LIF15 – Théorie des langages formels

Sylvain Brandel

2015-2016

sylvain.brandel@univ-lyon1.fr

Fonctionnement

- Nouveauté 2014 ! (reconduit en 2015 …)
- (tentative de) « classe inversée »
- Supports fournis en avance
  - Supports de cours
  - Ce qui est projeté en CM
  - Sujets de TD
- CM
  - Compléments du support fourni
- TD
  - Plus une discussion autour des solutions que vous aurez cherchées avant de venir
- Bref, travail de votre part AVANT et PENDANT

Evaluation

- http://liris.cnrs.fr/sylvain.brandel/wiki/doku.php?id=ens:lif15
- Contrôles de CM / TD
  - 4 contrôles (dates sous réserve)
    - Lundi 14 septembre (en TD) : 5%
    - Lundi 28 septembre (en TD) : 10%
    - Lundi 2 novembre (en TD) : 10%
    - Lundi 23 novembre (en TD) : 10%
  - Total : 35%
- Projet
  - A rendre en deux parties
  - Total : 30%
- Contrôle continu final
  - En janvier, anonyme et tout
  - 35%

Projet

- Diverses choses autour des automates à états finis
- Va s'appuyer sur JFLAP (cf. Google)
  - Plateforme de test d'un cours
  - Duke University, Trinity, Caroline du Nord, Etats-Unis
  - Pas tout récent
- A faire en binôme, ou en solo
- Séances de TP pour discuter de ce qui coince
- Langage : C / C++
  - CM supplémentaire le 29 septembre (R. Thion) commun avec LIF11
  - Utilise les conteneurs STL (vector, set …)

De votre côté

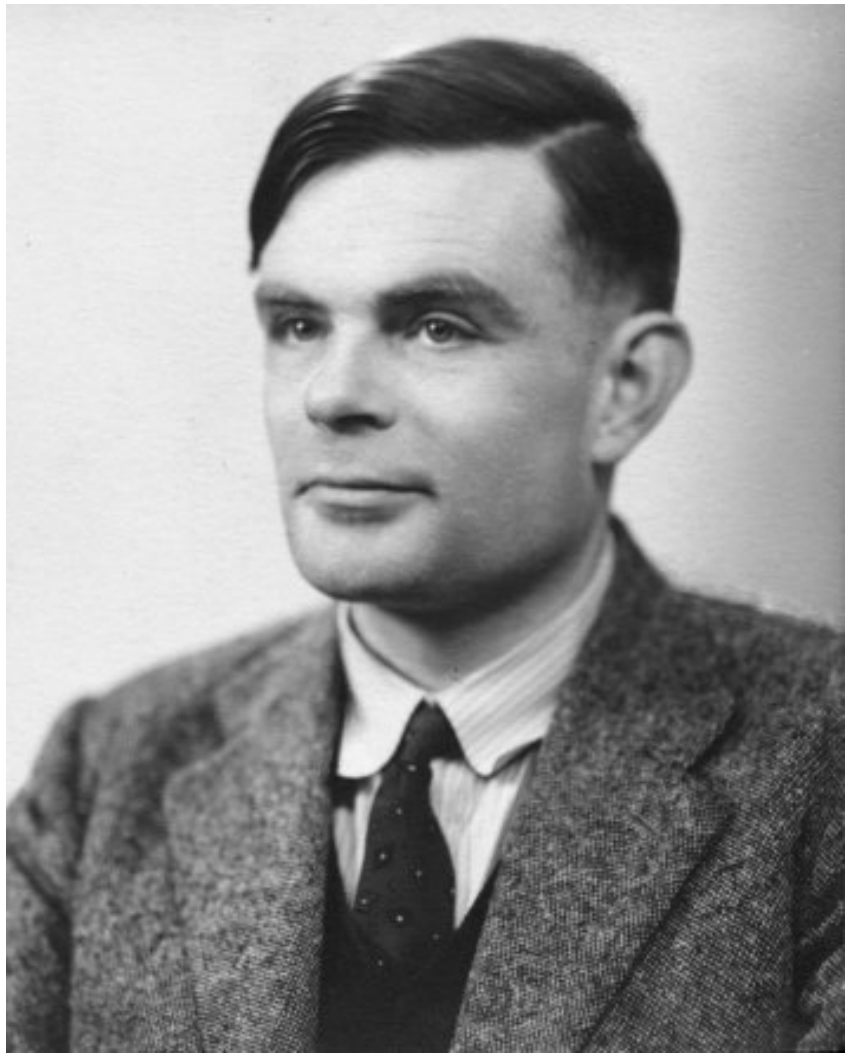
- Travail personnel conséquent
- Se préparer à l'avance
- Ne pas attendre que les réponses viennent toutes seules

- Lisez vos mails …

LIF15 – Théorie des langages formels

Sylvain Brandel
2015 – 2016
sylvain.brandel@univ-lyon1.fr

INTRODUCTION



Motivations

- Informatique fondamentale
- Historiquement
  - Théorie de l'incomplétude
  - Que peut-on calculer avec un algorithme ?
- Lien avec les langages de programmation
  - Ce cours prépare à deux cours de master
    - Calculabilité et complexité
    - Compilation
- Vous intéresser …
  - Si on sait qu'un problème est indécidable, inutile de chercher un algorithme pour le résoudre

Programme

- Classifier des langages

Exemple d'école	Classe de langage	Reconnu par	Engendré par
$a^* b^*$	langages rationnels	automates à états finis	grammaire régulière
$\{a^n b^n \mid n \geq 0\}$	langages algébriques	automates à pile	grammaire algébrique
$\{a^n b^n c^n \mid n \geq 0\}$	langages récursifs	machine de Turing	grammaire (générale)

- La décidabilité et la complexité en découlent

Programme

- Notions mathématiques de base
  - Ensembles
  - Alphabets, langages, expressions régulières
- Automates à états finis
  - Déterministes ou non
  - Liens avec les expressions rationnelles
  - Rationalité
  - Minimisation
- Langages algébriques
  - Grammaires algébriques
  - Automates à pile
  - Algébricité

Programme (suite)

- Machines de Turing
  - Formalisme de base
  - Langages récursifs
  - Extensions
  - Machine de Turing Universelle
  - Grammaires
- Indécidabilité
  - Thèse de Church – Turing
  - Problèmes indécidables
- Complexité
  - Classes P, NP …
  - NP-complétude
  - Théorème de Cook

MIF15 (M1)

Littérature

Elements of the Theory of Computation
Harry R. Lewis, Christos H. Papadimitriou
éd. Prentice-Hall

Introduction à la calculabilité
Pierre Wolper
éd. Dunod

Introduction to the Theory of Computation
Michael Sipser, MIT
éd. Thomson Course Technology

Introduction to Theory of Computation
Anil Maheshwari, Michiel Smid, School of Computer Science, Carleton University
free textbook

Gödel Escher Bach, les Brins d'une Guirlande Eternelle
Douglas Hofstadter
éd. Dunod

Logicomix
Apóstolos K. Doxiàdis, Christos Papadimitriou, Alecos Papadatos, Annie Di Donna
éd. Vuibert

LIF15 – Théorie des langages formels

Sylvain Brandel
2015 – 2016
sylvain.brandel@univ-lyon1.fr

Chapitre 1

NOTIONS MATHÉMATIQUES DE BASE

Terminologie ensembliste
Ensembles

- Définition
  - Par extension : $\Sigma = \{a, b, c\}$
  - Par intension : $P = \{x \in Z \mid \exists\, y \in Z\ ;\ x = 2y\}$
- Opérations ensemblistes
  - Appartient : $x \in E$, $x \notin E$
  - Ensemble vide : $\forall\, x \in E_1,\ x \notin \varnothing$
  - Inclusion : $E_1 \subset E_2$, $E_1 \not\subset E_2$
    $E_1 \subset E_2$ si $\forall\, x : (x \in E_1 \Rightarrow x \in E_2)$
  - Ensemble des parties de E : $P(E) = \{E_1 \mid E_1 \subset E\}$
  - Intersection : $E_1 \cap E_2 = \{x \mid x \in E_1 \text{ et } x \in E_2\}$
  - Union : $E_1 \cup E_2 = \{x \mid x \in E_1 \text{ ou } x \in E_2\}$
  - Complémentarité : $C_E^{E1} = \{x \in E \mid x \notin E_1\}$
    (ou $\neg E_1$ lorsque E est sous entendu)
  - Différence : $E \setminus E_1 = \{x \in E \mid x \notin E_1\}$
  - Produit cartésien : $E_1 \times E_2 = \{(x_1,x_2) \mid x_1 \in E_1 \text{ et } x_2 \in E_2\}$

Terminologie ensembliste
Relations

- Binaires : $R \in P(E_1 \times E_2)$, R est un ensemble de couples.
- n-aire : $R \in P(E_1 \times E_2 \times \ldots E_n)$
- Relations binaires
  - R réflexive $\Leftrightarrow \forall x : x\ R\ x$
  - R symétrique $\Leftrightarrow \forall x, y : x\ R\ y \Rightarrow y\ R\ x$
  - R antisymétrique $\Leftrightarrow \forall x, y : x\ R\ y$ et $y\ R\ x \Rightarrow x = y$
    $\Leftrightarrow \forall x, y : x\ R\ y$ et $x \neq y \Rightarrow (y, x) \notin R$
  - R transitive $\Leftrightarrow \forall x, y, z : x\ R\ y$ et $y\ R\ z \Rightarrow x\ R\ z$

  - Une relation réflexive, symétrique et transitive c'est … ?
  - Une relation réflexive, antisymétrique et transitive c'est … ?

Terminologie ensembliste
Stabilité en clôture

- Soient :
  - E un ensemble
  - R une relation n-aire sur E ($R \subset E^n$)
  - E_1 est une partie de E
- E_1 est dite stable par R ou close par R ssi
  - $\forall\ x_1 \in E_1, x_2 \in E_1, \ldots, x_{n-1} \in E_1$ et $(x_1, x_2, \ldots, x_n) \in R$
    $\Rightarrow x_n \in E_1$
- Plus simple à voir pour R binaire
- Si E_1 n'est pas stable par R, il existe un plus petit sous ensemble F de E tel que $E_1 \subset F$ et F stable par R.
  $\Rightarrow$ On dit que F est la clôture de E_1 par R.
- Soit b une relation sur D, c'est-à-dire $b \subset D^2$
  - On appelle fermeture transitive de b la plus petite relation binaire T telle que $b \subset T$ et T transitive.

Fonctions – applications – bijections – cardinal

- Une fonction f de E_1 vers E_2 est une relation de E_1 vers E_2 telle que

  $\forall x \in E_1$, il existe au plus un élément $y \in E_2$ tel que x f y

- On appelle ce y l'image de x par f : y = f(x)
- Le sous-ensemble de E_1 des éléments ayant des images par f s'appelle le domaine de f.

- Composition de fonctions : $\circ$

  $f \circ g\ (x) = f(g(x))$ $\quad E_1 \rightarrow (g) \rightarrow E_2 \rightarrow (f) \rightarrow E_3$

Fonctions – applications – bijections – cardinal

- Une application f de E_1 vers E_2 est une fonction telle que dom f = E_1

- Une application f est injective

  si $\forall\ x_1, x_2 \in E_1$, $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$

- Une application f est surjective

  si $\forall\ y \in E_2$, il existe au moins un élément x de E_1

  tel que f(x) = y.

- Une bijection est une application injective et surjective. (Ou $f(E_1) = E_2$.)

Fonctions – applications – bijections – cardinal

- Deux ensembles sont équipotents ou ont même cardinal
  ssi il existe une bijection de l'un vers l'autre.

- Un ensemble est fini s'il est équipotent à {1, 2, … n}
  pour tout entier n
- Un ensemble infini est un ensemble non fini

- On dit qu'un ensemble est infini dénombrable s'il est équipotent à N.
- S'il n'existe pas de bijection entre X et une partie de N, alors on dit que X est infini non dénombrable.

- Il existe des ensembles infinis non dénombrables.

Alphabets et langages

Alphabet

- Un alphabet est un ensemble fini, non vide, de symboles.

  On le note généralement Σ.

- Un mot sur un alphabet Σ est une suite finie d'éléments de Σ.

- On note Σ^* l'ensemble de tous les mots (y compris le mot vide) définis sur Σ.

Alphabets et langages
Alphabet

- Longueur : nombre de symboles d'un mot.

- Deux mots u et v sont égaux ssi
  - ils ont même longueur
  - $\forall i \in \{1, \ldots, |u|\} : u_i = v_i$.

