **langage** sur un alphabet  $\Sigma$  est une partie de  $\Sigma^*$ 
  - $A = \{ m, m \text{ commence par un 1 et finit par 0} \}$   $\Sigma = \{0,1\}$
  - $A = \{ m, |m|_a = |m|_b \}$   $\Sigma = \{a,b\}$
  - A est le plus petit ensemble contenant aa et bb et tel que si m est dans A, alors m.a.m et m.b.m aussi.
- La concaténation de mots dans un langage n'est pas une loi interne en général.
  - Exemple : Langage = { mots français }  
chien et chat sont des mots français  
mais chienchat n'est pas français

## Langage fermé pour la concaténation

- Un langage A est dit **fermé** pour la concaténation si
 
$$\forall m, m' \in A, m.m' \in A$$
- Si A fermé contient  $\varepsilon$ , alors  $(A, .)$  est un monoïde
- Un mot m de A est dit **premier** si :
 
$$\nexists u, v \in A \text{ tel que } m = u.v \text{ et } u \neq \varepsilon \text{ et } v \neq \varepsilon$$
- Exemples :
  - $A = \{ m \in \{a,b\}^*, |m|_a = |m|_b \}$  est fermé
    - m est premier si pas de préfixe propre ayant autant de a que de b
  - $A = \Sigma^*$  : les mots fermés de A sont les lettres (éléments de  $\Sigma$ )

## Opérations ensemblistes sur les langages

Soit  $L_1$  et  $L_2$  deux langages sur le même alphabet  $\Sigma$

- Union, intersection
  - $L_1 \cup L_2$  est l'union des ensembles  $L_1$  et  $L_2$ .
  - $L_1 \cap L_2$  est l'intersection des ensembles  $L_1$  et  $L_2$ .
  - Exemples :
 
$$\{a, b, aba, ba\} \cap \{a, aa, aaa, \dots\} = \{a\}$$

$$\{m, |m| \text{ pair}\} \cup \{m, |m| \text{ impair}\} = \Sigma^*$$
- Différence ensembliste, Complémentaire
  - $L_1 \setminus L_2 = \{m \in L_1, m \notin L_2\}$
  - $\overline{L_1} = \{m \in \Sigma^*, m \notin L_1\}$

## Concaténation de langages

Soit  $L_1$  et  $L_2$  deux langages sur le même alphabet  $\Sigma$

- Produit (concaténation)

- $L_1 \cdot L_2 = \{m_1 \cdot m_2, \quad m_1 \in L_1 \text{ et } m_2 \in L_2\}$

- Puissance

- $L^n = L \cdot L^{n-1}$  et  $L^0 = \{\varepsilon\}$  ( $L^1 = L \cdot L^0 = L$ )

- Exemples :

$$\{ab, ba\}^2 = \{abab, abba, baab, baba\}$$

Pour une lettre  $a$  de  $\Sigma$  :

$$\{a\} \cdot \Sigma^* \cdot \{a\} = \{m \in \Sigma^*, \quad m \text{ commence et finit par un } a\}$$

$$\{a\}^n = \{a^n\} \quad \{ \}^n = \{ \} \text{ si } n > 0 \quad \{ \}^0 = \{\varepsilon\}$$

## Fermeture de Kleene (étoile)

- Tous les mots obtenus en concaténant des mots de  $L$

- $L^* = \bigcup_{i \in \mathbb{N}} L^i = \{\varepsilon\} \cup L \cup L^2 \cup \dots \cup L^n \cup \dots$

- $L^+ = \bigcup_{i \in \mathbb{N} \setminus \{0\}} L^i = L \cup L^2 \cup \dots \cup L^n \cup \dots$

- Exemple :

$$\{ab, ba, aa, bb\}^* = \{m \in \Sigma^*, |m| \text{ est pair}\}$$

- Remarque :

Si  $\varepsilon$  est dans  $L$ , alors  $\varepsilon$  est aussi dans  $L^+$

## Propriétés des langages

- $(\{\text{Langages sur } \Sigma\}, \cdot)$  est un monoïde

- Si  $A \subseteq \Sigma^*$  et  $B \subseteq \Sigma^*$ , alors  $A \cdot B \subseteq \Sigma^*$  (interne)

- $(A \cdot B) \cdot C = A \cdot (B \cdot C)$  (associatif)

- $A \cdot \{\varepsilon\} = \{\varepsilon\} \cdot A = A$  ( $\{\varepsilon\}$  élément neutre)

- $A \subseteq B \implies A^n \subseteq B^n \quad A^* \cdot A = A \cdot A^* = A^+$

- $A \cdot \left( \bigcup_{i=1}^n B_i \right) = \bigcup_{i=1}^n (A \cdot B_i) \quad \left( \bigcup_{i=1}^n B_i \right) \cdot A = \bigcup_{i=1}^n (B_i \cdot A)$

- $m \in A_1 \cdot A_2 \dots A_n \Leftrightarrow \exists m_1 \in A_1, \dots, m_n \in A_n \text{ tel que } m = \prod_{i=1}^n m_i$
- $m \in A^* \Leftrightarrow m = \varepsilon \text{ ou } \exists m_1 \in A, \dots, m_n \in A \text{ tel que } m = \prod_{i=1}^n m_i$

## Facteurs et sous mots

- Le mot  $p$  est un **préfixe** du mot  $m$  s'il existe un mot  $r$  tel que  $m = p \cdot r$

- Le mot  $s$  est un **suffixe** du mot  $m$  s'il existe un mot  $r$  tel que  $m = r \cdot s$

- Le mot  $u$  est un **facteur** du mot  $m$  s'il existe un mot  $p$  et un mot  $s$  tel que  $m = p \cdot u \cdot s$

- Le mot  $a_1 a_2 \dots a_n$  est un **sous mot** du mot  $m$  s'il existe des mots  $m_0, m_1, \dots, m_n$  tel que :

$$m = m_0 \cdot a_1 \cdot m_1 \cdot a_2 \cdot m_2 \cdot \dots \cdot a_n \cdot m_n$$

- Exemple : Tout facteur est un sous mot

- Facteur (sous mot,...) **propre** : si plus petit et non vide

## Ordre sur les mots

- Choix arbitraire d'un ordre dans  $\Sigma$  :  $a <_{\Sigma} b <_{\Sigma} c \dots$
- **Ordre préfixe** :
  - $m <_p m'$  si  $m$  est un préfixe (suffixe) de  $m'$  et  $m \neq m'$
  - Pas un ordre total :  $ab$  et  $ba$  ne sont pas comparables
- **Ordre lexicographique** : l'ordre du dictionnaire
  - $\text{Loir} <_L \text{loire} <_L \text{loirs}$
  - $m <_L m'$  si  $\exists w, u' \in \Sigma^*$  et  $\exists v' \in \Sigma^+$  tel que  
 $m = w.u'$  et  $m' = w.v'$   
et  $u' = \varepsilon$  ou  $u'[1] <_{\Sigma} v'[1]$
  - Séquence infinie non complète :  $a <_L aa <_L aaa <_L \dots$

## Ordre sur les mots (2)

- **Ordre longueur lexicographique (hiérarchique)**
  - Regarder d'abord la longueur des mots puis l'ordre lexicographique
  - $\varepsilon <_H a <_H b <_H ab <_H ba <_H bb <_H aba <_H \dots$
  - $m <_H m'$  si  $|m| < |m'|$   
ou  
 $|m| = |m'|$  et  $m <_L m'$
- **Ordre sur les longueurs**
  - $m < m'$  si  $|m| < |m'|$
  - Souvent utilisé pour un raisonnement par induction

## Morphisme de langages

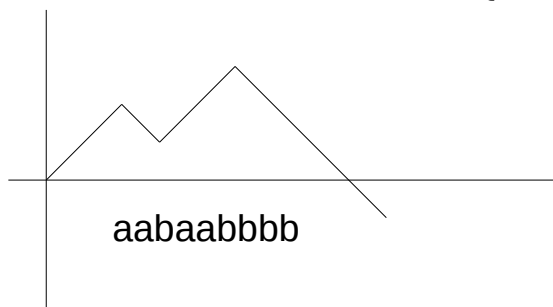
- $\Phi : \Sigma_1^* \rightarrow \Sigma_2^*$  est un **morphisme** si
  - a)  $\Phi(\varepsilon) = \varepsilon$
  - b)  $\forall m, m' \in \Sigma_1, \Phi(m.m') = \Phi(m).\Phi(m')$
- Exemple :
  - $\Sigma_1 = \{a, \dots, z\}$       $\Phi(x) = \text{code ascii de } x$   
ou  $\Phi(x) = \text{ascii}(x) + \text{code parité}$
- $\Phi$  est entièrement déterminé par sa restriction à  $\Sigma_1$
- Isomorphisme = morphisme + bijection

## Code

- $\Phi : \Sigma_1^* \rightarrow \Sigma_2^*$ 
  - $a \rightarrow 011$
  - $b \rightarrow 110$
  - $c \rightarrow 00$
  - $d \rightarrow 01$
  - $e \rightarrow 10$
- Trouver  $m$  tel que  $\Phi(m) = 01100110$  ?
- Non unicité :
  - $\Phi(acb) = 011\ 00\ 110$
  - $\Phi(dede) = 01\ 10\ 01\ 10$
  - $\Rightarrow$  On ne peut décoder
- $A = \{011, 110, 00, 01, 10\}$  n'est pas un **code** car il existe des mots de  $A^*$  qui admettent 2 décodages.
- $A$  est un code si  $\Phi$  est **injective**. Si  $A$  est un code,  $A^*$  est appelé le **monoïde libre** engendré par  $A$ .
- Pour  $\Sigma = \{a, b\}$ ,  $A = \Phi(\Sigma) = \{ab, abb\}$  est un code

## Dessiner les mots

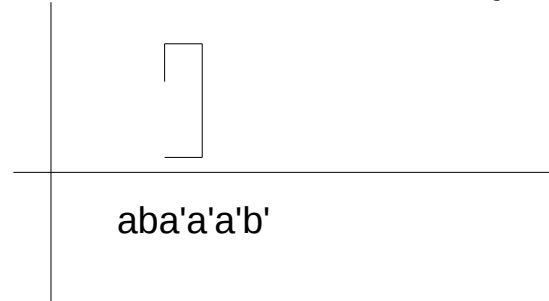
- Visualiser les mots si  $\Sigma = \{a,b\}$



- Visualiser  $|m|_a = |m|_b$
- Visualiser  $\forall i \in [1, |m|], |m[1..i]|_a \geq |m[1..i]|_b$

## Dessiner les mots (2)

- Visualiser les mots si  $\Sigma = \{a,b,a',b'\}$



- Inversement : un problème graphique est peut-être un problème qui se résout par la théorie des langages

## Résolution d'une équation

- Trouver 3 mots  $u, v$ , et  $w$  tel que  $u.v.w = w.u.v$ 
  - Solution simple :  $u = v = \varepsilon$
  - Autres solution ?  
 $\implies u = (w_1 w_2)^p w_1$  et  $v = w_2 (w_1 w_2)^q$  et  $w = (w_1 w_2)^r$
- Si  $u = \varepsilon$  ou  $v = \varepsilon$ , cas plus simple traité en TD
- Démonstration par un raisonnement par récurrence avec :  
 $\Pi(n) =$  La proposition est vrai si  $|u.v.w| \leq n$
- Beaucoup de cas à envisager selon les tailles relatives de  $u, v$  et  $w$

## Le « langage » français

- Approche 1
  - $\Sigma = \{a, b, c, \dots, z\}$
  - Langage = { mots référencés dans le dictionnaire français }
- Approche 2
  - $\Sigma = \{ \text{mots référencés dans le dictionnaire français} \}$ 
    - Exemple : "le.chat.mange.la.souris"
    - Les points de concaténation sont indispensables (ou les remplacer par des espaces)
  - Langage = { textes écrits en français }

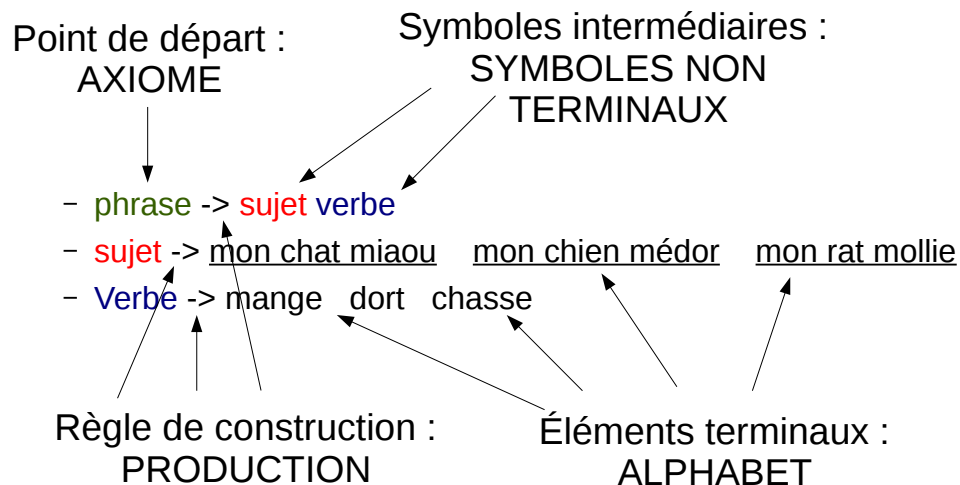


# Les grammaires

## Introduction

- Définition d'un langage en spécifiant ses règles de grammaire
  - Langage français très réduit à l'essentiel :
    - phrase -> sujet verbe
    - sujet -> mon chat miaou   mon chien médor   mon rat mollie
    - Verbe -> mange   dort   chasse
  - Mon chat miaou mange   est une phrase de ce français
- Langues naturelles :
  - Il faut une grammaire contextuelle
  - Exemple : tous les sujets ne chassent pas forcément

## Symboles et productions



## Grammaire non contextuelle

- $G = \langle \Sigma, X, P, S \rangle$  définit le langage  $L_G$ 
  - $\Sigma$  est l'**alphabet** des « lettres » des mots du langage  $L_G$ 
    - $\Sigma = \{ \text{mon chat miaou}, \text{mon chien médor}, \text{rené la taupe}, \text{mange}, \text{dort}, \text{chasse} \}$
    - $\Sigma$  est l'ensemble des « **symboles terminaux** »
  - $X$  est l'alphabet auxiliaire des symboles intermédiaires, i.e. des **symboles non terminaux**
    - $X = \{ \text{phrase}, \text{sujet}, \text{verbe} \}$  avec  $X \cap \Sigma = \{ \}$
  - $S$  est l'**axiome** à partir duquel on dérive les mots.
    - $S = \text{phrase}$  avec  $S \in X$
  - $P$  est l'ensemble des **productions** qui permettent de construire les mots de  $L_G$  à partir de  $S$

## Productions

- Production :
  - Notation :  $\alpha \rightarrow \beta$  où  $\alpha \in X$  et  $\beta \in (X \cup \Sigma)^*$
  - Définition :  
Une **production** est un élément de  $X \times (X \cup \Sigma)^*$
- Exemple :  $S \rightarrow aSb$      $S \rightarrow SS$      $S \rightarrow \varepsilon$   
 $X = \{S\}$      $P = \{(S, aSb), (S, SS), (S, \varepsilon)\}$   
 Discours : S se **réécrit** (se **dérive**) en aSb  
 Productions interdites :  $Sb \rightarrow aa$      $\varepsilon \rightarrow a$      $a \rightarrow b$
- $\alpha$  est un unique non terminal.
- $\beta$  est une succession de terminaux et non terminaux, ou juste le mot vide

## Fonctionnement des productions

- Principe :  
À partir de l'axiome S, il est engendré tous les mots de  $(X \cup \Sigma)^*$  constructibles en itérant l'utilisation des productions de P.
- $G = \langle \Sigma, X, P, S \rangle$  avec  $\Sigma = \{a, b\}$ ,  $X = \{S, T\}$   
 et  $P = \{S \rightarrow aST, S \rightarrow \varepsilon, T \rightarrow bb\}$   
 $S \rightarrow aST \rightarrow a aSTT \rightarrow aa aSTTT \rightarrow aaaTTT$   
 $\rightarrow aaaTbbT \rightarrow aaaTbbbb \rightarrow aaabbbbb$   
 $S \rightarrow aST \rightarrow aT \rightarrow abb$     autre possibilité
- Le langage  $L_G$  associé à G sera l'ensemble de tous les mots de  $\Sigma^*$  **générés** par ces opérations
- **Langage élargi**  $\widehat{L}_G$  : tous les termes dérivés de S

## Dérivations

- Dérivation élémentaire
  - $\alpha \rightarrow \beta \in P \implies m. \alpha . m' \rightarrow m. \beta . m'$   
est une **dérivation autorisée**
  - $\varepsilon$  est simplifié :  
 $\alpha \rightarrow \varepsilon \in P \implies m. \alpha \rightarrow m \quad \alpha . m' \rightarrow m'$
  - $m_1 \rightarrow m_2$  dérivation autorisée  $\implies m_2 \in (X \cup \Sigma)^*$
- **Chaîne de dérivations**
  - Suite de dérivations autorisées :  $m_0 \rightarrow m_1 \rightarrow \dots \rightarrow m_n$
  - Notations :  $m_0 \xrightarrow{*} m_n$      $m_0 \xrightarrow{n} m_n$      $m_0 \xrightarrow{\leq n} m_n$
  - n est appelé la **longueur** de la chaîne de dérivations
  - Cas limite :  $m \xrightarrow{*} m$  et  $m \xrightarrow{0} m$

## Langage associé à une grammaire

- Définition : Le langage, noté  $L_G$ , **associé à la grammaire**  $G = \langle \Sigma, X, P, S \rangle$  est défini sur l'alphabet  $\Sigma$  par :  
 $L_G = \{m \in \Sigma^* \mid S \xrightarrow{*} m\}$
- Définition : Le langage élargi, noté  $\widehat{L}_G$ , **associé à la grammaire**  $G = \langle \Sigma, X, P, S \rangle$  est défini sur l'alphabet  $\Sigma \cup X$  par :  
 $\widehat{L}_G = \{m \in (\Sigma \cup X)^* \mid S \xrightarrow{*} m\}$
- Un langage L est dit **algébrique** s'il existe une grammaire non contextuelle G telle que  $L = L_G$
- $G_1$  et  $G_2$  sont dites **équivalents** si  $L_{G_1} = L_{G_2}$

## Notations simplifiées

- $S \rightarrow \alpha \mid \beta$   
au lieu de :  $S \rightarrow \alpha$  et  $S \rightarrow \beta$
- $\xrightarrow{*}$  est parfois noté ponctuellement juste  $\rightarrow$
- $P = \{ \dots, S \rightarrow \beta, \dots \}$   
plutôt que  $P = \{ \dots, (S, \beta), \dots \}$
- $m.\varepsilon.m' = m \cdot m' = m m'$

## Notation BNF

- Notation « Backus-Naur Form »
- Souvent utilisé pour décrire les langages de programmation :
  - $\langle \text{conditionnelle} \rangle ::= \text{if} ( \langle \text{condition} \rangle ) \langle \text{instruction} \rangle ;$
  - $\langle \text{conditionnelle} \rangle ::= \text{if} ( \langle \text{condition} \rangle ) \{ \langle \text{instruction} \rangle ; \}$
- Non terminaux : entourés de chevron
- «  $\rightarrow$  » remplacée par  $::=$
- Avec des variantes et simplifications de notation
  - $\langle \text{cond} \rangle ::= \text{if} ( \langle \text{test} \rangle ) \langle \text{instr} \rangle ; [ \text{else} \langle \text{instr} \rangle ; ]$

## Exemple d'une grammaire

- $G = \langle \Sigma, X, P, S \rangle$ 
  - $\Sigma = \{ a, b \},$
  - $X = \{ S, T \}$
  - $P = \{ S \rightarrow aST, S \rightarrow \varepsilon, T \rightarrow bb \}$
- $L_G = ??? \quad \widehat{L}_G = ???$
- Analyse intuitive :
  - $L_G = \{ a^n b^{2n}, n \geq 0 \}$
  - $\widehat{L}_G = \{ a^n uv \mid n \geq 0 \text{ et } u \in \{ S, \varepsilon \} \text{ et } v \in \{ T, bb \}^n \}$
- Démonstration :
  - Nécessite le lemme fondamental
  - Nécessite un raisonnement par récurrence

## Lemme fondamental sur les dérivations

- Lemme :  
si  $u_1 u_2 \xrightarrow{k} v$  alors il existe  $v_1, v_2, k_1, k_2$  tel que  
 $k = k_1 + k_2, v = v_1 v_2, u_1 \xrightarrow{k_1} v_1$  et  $u_2 \xrightarrow{k_2} v_2$

$$\begin{array}{ccccc}
 u_1 u_2 & = & ( & ) & ( & ) \\
 \downarrow k = k_1 + k_2 & & \downarrow k_1 & & \downarrow k_2 \\
 v & = & v_1 v_2 & = & ( v_1 ) ( v_2 )
 \end{array}$$