Alphabets et langages
Alphabet

- La <u>concaténation</u> de 2 mots u et v de Σ^* est un mot noté uv et défini par :
  - $u = u_1u_2\ldots u_n$, $v = v_1v_2\ldots v_n \rightarrow w = u_1\ldots u_nv_1\ldots v_n$
    - $\forall i \in \{1, |u|\}$ $(uv)_i = u_i$
    - $\forall i \in \{|u|+1, \ldots |u|+|v|\}$ $(uv)_i = v_{i-|u|}$

- <u>Propositions</u>
  - la concaténation est régulière à droite et à gauche
    - $wu = wv \Rightarrow u = v$
    - $uw = vw \Rightarrow u = v$
  - $|uv| = |u| + |v|$.

Alphabets et langages
Alphabet

- On appelle facteur gauche de w un mot u tel que uv = w.
- On appelle facteur droit un mot v tel que uv = w.
- On appelle facteur de w un mot u tel que il existe v et v' tels que vuv' = w.

- Miroir (Reverse) :
  - La fonction miroir $\_^R : \Sigma^* \rightarrow \Sigma^*$ est définie par récurrence :
    - w tq $|w| = 0$: $w^R = e^R = e$
    - w tq $|w| > 0$: $\exists a \in \Sigma$ tq $w = au$
      et $w^R = (au)^R = u^R a$

- Propriété
  - $\forall u, v \in \Sigma^*$: $(uv)^R = v^R u^R$

Alphabets et langages

Langage

- On appelle langage sur Σ tout ensemble de mots sur Σ

- Remarque de cardinalité
  - Σ fini
  - Σ^* infini dénombrable (rappel : dont on peut énumérer les éléments)
  - P (Σ^*) est infini non dénombrable

- Opérations sur les langages
  - $\cup$, $\cap$, $\neg$ (complément), \ $\Rightarrow$ comme d'habitude
    (complément : $\neg A = \Sigma^* \setminus A$)
  - Concaténation : $L_1 \subset \Sigma^*$, $L_2 \subset \Sigma^*$
    - $L = L_1.L_2$ ou L_1L_2
      est défini par $L = \{w \mid \exists\, w_1 \in L_1 \text{ et } \exists\, w_2 \in L_2 : w = w_1w_2\}$
  - Clôture de Kleene (Kleene star) ou étoile de L
    - $L^* = \{w \in \Sigma^* \mid \exists\, k \in N, \exists\, w_1, w_2, \ldots, w_k \in L : w = w_1w_2\ldots w_k\}$

Représentation finie des langages

- Une description habituelle d'une certaine classe de langages est fournie par ce qu'on appelle les expressions régulières (ou rationnelles).
  - les éléments de base sont :
    - les singletons sur Σ
    - l'ensemble $\varnothing$
  - les opérations sont :
    - la concaténation de langages
    - la réunion de deux langages
    - la fermeture de Kleene

Représentation finie des langages

- Les expressions régulières / rationnelles constituent syntaxiquement un langage (que nous appellerons ER) de mots bien formés sur l'alphabet $\Sigma \cup \{(,), \varnothing, \cup, *\}$ tel que :
  - ① $\varnothing$ et chaque lettre de Σ est une ER
  - ② si α et β sont des ER, alors $(\alpha\beta)$ et $(\alpha \cup \beta)$ aussi
  - ③ si α est une ER alors α^* aussi
  - ④ ER est close pour ces propriétés

    (rien d'autre n'est une ER que les points ① à ③)

Représentation finie des langages

- Exemple
  - $\Sigma = \{a, b, c\}$
  - Ensemble des mots finissant par a :

$$L = (a \cup b \cup c)^* a$$

$$(= (a^* c \cup b)^* a)$$

- Définition
  - On appelle Langage rationnel (ou régulier) tout langage qui peut être décrit par une expression rationnelle.

LIF15 – Théorie des langages formels

Sylvain Brandel
2015 – 2016
sylvain.brandel@univ-lyon1.fr

Chapitre 2

AUTOMATES À ÉTATS FINIS

Automates à états finis

- Automate ou Machine
  - *Ang. Automaton / Machine*
- Automates ou Machines
  - *Ang. Automata / Machines …*

- Automates (à états) finis déterministes
  - *Ang. Deterministic Finite Automata (DFA)*
- Automates (à états) finis non déterministes
  - *Ang. Nondeterministic Finite Automata (NFA)*
- (chapitre suivant) Automates à pile
  - *Ang. Pushdown Automata (PDA)*
- (l'an prochain) Machines de Turing
  - *Ang. Turing Machines*

Automates à états finis

- Exemple concret
  - Machine à café dans le hall du déambu
  - Barrière de péage du périph' (pas liber-t …)
  - …

Automates finis déterministes

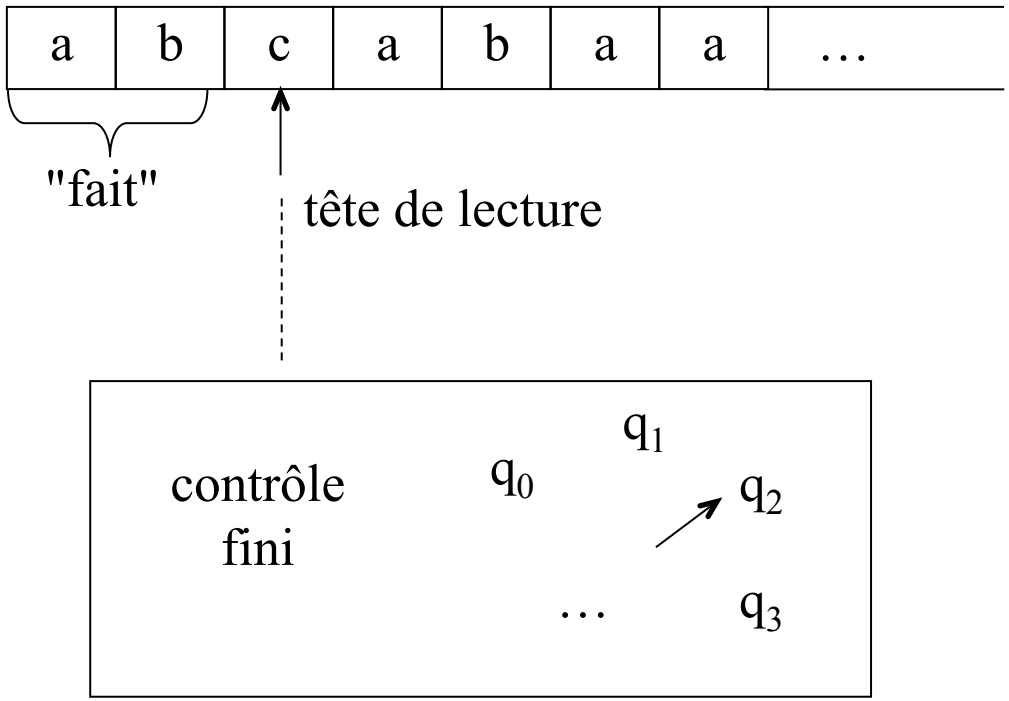
- Simulation d'une machine très simple :
  - mémorisation d'un état
  - programme sous forme de graphe étiqueté indiquant les transitions possibles
- Cette machine lit un mot en entrée.
- Ce mot décrit une suite d'actions et progresse d'état en état

  → jusqu'à la lecture complète du mot.
- Lorsque le dernier état est distingué (état final)

  → on dit que le mot est accepté.

⇒ Un automate permet de reconnaître un langage.

Automates finis déterministes



- Un état dépend uniquement
  - De l'état précédent
  - Du symbole lu

Automates finis déterministes

- Un automate déterministe fini est le quintuplet

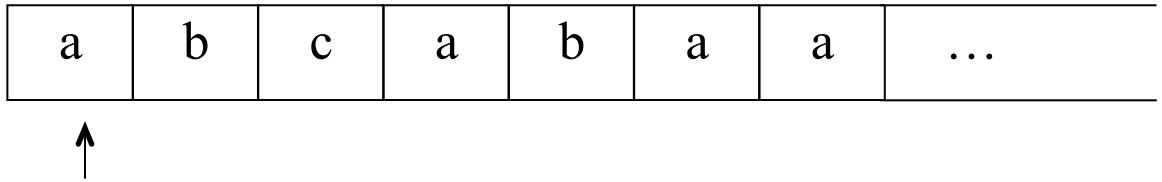
  $M = (K, \Sigma, \delta, s, F)$ où :

  - K : ensemble fini (non vide) d'états
  - Σ : alphabet (ensemble non vide de lettres)
  - δ : fonction de transition : $K \times \Sigma \rightarrow K$

    $\delta(q, \sigma) = q'$ (q' : état de l'automate après avoir lu la lettre σ dans l'état q)
  - s : état initial : $s \in K$
  - F : ensemble des états finaux : $F \subset K$

Automates finis déterministes

- Exécution



La machine

- lit a (qui est ensuite oublié),
- passe dans l'état $\delta(s, a)$ et avance la tête de lecture,
- répète cette étape jusqu'à ce que tout le mot soit lu.

- La partie déjà lue du mot ne peut pas influencer le comportement à venir de l'automate.

→ d'où la notion de configuration

Automates finis déterministes

- Configuration
  - état dans lequel est l'automate
  - mot qui lui reste à lire (partie droite du mot initial)

- Formellement : une configuration est un élément quelconque de $K \times \Sigma^*$.

- Exemple

  sur l'exemple précédent, la configuration est $(q_2, cabaa)$.

Automates finis déterministes

- Le fonctionnement d'un automate est décrit par le passage d'une configuration à une autre, cette dernière obtenue
  - en lisant un caractère,
  - et en appliquant la fonction de transition.

- <u>Exemple</u>
  - $(q_2, cabaa) \rightarrow (q_3, abaa)$ si $\delta(q_2, c) = q_3$

Automates finis déterministes

- Un automate M détermine une relation binaire entre configurations qu'on note $\vdash_M$ définie par :
  - $\vdash_M \subset (K \times \Sigma^*)^2$
  - $(q, w) \vdash_M (q', w')$ ssi $\exists\ a \in \Sigma$ tel que $w = aw'$ et $\delta(q, a) = q'$

- On dit alors que on passe de (q, w) à (q', w') en une étape.

Automates finis déterministes

- On note $\vdash_M^*$ la fermeture transitive réflexive de $\vdash_M$:
  $(q, w) \vdash_M^* (q', w')$ signifie qu'on passe de (q, w) à (q', w') en zéro, une ou plusieurs étapes

- Un mot w est <u>accepté</u> par M
  ssi $(s, w) \vdash_M^* (q, e)$, avec $q \in F$.

- Le <u>langage accepté</u> par M est l'ensemble de tous les mots acceptés par M.
  Ce langage est noté L(M).