- Toute dérivation s'applique soit à  $u_1$  soit à  $u_2$

## Extension du lemme fondamental

- $$u_1 u_2 \dots u_p = \left( \begin{array}{c} u_1 \\ \downarrow k_1 \\ v_1 \end{array} \right) \left( \begin{array}{c} u_2 \\ \downarrow k_2 \\ v_2 \end{array} \right) \dots \left( \begin{array}{c} u_p \\ \downarrow k_p \\ v_p \end{array} \right)$$

$$k = k_1 + \dots + k_p$$

$$v = v_1 v_2 \dots v_p = \left( \begin{array}{c} v_1 \end{array} \right) \left( \begin{array}{c} v_2 \end{array} \right) \dots \left( \begin{array}{c} v_p \end{array} \right)$$
- Lorsque un  $u_i$  est une lettre (un terminal)
  - $k_i = 0$  et  $v_i = u_i$
  - Exemples :
    - $aSb \xrightarrow{k} m \implies m = avb$  et  $S \xrightarrow{k} v$
    - $aSbS \xrightarrow{k} m \implies m = av_1bv_2$  et  $S \xrightarrow{k_1} v_1$  et  $S \xrightarrow{k_2} v_2$  et  $k = k_1 + k_2$

## Démonstration par récurrence sur k

- $k = 0$  cas trivial avec  $k_1 = k_2 = 0$  et  $v_1 = u_1, v_2 = u_2$
- $k = 1$   $u_1 u_2 \xrightarrow{1} v$ 
  - Soit  $(M \rightarrow m) \in P$  la production utilisée pour cette dérivation.  $M$ , non terminal, est dans  $u_1$  ou dans  $u_2$  :
    - $M$  est dans  $u_1$ , alors
      - $u_1 u_2 = (u'.M.u'') \cdot u_2 \rightarrow (u'.m.u'') \cdot u_2 = v$
      - $k_1 = 1$   $k_2 = 0$  et  $v_1 = u'.m.u''$  et  $v_2 = u_2$ , et on a  $v = v_1 v_2$
    - $M$  dans  $u_2$ , alors
      - $u_1 u_2 = u_1 \cdot (u'.M.u'') \rightarrow u_1 \cdot (u'.m.u'') = v$
      - $k_1 = 0$   $k_2 = 1$  et  $v_1 = u_1, v_2 = u'.m.u''$  et on a  $v = v_1 v_2$

## Démonstration par récurrence sur k

- Hypothèse : Vrai pour  $k$ , où  $k \geq 1$
- Montrons que c'est vrai pour  $k+1$
- $u_1 u_2 \xrightarrow{k+1} v \implies u_1 u_2 \xrightarrow{k} w \xrightarrow{1} v$
- Par hypothèse :
 

$$w_1 w_2 \xrightarrow{k'_1} v = v_1 v_2$$

$$w_1 w_2 \xrightarrow{k'_2} v$$

$\leftarrow$  Montré précédemment ( $k=1$ )

$$u_1 u_2 \xrightarrow{k} w = w_1 w_2$$

  - $k+1 = (k_1 + k'_1) + (k_2 + k'_2)$   $u_1 \xrightarrow{k_1} w_1 \xrightarrow{k'_1} v_1$  et  $u_2 \xrightarrow{k_2} w_2 \xrightarrow{k'_2} v_2$

## Démonstration sur les grammaires

- Soit  $G = \langle \{a,b\}, \{S\}, \{S \rightarrow aSb \mid \varepsilon\}, S \rangle$
- Soit  $E = \{a^n b^n, n \geq 0\}$
- Proposition :  $E = L_G$
- Démonstration classique :
  - $E \subseteq L_G$ 
    - Récurrence sur la longueur du mot
    - $\Pi(n) = \forall m \in E \text{ tel que } |m| \leq n \text{ alors } m \in L_G$
  - $L_G \subseteq E$ 
    - Récurrence sur la longueur de la chaîne de dérivations
    - $\Pi(n) = \forall m \in L_G \text{ tel que } S \xrightarrow{n} m \text{ alors } m \in E$

## Démonstration de $E \subseteq L_G$

$$G = \langle \{a,b\}, \{S\}, \{S \rightarrow aSb \mid \varepsilon\}, S \rangle$$

$$\Pi(n) = \forall m \in E \text{ tel que } |m| \leq n \text{ alors } m \in L_G$$

- $\Pi(0)$  vrai :  $m \in E$  et  $|m| \leq 0 \implies m = \varepsilon$ . On a :  $\varepsilon \in L_G$
- Hypothèse :  $\Pi(n)$  vrai (et  $n \geq 0$ )
  - Soit  $m \in E$ ,  $|m| = n+1$ , il faut montrer que  $m \in L_G$
  - $m \in E \implies m = a^n b^{n'} = a (a^{n'-1} b^{n'-1}) b = a u b$  avec  $|u| \leq n$   
 $\implies u \in L_G$ , i.e.  $S \xrightarrow{*} u$  par hyp. de récurrence  
 $\implies S \rightarrow aSb \xrightarrow{*} aub = m$   
 $\implies m \in L_G$

## Démonstration de $L_G \subseteq E$

$$G = \langle \{a,b\}, \{S\}, \{S \rightarrow aSb \mid \varepsilon\}, S \rangle$$

$$\Pi(n) = \forall m \in L_G \text{ tel que } S \xrightarrow{n} m \text{ alors } m \in E$$

- $\Pi(1)$  vrai :  $S \xrightarrow{1} m \implies m = \varepsilon$ . On a bien :  $\varepsilon \in E$
- Hyp :  $\Pi(n)$  vrai (et  $n \geq 1$ ). Montrons que  $\Pi(n+1)$  vrai :  
 Soit  $S \xrightarrow{n+1} m$ , montrons que  $m \in E$   
 Forcément la première dérivation est  $S \rightarrow aSb$  :  
 $S \xrightarrow{1} aSb \xrightarrow{n} m$  Puis lemme fondamental généralisé :  
 $aSb \xrightarrow{n} m = am'b$  et  $S \xrightarrow{n} m'$   
 $\implies m' \in E$  (par hyp. de rec.)  
 $\implies m' = a^k b^k$   
 $\implies m = am'b = a^{k+1} b^{k+1} \in E$  CQFD.

## Démonstration de $L_G \subseteq E$ (2)

$$G = \langle \{a,b\}, \{S\}, \{S \rightarrow aSb \mid \varepsilon\}, S \rangle$$

$$\Pi(n) = \forall m \in L_G \text{ tel que } S \xrightarrow{n} m \text{ alors } m \in E$$

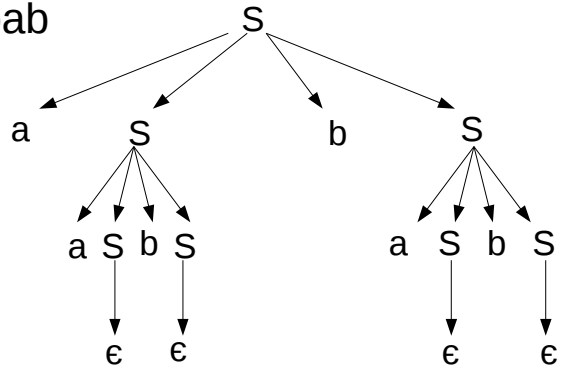
- $\Pi(0)$  vrai :  $S \xrightarrow{0} m$  impossible. Donc  $\Pi(0)$  vrai par vacuité
- Hyp :  $\Pi(n)$  vrai (et  $n \geq 0$ ). Montrons que  $\Pi(n+1)$  vrai :  
 Soit  $S \xrightarrow{n+1} m$ , montrons que  $m \in E$  :
  - $S \xrightarrow{1} \varepsilon \xrightarrow{n} m$  Donc  $m = \varepsilon$  et on a bien  $m \in E$
  - $S \xrightarrow{1} aSb \xrightarrow{n} m$  Puis lemme fondamental généralisé :  
 $aSb \xrightarrow{n} m = am'b$  et  $S \xrightarrow{n} m'$   
 $\implies m' \in E$  (par hyp. de rec.)  
 $\implies m' = a^k b^k$   
 $\implies m = am'b = a^{k+1} b^{k+1} \in E$  CQFD.

## Exemples de démonstration

- $G = \langle \Sigma, X, P, S \rangle$   
 $\Sigma = \{a, b\}$   
 $X = \{S, T\}$   
 $P = \{S \rightarrow aST, S \rightarrow \varepsilon, T \rightarrow bb\}$
- Montrer que :  $L_G = \{a^n b^{2n} \mid n \geq 0\}$
- Montrer que :  
 $\widehat{L}_G = \{a^n uv \mid n > 0 \text{ et } u \in \{S, \varepsilon\} \text{ et } v \in \{T, bb\}^n\}$
- Simplifier la grammaire au préalable à condition de préciser les règles de simplification utilisées et prouvées...  
 - Ici, on obtiendrait :  $P = \{S \rightarrow aSbb \mid \varepsilon\}$
- Établir des propriétés :  $S \xrightarrow{*} m \implies |m|_a = |m|_T + \frac{1}{2}|m|_b$   
 $S \xrightarrow{*} m \implies ba \text{ et } Ta \text{ ne sont pas des sous mots de } m$

## Arbre de dérivation

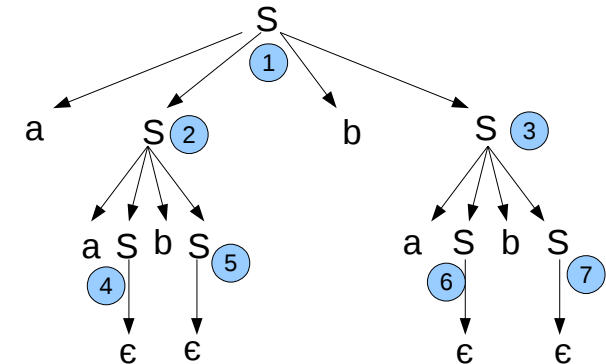
- $G = \langle \{a,b\}, \{S\}, \{ S \rightarrow aSbS \mid \epsilon \}, S \rangle$
- $S \xrightarrow{*} aabbab$



- $S \xrightarrow{1} aSbS \xrightarrow{2} a aSbS b aSbS \xrightarrow{4} a a\epsilon b\epsilon b a\epsilon b\epsilon = aabbab$

## Différentes dérivation

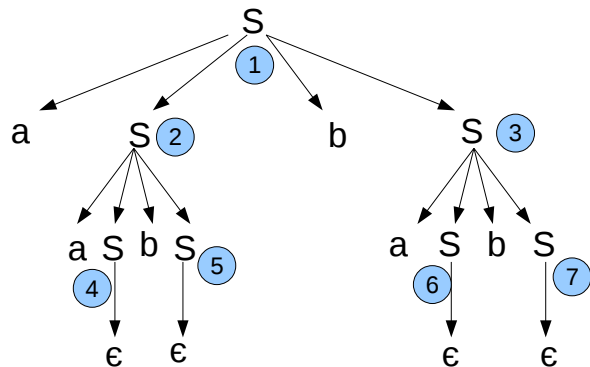
- 1 seul arbre, plusieurs chaînes de dérivation
- 



- Ordre 1,2,3,4,5,6,7 :  
 $\underline{S} \rightarrow a\underline{S}bS \rightarrow a a\underline{S}bS b\underline{S} \rightarrow a a\underline{S}bS b aSbS \rightarrow a a b\underline{S} b aSbS \rightarrow a ab b a\underline{S}bS \rightarrow a ab b ab\underline{S} \rightarrow aabbab$

## Différentes dérivation (2)

- Chaîne de dérivation = parcours de l'arbre

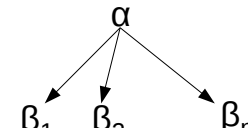


- Ordre 1,2,4,5,3,6,7 :  
 $\underline{S} \rightarrow a\underline{S}bS \rightarrow a a\underline{S}bS bS \rightarrow a ab\underline{S} bS \rightarrow a ab b\underline{S} \rightarrow aabb a\underline{S}bS \rightarrow aabb ab\underline{S} \rightarrow aabb ab$

## Définition d'un arbre de dérivation

- Soit  $G = \langle \Sigma, X, P, S \rangle$  une grammaire. Un **arbre de dérivation** de  $G$  est un arbre (non unique) qui vérifie les propriétés suivantes :

- Ses étiquettes sont dans  $X \cup \Sigma \cup \{\epsilon\}$  et :
  - L'étiquette de la racine de l'arbre est  $S$  (l'axiome)
  - Les étiquettes des feuilles sont dans  $\Sigma \cup \{\epsilon\}$
  - Les étiquettes des noeuds internes sont dans  $X$

- Si  est dans l'arbre de dérivation, alors  $\alpha \rightarrow \beta_1 \beta_2 \dots \beta_n \in P$

- L'ordre des branches doit être respecté

## Relations entre arbres et chaînes de dérivations

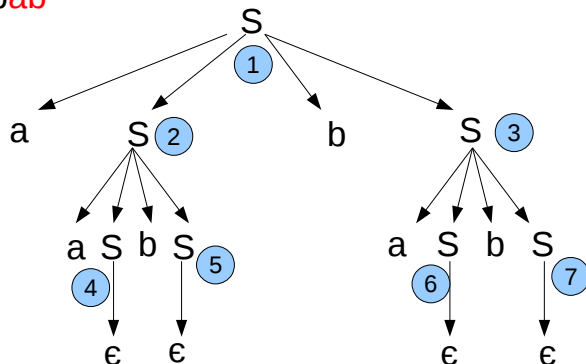
- À un arbre de dérivations A est **associé** le mot m constitué de la concaténation des feuilles de A, lues de gauche à droite. On dira que **l'arbre A reconnaît le mot m**
- À un arbre de dérivations reconnaissant le mot m correspond (en général) plusieurs chaînes de dérivations donnant m
- À une chaîne de dérivation  $S \xrightarrow{*} m$ , correspond un unique arbre de dérivations A reconnaissant m
- Pour avoir une correspondance bi-univoque, il faut considérer les chaînes de **dérivations à gauche**

## Dérivation gauche

- Définition :  $m \rightarrow m'$  est une **dérivation à gauche** si la production utilisée pour l'obtenir s'applique au non terminal le plus à gauche dans m.
  - $aSbS \rightarrow aaSbSbS$  dérivation gauche
  - $aSbS \rightarrow aSbaSbS$  n'est pas une dérivation gauche
- Une « **chaîne de dérivations** » à gauche est une chaîne de « dérivations à gauche » !
- À toute chaîne de dérivations (et à tout arbre de dérivation) correspond une unique chaîne de dérivations à gauche.
  - Cela consiste à parcourir l'arbre « **en profondeur d'abord** » et de gauche à droite

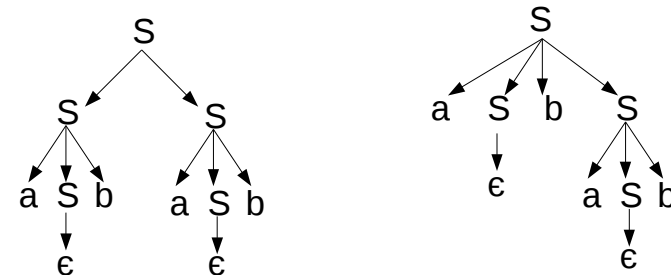
## Parcours en profondeur d'abord

- Ordre : 1 2 4 5 3 6 7
- $\underline{S} \rightarrow a\underline{S}bS \rightarrow aa\underline{S}bSbS \rightarrow aab\underline{S}bS \rightarrow aabb\underline{S} \rightarrow aabba\underline{S}bS \rightarrow aabbab\underline{S} \rightarrow aabbab$



## Grammaire ambiguë

- Définition : Une grammaire G est dite **ambiguë** s'il existe au moins un mot de  $L_G$  qui est associé à au moins deux arbres de dérivations différents.
- Exemple :  $G = \langle \{a,b\}, \{S\}, \{S \rightarrow aSbS \mid aSb \mid SS \mid \epsilon\}, S \rangle$



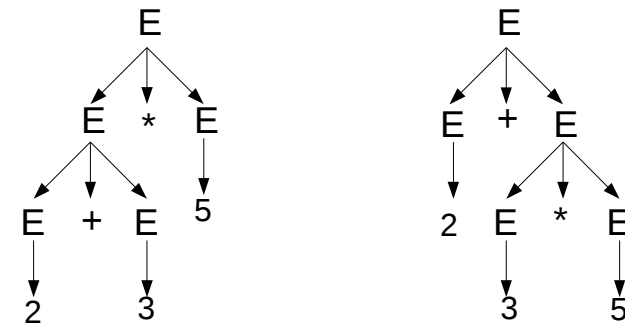
- $m = abab$  est reconnu par deux arbres différents

## Cas simple de non ambiguïté

- Exemple :  $G = \langle \{a,b\}, \{S\}, \{S \rightarrow aS \mid bS \mid \varepsilon\}, S \rangle$
- La grammaire est non ambiguë.
  - Et même : La chaîne de dérivations est unique  $S \xrightarrow{n} m$
  - Exemple avec  $m = aaba$  :  
 $S \rightarrow aS \rightarrow aaS \rightarrow aabS \rightarrow aabaS \rightarrow aaba$
  - En général, si on a :  $S \xrightarrow{n} m'S \rightarrow m$   
 $\implies m'S \rightarrow m'aS$  ou  $m'bS$  selon que  $m'a$  ou  $m'b$  soit un préfixe de  $m$ .
- Éléments de preuve par récurrence :
  - Si  $S \xrightarrow{1} \alpha \xrightarrow{n} m$  et  $S \xrightarrow{1} \beta \xrightarrow{n} m$  alors  $\alpha = \beta = m[1] S$
  - $\Pi(n)$  = s'il existe une chaîne  $S \xrightarrow{n} m$ , elle est unique

## Une ambiguïté bien connue

- Les expressions arithmétiques
  - $E \rightarrow E + E \mid E * E \mid (E) \mid v$  grammaire ambiguë
  - $2 + 3 + 5$  :  $2+3 + 5$  vs.  $2 + 3+5$  ?  
(l'associativité de l'addition permet l'ambiguïté)
  - $2 + 3 * 5$  :  $2 + 3 * 5$  vs.  $2 + 3 * 5$  ?  
(identifier de ces arbres est sémantiquement faux)



## Expressions arithmétiques

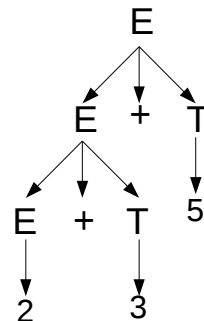
- Grammaire non ambiguë
 
$$E \rightarrow E + T \mid T \quad E = \text{expression}$$

$$T \rightarrow T * F \mid F \quad T = \text{Terme} \quad F = \text{Facteur}$$

$$F \rightarrow (E) \mid v$$
- \* est prioritaire sur +
- On « associe à gauche » :
 
$$2 + 3 + 5 = 2+3 + 5$$

$$2 + 3 + 5 \text{ impossible}$$

$$2 * 3 * 5 = 2*3 * 5$$



## Langage intrinsèquement ambiguë

- Grammaire ambiguë : redondance d'informations pour tester l'appartenance d'un mot au langage associé
- Si un langage est défini par une grammaire ambiguë, rechercher une grammaire équivalente non ambiguë
  - Mais ce n'est pas toujours possible
    - Un langage algébrique est dit **intrinsèquement ambiguë** s'il n'est pas définissable par une grammaire non ambiguë
    - Il existe des langages intrinsèquement ambiguë !
- Mais savoir si une grammaire est ambiguë peut être compliqué :
  - Il ne peut exister d'algorithme répondant à cette question pour toutes les grammaires (problème non décidable)

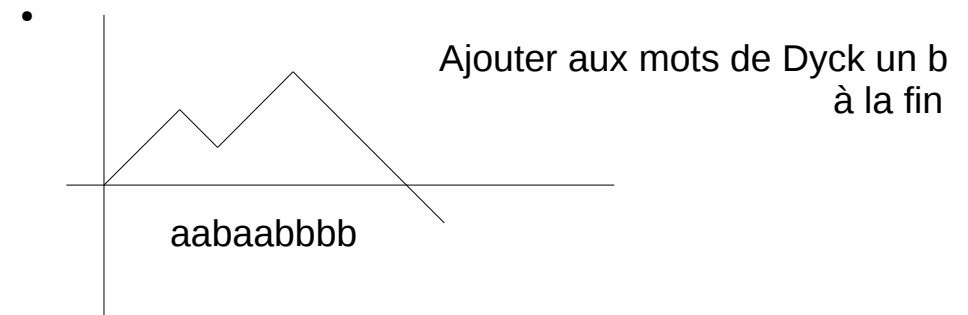


## Le langage de Dyck

- $D = \langle \{a,b\}, \{S\}, \{ S \rightarrow aSb \mid SS \mid \varepsilon \}, S \rangle$ 
  - Cette grammaire est ambiguë  
 $S \rightarrow SS$  est forcément ambiguë en l'itérant 2 fois
- $D' = \langle \{a,b\}, \{S\}, \{ S \rightarrow aSbS \mid \varepsilon \}, S \rangle$ 
  - Cette grammaire n'est pas ambiguë
- $L_D = L_{D'}$  (cf. TD)
- Le langage de Dick n'est pas intrinsèquement ambiguë
- Mots du langage : des « chaînes de montagnes »

## Le langage de Lukasiewicz

- $L = \langle \{a,b\}, \{S\}, \{ S \rightarrow aSS \mid b \}, S \rangle$



- Grammaire non ambiguë : le choix entre les deux productions est imposé par la lettre la plus à gauche du mot final
- $S \rightarrow aSS \rightarrow aaS SS \rightarrow aabSS \rightarrow aabaSS S \rightarrow \dots$   
 $aabaabbbb \quad aabaabbbb \quad aabaabbbb \quad aabaabbbb$

# Les automates déterministes

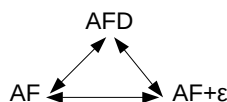
## Automates réels et abstraits

- Automate « réel » :
  - Boîte à musique  $\pm$  programmable
  - Machine à calculer (Blaise pascal)
  - Horloge
  - Intervention humaine très réduite
- Automate « abstrait/formel » :
  - L'automate **pass**e automatiquement d'un **état** au suivant en fonction de ce qu'il « **lit** » :
    - Plein/trou sur une carte perforée, roues  $\pm$  tournées, etc.
- Ordinateur > automate
  - Description des exécutions des programmes par un automate impossible (nombre d'états possiblement infinis)

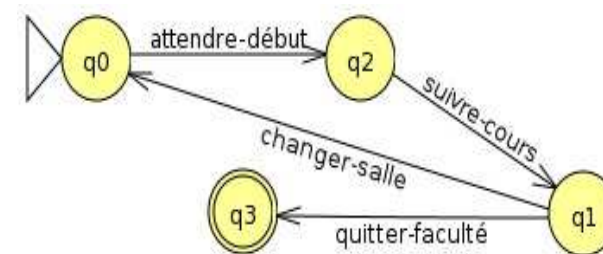


## Différents types d'automate

- Définition d'un langage à partir d'un automate. Quatre types d'automates :
  - Automate fini déterministe (AFD)
  - Automate fini indéterministe (AF)
  - Automate fini indéterministe et  $\epsilon$ -transitions
  - Automate à pile : non traité dans ce cours
- Les trois types d'automates (AFD, AF, AF+ $\epsilon$ ) englobent la même famille de langages
  - Étude des transformations entre ces types



## Exemple d'un automate abstrait



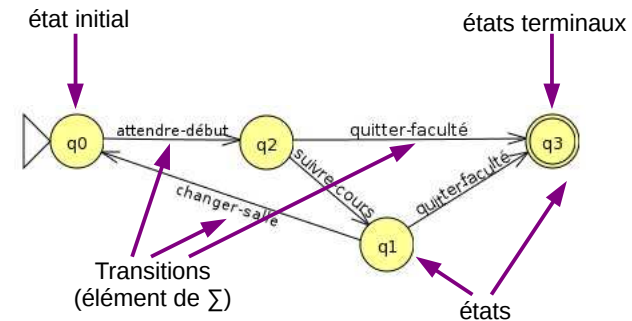
a = attendre-début  
s = suivre-cours  
c = changer-salle  
q = quitter-faculté

- Point de départ : l'état q0    Point final : q3
- Ce que peut vivre un étudiant : ascascasq
  - L'ensemble des possibilités :  $\{ as(cas)^nq \mid n \geq 0 \}$
  - Un étudiant doit suivre au moins un cours !

# Définition d'un automate

- Un **automate déterministe d'états fini (AFD)**, aussi appelé **automate fini déterministe**, est la donnée d'un quintuplet  $A = (\Sigma, E, i, F, \delta)$ 
  - $\Sigma$  est l'alphabet (d'entrée)
  - $E$  est un ensemble fini d'éléments appelés des **états**
  - $i$  est un élément de  $E$ , appelé l' "**état initial**".
  - $F$  est une partie de  $E$ , dont les éléments sont appelés des **états d'arrivée ou terminaux ou finaux, voire finals**
  - $\delta$  est une **fonction** de  $E \times \Sigma$  vers  $E$ , appelée **fonction de transition**

# Visualisation d'un automate

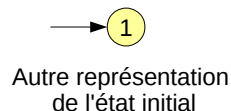
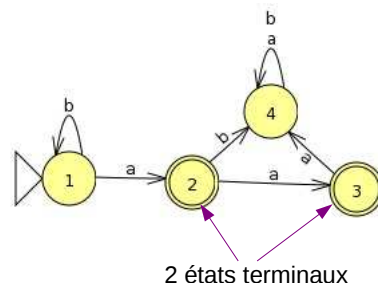


- $\delta : E \times \Sigma \rightarrow E$ 
  - $(q_0, \text{attendre-début}) \rightarrow q_2$  ou  $\delta(q_0, a) = q_2$
  - $(q_2, \text{quitter-faculté}) \rightarrow q_3$  ou  $\delta(q_2, q) = q_3$
- Mot reconnu : les transitions lues de  $q_0$  à  $q_3$

# Exemple complet

- $A = (\Sigma, E, i, F, \delta)$ 
  - $\Sigma = \{a, b\}$
  - $E = \{1, 2, 3, 4\}$      $i = 1$      $F = \{2, 3\}$
  - $\delta(1, a) = 2$      $\delta(1, b) = 1$      $\delta(2, a) = 3$      $\delta(2, b) = 4$
  - $\delta(3, a) = 4$      $\delta(4, a) = 4$      $\delta(4, b) = 4$

- Visualisation :



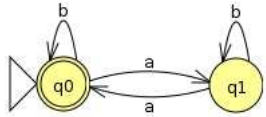
# Définition de la visualisation

- La **visualisation (représentation)** d'un automate  $A = (\Sigma, E, i, F, \delta)$  est un graphe étiqueté tel que :
  - Les sommets sont les états de  $E$
  - Le sommet  $i$  est repéré par une flèche entrante (ou un triangle entrant)
  - Les sommets de  $F$  sont marqués par un double cercle
  - Pour tout  $(e_i, e_j, \alpha)$  dans  $E \times E \times \Sigma$  tel que  $\delta(e_i, \alpha) = e_j$ , il y a un arc de l'état  $e_i$  vers l'état  $e_j$  étiqueté par  $\alpha$ .

Lorsqu'il y a plusieurs arcs de  $e_i$  vers  $e_j$ , on peut les regrouper en un seul arc étiqueté par l'ensemble des étiquettes fusionnées

## Langage associé à un automate

- **Intuition** : Le langage associé regroupe tous les mots qui se lisent sur les étiquettes lorsque l'on parcourt l'automate de l'état initial jusqu'à un état final.



$$L_A = \{m \in \{a,b\}^* \mid |m|_a \text{ est pair} \}$$

- $q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \implies \text{abbab} \in L_A$

↑  
chemin de  $q_0$  à  $q_0$

↑  
trace du chemin

## Chemin et trace

- Définition : pour un automate  $A=(\Sigma, E, i, F, \delta)$ , un **chemin** entre les états  $e_1$  et  $e_n$  est, s'il existe, une séquence de la forme

$$(e_1 \ \alpha_1 \ e_2 \ \alpha_2 \ e_3 \ \alpha_3 \ e_4 \ \alpha_4 \ \dots \ \alpha_{n-1} \ e_n)$$

tel que  $\delta(e_i, \alpha_i) = e_{i+1}$  pour tout  $i$  dans  $\{1, \dots, n-1\}$

Le nombre  $n-1$  de lettres dans la séquence est appelé la **longueur** du chemin.

Et la séquence  $\alpha_1 \alpha_2 \alpha_3 \alpha_4 \dots \alpha_{n-1}$  est appelé la **trace** du chemin.

Remarque : une trace est un mot de  $\Sigma^*$

## Langage reconnu par un automate

- Définition : Soit  $A=(\Sigma, E, i, F, \delta)$  un automate. **Le langage** d'alphabet  $\Sigma$  **reconnu par cet automate**, noté  $L_A$  ou  $L(A)$ , est l'ensemble des mots  $m$  tel qu'il existe un chemin entre l'état initial  $i$  et un état terminal de  $F$ , et dont la trace est  $m$ .

On dira que le mot  $m$  est **accepté** ou **reconnu par l'automate**

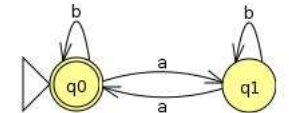
$$m \in L_A \Leftrightarrow \left\{ \begin{array}{l} \exists \begin{array}{c} \downarrow = i \\ (e_1, \alpha_1, e_2, \dots, e_n, \alpha_n, e_{n+1}) \\ \downarrow \in F \end{array} \\ \text{chemin dans } A, \text{ i.e. } \delta(e_k, \alpha_k) = e_{k+1}, \\ \text{de trace } \alpha_1 \alpha_2 \dots \alpha_n = m \end{array} \right\}$$

- Deux automates  $A_1$  et  $A_2$  sont dits **équivalents** si :

$$L_{A_1} = L_{A_2}$$

## Le langage des mots ayant un nombre pair de « a »

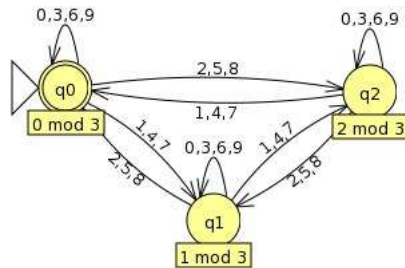
- $L(A) = \{m \in \{a,b\}^* \mid |m|_a \text{ est pair} \}$   
( $\mu$ )



- Il faut trouver tous les chemins entre  $q_0$  et  $q_0$ 
  - Chemin:  $(q_0 \xrightarrow{b} q_0 \xrightarrow{b} \dots q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_1 \dots q_1 \xrightarrow{a} q_0 \dots)$
  - Trace :  $b^n a b^m a \dots$   $L(A) = (\{b\}^* \{a\} \{b\}^* \{a\} \{b\}^*)^* ?$
- Il faudra prouver l'égalité ( $\mu$ ) entre un langage défini par une propriété et un langage défini par un automate

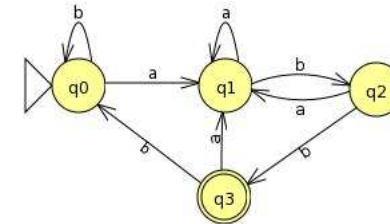
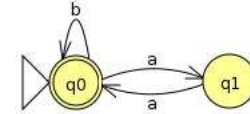
## Les nombres divisibles par 3

- $L_3 = \{ m \in \{0, \dots, 9\}^* \mid m \text{ est divisible par } 3 \text{ (ou } \varepsilon) \}$ 
  - Exemple :  $423 \in L_3$  car  $423 = 3 \times 141$
  - $m \in L_3$  ssi la somme des chiffres de  $m$  vaut 0 modulo 3
  - $423 = q_0 \text{ } 4 \text{ } q_1 \text{ } 2 \text{ } q_0 \text{ } 3 \text{ } q_0$   
 $423 \in L_3$
  - $25 = q_0 \text{ } 2 \text{ } q_2 \text{ } 5 \text{ } q_1$   
 $25 \notin L_3$



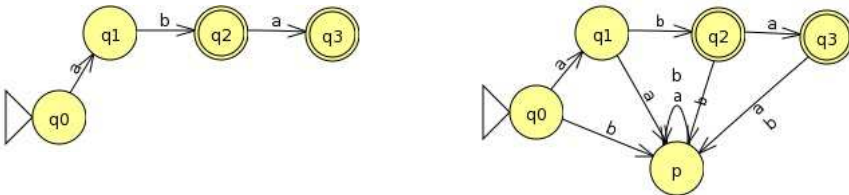
## Automate complet

- $A = (\Sigma, E, i, F, \delta)$   
 $\delta$  est seulement une **fonction** de  $E \times \Sigma \rightarrow E$   
Certains chemins finissent en impasse.  
 $m = ab \notin L(A)$   
Pas de chemin pour  $ab$
- Définition : un automate est dit **complet** si sa fonction de transition est une **application**



## Compléter un automate

- Tout automate  $A = (\Sigma, E, i, F, \delta)$  peut être complété en ajoutant un état « poubelle »
  - L'automate complété reconnaît le même langage



- $A_p = (\Sigma, E \cup \{p\}, i, F, \delta_p)$  automate complété
  - $p$  est un nouvel état qui n'était pas dans  $E$
  - $\delta_p$  contient des éléments en plus :

## Compléter un automate (2)

- $A_p = (\Sigma, E \cup \{p\}, i, F, \delta_p)$  automate complété
  - $p \notin E$
  - $\delta_p(e, \alpha) = \delta(e, \alpha)$  si  $(e, \alpha) \in \mathcal{D}_\delta$  pour tout  $e \in E \cup \{p\}$   
 $p$  sinon
- Utilisation d'une notation ensembliste pour  $\delta$ 
  - $\delta =_{\text{not}} \{ (e, \alpha, e') \mid \delta(e, \alpha) = e' \}$
  - $(e, \alpha, e') \in \delta$  ssi  $\delta(e, \alpha) = e'$
  - $\delta_p = \delta \cup \{ (e, \alpha, p) \in E \times \Sigma \times \{p\} \text{ tq } (e, \alpha) \notin \mathcal{D}_\delta \}$   
 $\cup \{ (p, \alpha, p) \mid \alpha \in \Sigma \}$
- $A$  et  $A_p$  reconnaissent le même langage ?
  - Preuve sur les automates (cf. TD)

## Fonction de transition itérée

lettre

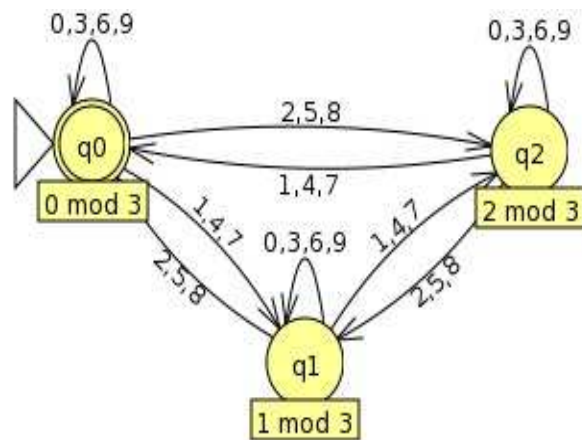
$$(q_0, 2) \xrightarrow{\delta} q_2$$

$$(q_0, 28) \xrightarrow{\delta^*} q_1$$

$$(q_0, 283) \xrightarrow{\delta^*} q_1$$

$$(q_0, 2832) \xrightarrow{\delta^*} q_0$$

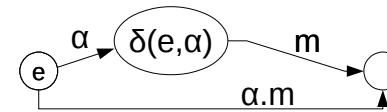
mot



## Fonction de transition itérée

- Définition : Soit  $A = (\Sigma, E, i, F, \delta)$  un automate **complet**. La **fonction de transition itérée**, notée  $\delta^*$ , est l'application de  $E \times \Sigma^*$  vers  $E$  qui vérifie :

- $\forall e \in E, \delta^*(e, \varepsilon) = e$
- $\forall e \in E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(e, \alpha.m) = \delta^*(\delta(e, \alpha), m)$



- $\delta^*(e, m)$  est l'état accédé à partir de l'état  $e$  par un chemin de trace  $m$ .
- $\delta^*$  est bien une application :  $\delta^*(e, m)$  existe pour tout état  $e$  de  $E$  et tout mot  $m$  de  $\Sigma^*$

## Propriétés des transitions itérées

- $\delta^*(e, \alpha) = \delta(e, \alpha)$  si  $\alpha \in \Sigma$

- Preuve :

$$\begin{aligned} \delta^*(e, \alpha) &= \delta^*(e, \alpha.\varepsilon) \\ &= \delta^*(\delta(e, \alpha), \varepsilon) \\ &= \delta(e, \alpha) \end{aligned}$$

- Définition de  $L(A)$  sans chemins ni traces

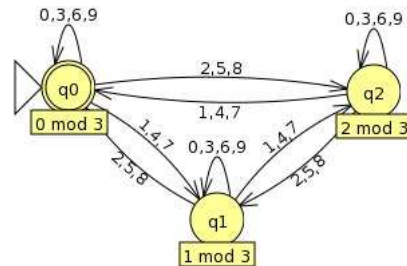
$$\delta^*(i, m) \in F \Leftrightarrow m \in L(A)$$

$$\delta^*(e, m) = e' \Leftrightarrow \left\{ \begin{array}{l} \text{Il existe un chemin dans } A \\ \text{de l'état } e \text{ à l'état } e', \text{ et de trace } m \end{array} \right\}$$

- Donner du sens aux états :**  $\delta^*(i, m) = q_{m \bmod 3}$

- $\delta^*(e, m) \in F$  "les mots  $m$  reconnus à partir de  $e$ "

- $\delta^*(i, m) = e$  "les mots  $m$  reconnus en arrivant en  $e$ "



## Propriété de la fonction itérée

- La fonction itérée est définie par :

$$\forall e \in E, \delta^*(e, \varepsilon) = e$$

$$\forall e \in E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(e, \alpha.m) = \delta^*(\delta(e, \alpha), m)$$

- La fonction itérée vérifie le théorème :

$$\forall e \in E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(e, m.\alpha) = \delta(\delta^*(e, m), \alpha)$$

- Preuve par récurrence sur la longueur de  $m$

$$\pi(n) = |m| \leq n \Rightarrow \delta^*(e, m.\alpha) = \delta(\delta^*(e, m), \alpha)$$

$$\pi(0) \text{ est vrai : } \delta^*(e, \varepsilon.\alpha) = \delta^*(e, \alpha) = \delta(e, \alpha)$$

$$\delta(\delta^*(e, \varepsilon), \alpha) = \delta(e, \alpha)$$



## Propriété inverse de la fonction itérée

- Rappel définition :

$$\forall e \in E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(e, \alpha.m) = \delta^*(\delta(e, \alpha), m)$$

- Hypothèse de récurrence :

$$\pi(n) = |m| \leq n \Rightarrow \delta^*(e, m.\alpha) = \delta(\delta^*(e, m), \alpha)$$

- Soit  $m, |m| = n+1$

$$\begin{aligned} \delta^*(e, m.\alpha) &= \delta^*(e, \beta.(m'.\alpha)) && \text{avec } m = \beta.m' \\ &= \delta^*(\delta(e, \beta), m'.\alpha) && \text{par définition de } \delta^* \\ &= \delta(\delta^*(\delta(e, \beta), m'), \alpha) && \text{par hyp. de récurrence} \\ &= \delta(\delta^*(e, \beta.m'), \alpha) && \text{par définition de } \delta^* \\ &= \delta(\delta^*(e, m), \alpha) && \text{avec } m = \beta.m' \end{aligned}$$

## Fonction de transition itérée pour automate non complet

- La fonction de transition itérée existe mais n'est pas une application
- Définition : Soit  $A = (\Sigma, E, i, F, \delta)$  un automate. La **fonction de transition itérée**, notée  $\delta^*$ , est la fonction de  $E \times \Sigma^*$  vers  $E$  qui vérifie :

- $\forall e \in E, \delta^*(e, \varepsilon) = e$
- $\forall e \in E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*,$   
si  $\delta(e, \alpha)$  et  $\delta^*(\delta(e, \alpha), m)$  sont définis  
alors  $\delta^*(e, \alpha.m) = \delta^*(\delta(e, \alpha), m)$   
sinon  $\delta^*(e, \alpha.m)$  est non défini

## Fonction de transition itérée sous réserve d'existence...

- Les propriétés et théorèmes s'étendent sous réserve que les expressions soient définies

- Théorème :

$$\delta^*(e, m) \text{ existe} \Rightarrow \forall p \text{ préfixe de } m, \delta^*(e, p) \text{ existe}$$

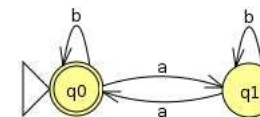
- Propriété fondamentale :

$$m \in L(A) \Leftrightarrow \delta^*(i, m) \text{ est défini et } \delta^*(i, m) \in F$$

$$m \in L(A) \Leftrightarrow \delta^*(i, m) \in F \text{ sous réserve d'existence}$$

## Preuve sur un automate

- Soit l'automate A suivant :



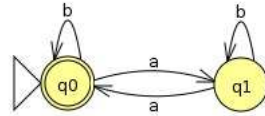
- Prouver que :  $L(A) = \{ m \in \Sigma^* \mid |m|_a \text{ pair} \} = L_p$ 
  - Lemme :  $\forall m \in \Sigma^*, \delta^*(q_0, m) = q_0$  si  $|m|_a$  pair  
 $q_1$  si  $|m|_a$  impair
  - Preuve de  $L(A) = L_p : m \in L(A) \Leftrightarrow \delta^*(q_0, m) = q_0 \Leftrightarrow |m|_a \text{ pair}$
  - Preuve du lemme par récurrence sur  $|m|$  :

$$\pi(n) = |m| \leq n \Rightarrow \delta^*(q_0, m) = q_{|m|_a \bmod 2}$$

$$\pi(0) \text{ est vrai : } \delta^*(q_0, \varepsilon) = q_0 = q_{|\varepsilon|_a \bmod 2}$$

## Preuve sur un automate

- Soit l'automate A suivant :