Automates finis déterministes

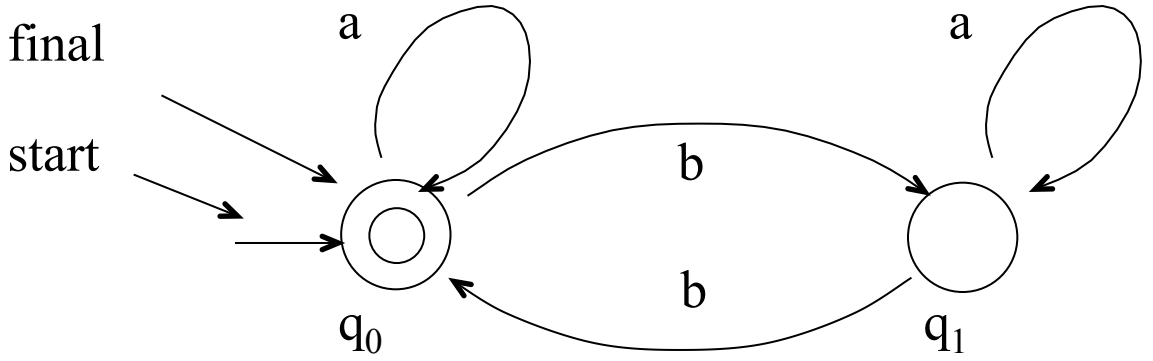
- Exemple M = (K, Σ, δ, s, F)
  - K = {q_0, q_1}
  - Σ = {a, b}
  - s = q_0
  - F = {q_0}

δ :

q	σ	δ(q, σ)
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0

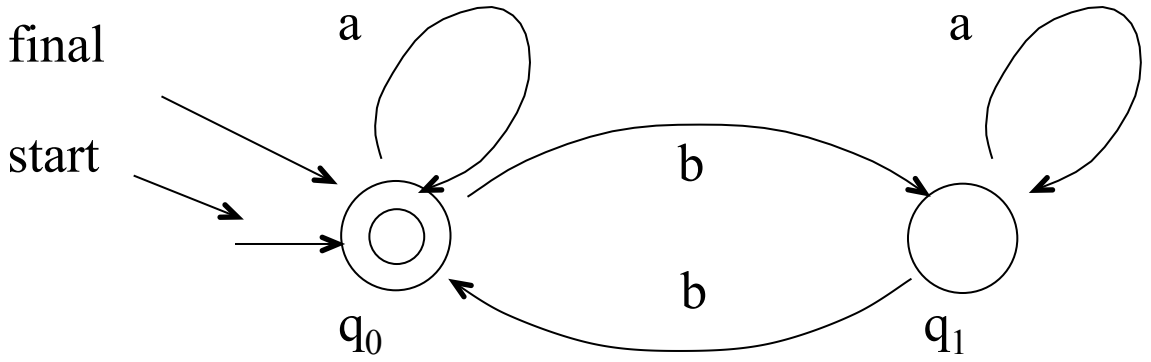
⇔

	a	b
q_0	q_0	q_1
q_1	q_1	q_0



Automates finis déterministes

- Démo

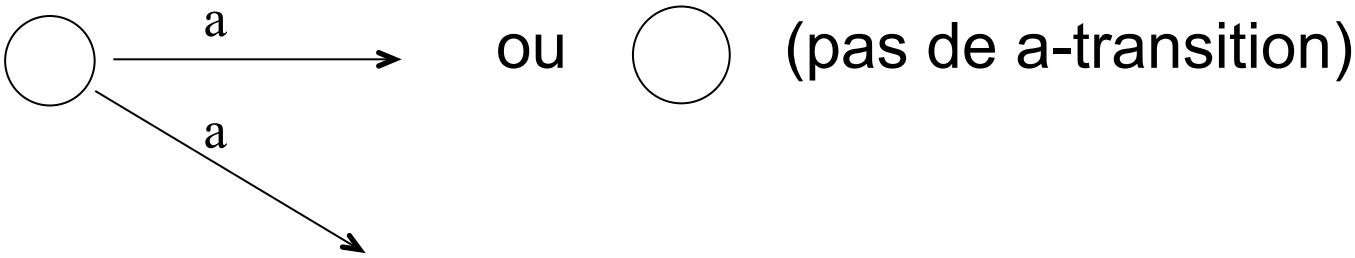


Automates finis non déterministes

- Idée : remplacer la fonction $\vdash_M$ (ou δ) par une relation.
- Une relation, c'est beaucoup plus général qu'une fonction.

→ on a ainsi une classe plus large d'automates.

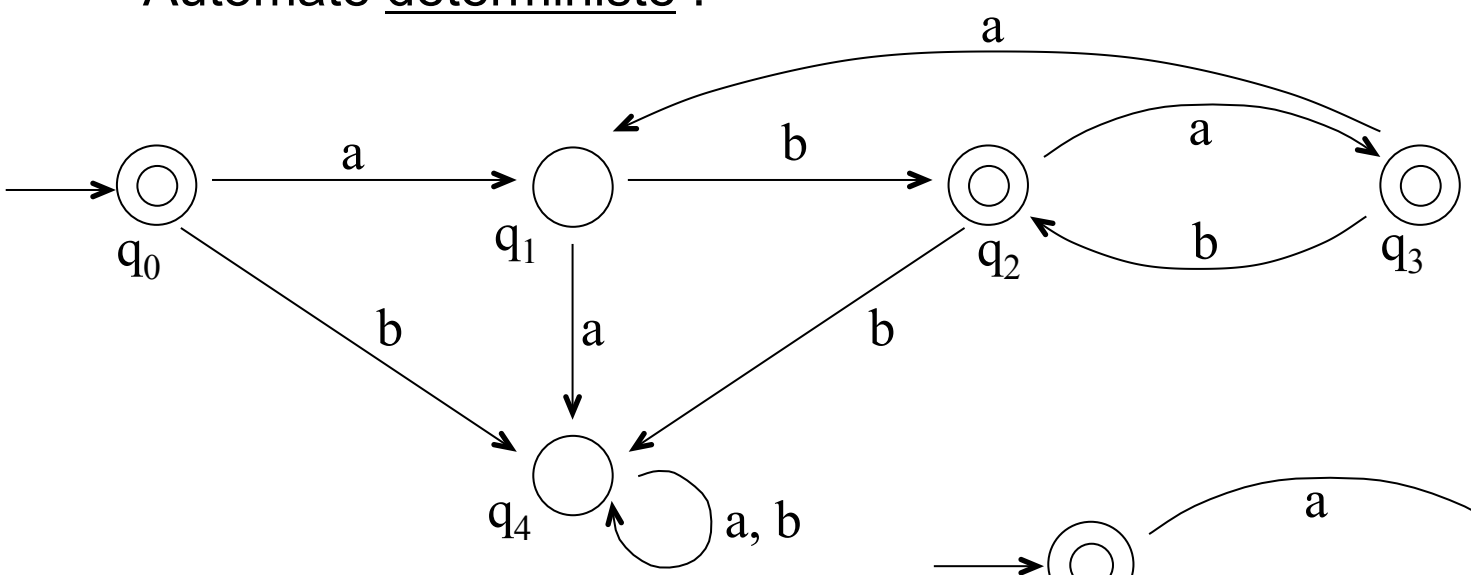
⇒ Dans un état donné, on pourra avoir :



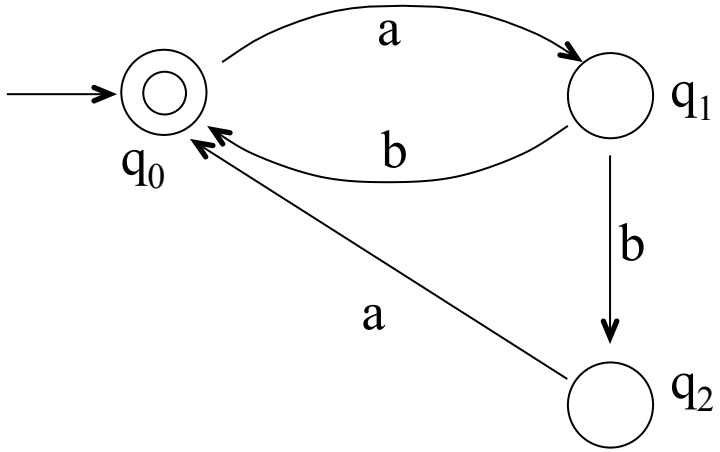
Automates finis non déterministes

- $L = (ab \cup aba)^*$

Automate déterministe :



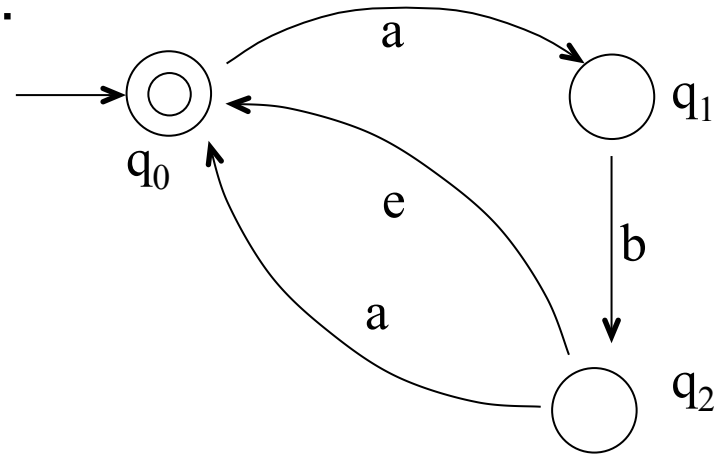
Automate non déterministe :



Automates finis non déterministes

- Dans le cas de l'automate non déterministe, un mot est accepté s'il existe un ou plusieurs chemins (au moins un) pour aller de l'état initial (ici q_0) à l'état final (ici q_0).
- Autre différence par rapport aux automates déterministes :
  - Il est possible d'étiqueter une flèche par le symbole e.

- Autre formulation (encore plus intuitive) de l'automate précédent :



Automates finis non déterministes

- Un automate non déterministe fini est le quintuplet

  $M = (K, \Sigma, \Delta, s, F)$ où :

  - K : ensemble fini (non vide) d'états
  - Σ : alphabet (ensemble non vide de lettres)
  - Δ : relation de transition : $K \times (\Sigma \cup \{e\}) \times K$

    $(q, \sigma, p) \in \Delta$: σ-transition ($\sigma \in \Sigma$)
  - s : état initial : $s \in K$
  - F : ensemble des états finaux : $F \subset K$

  (hormis la relation, le reste est identique à la formulation déterministe)

Automates finis non déterministes

- Si $(q, e, p) \in \Delta$: on a une ε-transition (transition spontanée)

  → On passe de q à p sans lire de symbole dans le mot courant.

- Une configuration est un élément de $K \times \Sigma^*$.

- M décrit une relation binaire entre configurations qu'on note $\vdash_M$:

  $(q, w) \vdash_M (q', w')$

  ssi il existe un mot d'au plus une lettre $u \in \Sigma \cup \{e\}$

  tq $w = uw'$ et $(q, u, q') \in \Delta$

Automates finis non déterministes

- $\vdash_M$ est une relation et non plus une fonction (automates déterministes) :
  - (q, e) peut être en relation avec une autre configuration (après une ε-transition)
  - pour une configuration (q, w), il peut y avoir plusieurs configurations (q', w') (ou aucune) tq (q, w) $\vdash_M$ (q', w')

- On note comme avant $\vdash_M^*$ la fermeture transitive réflexive de $\vdash_M$

- Un mot w est accepté par M ssi (s, w) $\vdash_M^*$ (q, e) avec q $\in$ F.

- L(M) est le langage de tous les mots acceptés par M.

Et maintenant ?

- Déterministe ou non déterministe ?
  - Même classe de langages reconnus
  - Donc équivalents
- Langages rationnels et langages reconnus par les automates finis ?
  - Même classe de langages
- Un langage est-il rationnel ?
  - Preuve de rationalité
- Un automate est-il minimal ?
  - Automate standard

Elimination du non-déterminisme

- Définition
  - 2 automates finis (déterministes ou non) sont équivalents ssi L(M) = L(M').

- Théorème

  Pour tout automate non déterministe, il existe un automate déterministe équivalent, et il existe un algorithme pour le calculer

  Cet algorithme est appelé déterminisation d'un automate.

Elimination du non-déterminisme
Preuve

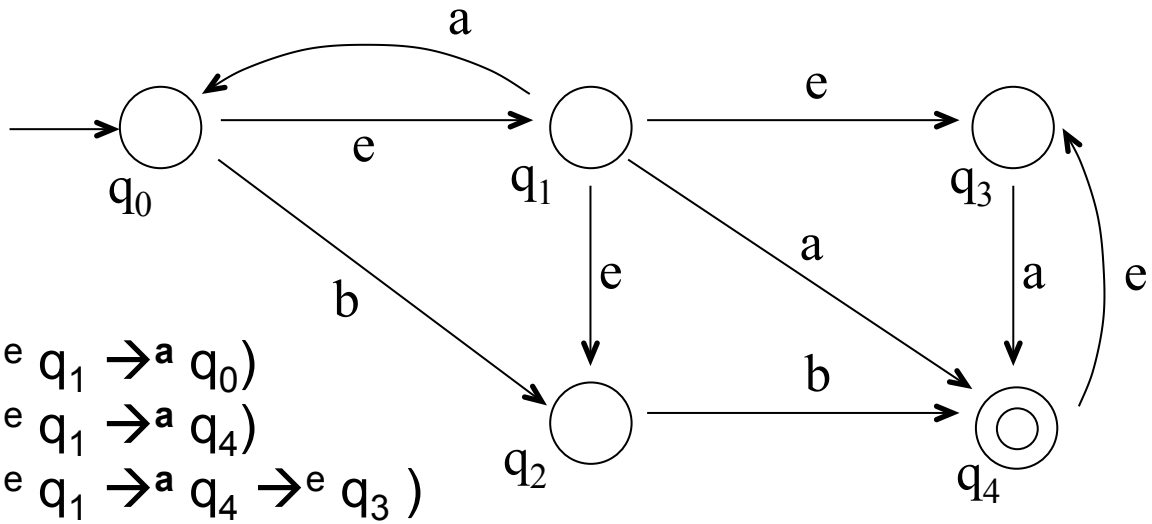
- Soit M = (K, Σ, Δ, s, F) un automate non déterministe.
- Problème : M' = (K', Σ, δ', s', F') ? (M' déterministe)
- Démarche :
  - Méthode pour construire M'
  - Montrer
    - M' déterministe
    - M' équivalent à M

Elimination du non-déterminisme
Preuve

- L'idée est la suivante :
  - Pour toute lettre σ de Σ, on considère l'ensemble des états qu'on peut atteindre en lisant σ.
  - On rassemble ces états et on considère des états étiquetés par des parties de K (P (K))
  - L'état initial de M' est l'ensemble des états atteignables en ne lisant aucune lettre.
  - Les états finaux de M' sont les parties (atteignables) de P (K) contenant au moins un état final (de M).