Hypothèse :  $\pi(n) = |m| \leq n \Rightarrow \delta^*(q_0, m) = q_{|m|_a \bmod 2}$

$$\begin{aligned} \forall m, |m| = n+1, \delta^*(q_0, m) &= \delta^*(q_0, m' \cdot \alpha) && \text{Avec } m = m' \cdot \alpha \\ &= \delta(\delta^*(q_0, m'), \alpha) && \text{La propriété...} \\ &= \delta(q_{|m'|_a \bmod 2}, \alpha) && \text{hyp. Rec.} \\ &? \\ &= q_{|m'|_a \cdot \alpha|_a \bmod 2} \\ &= q_{|m|_a \bmod 2} && \text{Avec } m = m' \cdot \alpha \end{aligned}$$

- Il reste à vérifier que :  $\delta(q_{k \bmod 2}, \alpha) = q_{k + |\alpha|_a \bmod 2}$ 
  - Envisager les 4 cas possibles :  $k \in \{0, 1\}, \alpha \in \{a, b\}$   
 $k=0, \alpha = b \quad \delta(q_0, b) = q_0 = q_{0+|b|_a}$

## Simplifications

- Deux automates sont dit **équivalents** s'ils définissent (reconnaissent) le même langage
- Exemple de simplification : éliminer les états inutiles
  - Un état e est dit **accessible** s'il existe un chemin depuis l'état initial jusqu'à e :  $\exists m, \delta^*(i, m) = e$
  - Un état e est dit **co-accessible** s'il existe un chemin depuis e jusqu'à un état terminal :  $\exists m, \delta^*(e, m) \in F$
  - Théorème : pour tout automate A, soit A' l'automate obtenu en supprimant dans A les états non accessibles et non co-accessibles. Alors A et A' sont équivalents

## Élimination des états non accessibles

- $A = (\Sigma, E, i, F, \delta)$   
 $A' = (\Sigma, E', i, F', \delta')$  l'état i est co-accessible...  
 $E' = \{e \in E, \exists m_1, \delta^*(i, m_1) = e \text{ et } \exists m_2, \delta^*(e, m_2) \in F\}$   
 $F' = F \cap E'$   
 $\delta' = \{(e, \alpha, e') \in \delta, (e, e') \in E' \times E'\} = \delta|_{E' \times \Sigma \times E'}$
- $L(A') \subseteq L(A)$  car ...  $\delta'^* = \delta^*|_{E' \times \Sigma^* \times E'}$   
 $m \in L(A') \Rightarrow \delta'^*(i, m) \in F' \Rightarrow \delta^*(i, m) \in F' \subseteq F \Rightarrow m \in L(A)$
- $L(A) \subseteq L(A')$  car  $F' \subseteq F$  et  $\delta' \subseteq \delta$  et ...  $\delta'^* \subseteq \delta^*$   
 $m \in L(A) \Rightarrow \delta^*(i, m) \in F \Rightarrow \delta^*(i, m) \text{ accessible et co-accessible}$   
 $\Rightarrow \delta^*(i, m) \in E' \Rightarrow \delta^*(i, m) \in F' \Rightarrow \delta'^*(i, m) \in F' \Rightarrow m \in L(A')$

## Un exercice

- Quel est l'automate si on inverse toutes les flèches ?
  - On suppose un seul état terminal et on inverse l'état initial et l'état final.
- Formaliser la question
  - $A = (\Sigma, E, i, \{f\}, \delta)$   
 $A' = (\Sigma, E, f, \{i\}, \delta')$  et  $\delta'(e, \alpha) = e' \Leftrightarrow \delta(e', \alpha) = e$
  - $L_{A'} = \{m \in \Sigma^* \mid m = \alpha_1 \dots \alpha_n \text{ et } \alpha_n \dots \alpha_1 \in L_A\}$   
 On inverse l'ordre des lettres des mots de A
- Preuve :  
 $m \in L_{A'} \Leftrightarrow \delta'^*(f, m) = i \stackrel{?}{\Leftrightarrow} \delta^*(i, \bar{m}) = f \Leftrightarrow \bar{m} \in L_A$   
 avec  $\bar{m}$  le mot m où les lettres sont dans l'ordre inverse



## Exercice (2)

$$\Pi(n) = \forall m, |m| \leq n, \delta'^*(e, \bar{m}) = e' \Leftrightarrow \delta^*(e', m) = e$$

Généralisation pour des états  $e$  et  $e'$  quelconques !

- $\Pi(0)$  vrai
- Hyp :  $\Pi(n)$  vrai.

Montrons  $\Pi(n+1)$  pour  $m.\alpha$  où  $|m| = n$

$$\begin{aligned} \delta'^*(e, \overline{m.\alpha}) = e' &\Leftrightarrow \delta'^*(e, \alpha.\bar{m}) = e' \\ &\Leftrightarrow \delta'^*(\delta'(e, \alpha), \bar{m}) = e' && \text{par def de } \delta'^* \\ &\Leftrightarrow \delta^*(e', \bar{m}) = \delta'(e, \alpha) && \text{par hyp de rec.} \\ &\Leftrightarrow \delta^*(e', m) = \delta'(e, \alpha) \\ &\Leftrightarrow \delta(\delta^*(e', m), \alpha) = e && \text{par def de } \delta' \\ \delta^*(e', m.\alpha) = e &\Leftrightarrow \delta^*(e', m.\alpha) = e && \text{par def de } \delta^* \end{aligned}$$

## Automates

- ☒ Automates déterministes
- ☐ Automates indéterministes
- ☐ Automates avec  $\varepsilon$ -transitions
- ☐ Transformations d'automates

# Les automates indéterministes

## Automates

- ☒ Automates déterministes
- ☐ Automates indéterministes
- ☐ Automates avec  $\varepsilon$ -transitions
- ☐ Transformations d'automates

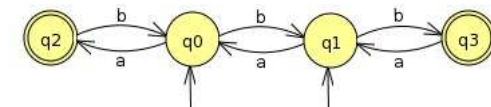
## Définition

- Un automate indéterministe d'états fini (AF) est la donnée d'un quintuplet  $A = (\Sigma, E, I, F, \delta)$ 
  - $\Sigma$  est l'alphabet (d'entrée)
  - $E$  est un ensemble fini d'éléments appelés des états
  - $I$  est une partie de  $E$ , dont les éléments sont appelé les "états initiaux"
  - $F$  est une partie de  $E$ , dont les éléments sont appelés des "états terminaux"
  - $\delta$  est une fonction de  $E \times \Sigma$  vers  $\mathcal{P}(E)$ , appelée fonction de transition

## Exemples

- Plusieurs états initiaux

$$I = \{q_0, q_1\}$$



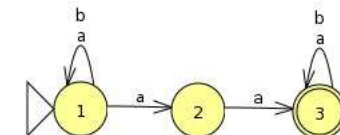
- Non déterminisme de  $\delta$

$$\delta(1, a) = \{1, 2\}$$

$$\delta(1, b) = \{1\}$$

$$\delta(2, b) = \{\}$$

...



# Indéterminisme vs. Déterminisme

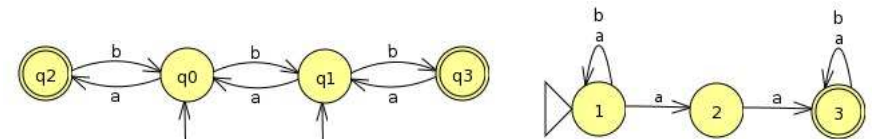
- Un automate déterministe est un automate indéterministe dont la fonction de transition est déterministe :

$$\forall e \in E, \forall \alpha \in \Sigma, \delta(e, \alpha) \text{ a au plus un élément}$$

- Les résultats sur les automates déterministes s'étendent aux automates indéterministes
  - Représentation visuelle
  - Chemin, trace, langage associé
  - Automate complet
  - Fonction de transition itérée (et étendue)
  - État (co-)accessible

# Représentation visuelle

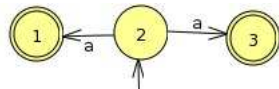
- Mêmes règles de représentation que pour les AFD
  - Les sommets pour les états.
  - Double cercle pour un état terminal
- Modification pour inclure l'indéterminisme
  - Plusieurs états initiaux (avec une flèche entrante)
  - Pour tout état  $e$ , toute lettre  $\alpha$ , si  $\delta(e, \alpha) = \{e_1, \dots, e_n\}$  alors il y aura  $n$  arcs de l'état  $e$  vers chacun des  $n$  états  $e_i$



# Chemin et trace

- Même notion de chemin et trace
  - $(e_1 \alpha_1 e_2 \alpha_2 e_3 \alpha_3 e_4 \alpha_4 \dots \alpha_{n-1} e_n)$  est un **chemin** si  $e_{i+1} \in \delta(e_i, \alpha_i)$  pour tout entier  $i$  dans  $\{1, \dots, n-1\}$
  - $\alpha_1 \alpha_2 \alpha_3 \alpha_4 \dots \alpha_{n-1}$  est la **trace** du chemin précédent.
  - 2 chemins différents pour une même trace :

$(2 \text{ a } 1) \implies \text{trace } a$   
 $(2 \text{ a } 3) \implies \text{trace } a$



# Langage associé à un automate indéterministe

- Soit  $A = (\Sigma, E, I, F, \delta)$  un automate.  
**Le langage associé à cet automate  $A$** , noté  $L_A$  ou  $L(A)$ , est l'ensemble des mots  $m$  tel qu'il existe un chemin entre un état initial et un état terminal, et dont la trace est  $m$ .

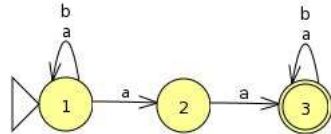
$$m \in L_A \Leftrightarrow \left\{ \begin{array}{l} \exists (e_1, \alpha_1, e_2, \dots, e_n, \alpha_n, e_{n+1}) \\ \text{de trace } \alpha_1 \alpha_2 \dots \alpha_n = m \text{ avec } e_{k+1} \in \delta(e_k, \alpha_k) \end{array} \right\}$$

- Deux automates  $A_1$  et  $A_2$  sont dits **équivalents** si :  

$$L_{A_1} = L_{A_2}$$

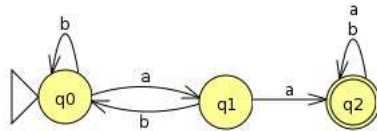
## Exemple

- Le langage des mots sur {a,b} ayant un facteur aa
- Version indéterministe



- Version déterministe

q0 : « a » non suffixe  
 q1 : « a » est un suffixe  
 q2 : reconnu « aa »



- L'indéterminisme facilite parfois la construction

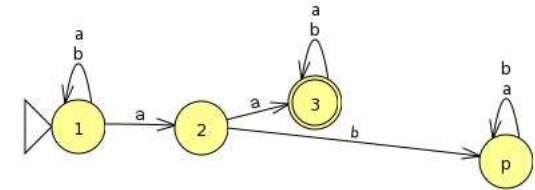
## Automate complet

- Définition : un automate  $A = (\Sigma, E, I, F, \delta)$  indéterministe est dit **complet** si

$$\forall e \in E \forall \alpha \in \Sigma \exists e' \in E \text{ tel que } e' \in \delta(e, \alpha)$$

- Compléter :

Ajouter un état poubelle



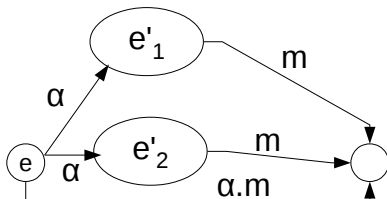
- $A_c = (\Sigma, E \cup \{p\}, I, F, \delta_c)$ 
  - $\delta_c = \delta \cup \{ (e, \alpha, p) \mid \delta(e, \alpha) = \emptyset \} \cup \{ (p, \alpha, p), \alpha \in \Sigma \}$
- Même abus de notation :
  - $\delta =_{\text{not}} \{ (e, \alpha, e') \mid e' \in \delta(e, \alpha) \}$

## Fonction de transition itérée

- Définition : Soit  $A = (\Sigma, E, I, F, \delta)$  un automate indéterministe. La **fonction de transition itérée**, notée  $\delta^*$ , est l'**application** de  $E \times \Sigma^*$  vers  $P(E)$  qui vérifie :

$$\forall e \in E, \delta^*(e, \varepsilon) = \{e\}$$

$$\forall e \in E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(e, \alpha.m) = \bigcup_{(e, \alpha, e') \in \delta} \delta^*(e', m)$$



$$\delta(e, \alpha) = \{e'_1, \dots, e'_2\}$$

$$(e, \alpha, e'_1) \in \delta$$

$$(e, \alpha, e'_2) \in \delta$$

- $\delta^*(e, m)$  est l'ensemble des états accédés à partir de l'état e par un chemin de trace m

## Fonction de transition étendue

- Définition : Soit  $A = (\Sigma, E, I, F, \delta)$  indéterministe. La **fonction de transition étendue**, notée  $\delta$ , est l'**application** de  $P(E) \times \Sigma$  vers  $P(E)$  qui vérifie :

$$\forall E' \subseteq E, \forall \alpha \in \Sigma, \delta(E', \alpha) = \bigcup_{e' \in E'} \delta(e', \alpha)$$

- Définition : Soit  $A = (\Sigma, E, I, F, \delta)$  indéterministe. La **fonction de transition itérée étendue**, notée  $\delta^*$ , est l'**application** de  $P(E) \times \Sigma^*$  vers  $P(E)$  qui vérifie :

$$\forall E' \subseteq E, \forall m \in \Sigma^*, \delta^*(E', m) = \bigcup_{e' \in E'} \delta^*(e', m)$$

- Les fonctions étendues sont notées à l'identique !
- $\delta(\bigcup_{e \in E'} \{e\}, \alpha) = \bigcup_{e \in E'} \delta(e, \alpha)$        $\delta^*(\bigcup_{e \in E'} \{e\}, m) = \bigcup_{e \in E'} \delta^*(e, m)$

## Propriétés des transitions itérées

- $\delta^*(e, \alpha) = \delta(e, \alpha)$  si  $\alpha \in \Sigma$ 
  - Preuve :  

$$\delta^*(e, \alpha) = \delta^*(e, \alpha.\varepsilon) = \bigcup_{e' \in \delta(e, \alpha)} \delta^*(e', \varepsilon) = \bigcup_{e' \in \delta(e, \alpha)} \{e'\} = \delta(e, \alpha)$$
- $\delta^*(I, m) \cap F \neq \{\} \Leftrightarrow m \in L(A)$ 
  - Définition de  $L(A)$  sans chemins ni traces
- $\delta^*(e, m) \cap F \neq \{\}$  les traces de  $e$  à un état terminal
- $e \in \delta^*(I, m)$  les traces en arrivant dans l'état  $e$ , à partir d'un état initial

## Propriétés des transitions itérées (2)

- Retour sur la définition de  $\delta^*$ 

$$\forall e \in E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(e, \alpha.m) = \bigcup_{(e, \alpha, e') \in \delta} \delta^*(e', m)$$

$$= \delta^*\left(\bigcup_{(e, \alpha, e') \in \delta} e', m\right)$$

$$\forall e \in E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(e, \alpha.m) = \delta^*(\delta(e, \alpha), m)$$
- Propriété symétrique de la définition de  $\delta$ 

$$\forall e \in E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(e, m.\alpha) = \delta(\delta^*(e, m), \alpha)$$

Preuve : en TD
- Pour la fonction de transition itérée étendue :
  - Les formules précédentes s'étendent de  $e$  à  $E' \subseteq E$  :  

$$\forall E' \subseteq E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(E', m.\alpha) = \delta(\delta^*(E', m), \alpha)$$

## Résumé des propriétés de $\delta$

$$\begin{aligned} \forall E' \subseteq E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(E', m.\alpha) &= \delta(\delta^*(E', m), \alpha) \\ \forall E' \subseteq E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(E', \alpha.m) &= \delta^*(\delta(E', \alpha), m) \\ \forall E' \subseteq E, \forall m_1, m_2 \in \Sigma^*, \delta^*(E', m_1.m_2) &= \delta^*(\delta^*(E', m_1), m_2) \\ \delta^*(E', \alpha) &= \delta(E', \alpha) \quad \text{et} \quad \delta^*(E', \varepsilon) = E' \\ \delta^*(E', ---) &= \bigcup_{e \in E'} \delta^*(e, ---) \quad \delta(E', ---) = \bigcup_{e \in E'} \delta(e, ---) \\ \delta(e, \alpha) &= \bigcup_{e' \in \delta(e, \alpha)} \{e'\} = \bigcup_{(e, \alpha, e') \in \delta} \{e'\} \\ A \subseteq B &\Rightarrow \delta^*(A, ---) \subseteq \delta^*(B, ---) \quad A \subseteq B \Rightarrow \delta(A, ---) \subseteq \delta(B, ---) \end{aligned}$$

$$\delta^*(I, m) \cap F \neq \{\} \Leftrightarrow m \in L(A)$$

## État accessible et co-accessible

- Même définition et même propriété
  - Un état  $e$  est dit **accessible** s'il existe un chemin depuis un état initial jusqu'à  $e$ .
  - Un état  $e$  est dit **co-accessible** s'il existe un chemin depuis l'état  $e$  jusqu'à un état terminal.
  - Théorème : pour tout automate  $A$ , soit  $A'$  l'automate obtenu en supprimant dans  $A$  les états non accessibles et non co-accessibles. Alors  $A$  et  $A'$  sont équivalents
- Avec la fonction de transition itérée étendue :
  - L'état  $e$  est accessible si  $\exists m, e \in \delta^*(I, m)$
  - L'état  $e$  est co-accessible si  $\exists m, \delta^*(e, m) \cap F \neq \emptyset$

## Puissance des automates finis

- Définition : Un langage est dit **rationnel** s'il est définissable par un automate d'états fini
  - C'est un théorème dans d'autres approches  
Notation :  $\text{rec}(\Sigma) = \text{rat}(\Sigma) = \{ \text{langages rationnels sur } \Sigma \}$
  - Les langages algébriques (définis par une grammaire)  $\text{alg}(\Sigma)$  contiennent les langages rationnels  
 $\text{rat}(\Sigma) \subset \text{alg}(\Sigma)$
- Calculer si un mot est dans un langage défini par un automate :
  - Très simple et efficace si l'automate est déterministe
  - Plus lourd si l'automate est indéterministe

## Algorithme d'acceptation d'un mot (automate déterministe complet)

- La puissance des automates déterministes provient de l'efficacité du test d'appartenance d'un mot  $m_0$  au langage

$e \leftarrow i; m \leftarrow m_0; p \leftarrow \varepsilon;$

*tant que  $m \neq \varepsilon$  faire*

*/\* Invariant :  $m_0 = p.m$  et  $e = \delta^*(i, p)$  \*/*

$\alpha \leftarrow \text{premièreLettre}(m);$

$m \leftarrow \text{resteLettres}(m);$

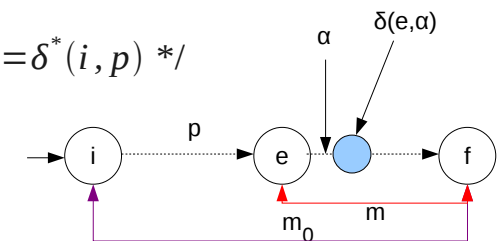
$e \leftarrow \delta(e, \alpha);$

$p \leftarrow p.\alpha;$

*fin /\*  $m = \varepsilon$  et donc  $e = \delta^*(i, m_0)$  \*/*

*retourner  $e \in F$*

Reconnaissance de  $m_0$



## Algorithme d'acceptation d'un mot (automate déterministe non complet)

$e \leftarrow i; m \leftarrow m_0; p \leftarrow \varepsilon;$

*tant que  $m \neq \varepsilon$  faire*

*/\* Invariant :  $m_0 = p.m$  et  $e = \delta^*(i, p)$  \*/*

$\alpha \leftarrow \text{premièreLettre}(m);$

$m \leftarrow \text{resteLettres}(m);$

*si  $\delta(e, \alpha)$  est défini alors*

$e \leftarrow \delta(e, \alpha);$

$p \leftarrow p.\alpha;$

*sinon*

*retourner faux;*

*fin /\*  $m = \varepsilon$  (si on sort à la fin de la boucle) \*/*

*retourner  $e \in F$*

## Algorithme d'acceptation d'un mot (automate indéterministe)

$E' \leftarrow I; m \leftarrow m_0; p \leftarrow \varepsilon;$

*tant que  $m \neq \varepsilon$  faire*

*/\* Invariant :  $m_0 = p.m$  et  $E' = \delta^*(i, p)$  \*/*

$\alpha \leftarrow \text{premièreLettre}(m);$

$m \leftarrow \text{resteLettres}(m);$

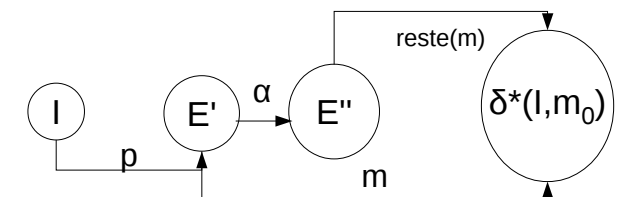
$E'' \leftarrow \emptyset$

*pour tout  $e' \in E'$  faire  $E'' \leftarrow E'' \cup \delta(e', \alpha);$*

$E' \leftarrow E'';$

*fin /\*  $m = \varepsilon$  \*/*

*retourner  $E' \cap F \neq \emptyset$*



# Les automates indéterministes avec $\epsilon$ -transitions

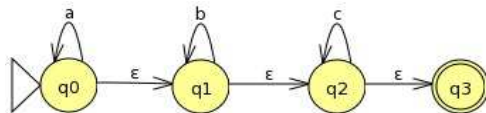
## Automates

- ☒ Automates déterministes
- ☒ Automates indéterministes
- ☐ Automates avec  $\epsilon$ -transitions
- ☐ Transformations d'automates

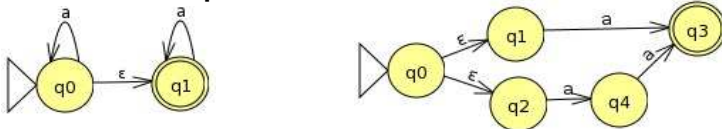
## Introduction

- Accepter des transitions ne « consommant » pas de lettres (et appelés des  $\epsilon$ -transitions) :

$$L = \{ a^n b^m c^p \mid n \geq 0, m \geq 0, p \geq 0 \}$$



- Intuitivement :  $L = \{ a^n \epsilon b^m \epsilon c^p \epsilon \mid n \geq 0, m \geq 0, p \geq 0 \}$
- Une  $\epsilon$ -transition peut induire de l'indéterminisme

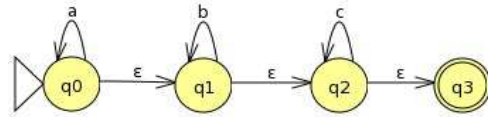


## Définition

- Un **automate avec  $\epsilon$ -transitions** est la donnée d'un quintuplet  $A = (\Sigma, E, I, F, \delta)$ 
  - $\Sigma$  est l'alphabet (d'entrée)
  - $E$  est un ensemble fini d'éléments appelés des **états**
  - $I$  est une partie de  $E$ , dont les éléments sont appelé les **“états initiaux”**
  - $F$  est une partie de  $E$ , dont les éléments sont appelés des **“états terminaux”**
  - $\delta$  est une fonction de  $E \times (\Sigma \cup \{\epsilon\})$  vers  $P(E)$ , appelée **fonction de transition**

## La fonction de transition

- $L = \{ a^n b^m c^p \mid n \geq 0, m \geq 0, p \geq 0 \}$



- $\delta : E \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(E)$

$$\begin{aligned} \delta(q_0, a) &= \{q_0\} & \delta(q_1, b) &= \{q_1\} & \delta(q_2, c) &= \{q_2\} \\ \delta(q_0, \varepsilon) &= \{q_1\} & \delta(q_1, \varepsilon) &= \{q_2\} & \delta(q_2, \varepsilon) &= \{q_3\} \end{aligned}$$

- Représentation visuelle :

$\varepsilon$  apparaît comme une lettre de l'alphabet

## Chemin et trace

- Mêmes notions de chemin et trace

- $(e_1 \alpha_1 e_2 \alpha_2 e_3 \alpha_3 e_4 \alpha_4 \dots \alpha_{n-1} e_n)$  est un **chemin** si  $e_{i+1} \in \delta(e_i, \alpha_i)$  pour tout entier  $i$  dans  $\{1, \dots, n-1\}$
- $\forall i \in \{1, \dots, n-1\}, \alpha_i \in \Sigma \cup \{\varepsilon\}$
- $\alpha_1 \alpha_2 \alpha_3 \alpha_4 \dots \alpha_{n-1}$  est la **trace** du chemin précédent.
- Les occurrences de  $\varepsilon$  dans la trace sont simplifiées.

## Langage associé à un automate avec $\varepsilon$ -transitions

- Soit  $A = (\Sigma, E, I, F, \delta)$  un automate.
  - Le langage reconnu par cet automate  $A$ , noté  $L_A$  ou  $L(A)$  est l'ensemble des mots  $m$  tel qu'il existe un chemin entre un état initial et un état terminal, et dont la trace (**simplifiée de ses mots vides  $\varepsilon$  inutiles**) est  $m$ .

$$m \in L_A \Leftrightarrow \left\{ \begin{array}{l} \exists (e_1, \alpha_1, e_2, \dots, e_n, \alpha_n, e_{n+1}) \\ \text{de trace } \alpha_1 \alpha_2 \dots \alpha_n = m \text{ avec } e_{k+1} \in \delta(e_k, \alpha_k) \end{array} \right\}$$

$\begin{array}{ccc} \in I & & \in F \\ \downarrow & & \downarrow \end{array}$

## Fonction de transition itérée étendue

- Définition : Soit  $A = (\Sigma, E, I, F, \delta)$  un automate avec  $\varepsilon$ -transitions. La **fonction de transition itérée étendue**, notée  $\delta^*$ , est l'application de  $P(E) \times \Sigma^*$  vers  $P(E)$  qui vérifie :

$$\begin{aligned} \forall E' \subseteq E, \quad \delta^*(E', \varepsilon) &= \hat{e}(E') \\ \forall E' \subseteq E, \forall \alpha \in \Sigma, \quad \delta^*(E', \alpha) &= \hat{e}(\delta(\hat{e}(E'), \alpha)) \\ \forall E' \subseteq E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \quad \delta^*(E', \alpha.m) &= \delta^*(\delta^*(E', \alpha), m) \end{aligned}$$

$\delta^*$  étendue !

$$\hat{e}(E') = \{f \in E, \exists \text{ un chemin d'un état de } E' \text{ à } f, \text{ et de trace } \varepsilon\}$$

- $\delta^*(E', m)$  est l'ensemble des états accédés à partir des états  $e'$  de  $E'$  par un chemin de trace  $m$



## Remarques sur la définition

- Fonction de transition itérée (non étendue)
    - $\delta^*$  restreint à  $E \times \Sigma^*$  étendue
    - $\forall E' \subseteq E, \forall m \in \Sigma^*, \delta^*(E', m) = \bigcup_{e' \in E'} \delta^*(e', m)$  non étendue
  - Même notation  $\delta^*$  (et de même pour  $\delta$ )
  - $\delta$  s'étend à  $P(E)$  :  $\delta(E', \alpha) = \delta(\bigcup_{e \in E'} \{e\}, \alpha) = \bigcup_{e \in E'} \delta(e, \alpha)$
  - Nécessité des 2 cas limites pour  $\delta^*(E', \alpha)$  et  $\delta^*(E', \varepsilon)$ 
    - $\forall E' \subseteq E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(E', \alpha.m) = \delta^*(\delta^*(E', \alpha), m)$
- Appliqué à  $m = \varepsilon$ , cela donne :
- $$\delta^*(E', \alpha.\varepsilon) = \delta^*(\delta^*(E', \alpha), \varepsilon) = \hat{\varepsilon}(\delta^*(E', \alpha)) \Rightarrow \text{impasse}$$

## Un exemple

- $\delta^*(E', m)$  est l'ensemble des états accédés à partir des états  $e'$  de  $E'$  par un chemin de trace  $m$
- 
- $\hat{\varepsilon}(e) = \{e, 1\}$   
 $\delta(e, \alpha) = \{4\}$  et  $\delta(1, \alpha) = \{2\}$   
 $\hat{\varepsilon}(4) = \{4, 5, 6\}$  et  $\hat{\varepsilon}(2) = \{2, 3, 6\}$   
 $\delta^*(e, \alpha) = \{4, 5, 6, 2, 3\}$
- 
- $\forall E' \subseteq E, \delta^*(E', \varepsilon) = \hat{\varepsilon}(E')$   
 $\forall E' \subseteq E, \forall \alpha \in \Sigma, \delta^*(E', \alpha) = \hat{\varepsilon}(\delta(\hat{\varepsilon}(E'), \alpha))$   
 $\forall E' \subseteq E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(E', \alpha.m) = \delta^*(\delta^*(E', \alpha), m)$

## Fermeture transitive des $\varepsilon$ -transitions

- Définition : la **fermeture transitive des  $\varepsilon$ -transitions** (ou  **$\varepsilon$ -fermeture**) est une fonction, **notée**  $\hat{\varepsilon}$ , définie de  $E$  vers  $P(E)$  telle que :
 
$$\hat{\varepsilon}(e) = \{f \in E \mid \exists \text{ un chemin de } e \text{ à } f, \text{ et de trace } \varepsilon\}$$
- Extension à  $P(E)$  :  $\hat{\varepsilon}(E') = \bigcup_{e' \in E'} \hat{\varepsilon}(e')$ 

$$\hat{\varepsilon}(E') = \{f \in E \mid \exists \text{ un chemin d'un état de } E' \text{ à } f, \text{ et de trace } \varepsilon\}$$
- Vision inversée (de celle de la définition de  $\delta^*$ ) :
 
$$\forall e' \in E, \hat{\varepsilon}(e') = \delta^*({e'}, \varepsilon)$$

$$\forall E' \subseteq E, \hat{\varepsilon}(E') = \delta^*(E', \varepsilon)$$
- Remarque :  $\hat{\varepsilon}(\hat{\varepsilon}(E')) = \hat{\varepsilon}(E')$   $\hat{\varepsilon}$  est une fermeture !

## Construction de $\hat{\varepsilon}$

- Pour tout état  $e$ , construction successive des ensembles d'états, notés  $\hat{\varepsilon}_i(e)$  accessibles par au plus  $i$   $\varepsilon$ -transitions
 
$$\hat{\varepsilon}_0(e) = \{e\}$$

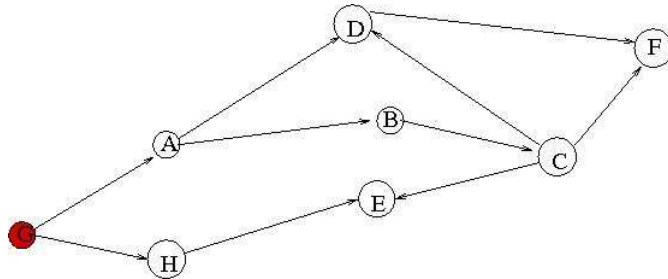
$$\hat{\varepsilon}_{i+1}(e) = \hat{\varepsilon}_i(e) \cup \delta(\hat{\varepsilon}_i(e), \varepsilon)$$
- La séquence des  $\hat{\varepsilon}_i(e)$  est croissante  $\hat{\varepsilon}_i(e) \subseteq \hat{\varepsilon}_{i+1}(e)$  et majorée par  $E$ . Elle est donc stationnaire.
  - Elle est stationnaire au pire pour  $i = |E| - 1$
- $$\hat{\varepsilon}(e) = \bigcup_{i \in \mathbb{N}} \hat{\varepsilon}_i(e)$$

$$= \hat{\varepsilon}_k(e) \quad \text{où } k = \min_{i \in \mathbb{N}} (i \text{ tq } \hat{\varepsilon}_i(e) = \hat{\varepsilon}_{i+1}(e))$$

## Exemple de construction (1)

Que des  $\varepsilon$ -transitions

$$\hat{\varepsilon}_0(G) = \{G\}$$



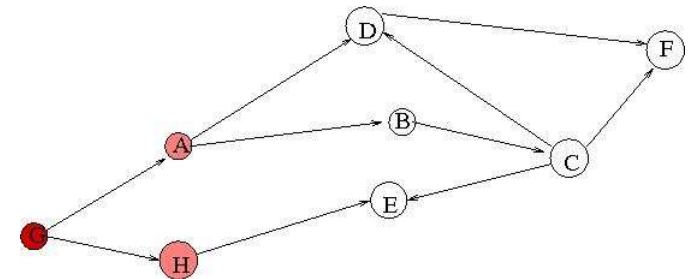
$$\hat{\varepsilon}_{i+1}(e) = \hat{\varepsilon}_i(e) \cup \delta(\hat{\varepsilon}_i(e), \varepsilon)$$

## Exemple de construction (2)

Que des  $\varepsilon$ -transitions

$$\hat{\varepsilon}_0(G) = \{G\}$$

$$\hat{\varepsilon}_1(G) = \{G, A, H\}$$



$$\hat{\varepsilon}_{i+1}(e) = \hat{\varepsilon}_i(e) \cup \delta(\hat{\varepsilon}_i(e), \varepsilon)$$

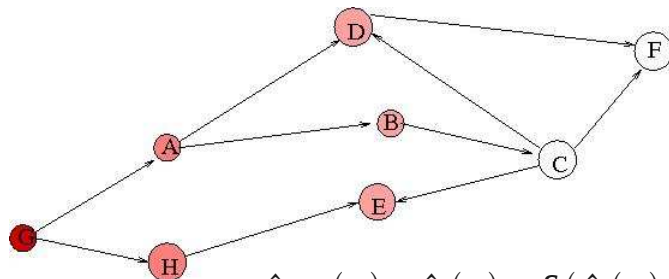
## Exemple de construction (3)

Que des  $\varepsilon$ -transitions

$$\hat{\varepsilon}_0(G) = \{G\}$$

$$\hat{\varepsilon}_1(G) = \{G, A, H\}$$

$$\hat{\varepsilon}_2(G) = \{G, A, H, D, B, E\}$$



$$\hat{\varepsilon}_{i+1}(e) = \hat{\varepsilon}_i(e) \cup \delta(\hat{\varepsilon}_i(e), \varepsilon)$$

## Exemple de construction (4)

Que des  $\varepsilon$ -transitions

$$\hat{\varepsilon}_0(G) = \{G\}$$

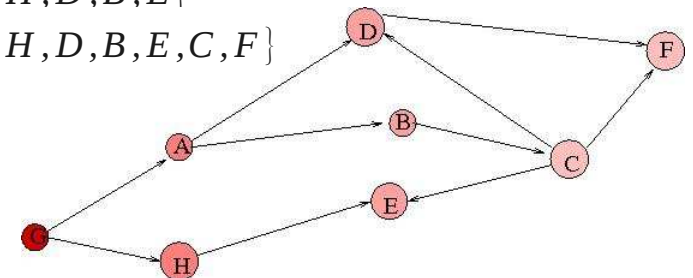
$$\hat{\varepsilon}_1(G) = \{G, A, H\}$$

$$\hat{\varepsilon}_2(G) = \{G, A, H, D, B, E\}$$

$$\hat{\varepsilon}_3(G) = \{G, A, H, D, B, E, C, F\}$$

$$\hat{\varepsilon}_4(G) = \hat{\varepsilon}_3(G)$$

$$\hat{\varepsilon}(G) = \hat{\varepsilon}_3(G)$$



$$\hat{\varepsilon}_{i+1}(e) = \hat{\varepsilon}_i(e) \cup \delta(\hat{\varepsilon}_i(e), \varepsilon)$$

## Propriétés des transitions itérées

$$\delta^*(l, m) \cap F \neq \emptyset \Leftrightarrow m \in L(A)$$

Définition de  $L(A)$  sans chemins ni traces

- $\delta^*(e, m) \cap F \neq \emptyset$  les mots  $m$  reconnus à partir de  $e$
- $\delta^*(l, m) \ni e$  les mots  $m$  reconnus en arrivant en  $e$

$$\forall E' \subseteq E, \forall \alpha \in \Sigma, \forall m \in \Sigma^*, \delta^*(E', m.\alpha) = \delta^*(\delta^*(E', m), \alpha)$$

- Même définition des états accessibles et co-accessibles

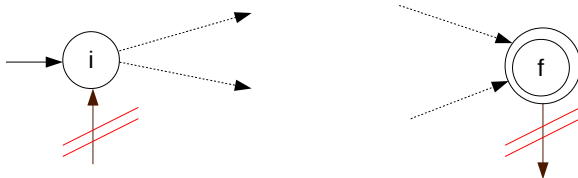
(QUASI) LES MEMES PROPRIETES QUE POUR UN AUTOMATE INDETERMINISTE SANS  $\varepsilon$ -TRANSITIONS

## Compositions d'automates

- Construire les automates réalisant :
  - L'union de deux langages définis par un automate
  - La concaténation " " "
  - Le complémentaire d'un langage défini par un automate
  - La fermeture de Kleene " " "
- Partir d'automates « composables » :
  - Les automates standards
- En déduire des résultats généraux :
  - L'union, la concaténation, le complémentaire, ... de langages rationnels sont rationnels
    - Car reconnus par un automate que l'on construit !

## Automates standards

- Un automate  $A = (\Sigma, E, I, F, \delta)$  est dit **standard** si :
  - $I = \{i\}$  i.e.  $I$  est un singleton
  - $F = \{f\}$  i.e.  $F$  est un singleton
  - $\forall e \in E, \forall \alpha \in \Sigma \cup \{\varepsilon\} : (e, \alpha, i) \notin \delta$  et  $(f, \alpha, e) \notin \delta$



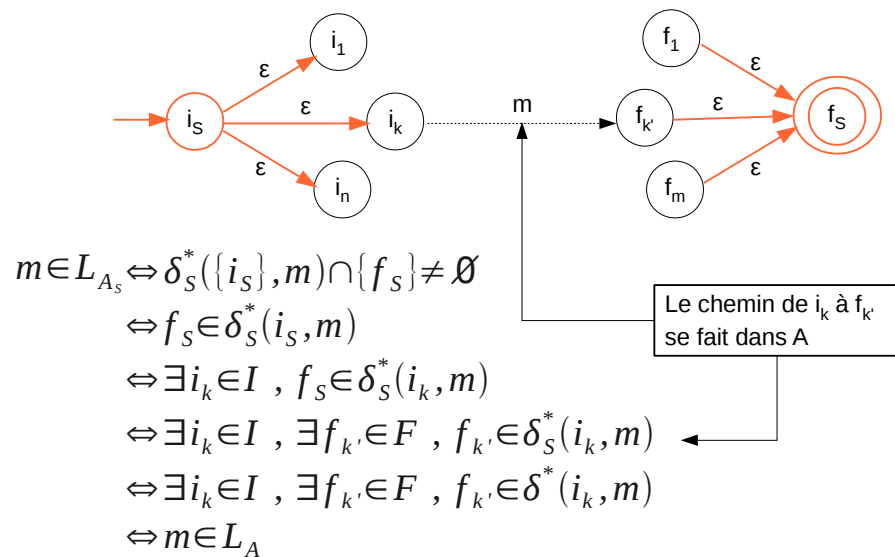
## Automate --> Automate standard

- Automate standard  $A_S$  associé à l'automate  $A = (\Sigma, E, I, F, \delta)$  qui reconnaît le même langage :
  - $A_S = (\Sigma, E \cup \{i_S, f_S\}, \{i_S\}, \{f_S\}, \delta_S)$   
et  $\delta_S = \delta \cup \{(i_S, \varepsilon, i_k) \mid i_k \in I\} \cup \{(f_k, \varepsilon, f_S) \mid f_k \in F\}$
  - Dé-qualification de tous les états initiaux et terminaux



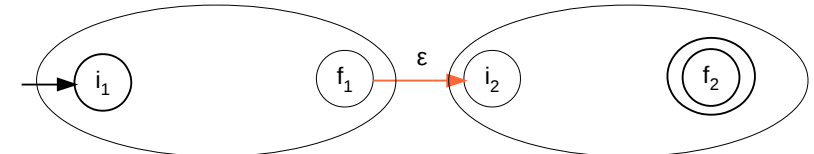
- C'est bien un automate standard

## Automate --> Automate standard



## Concaténation d'automates

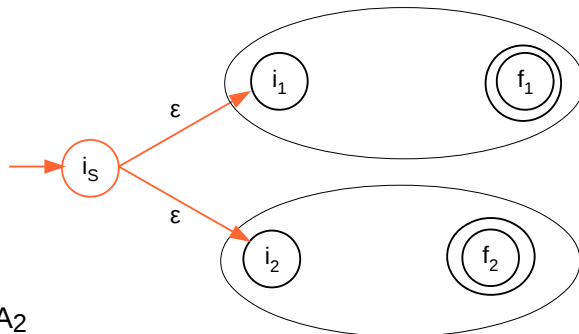
- Concaténation de deux automates standards
  - $A_1 = (\Sigma, E_1, \{i_1\}, \{f_1\}, \delta_1)$  et  $A_2 = (\Sigma, E_2, \{i_2\}, \{f_2\}, \delta_2)$
  - $A_{1.2} = (\Sigma, E_1 \cup E_2, \{i_1\}, \{f_2\}, \delta_1 \cup \delta_2 \cup \{(f_1, \varepsilon, i_2)\})$



- $L_{A_{1.2}} = L_{A_1} \cdot L_{A_2}$  et c'est aussi un automate standard
- Extension aux automates quelconques :
  - Construire préalablement leur automate standard

## Union de deux automates

- Union de deux automates standards
  - $A_1 = (\Sigma, E_1, \{i_1\}, F_1, \delta_1)$  et  $A_2 = (\Sigma, E_2, \{i_2\}, F_2, \delta_2)$
  - $A_{1 \cup 2} = (\Sigma, E_1 \cup E_2 \cup \{i_s\}, \{i_s\}, F_1 \cup F_2, \delta_{1 \cup 2})$
  - $\delta_{1 \cup 2} = \delta_1 \cup \delta_2 \cup \{(i_s, \varepsilon, i_1), (i_s, \varepsilon, i_2)\}$