Elimination du non-déterminisme
Preuve

- Exemple



$q_0 \rightarrow^{a}$ q_0 ($q_0 \rightarrow^{e} q_1 \rightarrow^{a} q_0$)
q_4 ($q_0 \rightarrow^{e} q_1 \rightarrow^{a} q_4$)
q_3 ($q_0 \rightarrow^{e} q_1 \rightarrow^{a} q_4 \rightarrow^{e} q_3$)
q_1 ($q_0 \rightarrow^{e} q_1 \rightarrow^{a} q_0 \rightarrow^{e} q_1$)
q_2 ($q_0 \rightarrow^{e} q_1 \rightarrow^{a} q_0 \rightarrow^{e} q_1 \rightarrow^{e} q_2$)

$q_0 \rightarrow^{b}$ q_2 ($q_0 \rightarrow^{b} q_2$)
q_4 ($q_0 \rightarrow^{e} q_1 \rightarrow^{e} q_2 \rightarrow^{b} q_4$)
q_3 ($q_0 \rightarrow^{e} q_1 \rightarrow^{e} q_2 \rightarrow^{b} q_4 \rightarrow^{e} q_3$)

état initial : {q_0, q_1, q_2, q_3}

Elimination du non-déterminisme
Preuve

- On introduit la notion de ε-clôture (pour prendre en compte les ε-transitions) :

Soit $q \in K$, on note E(q) l'ensemble des états de M atteignables sans lire aucune lettre :

$$E(q) = \{p \in K : (q, e) \vdash_M^* (p, e)\}$$

(c-à-d E(q) est la clôture de {q} par la relation binaire $\{(p, r) \mid (p, e, r) \in \Delta\}$)

- <u>Construction de E(q)</u>

E(q) := {q}
<u>tant que</u> il existe une transition $(p, e, r) \in \Delta$
avec $p \in E(q)$ et $r \notin E(q)$
<u>faire</u> $E(q) := E(q) \cup \{r\}$

Elimination du non-déterminisme
Preuve

- Donc $M' = (K', \Sigma, \delta', s', F')$

avec $K' = P(K)$

$s' = E(s)$

$F' = \{Q \subset K : Q \cap F \neq \emptyset\}$

$\delta' = P(K) \times \Sigma \rightarrow P(K)$

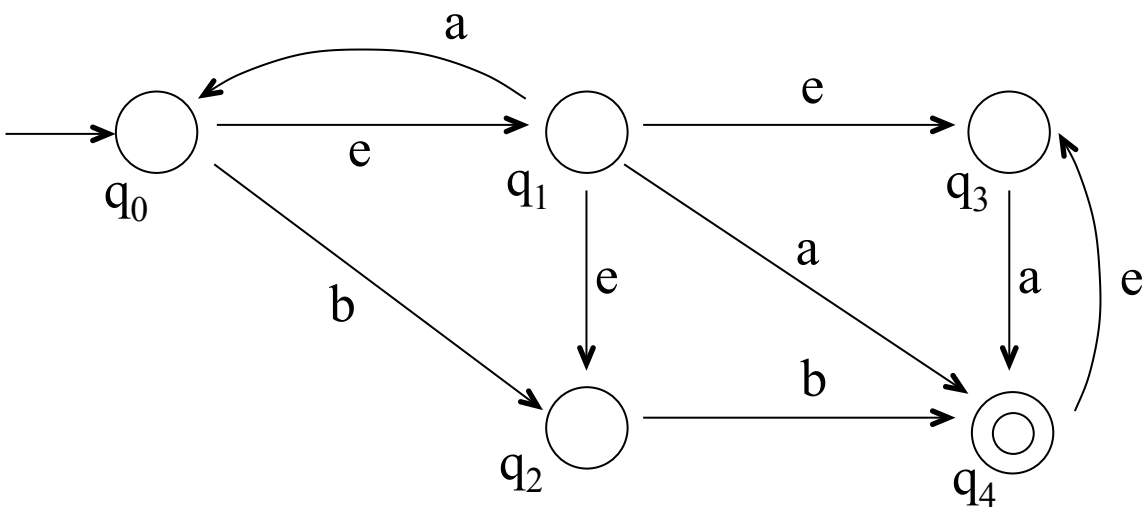
$$\forall\, Q \subset K,\ \forall\, a \in \Sigma,$$

$$\delta'(Q, a) = \cup \{E(p) \mid \exists\, q \in Q : (q, a, p) \in \Delta\}$$

$\delta'(Q, a)$: ensemble de tous les états (de M) dans lesquels M peut aller en lisant a (y compris e)