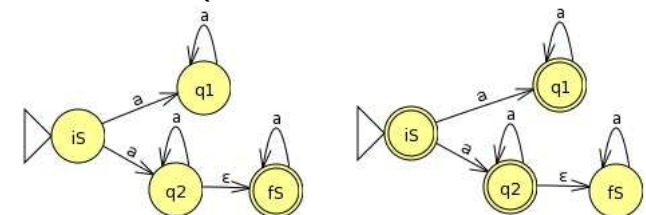
- $L_{A_{1 \cup 2}} = L_{A_1} \cup L_{A_2}$   
 $A_{1 \cup 2}$  n'est pas un automate standard : ajouter un état  $f_s$

## Complémentaire d'un automate

- L'automate doit être déterministe et complet
  - Tout automate est « déterminisable » et « complétable »
- L'**automate complémentaire** de A est l'automate  $A^c = (\Sigma, E, I, E-F, \delta)$
- Le complémentaire du langage  $L_A$  dans  $\Sigma^*$  est le langage associé à l'automate complémentaire  $A^c$ 
  - Faux si indéterministe (même si complet) :

$$L_A = \{a\}^+$$

$$L_{A^c} = \{a\}^*$$



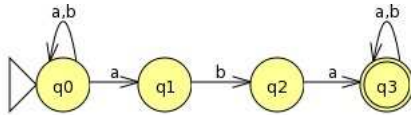
## Intersection d'automates

- Via l'union et le complémentaire :  $C_{\Sigma^*}^{A \cap B} = C_{\Sigma^*}^A \cup C_{\Sigma^*}^B$

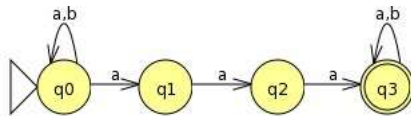
- Exemple :

- Les mots qui contiennent les facteurs aba et aaa

- $A_{aba}$  :



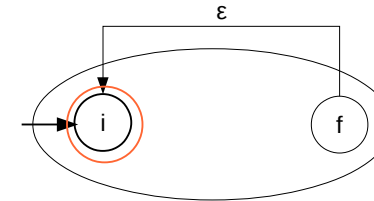
- $A_{aaa}$  :



- Déterminiser, compléter, puis complémentaire de  $A_{aba} A_{aaa}$
- Union des 2 précédents, puis le déterminiser, compléter, et prendre son complémentaire

## Fermeture de Kleene d'un automate

- Soit  $A = (\Sigma, E, \{i\}, \{f\}, \delta)$  un automate standard, l'automate réalisant la fermeture de Kleene de A est l'automate  $A^* = (\Sigma, E, \{i\}, \{i\}, \delta \cup \{(f, \varepsilon, i)\})$



L'état f n'est plus terminal

$$L_{A^*} = (L_A)^*$$

- Justification :  $m \in L_{A^*} \Leftrightarrow \underbrace{i \dots f}_{m_1} \underbrace{\varepsilon i \dots f}_{m_2} \underbrace{\varepsilon i \dots f}_{m_n} \varepsilon i$

$$m \in L_{A^*} \Leftrightarrow \exists m_1 \in L_A, \dots, \exists m_n \in L_A \text{ tel que } m = m_1 \dots m_n$$

$$\Leftrightarrow \exists n \geq 0, m \in (L_A)^n \Leftrightarrow m \in L_{A^*}$$

## Théorème de Kleene

- Théorème (Kleene) :** L'ensemble des langages reconnus par un automate fini (appelés langages rationnels) est la fermeture transitive des langages réduits à une lettres ou au mot vide, pour les opérations de concaténation, union et fermeture de Kleene
- Construit par fermeture  $\Rightarrow$  reconnu par un automate :
  - Tout langage réduit à un mot d'une lettre se construit simplement par un automate trivial.
  - Si  $L_1$  et  $L_2$  sont deux langages rationnels, alors les langages  $L_1 \cup L_2$ ,  $L_1 \cdot L_2$  et  $L_1^*$  sont rationnels
- Reconnu par un automate  $\Rightarrow$  construit par fermeture :
  - Résultera de la transformation « automate  $\rightarrow$  expr reg »

## Algorithme d'acceptation d'un mot (automate avec $\varepsilon$ -transitions)

$E' \leftarrow I; m \leftarrow m_0; p \leftarrow \varepsilon;$

*tant que*  $m \neq \varepsilon$  *faire*

/\* Invariant :  $m_0 = p.m$  et  $E' = \delta^*(i, p)$  \*/

$\alpha \leftarrow \text{premièreLettre}(m);$

$m \leftarrow \text{resteLettres}(m);$

$E'' \leftarrow \emptyset$

*pour tout*  $e' \in E'$  *faire*  $E'' \leftarrow E'' \cup \hat{\varepsilon}(\delta(\hat{\varepsilon}(e'), \alpha));$

$E' \leftarrow E'';$

*fin* /\*  $m = \varepsilon$  \*/

*retourner*  $E' \cap F \neq \emptyset$

La fonction  $\delta$  étendue

# Automates

## Transformations d'automates

☑ Automates déterministes

☑ Automates indéterministes

☑ Automates avec  $\varepsilon$ -transitions

☐ Transformations d'automates

☐ Automate avec  $\varepsilon$ -transitions  $\implies$  Automate indéterministe

☐ Automate indéterministe  $\implies$  Automate déterministe

☐ Automate déterministe  $\implies$  Automate minimal

• Tout automate est équivalent à un automate déterministe

• Justification des autres automates : simplicité de modélisation

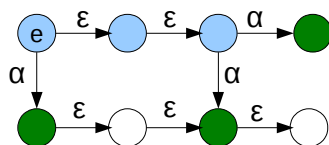
## Suppression des $\varepsilon$ -transitions

- Soit  $A_\varepsilon = (\Sigma, E, I, F_\varepsilon, \delta_\varepsilon)$  un automate avec  $\varepsilon$ -transitions. L'automate sans  $\varepsilon$ -transitions qui reconnaît le même langage que  $A_\varepsilon$  est défini par :

$$A = (\Sigma, E, I, F, \delta) \quad F = \{e \in E \mid \hat{\varepsilon}(e) \cap F_\varepsilon \neq \emptyset\}$$

$$\forall e \in E, \forall \alpha \in \Sigma, \quad \delta(e, \alpha) = \delta_\varepsilon(\hat{\varepsilon}(e), \alpha)$$

- Raccourcis : « les  $\varepsilon$ -transitions puis la transition  $\alpha$  »



$\hat{\varepsilon}(e)$  en **bleu**  
 $\delta_\varepsilon(\hat{\varepsilon}(e), \alpha)$  en **vert**

- Autres approches :

$$\delta(e, \alpha) = \hat{\varepsilon}(\delta_\varepsilon(e, \alpha))$$

$$\delta(e, \alpha) = \hat{\varepsilon}(\delta_\varepsilon(\hat{\varepsilon}(e), \alpha))$$

## Pourquoi cela fonctionne ?

- Principe :

• Remplacer les :

• Par des :

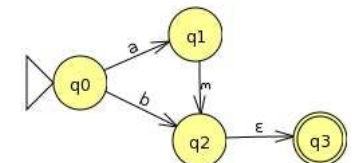
• Correct en traitant tous les raccourcis possibles.

- États terminaux :  $F = \{e \in E \mid \hat{\varepsilon}(e) \cap F_\varepsilon \neq \emptyset\}$

• Pas seulement ceux de  $F_\varepsilon$ , mais en plus ceux qui sont séparés d'un état terminal par des  $\varepsilon$ -transitions

$$F_\varepsilon = \{q3\}$$

$$F = \{q1, q2, q3\}$$



## Exemple

- Soit l'automate suivant :

$$\hat{\varepsilon}(q0) = \{q0, q1\}$$

$$\hat{\varepsilon}(q1) = \{q1\}$$

$$\hat{\varepsilon}(q2) = \{q2\}$$

$$\hat{\varepsilon}(q3) = \{q3, q0, q1\}$$

$$\delta(q0, a) = \delta_{\varepsilon}(\hat{\varepsilon}(q0), a) = \delta_{\varepsilon}(\{q0, q1\}, a) = \{q2\}$$

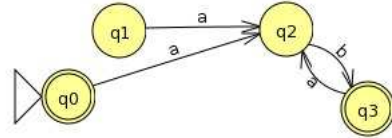
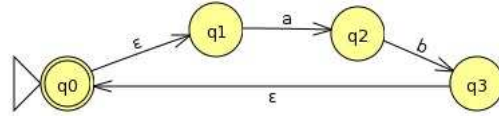
$$\delta(q1, a) = \delta_{\varepsilon}(\hat{\varepsilon}(q1), a) = \delta_{\varepsilon}(\{q1\}, a) = \{q2\}$$

$$\delta(q2, b) = \delta_{\varepsilon}(\hat{\varepsilon}(q2), b) = \delta_{\varepsilon}(\{q2\}, b) = \{q3\}$$

$$\delta(q3, a) = \delta_{\varepsilon}(\hat{\varepsilon}(q3), a) = \delta_{\varepsilon}(\{q3, q0, q1\}, a) = \{q2\}$$

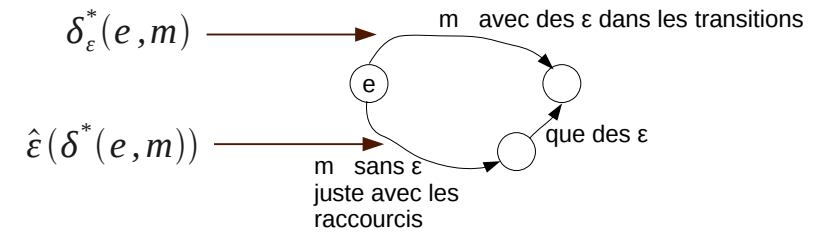
$$F = \{e \mid \hat{\varepsilon}(e) \cap \{q0\} \neq \emptyset\}$$

$$= \{q0, q3\}$$



## Preuve de l'équivalence : $A_{\varepsilon} \sim A$

- Théorème :  $\forall m \in \Sigma^*, \forall e \in E, \delta_{\varepsilon}^*(e, m) = \hat{\varepsilon}(\delta^*(e, m))$



- Preuve par récurrence sur  $|m|$

$$\Pi(n) = \forall m \in \Sigma^*, |m| \leq n, \forall e \in E, \delta_{\varepsilon}^*(e, m) = \hat{\varepsilon}(\delta^*(e, m))$$

$$\Pi(0) \text{ vrai : } \delta_{\varepsilon}^*(e, \varepsilon) = \hat{\varepsilon}(e) \text{ et } \hat{\varepsilon}(\delta^*(e, \varepsilon)) = \hat{\varepsilon}(\{e\})$$

$$\Pi(1) \text{ vrai : } \delta_{\varepsilon}^*(e, \alpha) \stackrel{\text{def } \delta_{\varepsilon}^*}{=} \hat{\varepsilon}(\delta_{\varepsilon}(\hat{\varepsilon}(e), \alpha)) \stackrel{\text{def } \delta}{=} \hat{\varepsilon}(\delta(e, \alpha))$$

$$\hat{\varepsilon}(\delta^*(e, \alpha)) \stackrel{\text{prop de } \delta^*}{=} \hat{\varepsilon}(\delta(e, \alpha)) \quad "$$

## Preuve de la récurrence

$$\Pi(n) = \forall m \in \Sigma^*, |m| \leq n, \forall e \in E, \delta_{\varepsilon}^*(e, m) = \hat{\varepsilon}(\delta^*(e, m))$$

- Hypothèse :  $\Pi(n)$  vrai. Soit  $|m| = n+1$  et  $m = m' . \alpha$

$$\delta_{\varepsilon}^*(e, m' . \alpha) = \delta_{\varepsilon}^*(\underbrace{\delta_{\varepsilon}^*(e, m')}_u, \alpha) \quad \text{par déf de } \delta_{\varepsilon}^*$$

$$= \hat{\varepsilon}(\delta_{\varepsilon}(\hat{\varepsilon}(u), \alpha)) \quad \text{par déf de } \delta_{\varepsilon}^*$$

$$= \hat{\varepsilon}(\delta_{\varepsilon}(\hat{\varepsilon}(\underbrace{\delta_{\varepsilon}^*(e, m')}_u), \alpha))$$

$$= \hat{\varepsilon}(\delta_{\varepsilon}(\hat{\varepsilon}(\underbrace{\hat{\varepsilon}(\delta^*(e, m'))}_u), \alpha)) \quad \text{par hyp de rec}$$

$$= \hat{\varepsilon}(\delta_{\varepsilon}(\hat{\varepsilon}(\delta^*(e, m')), \alpha)) \quad \text{car } \hat{\varepsilon}(\hat{\varepsilon}(e)) = \hat{\varepsilon}(e)$$

$$= \hat{\varepsilon}(\delta(\delta^*(e, m'), \alpha)) \quad \text{par déf de } \delta$$

$$\hat{\varepsilon}(\delta^*(e, m' . \alpha)) = \hat{\varepsilon}(\delta^*(e, m' . \alpha)) \quad \text{par déf de } \delta^*$$

Ordre des  
calculs  
délicat ...

## Dernier étape de la preuve

- Étendre :  $\forall m \in \Sigma^*, \forall e \in E, \delta_{\varepsilon}^*(e, m) = \hat{\varepsilon}(\delta^*(e, m))$   
aux états initiaux :  $\forall m \in \Sigma^*, \delta_{\varepsilon}^*(I, m) = \hat{\varepsilon}(\delta^*(I, m))$

$$m \in L_{A_{\varepsilon}} \Leftrightarrow \delta_{\varepsilon}^*(I, m) \cap F_{\varepsilon} \neq \emptyset$$

$$\Leftrightarrow \hat{\varepsilon}(\delta^*(I, m)) \cap F_{\varepsilon} \neq \emptyset$$

$$\Leftrightarrow \exists x \text{ tel que } x \in \delta^*(I, m) \text{ et } \hat{\varepsilon}(x) \cap F_{\varepsilon} \neq \emptyset$$

$$\Leftrightarrow \exists x \text{ tel que } x \in \delta^*(I, m) \text{ et } x \in \{e, \hat{\varepsilon}(e) \cap F_{\varepsilon} \neq \emptyset\} = F$$

$$\Leftrightarrow \delta^*(I, m) \cap F \neq \emptyset$$

$$\Leftrightarrow m \in L_A$$



# Automates

☑ Automates déterministes

☑ Automates indéterministes

☑ Automates avec  $\varepsilon$ -transitions

## ☐ Transformations d'automates

☑ Automate avec  $\varepsilon$ -transitions ==> Automate indéterministe

☐ Automate indéterministe ==> Automate déterministe

☐ Automate déterministe ==> Automate minimal

# Déterminisation

- Construction d'un automate  $A_d = (\Sigma, E_d, i_d, F_d, \delta_d)$  déterministe à partir d'un automate indéterministe (sans  $\varepsilon$ -transitions)  $A_i = (\Sigma, E_i, I_i, F_i, \delta_i)$  tel que :

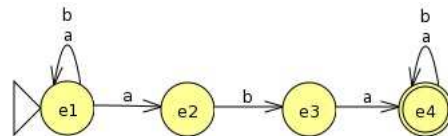
$$L_{A_d} = L_{A_i}$$

- $E_d = P(E_i)$  Remarque :  $|E_d| = 2^{|E_i|}$
- $i_d = I_i$  Danger :  $i_d$  état vs.  $I_i$  ensemble d'états
- $F_d = \{E' \in P(E_i) \mid E' \cap F_i \neq \emptyset\}$
- $\delta_d$  est la fonction  $\delta_i$  étendue :

$$\forall E' \in E_d, \forall \alpha \in \Sigma, \delta_d(E', \alpha) = \delta_i(E', \alpha)$$

## Un exemple plus parlant

• Soit l'automate :



• Déterminiser :

- À partir d'un état  $e$  et pour une lettre  $\alpha$ , déterminer l'ensemble des états auxquels on accède en lisant  $\alpha$

$(e1, a) \rightarrow \{e1, e2\}$	Nouvel état « e1-ou-e2 »
$(e1, b) \rightarrow \{e1\}$	
$(\{e1, e2\}, a) \rightarrow \{e1, e2\}$	$(\{e1, e2\}, b) \rightarrow \{e1, e3\}$
$(\{e1, e3\}, a) \rightarrow \{e1, e2, e4\}$	$(\{e1, e3\}, b) \rightarrow \{e1\}$
$(\{e1, e2, e4\}, a) \rightarrow \{e1, e2, e4\}$	$(\{e1, e2, e4\}, b) \rightarrow \{e1, e3, e4\}$
$(\{e1, e3, e4\}, a) \rightarrow \{e1, e2, e4\}$	$(\{e1, e3, e4\}, b) \rightarrow \{e1, e4\}$
$(\{e1, e4\}, a) \rightarrow \{e1, e2, e4\}$	$(\{e1, e4\}, b) \rightarrow \{e1, e4\}$

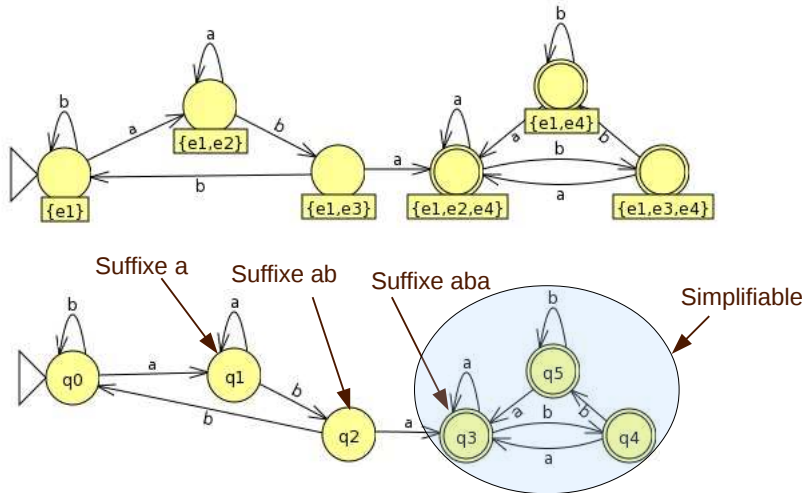
## Représentation de l'automate

- 
- |  |  |
|--|--|
| $(\{e1\}, a) \rightarrow \{e1, e2\}$             | $(\{e1\}, b) \rightarrow \{e1\}$                 |
| $(\{e1, e2\}, a) \rightarrow \{e1, e2\}$         | $(\{e1, e2\}, b) \rightarrow \{e1, e3\}$         |
| $(\{e1, e3\}, a) \rightarrow \{e1, e2, e4\}$     | $(\{e1, e3\}, b) \rightarrow \{e1\}$             |
| $(\{e1, e2, e4\}, a) \rightarrow \{e1, e2, e4\}$ | $(\{e1, e2, e4\}, b) \rightarrow \{e1, e3, e4\}$ |
| $(\{e1, e3, e4\}, a) \rightarrow \{e1, e2, e4\}$ | $(\{e1, e3, e4\}, b) \rightarrow \{e1, e4\}$     |
| $(\{e1, e4\}, a) \rightarrow \{e1, e2, e4\}$     | $(\{e1, e4\}, b) \rightarrow \{e1, e4\}$         |
- C'est la fonction de transition étendue de  $A_i$  et la fonction de transition du nouvel automate  $A_d$
- États terminaux : ceux qui contiennent un élément de  $F = \{e4\}$



## Nettoyage de l'automate

- Renommer les états et leur donner du sens



- Simplifier l'automate obtenue : minimalisation !

## Retour sur la formalisation de $A_d$

- Soit  $A_i = (\Sigma, E, I, F_i, \delta_i)$  un automate indéterministe  
Soit  $A_d = (\Sigma, P(E), I, F_d, \delta_d)$  l'automate tel que :

- $F_d = \{ E' \in P(E) \mid E' \cap F_i \neq \emptyset \}$

- $\delta_d : P(E) \times \Sigma \rightarrow P(E)$

La fonction  $\delta_i$  étendue

$$(E', \alpha) \rightarrow \delta_d(E', \alpha) = \delta_i(E', \alpha)$$

- Attention à bien différencier :

- $A_i$  a comme ENSEMBLE d'états initiaux :  $I$
- $A_d$  a comme UNIQUE état initial :  $I$
- $\delta_d(E', \alpha)$  est un UNIQUE état de  $A_d$   
et est un ENSEMBLE d'états de  $A_i$

## Preuve

- Soit  $A_i = (\Sigma, E, I, F_i, \delta_i)$  et  $A_d = (\Sigma, P(E), I, F_d, \delta_d)$   
avec :  $\forall E' \in P(E), \forall \alpha \in \Sigma, \delta_d(E', \alpha) = \delta_i(E', \alpha)$
- Lemme :  $\forall E' \in P(E), \forall m \in \Sigma^*, \delta_d^*(E', m) = \delta_i^*(E', m)$   
Par récurrence sur  $|m|$ . C'est trivialement vrai si  $|m| = 1$   
Si  $m = \varepsilon$ ,  $\delta_d^*(E', \varepsilon) = E'$  et  $\delta_i^*(E', \varepsilon) = U_{e' \in E'} \delta_i^*(e', \varepsilon) = U_{e' \in E'} \{e'\} = E'$   
Hyp : vrai si  $|m| = n$ . Soit  $|m| = n+1$ , alors  $m = m'.\alpha$  et :  

$$\begin{aligned} \delta_d^*(E', m) &= \delta_d^*(E', m'.\alpha) = \delta_d(\delta_d^*(E', m'), \alpha) && \text{par déf de } \delta_d^* \\ &= \delta_d(\delta_i^*(E', m'), \alpha) && \text{par hyp de rec} \\ &= \delta_i(\delta_i^*(E', m'), \alpha) && \text{par déf de } \delta_d \\ &= \delta_i^*(E', m'.\alpha) && \text{par déf de } \delta_i^* \\ &= \delta_i^*(E', m) \end{aligned}$$

## Fin de la preuve

- À prouver :  $L_{A_d} = L_{A_i}$  sachant :  $F_d = \{ E' \in P(E) \mid E' \cap F_i \neq \emptyset \}$
- Lemme :  $\forall E' \in P(E), \forall m \in \Sigma^*, \delta_d^*(E', m) = \delta_i^*(E', m)$
- Notons :  $\delta_d^*(I, m) = \delta_i^*(I, m) = \{e_1, e_2, \dots, e_k\}$
- Alors :  $m \in L_{A_i} \Leftrightarrow \delta_i^*(I, m) \cap F_i \neq \emptyset$   
Et :  $m \in L_{A_d} \Leftrightarrow \delta_d^*(I, m) \in F_d$   

$$\begin{aligned} &\Leftrightarrow \{e_1, e_2, \dots, e_k\} \cap F_i \neq \emptyset \\ &\Leftrightarrow \{e_1, e_2, \dots, e_k\} \in F_d \end{aligned}$$

# Automates

- ☑ Automates déterministes
- ☑ Automates indéterministes
- ☑ Automates avec  $\varepsilon$ -transitions
- ☐ Transformations d'automates
  - ☑ Automate avec  $\varepsilon$ -transitions  $\Rightarrow$  Automate indéterministe
  - ☑ Automate indéterministe  $\Rightarrow$  Automate déterministe
  - ☐ Automate déterministe  $\Rightarrow$  Automate minimal

# Minimalisation d'un automate

- Minimalisation en nombre d'états d'un automate A
  - Déterministe ( $\delta$  est une fonction)
  - Complet ( $\delta$  est une application)
  - Monogène (tout état est accessible)
  - Non trivial ( $L_A$  n'est ni  $\{\}$  ni  $\Sigma^*$ )
- Tout automate non trivial a un équivalent ayant ces propriétés
- Unicité de l'automate minimal à un renommage près
- Cela permet de comparer deux automates quelconques

## Intuition de la minimalisation

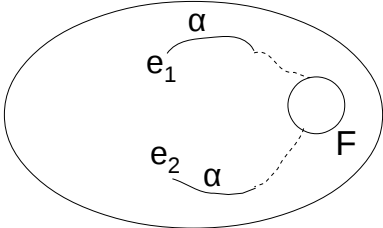
- Si les états  $e_1$  et  $e_2$  « engendrent » les mêmes mots, alors arriver en  $e_1$  ou  $e_2$  revient au même:  $e_1$  et  $e_2$  sont fusionnables
 

$$\{ m \mid \delta^*(e_1, m) \in F \}$$

$$\parallel$$

$$\{ m \mid \delta^*(e_2, m) \in F \}$$

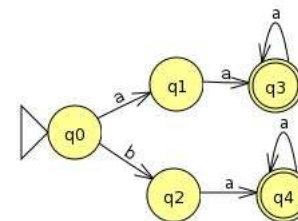
i.e.  $\forall m, \delta^*(e_1, m) \in F \Leftrightarrow \delta^*(e_2, m) \in F$


- Définition : les états  $e_1$  et  $e_2$  sont distinguables si
 

$$\exists m, \delta^*(e_1, m) \in F \Leftrightarrow \delta^*(e_2, m) \notin F$$

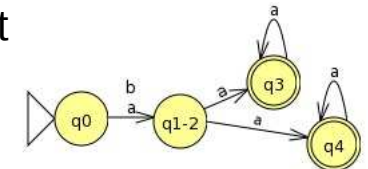
Un tel mot m est dit séparer les états  $e_1$  et  $e_2$

## Exemple simple de fusion

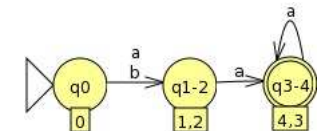


q1 et q2 fusionnables  
 q3 et q4 fusionnables  
 q0 et q1 séparés par ba  
 q0 et q2 séparés par ba  
 q1 et q3 séparés par  $\varepsilon$

- Fusionner 2 états est correct
  - Fusionner juste q1 et q2



- Automate minimal :  
 quand tous les états fusionnables auront été fusionnés



## Relation de N rode

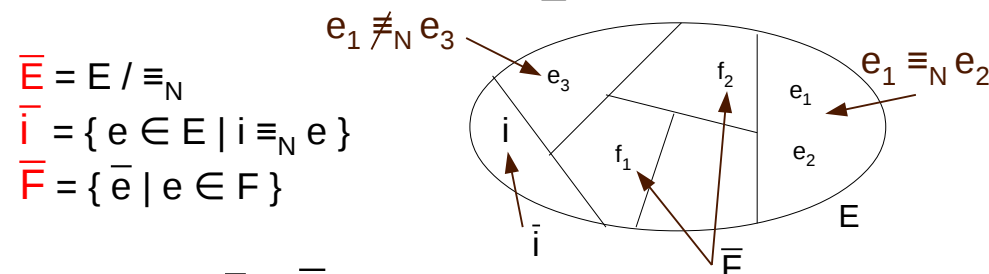
- D finition : soit  $A = (\Sigma, E, i, F, \delta)$  un automate d terministe. La **relation de N rode** est la relation binaire, not e  $\equiv_N$ , d finie sur  $E$  par :

$$\begin{aligned} e_1 \equiv_N e_2 &\Leftrightarrow e_1 \text{ et } e_2 \text{ ne sont pas distinguables} \\ &\Leftrightarrow \forall m, \delta^*(e_1, m) \in F \Leftrightarrow \delta^*(e_2, m) \in F \end{aligned}$$

- La relation de N rode est une relation d' quivalence
  - R flexive ( $e \equiv_N e$ ), sym trique ( $e_1 \equiv_N e_2 \Rightarrow e_2 \equiv_N e_1$ ) et transitive ( $e_1 \equiv_N e_2$  et  $e_2 \equiv_N e_3 \Rightarrow e_1 \equiv_N e_3$ )
- $E$  est partitionn  en classes d' quivalence  
L'ensemble des classes d' quivalence est not   $\bar{E}$   
 $\bar{E} = \{ \bar{e} \mid e \in E \}$   $e$  est un **repr sentant** de  $\bar{e}$

## Automate quotient

- Soit  $A = (\Sigma, E, i, F, \delta)$  un automate d terministe et complet. L'**automate quotient  par la relation de N rode** est l'automate  $\bar{A} = (\Sigma, \bar{E}, \bar{i}, \bar{F}, \bar{\delta})$  avec :



- Remarque :  $\bar{e} \in \bar{F} \Leftrightarrow e \in F$   
car les  l ments de  $\bar{e}$  ne sont pas s par s par  $\varepsilon$   
 $\bar{F} = \{ \{f_{i1}, \dots, f_{in}\}, \dots, \{f_{k1}, \dots, f_{kp}\} \}$

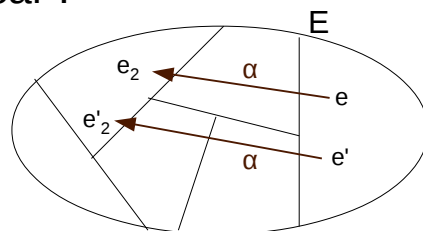
## La transition quotient

- $\bar{\delta}$  est d finie sur  $\bar{E} \times \Sigma$  par :  $\bar{\delta}(\bar{e}, \alpha) = \overline{\delta(e, \alpha)}$
- Cette d finition a du sens car :

$$\bar{e} = \bar{e'} \Rightarrow \overline{\delta(e, \alpha)} = \overline{\delta(e', \alpha)}$$

i.e.

$$e \equiv_N e' \Rightarrow \delta(e, \alpha) \equiv_N \delta(e', \alpha)$$



Preuve par contraposition : si  $\delta(e, \alpha) \not\equiv_N \delta(e', \alpha)$

alors  $\exists m, \delta^*(\delta(e, \alpha), m) \in F$  et  $\delta^*(\delta(e', \alpha), m) \notin F$

$\Rightarrow \exists m, \delta^*(e, \alpha.m) \in F$  et  $\delta^*(e', \alpha.m) \notin F$

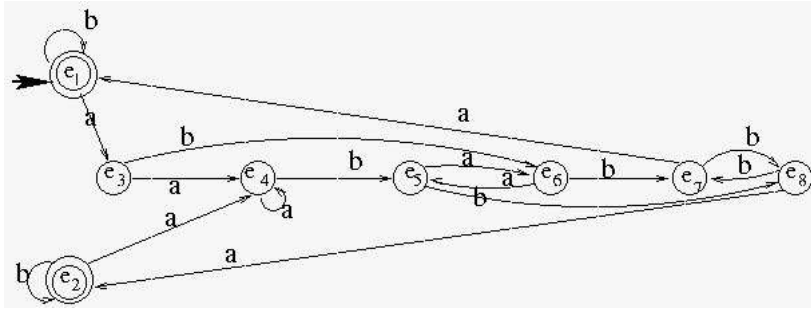
$\Rightarrow e$  et  $e'$  sont s par s par  $\alpha.m$

$\Rightarrow e \not\equiv_N e'$

## Propri t s de l'automate quotient

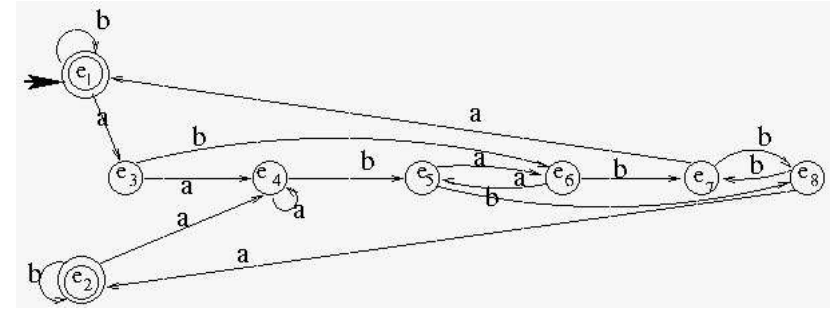
- $\delta(e, \alpha) = e' \Rightarrow \bar{\delta}(\bar{e}, \alpha) = \bar{e'}$  (par d finition de  $\bar{\delta}$ )
- $\bar{A}$  est complet  
 $\bar{\delta}$  est d finie sur **tout**  $\bar{E} \times \Sigma$  par :  $\bar{\delta}(\bar{e}, \alpha) = \overline{\delta(e, \alpha)}$   
car  $\delta(e, \alpha)$  est d fini sur tout  $E \times \Sigma$  ( $A$  complet)
- $\bar{A}$  est d terministe  
car  $\bar{\delta}(\bar{e}, \alpha) = \overline{\delta(e, \alpha)}$  d termine un unique  l ment de  $\bar{E}$
- Les langages reconnus par  $A$  et  $\bar{A}$  sont identiques  
 $L_{\bar{A}} = L_A$  d monstration    tablir...
- $\bar{A}$  est minimal en nombre d' tats parmi les automates d terministes complets reconnaissant  $L_A$

## Construction de $\bar{A}$



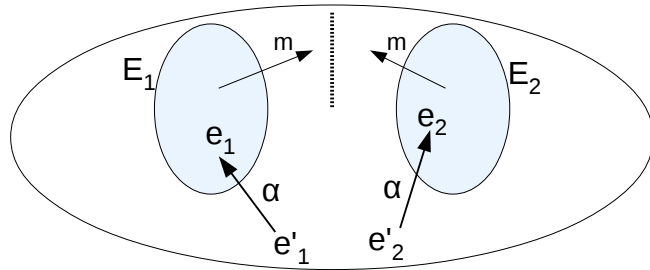
- Construire  $\bar{A}$  c'est surtout déterminer  $\equiv_N$
- $e_i \equiv_N e_j$  si on ne peut trouver un mot qui les sépare
  - Chercher à séparer les états en envisageant les mots de longueur 0, 1, 2, ...
  - Au final, deux états qui n'auront pu être séparés seront fusionnables.

## Construction de $\equiv_N$



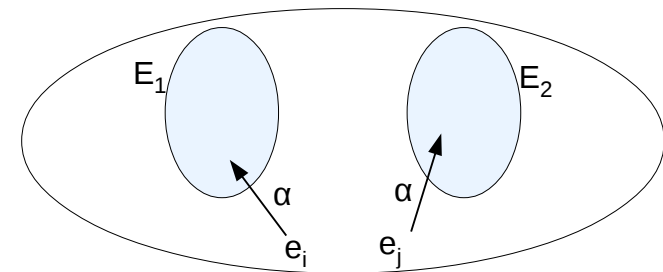
- Le mot (de longueur 0)  $\varepsilon$  sépare :
  - $E_1 = F = \{e_1, e_2\}$  de  $E_2 = \{e_3, e_4, e_5, e_6, e_7, e_8\}$
  - $\delta^*(e_1, \varepsilon) \in F$  et  $\delta^*(e_3, \varepsilon) \notin F \implies e_1$  et  $e_3$  séparés
- Les mots (de longueur 1)  $a$  et  $b$  séparent :
  - $E_{2.1} = \{e_3, e_4, e_5, e_6\}$  et  $E_{2.2} = \{e_7, e_8\}$
  - $\delta^*(e_7, a) \in F$  et  $\delta^*(e_3, a) \notin F \implies e_7$  et  $e_3$  séparés

## Itération des séparations



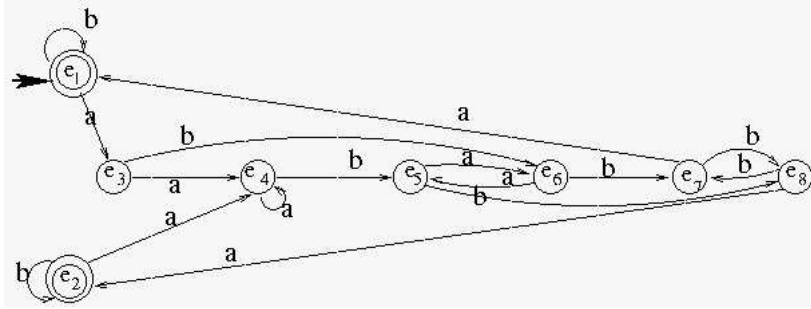
- $E_1$  et  $E_2$  séparés par le mot  $m$  :
 
$$\delta^*(e_1, m) \in F \Leftrightarrow \delta^*(e_2, m) \notin F$$
- $\delta(e'_1, \alpha) = e_1 \Rightarrow e'_1$  et  $e'_2$  séparés par  $\alpha.m$   
 $\delta(e'_2, \alpha) = e_2$
- Car  $\delta^*(e'_1, \alpha.m) = \delta^*(\delta(e'_1, \alpha), m) = \delta^*(e_1, m)$   
 et  $\delta^*(e'_2, \alpha.m) = \delta^*(\delta(e'_2, \alpha), m) = \delta^*(e_2, m)$

## Itération des séparations (2)



- Construction progressive des parties  $E_i$
- Calcul de tous les  $\delta(e_i, \alpha)$  et détection des cas où  $e_i$  et  $e_j$  appartiennent à une même classe et  $\delta(e_i, \alpha)$  et  $\delta(e_j, \alpha)$  appartiennent à deux classes différentes. Cela induit la séparation de  $e_i$  et  $e_j$

## Application à l'exemple



- Écrire explicitement tous les  $\delta(e_i, \alpha)$  ...
- $E_1 = \{e_1, e_2\}$        $E_2 = \{e_3, e_4, e_5, e_6, e_7, e_8\}$
- $E_1 = \{e_1, e_2\}$      $E_{2.1} = \{e_3, e_4, e_5, e_6\}$      $E_{2.2} = \{e_7, e_8\}$
- $E_1 = \{e_1, e_2\}$      $E_{2.1.1} = \{e_3, e_4\}$      $E_{2.1.2} = \{e_5, e_6\}$      $E_{2.2} = \{e_7, e_8\}$ 
  - Séparation :  $\delta(e_3, b) = e_6 \in E_{2.1}$  et  $\delta(e_5, b) = e_8 \in E_{2.2}$

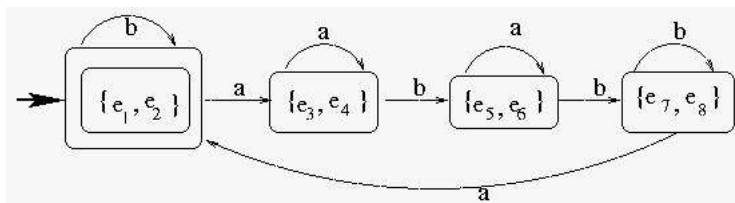
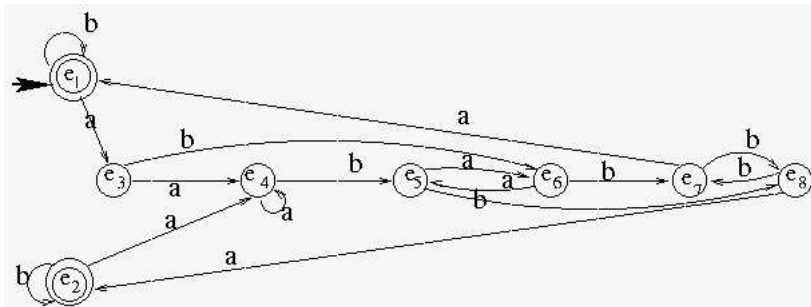
## Construction de $\bar{A}$

- $E_1 = \{e_1, e_2\}$      $E_{2.1.1} = \{e_3, e_4\}$      $E_{2.1.2} = \{e_5, e_6\}$      $E_{2.2} = \{e_7, e_8\}$ 
  - Plus de séparation possible.
  - $\bar{E} = \{\{e_1, e_2\}, \{e_3, e_4\}, \{e_5, e_6\}, \{e_7, e_8\}\}$
- $\bar{\delta}(\bar{e}_i, \alpha) = \overline{\delta(e_i, \alpha)}$ 
  - Exemple :  $\bar{\delta}(\{e_3, e_4\}, b) = \{e_5, e_6\}$
- État initial :  $\{\bar{e}_1, \bar{e}_2\}$  car  $e_1$  est l'état initial de A
- États terminaux :  $\bar{F} = \{\bar{e}_1\} \cup \{\bar{e}_2\}$  car  $F = \{e_1, e_2\}$ 

$$= \{\{e_1, e_2\}\} \cup \{\{e_1, e_2\}\}$$

$$= \{\{e_1, e_2\}\} \quad 1 \text{ seul élément}$$

## Automates A et $\bar{A}$

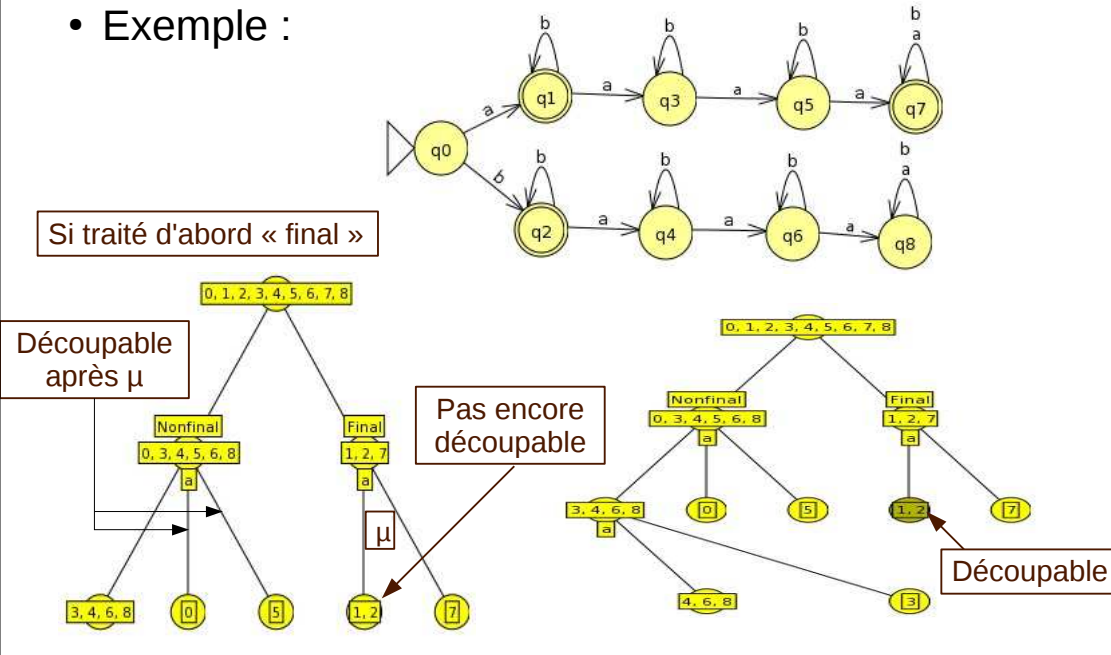


## Ordre de séparation des états

- L'ordre est sans importance
  - **Ordre par niveau** : au niveau i, tous les états séparables par des mots de longueur inférieure ou égale à i ont été réalisés.
  - **Ordre suivi par jflap** : un ensemble d'états est découpé par une lettre donnée en plusieurs sous parties.
- Le processus de séparation termine forcément
  - Au plus  $|E|-1$  niveaux
- L'important c'est de (re)traiter toutes les parties jusqu'à ce qu'aucune partie ne soit découppable

# Ordre de séparation (avec jflap)

- Exemple :



## Minimalité de $\bar{A}$

- $\bar{A}$  est minimal en nombre d'états.
  - Théorème** : Soit A un automate déterministe, complet, monogène et non trivial. Alors, il n'existe pas d'automate déterministe complet, reconnaissant le langage  $L(A)$ , ayant moins d'états que l'automate  $\bar{A}$ .
  - Preuve** : en considérant les résiduels de  $L(A)$
- Corollaire** : Pour tout langage rationnel L, il existe un unique automate (au nom des états près) déterministe complet avec un nombre minimum d'états qui reconnaisse le langage L
  - Cela donne une méthode pour tester l'égalité de deux langages définis par un automate.