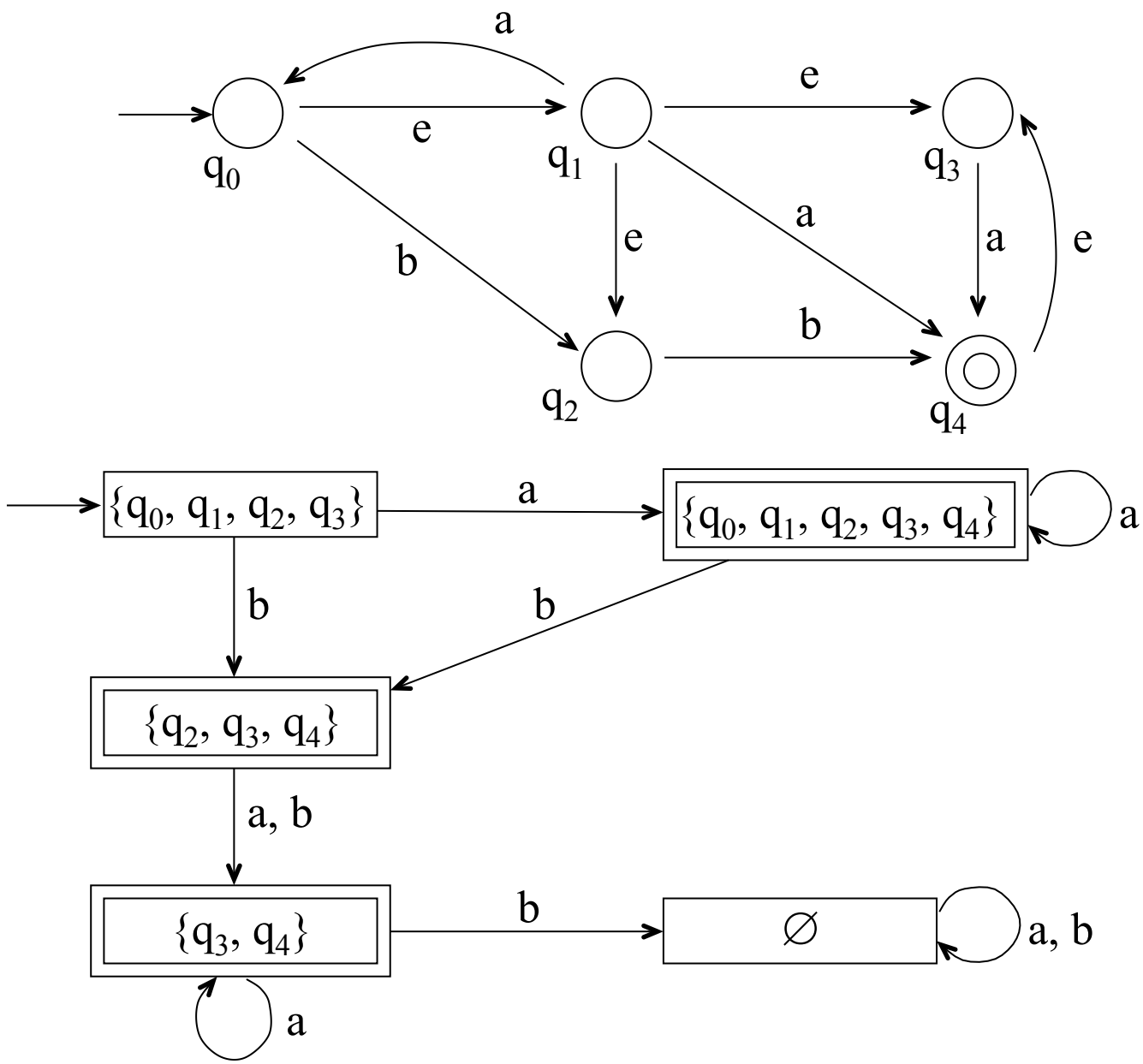
Elimination du non-déterminisme

Exemple



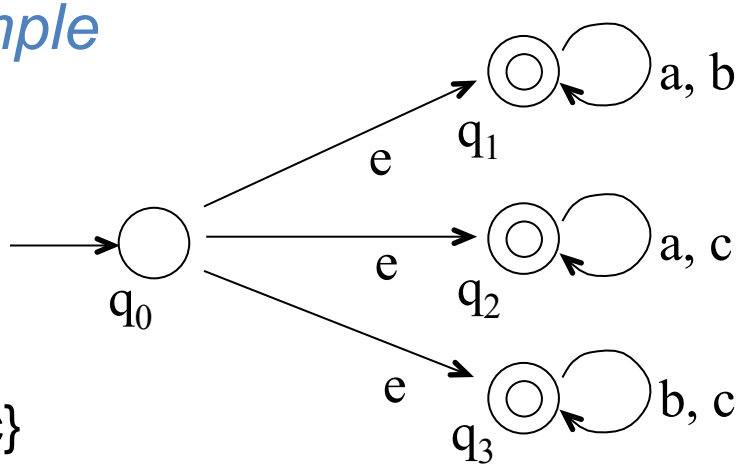
Elimination du non-déterminisme

Exemple



Elimination du non-déterminisme

Autre exemple

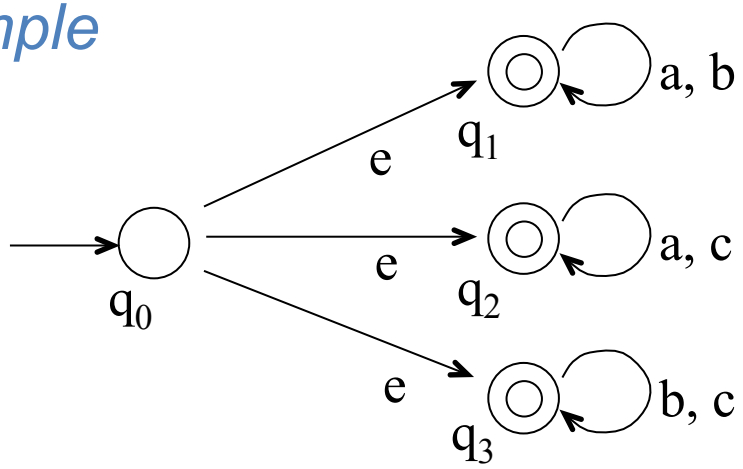


$\Sigma = \{a, b, c\}$

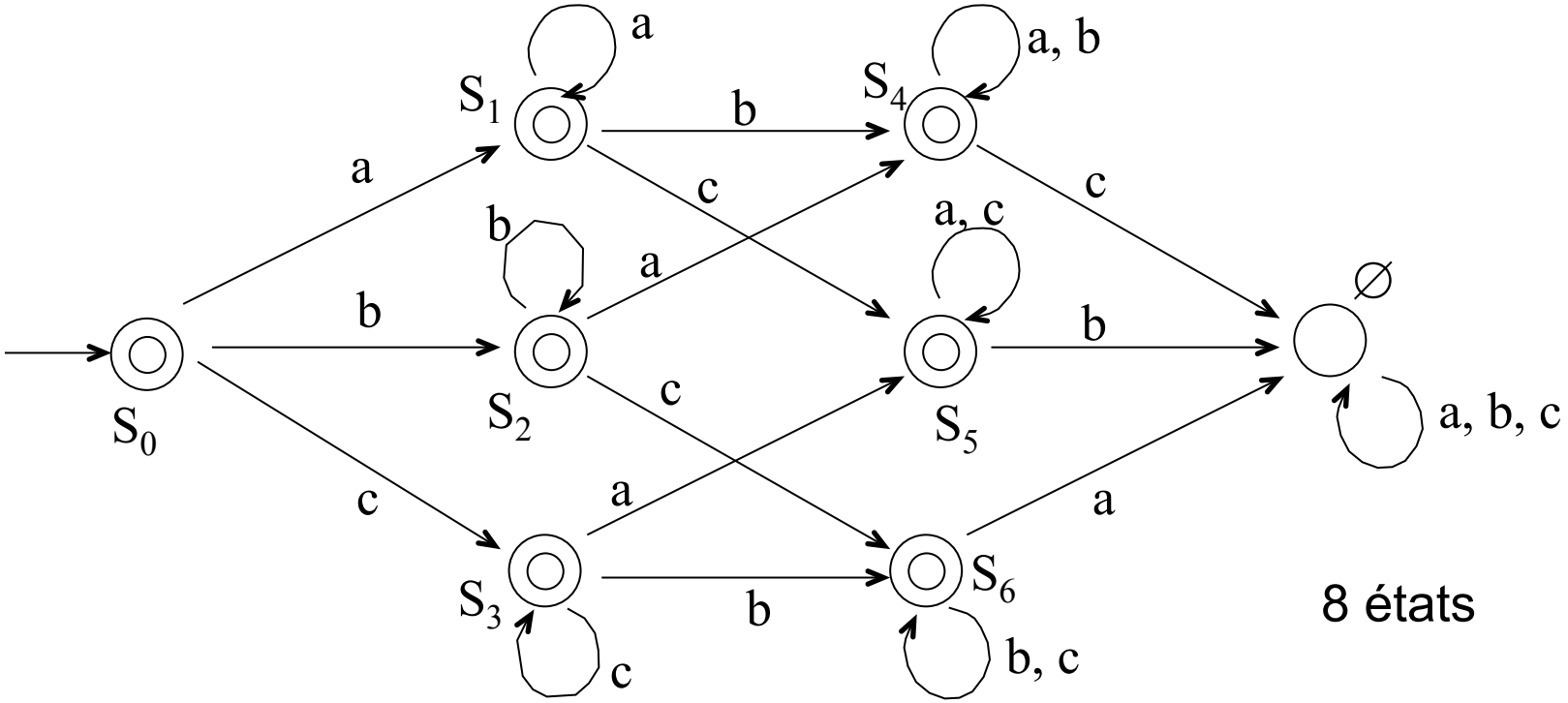
$L(M) = (a \cup b)^* \cup (a \cup c)^* \cup (b \cup c)^*$

Elimination du non-déterminisme

Autre exemple



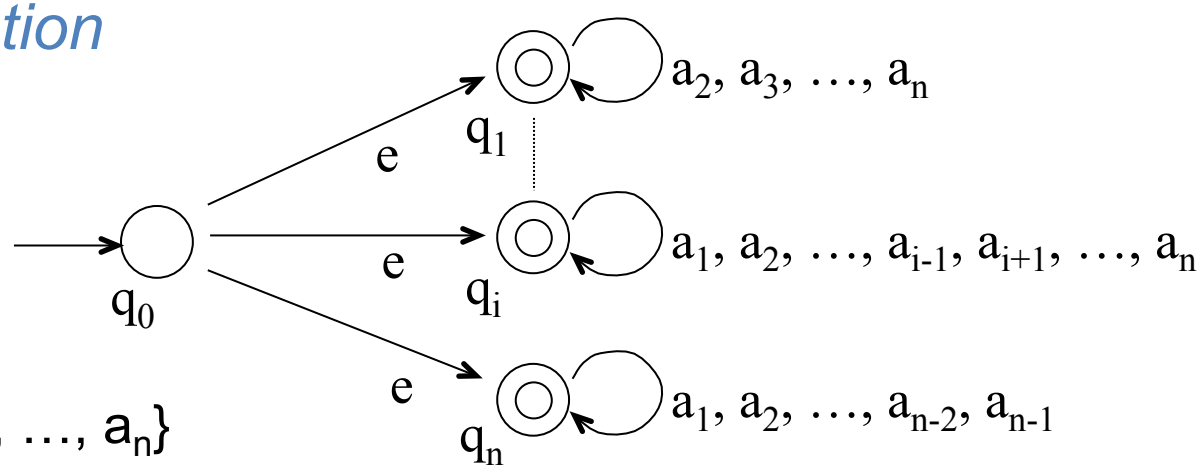
4 états



8 états

Elimination du non-déterminisme

Généralisation



$\Sigma = \{a_1, a_2, \ldots, a_n\}$

$\Sigma_1 = \Sigma - \{a_1\}$

...

$\Sigma_i = \Sigma - \{a_i\}$

$\rightarrow L(M) = \cup_{i=1}^{n} \Sigma_i^*$

Lien avec les expressions rationnelles
Stabilité

- Propriété de stabilité des langages reconnus par des automates finis.

- Théorème

  La classe des langages acceptés par les automates finis est stable par :

  - *Union*
  - *Concaténation*
  - *Kleene star*
  - *Complément*
  - *Intersection*

  ordre de la démonstration

- Preuve constructive
  - Construction de l'automate pour chaque opération

Lien avec les expressions rationnelles
Stabilité

- Union

  $L_1 = L(M_1)$ $(K_1, \Sigma, \Delta_1, s_1, F_1)$ → Même Σ

  $L_2 = L(M_2)$ $(K_2, \Sigma, \Delta_2, s_2, F_2)$ → Hyp. $K_1 \cap K_2 = \varnothing$

Posons s tel que $s \notin K_1$ et $s \notin K_2$.

→ $M_\cup : (\{s\} \cup K_1 \cup K_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(s, e, s_1), (s, e, s_2)\}, s, F_1 \cup F_2)$

$w \in L(M_\cup) \Leftrightarrow (s, w) \vdash_M (s_1, w) \vdash_M^* (f_1, e) \quad (f_1 \in F_1) \quad \leftarrow L_1$

ou

$(s, w) \vdash_M (s_2, w) \vdash_M^* (f_2, e) \quad (f_2 \in F_2) \quad \leftarrow L_2$

$\Leftrightarrow w \in L_1 \cup L_2$

Lien avec les expressions rationnelles
Stabilité

- Concaténation

  $L_1 = L(M_1)$ $(K_1, \Sigma, \Delta_1, s_1, F_1)$ → Même Σ

  $L_2 = L(M_2)$ $(K_2, \Sigma, \Delta_2, s_2, F_2)$ → Hyp. $K_1 \cap K_2 = \varnothing$

  → $M_c : (K_1 \cup K_2, \Sigma, \Delta_1 \cup \Delta_2 \cup \{(f_i, e, s_2) \mid f_i \in F_1\}, s_1, F_2)$

Lien avec les expressions rationnelles
Stabilité

- Etoile de Kleene

  $L_1 = L(M_1)$ $(K_1, \Sigma, \Delta_1, s_1, F_1)$

  → $M_K : (K_1 \cup \{s\}, \Sigma, \Delta_1 \cup \{(s, e, s_1)\} \cup \{(f_i, e, s_1) \mid f_i \in F_1\}, s, F_1 \cup \{s\})$

Lien avec les expressions rationnelles
Stabilité

- <u>Complément</u>

  $L_1 = L(M_1)$ $(K_1, \Sigma, \Delta_1, s_1, F_1)$

  → $M_{\neg} : (K_1, \Sigma, \Delta_1, s, \neg F_1)$ avec $\neg F_1 = K_1 - F_1$

  Attention M_1 doit être déterministe

Lien avec les expressions rationnelles
Stabilité

- Intersection

  $L_1 = L(M_1)$ $(K_1, \Sigma, \Delta_1, s_1, F_1)$ → Même Σ

  $L_2 = L(M_2)$ $(K_2, \Sigma, \Delta_2, s_2, F_2)$ → Hyp. $K_1 \cap K_2 = \varnothing$

- Deux méthodes :

  – $L(M_1) \cap L(M_2) = \overline{\overline{L(M_1)} \cup \overline{L(M_2)}}$

  – Automate produit
    - $M_\cap$: ($K_1 \times K_2, \Sigma,$
      $\{((p_1,p_2), \sigma, (q_1,q_2)) \mid (p_1, \sigma, q_1) \in \Delta_1$ et $(p_2, \sigma, q_2) \in \Delta_2\}$,
      $(s_1,s_2), F_1 \times F_2)$
    - Quadratique en le nombre d'états

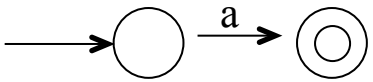
Lien avec les expressions rationnelles
Inclusion des classes de langages

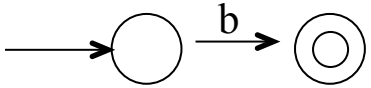
- Théorème

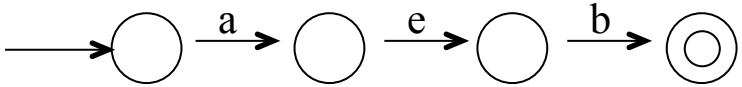
  La classe des langages acceptés par les automates finis contient les langages rationnels.

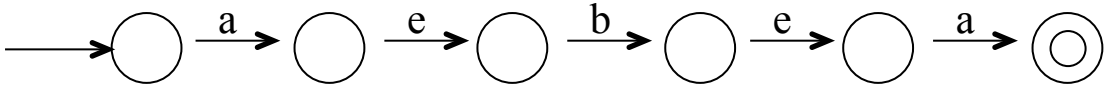
- Exemple

  $(ab \cup aba)^*$

a 

b 

ab 

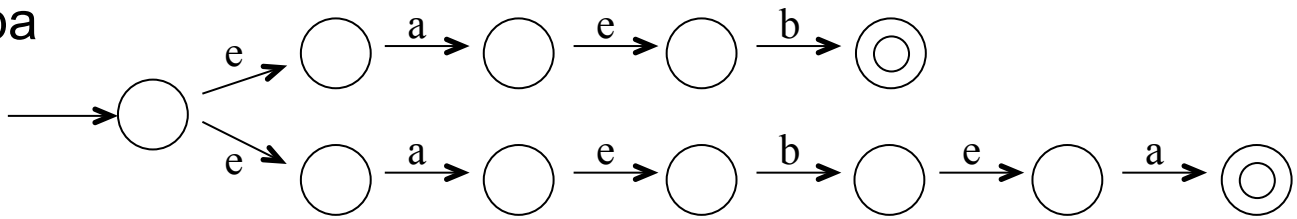
aba 

Lien avec les expressions rationnelles
Inclusion des classes de langages

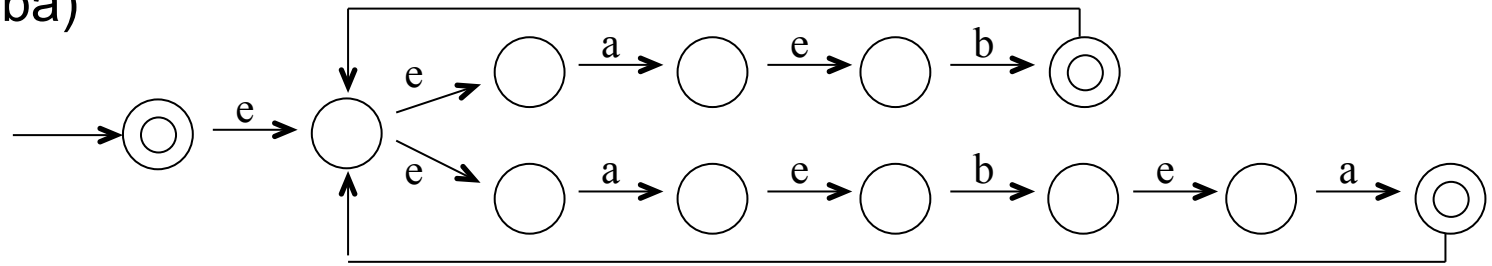
- Exemple

  $(ab \cup aba)^*$

ab ∪ aba



$(ab \cup aba)^*$



Lien avec les expressions rationnelles
Caractérisation des langages rationnels

- Théorème

  Un langage est rationnel ssi il est accepté par un automate fini.

- Preuve

  On suppose qu'on a numéroté (ordonné) les états.