## Expressions rationnelles (ou régulières)

## Introduction

- Outil très pratique pour décrire un « motif » ou « ensemble de mots », i.e. un langage
  - Sous Linux : « ls \*.jpg »
  - Recherche dans un logiciel :  
motif du mot/fichier cherché : « prog\*.??? »
  - Notion de « regexp » pour « expression régulière »
- La syntaxe ne sera pas tout à fait la même et sera plus riche dans les environnements de programmation et sous linux.
  - $(a.b)^*$  ---> une séquence de  $aXb$  où  $X$  est un caractère quelconque (sous Linux)
  - $(a.b)^*$  ---> une séquence de  $ab$  (dans ce cours)

## Expressions rationnelles

- Définition récursive : Une expression rationnelle (ou régulière) ER, définie sur un alphabet  $\Sigma$  est une expression définie inductivement par :
  - $\varepsilon$  est une ER (langage réduit à 1 mot :  $\varepsilon$ )
  - $\{\}$  est une ER (langage vide)
  - Toute lettre de  $\Sigma$  est une ER
  - $(r)$  est une ER si  $r$  l'est
  - $r+s$  est une ER si  $r$  et  $s$  le sont (pour l'union)
  - $r.s$  ou  $rs$  est une ER si  $r$  et  $s$  le sont
  - $r^*$  est une ER si  $r$  l'est

## Parenthésage et notations

- Exemples d'ER :
  - $(a+b)^*aba(a+b)^*$  Les mots ayant le facteur  $aba$
  - $0+1(00+10+01+11)^*$  Les nombres binaires avec un nombre impair de chiffres
- Ambiguïté des expressions rationnelles
  - $ab+c$  est-il :  $a \ b+c$  ou  $ab + c$  ?
  - Utilisation des règles de priorité pour lever les ambiguïtés : «  $*$  » est plus prioritaire que «  $.$  » qui est plus prioritaire que «  $+$  ».  
Exemple :  $ab+c^* = (ab)+(c^*)$
- Ajout de notations :
  - $r+ = r.r^*$        $r \mid s = r+s$        $r ? = (r + \varepsilon)$



## Langage associé à une expression régulière

- Définition : soit  $r$  une expression régulière. Le langage, noté  $L(r)$ , associé à  $r$  est défini inductivement par :

- $L(\{\}) = \{\}$  et  $L(\varepsilon) = \{\varepsilon\}$
- $L(\alpha) = \{\alpha\}$  pour tout  $\alpha \in \Sigma$
- $L((r)) = L(r)$
- $L(r+s) = L(r) \cup L(s)$
- $L(r.s) = L(rs) = L(r).L(s)$
- $L(r^*) = (L(r))^*$

## Exemples

- $r = (a+b)^*aba(a+b)^*$  Les mots ayant le facteur aba

$$\begin{aligned} L(r) &= (\{a\} \cup \{b\})^* \{a\} \{b\} \{a\} (\{a\} \cup \{b\})^* \\ &= (\{a, b\})^* \{a\} \{b\} \{a\} (\{a, b\})^* \\ &= \Sigma^* \{aba\} \Sigma^* \end{aligned}$$

- $r = 0+1(00+10+01+11)^*$  Les nombres binaires avec un nombre impair de chiffres

$$\begin{aligned} L(r) &= \{0\} \cup \{1\}(\{0\}\{0\} \cup \{1\}\{0\} \cup \{0\}\{1\} \cup \{1\}\{1\})^* \\ &= \{0\} \cup \{1\}\{00, 10, 01, 11\}^* \end{aligned}$$

## CNS pour que $m \in L(e)$

- $m \in L((a+b)^*aba(a+b)^*)$ 

$$\Leftrightarrow m \in L((a+b)^*) L(a) L(b) L(a) L((a+b)^*)$$

$$\Leftrightarrow m \in (\{a\} \cup \{b\})^* \{a\} \{b\} \{a\} (\{a\} \cup \{b\})^*$$

$$\Leftrightarrow m \in (\{a, b\})^* \{a\} \{b\} \{a\} (\{a, b\})^*$$

$$\Leftrightarrow m = (\alpha_1 \dots \alpha_n) aba (\alpha'_1 \dots \alpha'_n)$$

avec  $\alpha_i \in \{a, b\}$  et  $\alpha'_i \in \{a, b\}$
- Il n'y a pas une formule générale comme pour les automates ou les grammaires
- On peut réduire les équivalences et obtenir directement l'égalité finale.

## Expressions régulières équivalentes (même langage associé)

- Simplifications (pour toute expr. reg. «  $r$  »)
  - $\varepsilon r = r \varepsilon = r$                        $\{\} r = r \{\} = \{\}$
  - $\{\} + r = r$                                  $(\varepsilon + r)^* = (\varepsilon + r^*) = r^*$
  - $\{\}^* = \varepsilon^* = \varepsilon$                          $r + r = r$
- Associativité de la concaténation
  - $(ab)c = a(bc) = abc$
- Distributivité :
  - $a(b+c) = ab + ac$
- Commutativité de l'addition
  - $(a+b) = (b+a)$

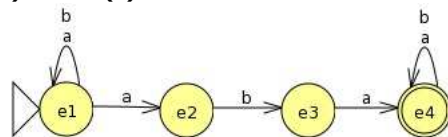


# Équivalence entre automate et ER

- À tout automate A correspond une expression rationnelle r telle que  $L(r) = L(A)$
- À toute expression rationnelle r correspond un automate A tel que  $L(A) = L(r)$

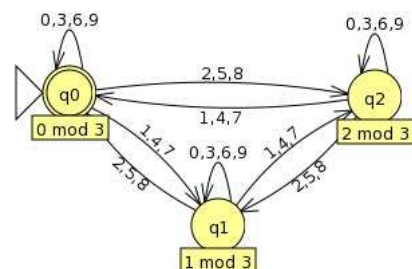
Exemple simple :

$$r = (a+b)^*aba(a+b)^*$$



Exemple pas simple :

expression associée ?

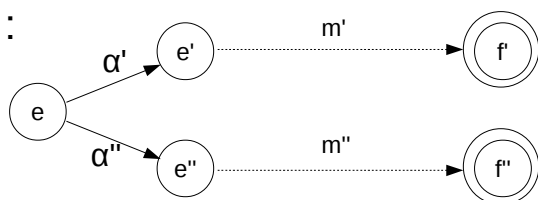


## ER associée à un automate

- L'automate peut être indéterministe et avec des  $\epsilon$ -transitions, mais doit être **émondé** (tout état est accessible et co-accessible).
- Deux méthodes :
  - Par variation des états d'entrée
    - Pour tout état e :  $L_e = \{ m \mid \delta^*(e, m) \cap F \neq \emptyset \}$
    - Sens de l'état e : « les mots reconnus à partir de e »
  - Par variation des états de sortie
    - Pour tout état e :  $L_e = \{ m \mid \delta^*(I, m) \cap \{e\} \neq \emptyset \}$
    - Sens de l'état e : « les mots reconnus jusqu'à e »

## Par variation des états d'entrée

Intuition :



- Les mots reconnus à partir de l'état e :
  - $L_e = \{\alpha'\}.L_{e'} \cup \{\alpha''\}.L_{e''}$  (même si  $e'=e$ ,  $\alpha'=\alpha''$ )
- Si e est un état terminal :
  - $L_e = \{\epsilon\} \cup \{\alpha'\}.L_{e'} \cup \{\alpha''\}.L_{e''}$
- Langage associé à l'automate :  $\bigcup_{e \in I} L_e$

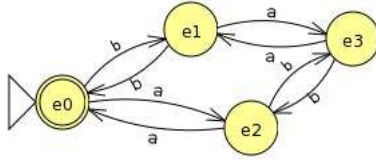
## Système d'équations associées

- Bijection entre les équations définissant le langage  $L_e$  et les équations définissant l'expression rationnelle  $R_e$

$$\begin{array}{ccc}
 L_e = \{\epsilon\} \cup \bigcup_{(e, \alpha, e') \in \delta} \{\alpha\}.L_{e'} & & L_e = \bigcup_{(e, \alpha, e') \in \delta} \{\alpha\}.L_{e'} \\
 \updownarrow & & \updownarrow \\
 R_e = \epsilon + \sum_{(e, \alpha, e') \in \delta} \alpha.R_{e'} & & R_e = \sum_{(e, \alpha, e') \in \delta} \alpha.R_{e'}
 \end{array}$$

- $L_e = L(R_e)$  Non unicité de  $R_e$  associée à  $L_e$
- Écriture du système d'équations incluant tous les  $R_e$ 
  - Résolution du système d'équation dans une algèbre spécifique avec des règles spécifiques
- Expression recherchée :  $\sum_{e \in I} R_e$   $I = \{\text{états initiaux}\}$

## Exemple



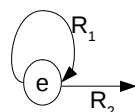
- $R_{e0} = bR_{e1} + aR_{e2} + \varepsilon$        $R_{e0}$  est final
- $R_{e1} = aR_{e3} + bR_{e0}$
- $R_{e2} = aR_{e0} + bR_{e3}$
- $R_{e3} = aR_{e1} + bR_{e2}$
- 4 équations et 4 inconnues :  $R_{e0}$  ,  $R_{e1}$  ,  $R_{e2}$  ,  $R_{e3}$
- Expression recherchée :  $R_{e0}$     seul état initial

## Règles de transformation

### • Règle de substitution

- $R_e$  peut être remplacée par « sa valeur » dans les partie droite des autres équations.
- $R_e = aR_{e'} + bR_{e''}$
- $R_x = \dots R_e \dots \implies R_x = \dots aR_{e'} + bR_{e''} \dots$

### • Règle du point fixe

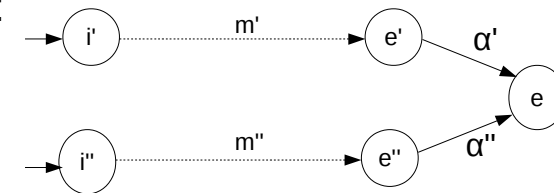
- $R_e = R_1.R_e + R_2 \implies R_e = R_1^*.R_2$    
où  $R_1$  et  $R_2$  sont des expr. rat. **sans  $R_e$**
- Intuition :  $R_e = R_1.R_e + R_2 = R_1 \dots R_1.R_e + R_2 = R_1 \dots R_1.R_2$
- Autres règles :  $R_1 + R_2 = R_2 + R_1$      $R(R_1 + R_2) = RR_1 + RR_2$   
 $R . \varepsilon = \varepsilon . R = R$     etc

## Application des règles sur l'exemple

- $R_{e0} = bR_{e1} + aR_{e2} + \varepsilon$        $R_{e2} = aR_{e0} + bR_{e3}$   
 $R_{e1} = aR_{e3} + bR_{e0}$        $R_{e3} = aR_{e1} + bR_{e2}$
- Substituer  $R_{e1}$  et  $R_{e2}$  par leur « valeur » dans  $R_{e0}$  ,  $R_{e3}$ 
  - $R_{e0} = (ba+ab)R_{e3} + (bb+aa)R_{e0} + \varepsilon$
  - $R_{e3} = (aa+bb)R_{e3} + (ab+ba)R_{e0}$
- Règle du point fixe pour  $R_{e3}$ 
  - $R_{e3} = (aa+bb)^*(ab+ba)R_{e0}$
- Substitution de  $R_{e3}$  et règle du point fixe pour  $R_{e0}$ 
  - $R_{e0} = (ba+ab) (aa+bb)^*(ab+ba)R_{e0} + (bb+aa)R_{e0} + \varepsilon$
  - $R_{e0} = ((ba+ab) (aa+bb)^*(ab+ba) + (bb+aa))^* . \varepsilon$

## Par variation des états de sortie

### • Intuition :



- Les mots reconnus jusqu'à l'état e :
  - $L_e = L_{e'} . \{\alpha'\} \cup L_{e''} . \{\alpha''\}$  (même si  $e'=e$  ,  $\alpha'=\alpha''$ )
- Si e est un état initial :  
 $L_e = \{\varepsilon\} \cup L_{e'} . \{\alpha'\} \cup L_{e''} . \{\alpha''\}$
- Langage associé à l'automate :  $\bigcup_{f \in F} L_f$

# Système d'équations associées

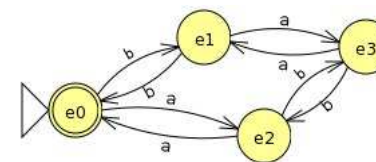
- Bijection entre les équations définissant le langage  $L_e$  et les équations définissant l'expression rationnelle  $R_e$

$$L_e = \{\varepsilon\} \cup \bigcup_{(e', \alpha, e) \in \delta} L_{e'} \cdot \{\alpha\} \quad L_e = \bigcup_{(e', \alpha, e) \in \delta} L_{e'} \cdot \{\alpha\}$$

$$\begin{array}{c} e \in I \\ \updownarrow \\ R_e = \varepsilon + \sum_{(e', \alpha, e) \in \delta} R_{e'} \cdot \alpha \end{array} \quad \begin{array}{c} e \notin I \\ \updownarrow \\ R_e = \sum_{(e', \alpha, e) \in \delta} R_{e'} \cdot \alpha \end{array}$$

- $L_e = L(R_e)$  Non unicité de  $R_e$  associée à  $L_e$
- Écriture du système d'équations incluant tous les  $R_e$ 
  - Résolution du système d'équation dans une algèbre spécifique avec des règles spécifiques
- Expression recherchée :  $\sum_{e \in F} R_e$   $F = \{\text{états terminaux}\}$

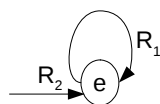
# Exemple



- $R_{e0} = R_{e1} b + R_{e2} a + \varepsilon$   $R_{e0}$  est initial
- $R_{e1} = R_{e3} a + R_{e0} b$
- $R_{e2} = R_{e0} a + R_{e3} b$
- $R_{e3} = R_{e1} a + R_{e2} b$
- 4 équations et 4 inconnues :  $R_{e0}, R_{e1}, R_{e2}, R_{e3}$
- Expression recherchée :  $R_{e0}$  seul état final

# Règles de transformation

## Règle du point fixe



- $R_e = R_e \cdot R_1 + R_2 \implies R_e = R_2 \cdot R_1^*$   
où  $R_1$  et  $R_2$  sont des expr. rat. **sans  $R_e$**

- Intuition :

$$\begin{aligned} R_e = R_e \cdot R_1 + R_2 &= R_e \cdot R_1 \cdot R_1 + R_2 = R_e \cdot R_1 \dots R_1 + R_2 \\ &= R_2 \cdot R_1 \dots R_1 \end{aligned}$$

- Les autres règles sont encore valides

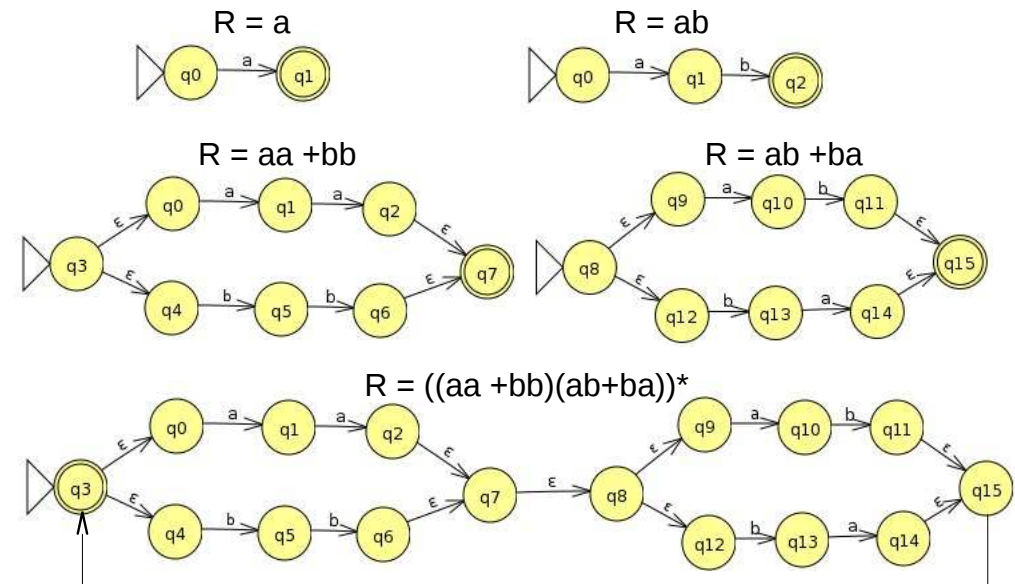
# Application des règles sur l'exemple

- $R_{e0} = R_{e1} b + R_{e2} a + \varepsilon$   $R_{e2} = R_{e0} a + R_{e3} b$   
 $R_{e1} = R_{e3} a + R_{e0} b$   $R_{e3} = R_{e1} a + R_{e2} b$
- Substituer  $R_{e1}$  et  $R_{e2}$  par leur « valeur » dans  $R_{e0}, R_{e3}$ 
  - $R_{e0} = R_{e3} (ba+ab) + R_{e0} (bb+aa) + \varepsilon$
  - $R_{e3} = R_{e3} (aa+bb) + R_{e0} (ab+ba)$
- Règle du point fixe pour  $R_{e3}$ 
  - $R_{e3} = R_{e0} (ab+ba)(aa+bb)^*$
- Substitution de  $R_{e3}$  et règle du point fixe pour  $R_{e0}$ 
  - $R_{e0} = R_{e0} (ab+ba)(aa+bb)^* (ba+ab) + R_{e0} (bb+aa) + \varepsilon$
  - $R_{e0} = ((ab+ba)(aa+bb)^* (ba+ab) + (bb+aa))^*$

## Automate associée à une ER

- Construction de l'automate avec  $\varepsilon$ -transitions
  - Avec des automates intermédiaires standards
  - Avec les constructions d'unions, de fermetures de Kleene et de concaténations d'automates déjà vues
  - Et avec des automates associés aux lettres
- En simplifiant éventuellement en cours de construction
- Exemple :  $r = ((ab+ba)(aa+bb))^*$ 
  - $L(r) = ( (\{a\}.\{b\} \cup \{b\}.\{a\}) . (\{a\}.\{a\} \cup \{b\}.\{b\}) )^*$
  - Construction d'un automate associé à  $L(r)$ 
    - à partir des automates associés à  $\{a\}$  et  $\{b\}$

## Automates associés

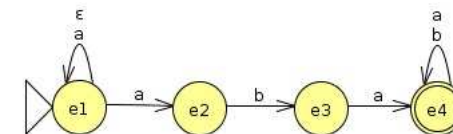


## Théorème de Kleene

- Théorème (Kleene) :** L'ensemble des langages reconnus par un automate fini est la fermeture transitive des langages réduits à une lettres ou au mot vide, pour les opérations de concaténation, union et fermeture de Kleene
  - Construit par fermeture  $\implies$  reconnu par un automate
    - traité lors de l'étude des automates avec  $\varepsilon$ -transitions
  - Reconnu par un automate  $\implies$  construit par fermeture
    - Pour tout automate, on sait construire une expression rationnelles associée à l'automate.
    - La transformation « expr. rat. »  $\implies$  Construction par fermeture consiste à remplacer des  $+$  par des  $\cup$  et ajouter des accolades autour des lettres.

## Algorithme d'acceptation d'un mot

- Algorithme pour savoir si un mot  $m$  appartient à  $L(r)$  pour une expression rationnelle  $r$  donnée.
  - Exemple :  $r = (a+\varepsilon)^*aba(a+b)^*$
- Utiliser l'automate associé à cette expression rationnelle



- La même complexité que pour les automates indéterministes avec  $\varepsilon$ -transition.