  Soit M = (K, Σ, Δ, s, F) un automate fini (déterministe ou non).

  $K = \{q_1, q_2, \ldots, q_n\}$, $s = q_1$

  Le langage reconnu par M est la réunion de tous les langages reconnus en parcourant tous les chemins possibles dans le graphe (en faisant attention aux boucles).

  (Cela revient à dire que à chaque chemin allant de s à f ($\in$ F), on associe le langage trouvé.)

Lien avec les expressions rationnelles

Caractérisation des langages rationnels

- On pose R(i, j, k) = l'ensemble des mots obtenus par lecture de l'automate M
  - en partant de l'état q_i,
  - en arrivant dans l'état q_j (avec le mot vide),
  - en ne passant que par des états dont le numéro est inférieur ou égal à k.

- Autrement dit : $R(i, j, k) = \{w \mid (q_i, w) \vdash_M^* (q_j, e)$ sans passer par des états dont le numéro est supérieur à k$\}$

- On a : $R(i, j, n) = \{w \mid (q_i, w) \vdash_M^* (q_j, e)\}$

- et : $L(M) = \bigcup_{i \mid q_i \in F} R(1, i, n)$

Lien avec les expressions rationnelles

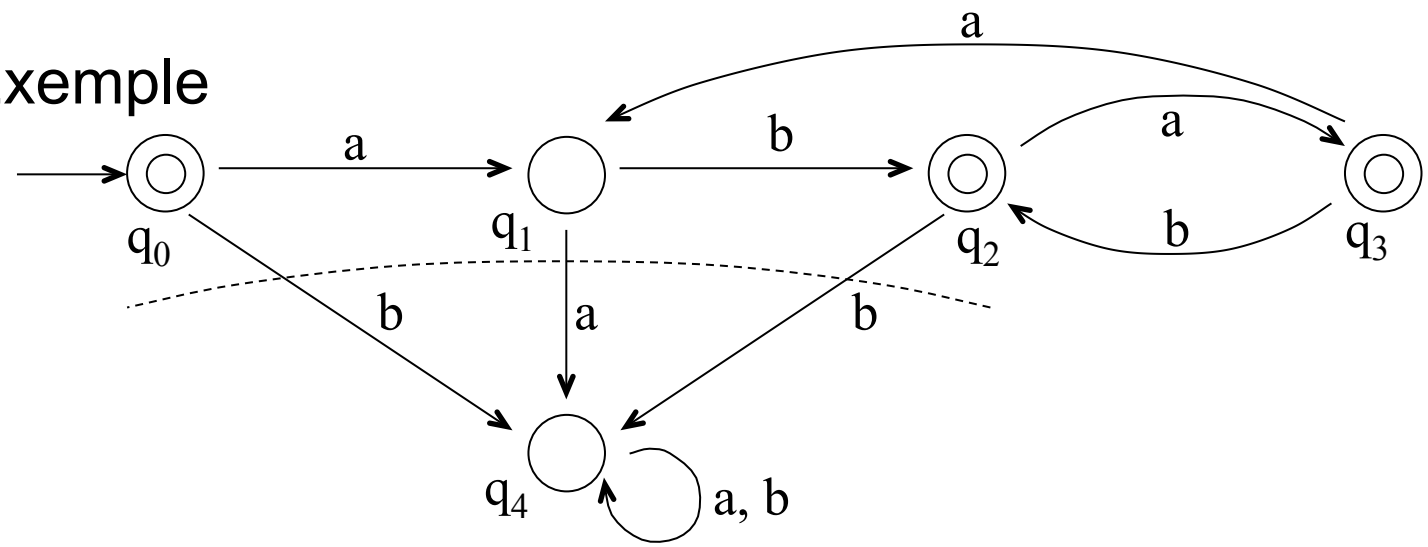
Caractérisation des langages rationnels

- R(i, j, k) est un langage rationnel dont on peut calculer l'expression rationnelle par récurrence sur k.

- Preuve

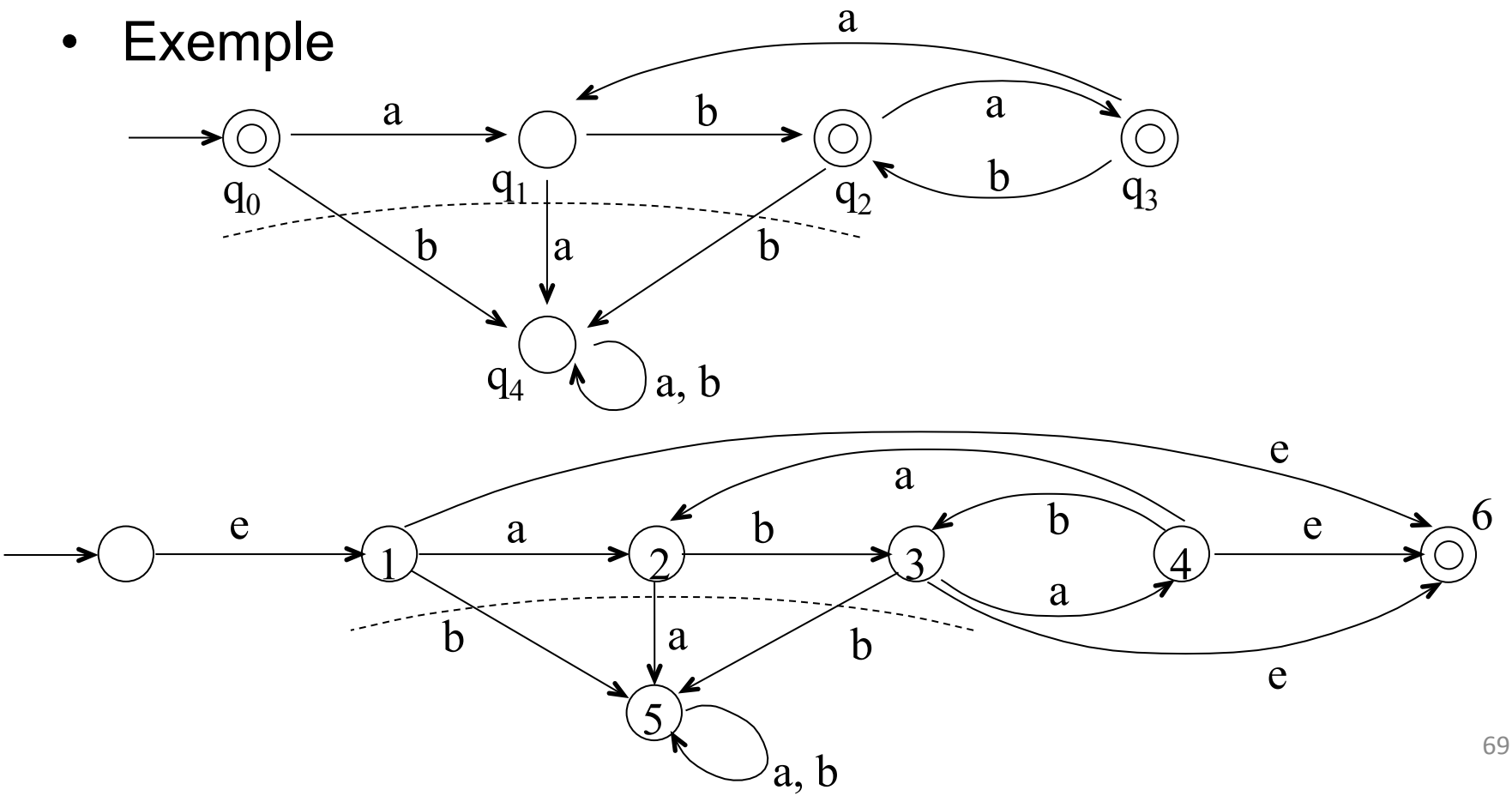
$$R(i, j, k) = R(i, j, k-1) \cup R(i, k, k-1) \,.\, (R(k, k, k-1))^{*} \,.\, R(k, j, k-1)$$

- Exemple



Lien avec les expressions rationnelles
Caractérisation des langages rationnels

- Pour simplifier on « arrange » le graphe
  - Un seul état final
  - Pas de retour de f vers le graphe ni du graphe vers l'état initial
- Exemple



Lien avec les expressions rationnelles
Caractérisation des langages rationnels

- Exemple
  - R(i, j, 0) → étiquettes sur les flèches.
  - Principe : calculer R(1, 6, 6)

R(1, 6, 6) = R(1, 6, 5) ∪ R(1, 6, 5) R(6, 6, 5)* R(6, 6, 5)

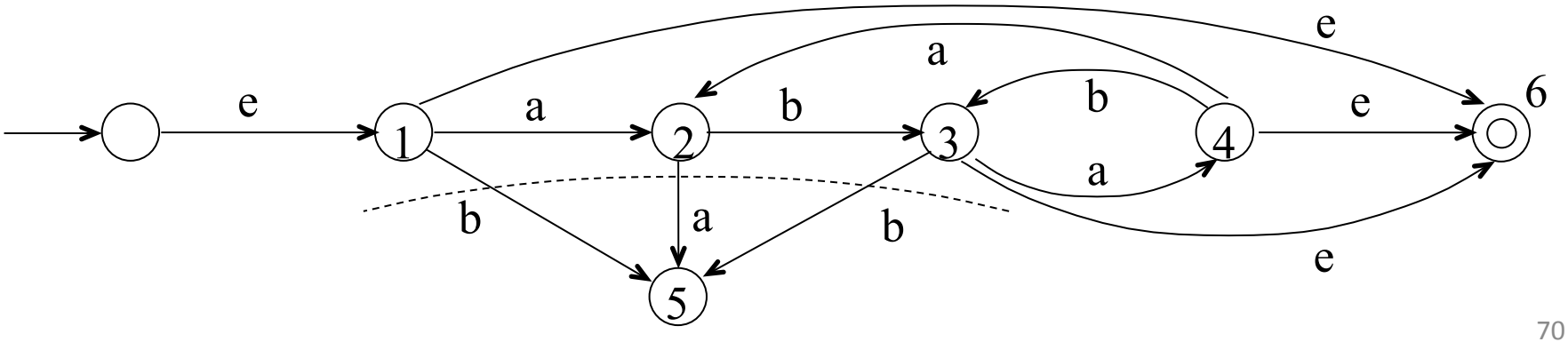
= R(1, 6, 4) ∪ R(1, 5, 4) R(5, 5, 4)* R(5, 6, 4) ∪ … ∪ …

⇒ Fastidieux, donc on calcule les R(i, j, k) de proche en proche.

→ On supprime q_1 (ce qui revient à calculer R(i, j, 1))

→ On supprime q_2 (ce qui revient à calculer R(i, j, 2))

…



Rationalité

- On peut monter qu'un langage est rationnel avec un des théorèmes :
  - (1) Stabilité

  (rappel : la classe des langages acceptés par un automate est stable par union, concaténation, Kleene star, complément, intersection)
  - (2) Caractérisation

  (rappel : un langage est rationnel ssi il est accepté par un automate)
  - (3) Un langage est rationnel ssi il peut être décrit par une expression rationnelle.

  (2) et (3) → équivalence entre automate et ER

  → il existe des algorithmes

  automate → ER

  ER → automate

Rationalité

- Pour montrer qu'un langage est rationnel :
  - A partir de (1) : utiliser les propriétés de stabilité

    → décomposer le langage en sous ensembles par union, intersection, concaténation,

    et montrer que ces sous ensembles sont rationnels.

  - A partir de (2) : construire un automate acceptant ce langage

    (on peut éventuellement déterminiser / minimiser cet automate)

  - A partir de (3) : construire une expression rationnelle décrivant ce langage.

Non rationalité

- Il existe des langages non rationnels :
  - L'ensemble des expressions rationnelles est dénombrable
  - L'ensemble des langages est non dénombrable

- Tout langage fini est rationnel (il peut être décrit par une ER composée de l'union de tous les mots du langage).

  → La question de non rationalité ne se pose que pour les langages infinis.

- Montrer la non rationalité :
  - Stabilité et raisonnement par l'absurde
  - Lemme de l'étoile

Non rationalité

Propriétés de stabilité

- Pour montrer que L est non rationnel,

  on pose l'hypothèse que L est rationnel,

  et on détermine L_0 non rationnel et L_1 rationnel tels que

  $L_0 = L \ \theta \ L_1$ $\quad (\theta \in \{\cap, \cup, .\}$

- L supposé rationnel
- L_1 rationnel

  $L \ \theta \ L_1$ rationnel par stabilité de la classe des langages rationnels par θ
- Or $L \ \theta \ L_1 = L_0$ avec L_0 connu (démontré) non rationnel

  ➔ Contradiction

  → l'hypothèse (L rationnel) est fausse

Non rationalité
Lemme de l'étoile

- Théorème Lemme de l'étoile

  Soit L un langage rationnel infini accepté par un automate déterministe M à k états.

  Soit z un mot quelconque de L tel que $|z| \geq k$.

  Alors z peut être décomposé en uvw

  avec $|uv| \leq k$, $|v| \neq 0$ *et* $uv^iw \in L, \ \forall\, i \geq 0$.

Non rationalité
Lemme de l'étoile

- Preuve
  - Lemme 1

    Soit G le graphe d'un automate déterministe à k états.

    Tout chemin de longueur k dans G contient un cycle.
  - Lemme 2

    Soit G le graphe d'un automate déterministe à k états.

    Soit p un chemin de longueur k ou plus.

    p peut être décomposé en sous chemins q, r et s tels que p = qrs, |qr| ≤ k, r est un cycle.

Non rationalité

Lemme de l'étoile

- Exemple : Montrons que $L = \{a^n b^n \mid n \geq 0\}$ est non rationnel.

Supposons que L est rationnel. L est reconnu par un automate M à k états.
D'après le lemme de l'étoile, $\forall z \in L, |z| \geq k, \exists u, v, w \in \Sigma^*$ tels que

$$z = uvw,\ |uv| \leq k,\ |v| > 0 \text{ et } \forall i \geq 0,\ uv^i w \in L$$

Soit $z_0 = a^k b^k$.
On a bien $z_0 \in L$ et $|z_0| = 2k \geq k$.
Toutes les décompositions possibles $z_0 = uvw$ telles que $|uv| \leq k$, $|v| > 0$
sont de la forme $u = a^p$, $v = a^q$, $w = a^r b^k$ avec $q > 0$ et $p+q+r = k$.

Or $uv^i w = a^p a^{qi} a^r b^k = a^{p + qi + r} b^k$
On a $\forall i \neq 1,\ p + qi + r \neq k$
Donc $\forall i \neq 1,\ uv^i w \notin L$
Donc contradiction dans la propriété
Donc l'hypothèse (L rationnel) est fausse
Donc L non rationnel

Minimisation des états

Théorème de Myhill – Nerode

Contexte à droite relativement à un langage

- <u>Définition</u>

  Soit $L \subseteq \Sigma^*$ un langage

  On définit pour un mot $u \in \Sigma^*$ son contexte à droite relativement à L :

$$R_L(u) = \{ z \in \Sigma^* \mid uz \in L \}$$

Minimisation des états

Théorème de Myhill – Nerode

Equivalence des mots suivant un langage

- **Définition**

  Soit $L \subseteq \Sigma^*$ un langage et $x, y \in \Sigma^*$ deux mots.

  On dit que x et y sont équivalents suivant L, et on note $x \approx_L y$, si pour tout mot z de Σ^* :

  $$xz \in L \text{ ssi } yz \in L$$

- **Propriété**

  $\approx_L$ est une relation d'équivalence

  On note $[w]_L$ la classe d'équivalence de w.

Minimisation des états
Théorème de Myhill – Nerode

Equivalence des mots suivant un langage

- Exemple $L = (ab \cup ba)^*$
  Chercher les classes suivant $\approx_L$ dans Σ^*

  - [e] = L
  - [a] = La
  - [b] = Lb
  - [aa] = L (aa ∪ bb) Σ^*
  - [bb] = [aa]
  - [bba] = [aa]
  - [aaa] = [aa]
  - [ab] = L = [ba] = [e] car ab ∈ L

  ...

  On montre facilement par récurrence qu'on a toutes les classes

  - On obtient 4 classes d'équivalence :

    $\Sigma^* = L \cup La \cup Lb \cup L (aa \cup bb) \Sigma^*$

Minimisation des états

Théorème de Myhill – Nerode

Equivalence des mots suivant un automate

- **Définition**

  Soit M = (K, Σ, δ, s, F) un automate déterministe (fini), et x, y $\in \Sigma^*$ deux mots.

  On dit que x et y sont équivalents relativement à M, on note $x \sim_M y$, ssi il existe un état q de K tel que :

  $$(s, x) \vdash_M^* (q, e) \text{ et } (s, y) \vdash_M^* (q, e)$$

- **Remarque**

  M étant déterministe, si on note $q_M(x)$ l'état auquel on parvient dans M en lisant x, on a :

  $$x \sim_M y \text{ ssi } q_M(x) = q_M(y)$$

Minimisation des états

Théorème de Myhill – Nerode

Equivalence des mots suivant un automate

- Propriétés
  - $\sim_M$ est une relation d'équivalence.
  - on note E_q la classe d'équivalence des mots x tels que $q_M(x) = q$

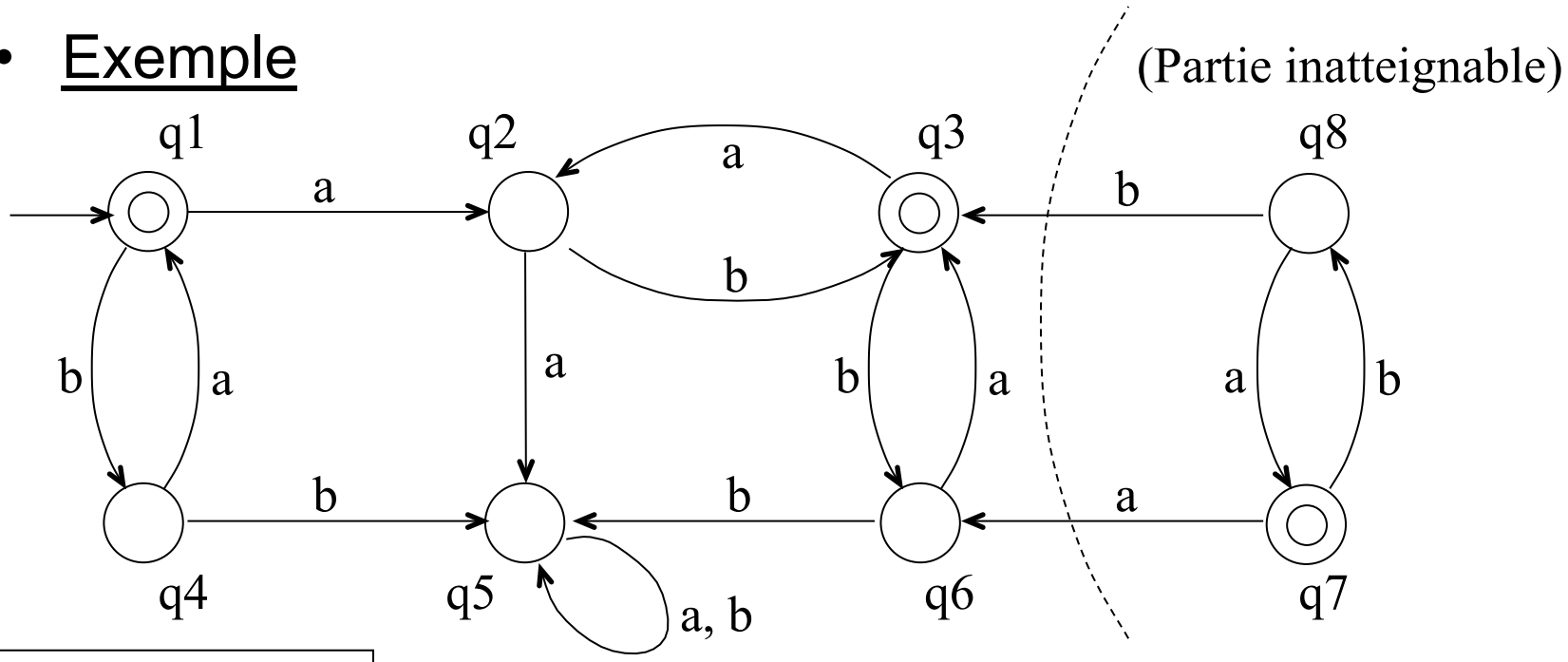
    $E_q = \varnothing$ si l'état est inatteignable
  - il y a autant de classes d'équivalence que d'états atteignables (accessibles).

Minimisation des états

Théorème de Myhill – Nerode

Equivalence des mots suivant un automate

- Exemple



[e] = $E_{q1} \cup E_{q3}$
[a] = E_{q2}
[b] = $E_{q4} \cup E_{q6}$
[aa] = E_{q5}

Minimisation des états

Théorème de Myhill – Nerode

Equivalence des mots suivant un automate

- $E_q = L(M^q)$ avec $M^q = (K, \Sigma, \delta, s, \{q\})$

- <u>Théorème</u>

  Pour tout automate fini déterministe M et deux mots x, y $\in \Sigma^*$, on a :

  si $x \sim_M y$ alors $x \approx_{L(M)} y$

  (on dit que $\sim_M$ raffine $\approx_{L(M)}$)

  Les classes de $\sim_M$ sont plus petites (et incluses)

  dans les classes de $\approx_{L(M)}$

Minimisation des états

Théorème de Myhill – Nerode

Automate standard

- Théorème de Myhill – Nerode

  Soit $L \subseteq \Sigma^*$ un langage rationnel.

  Il existe un automate déterministe ayant | $\Sigma^* / \approx_L$ | états acceptant L.

  (C'est-à-dire autant d'états que le nombre de classes d'équivalence suivant $\approx_L$.)

- Remarques
  - cet automate a le plus petit nombre d'états possibles
  - il n'y a qu'un seul automate vérifiant cela
  - on appelle cet automate l'automate standard de L (ou automate minimal de L).

Minimisation des états

Théorème de Myhill – Nerode

Automate standard

- Preuve (constructive) :

  M automate standard d'un langage L.

  M = (K, Σ, δ, s, F).

  Avec

  - K = { [x], x $\in \Sigma^*$ }
  - s = [e]
  - F = { [x], x $\in$ L }
  - δ : définie par δ([x], a) = [xa]

Minimisation des états

Théorème de Myhill – Nerode

Automate standard

- K est fini car L est reconnu par un automate déterministe M'

  $| \Sigma^* / \sim_{M'} | \geq | \Sigma^* / \approx_L |$

- δ bien définie : si [x] = [y] alors [xa] = [ya]

  (clair car : $x \approx_L y \Rightarrow xa \approx_L ya$

- L = L(M)

  (i) on montre d'abord que $([x], y) \vdash_M^* ([xy], e)$ (par induction sur |y|)

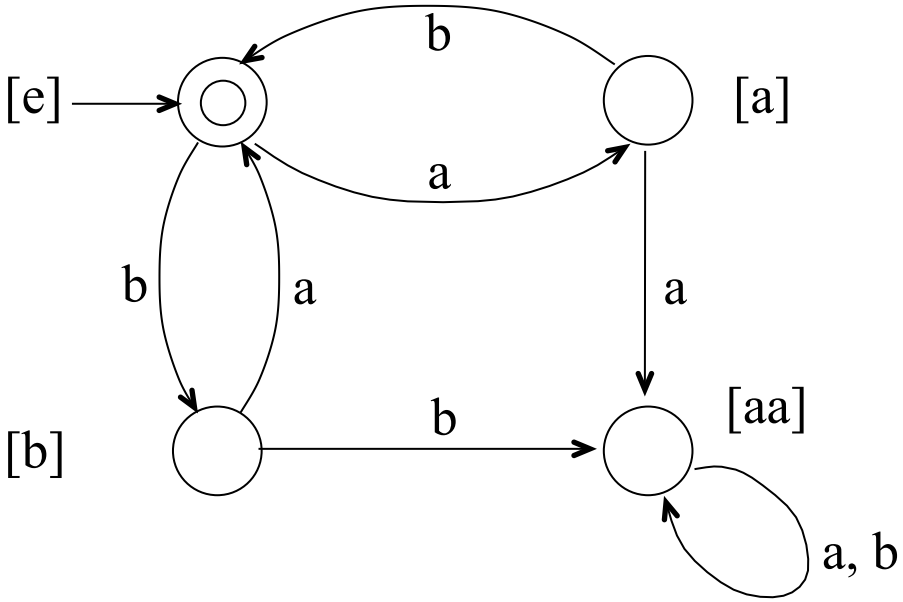
  (ii) $\forall x \in \Sigma^*$, $x \in L(M)$ ssi $([e], x) \vdash_M^* ([x], e)$ et $[x] \in F$ ie $x \in L$ par définition de F.

Minimisation des états

Théorème de Myhill – Nerode

Automate standard

- Exemple



Minimisation des états

Théorème de Myhill – Nerode

Corollaire (Myhill – Nerode)

- Théorème

  L est rationnel ssi $\approx_L$ a un nombre fini de classes d'équivalence.

Minimisation des états

Minimisation d'un automate donné

- Définition

  Soit $M = (K, \Sigma, \delta, s, F)$ un automate fini déterministe.

  On note $M(q)$, pour $q \in K$, l'automate $(K, \Sigma, \delta, \mathbf{q}, F)$.

  (Avec cette notation, $M = M(s)$.)

  On dit que p et q sont deux états équivalents de M, et on note $p \equiv q$,

  ssi $L(M(p)) = L(M(q))$.

- En d'autres termes :

  $p \equiv q$ ssi $\forall w \in \Sigma^*$,

  - soit $(p, w) \vdash_M^* (f_1, e)$ et $(q, w) \vdash_M^* (f_2, e)$ avec $f_1, f_2 \in F$
  - soit $(p, w) \vdash_M^* (g_1, e)$ et $(q, w) \vdash_M^* (g_2, e)$ avec $g_1, g_2 \notin F$

  classe d'éq. des mots x tq. $q_M(x) = p$

Minimisation des états

Minimisation d'un automate donné

- Propriété

  $M = (K, \Sigma, \delta, s, F)$ sans états inatteignables ($E_p \neq \emptyset \ \forall\ p \in K$).

  Soient p et q $\in K$.

  $(\exists\ v \in \Sigma^* \mid E_p \subset [v]$ et $E_q \subset [v]) \Leftrightarrow p \equiv q$

- Corollaire

  $\equiv$ sur K induit une relation d'équivalence sur $\Sigma^* / \sim_M$ (notée $\approx$)

  définie par $E_p \approx E_q$ ssi $p \equiv q$

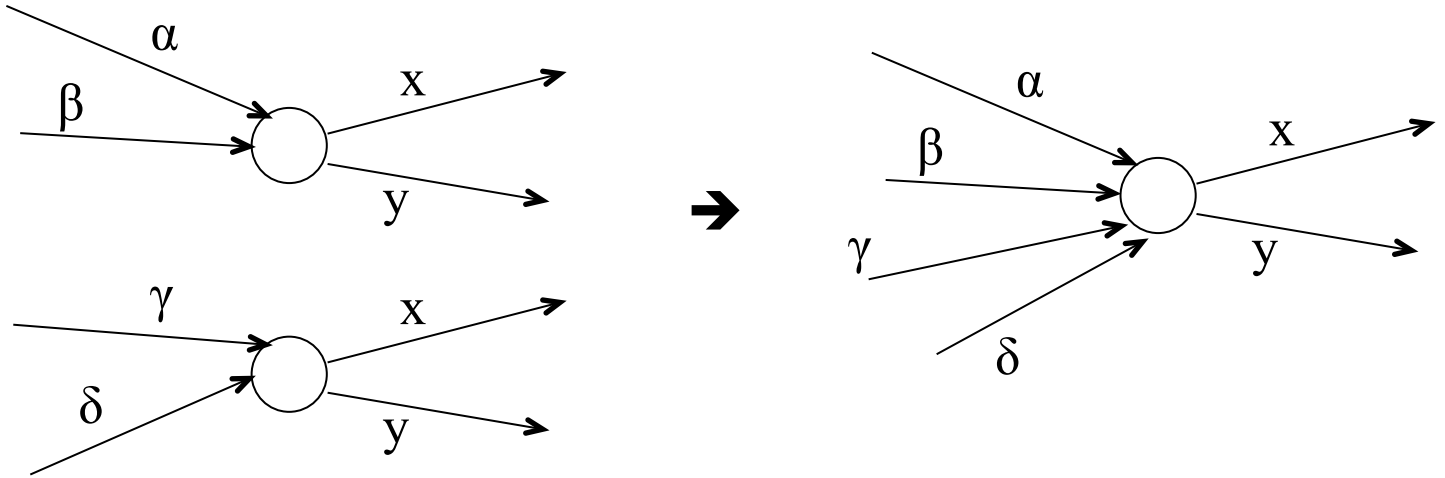
  Chaque classe d'équivalence correspond à une classe de la forme [v].

Minimisation des états

Minimisation d'un automate donné

Deux états équivalents sont ceux qu'on doit fusionner pour obtenir l'automate (minimal) standard.

On garde les flèches entrantes et on oublie les flèches sortantes de l'un des états (qui sont étiquetées par toutes les lettres de Σ)



Minimisation des états

Minimisation d'un automate donné

- Pratique

  On calcule $\equiv$ puis on fait la fusion.

  $\equiv$ est calculée comme la limite de $(\equiv_i)_{i \in N}$.

  $\equiv_i$ définie par :

  $p \equiv_i q$ ssi pour tout mot w de longueur $\leq i$ tel que

  $$(p, w) \vdash_M^* (f_1, e) \text{ et } (q, w) \vdash_M^* (f_2, e)$$

  on a $f_1 \in F \Leftrightarrow f_2 \in F$

  (ou $L(M(p)) \cap (\cup_{k=0}^{i} \Sigma^k) = L(M(q)) \cap (\cup_{k=0}^{i} \Sigma^k)$)

  On a évidemment, $\forall\, i \in N$:

  $p \equiv q \Leftrightarrow p \equiv_i q$

  et $p \equiv_i q \Leftrightarrow p \equiv_{i-1} q$

Minimisation des états

Minimisation d'un automate donné

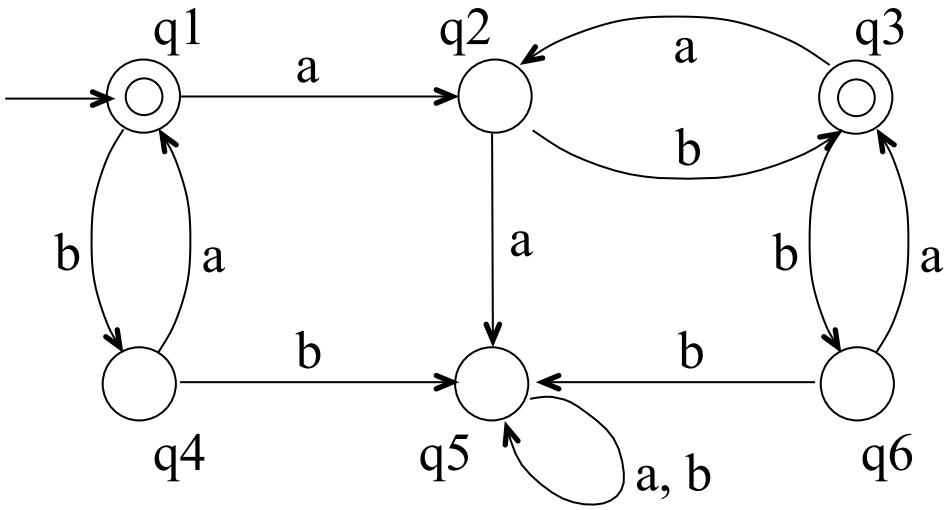
- **Propriété**

  Pour tout couple $(p, q) \in K^2$ et $n \geq 1$ on a :

  $p \equiv_n q$ ssi $p \equiv_{n-1} q$

  et $\forall \sigma \in \Sigma, \delta(p, \sigma) \equiv_{n-1} \delta(q, \sigma)$

- **Exemple**



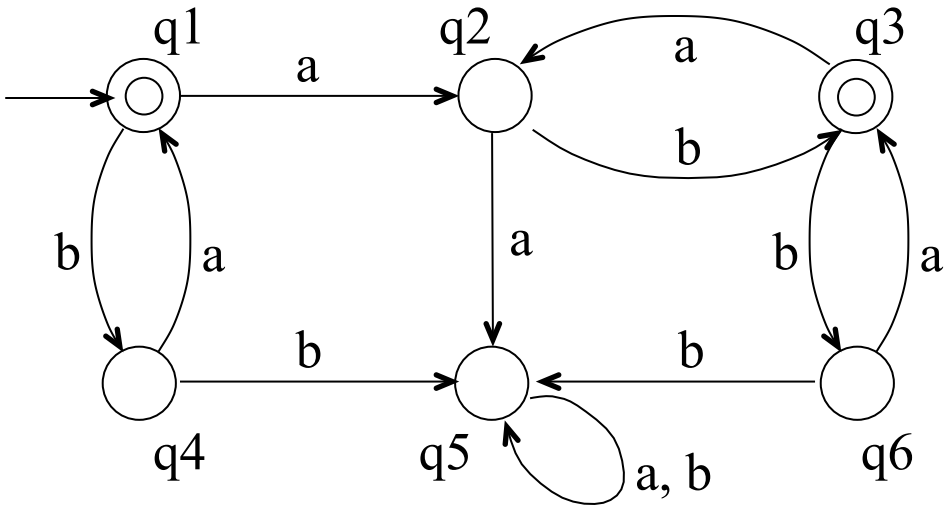
$q_1 \equiv_0 q_3$ car $q_1 \in F$ et $q_3 \in F$

$q_2 \equiv_0 q$ car $q_2 \notin F$ et $q_4 \notin F$

Minimisation des états

Minimisation d'un automate donné

- Exemple



$q_1 \equiv_1 q_3$ car $q_1 \equiv_0 q_3$ et $\delta(q_1, a) \equiv_0 \delta(q_3, a)$

$(\delta(q_1, a) = q_2$ et $\delta(q_3, a) = q_2$ et $\{q_2\})$

$\delta(q_1, b) \equiv_0 \delta(q_3, b)$

$(\delta(q_1, b) = q_4$ et $\delta(q_3, b) = q_6$ et $\{q_4, q_6\})$

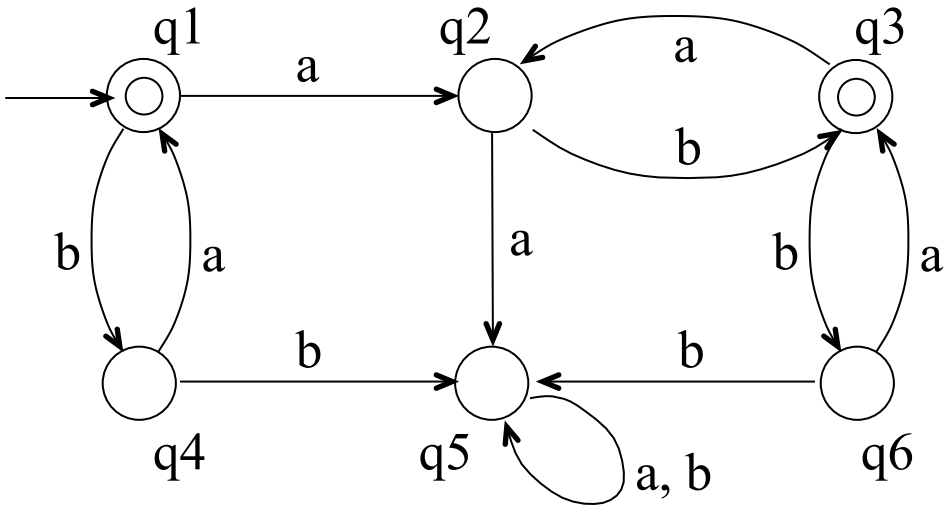
$q_2 \neg\equiv_1 q_4$ car $\delta(q_2, a) \neg\equiv_0 \delta(q_4, a)$

$(\delta(q_2, a) = q_5$ et $\delta(q_4, a) = q_1$ et $\{q_1, q_3\}$ et $\{q_5\})$

Minimisation des états

Minimisation d'un automate donné

- Exemple



$\equiv_0 : \{q_1, q_3\}, \{q_2, q_4, q_5, q_6\}$

$\equiv_1 : \{q_1, q_3\}, \{q_2\}, \{q_4, q_6\}, \{q_5\}$

$\equiv_2 : \{q_1, q_3\}, \{q_2\}, \{q_4, q_6\}, \{q_5\}$

… (Ça ne change plus)

- Finalement :

$q_1 \equiv q_3$

$q_4 \equiv q_6$

Complexité d'algorithmes pour les automates

- Théorèmes (sans démonstrations précises)

(i) Il existe un algorithme exponentiel (en le nombre d'états)

Entrée : un automate fini non déterministe

Sortie : un automate fini déterministe équivalent

(ii) Il existe un algorithme polynomial (en fonction de la taille de l'expression ou du nombre d'opérateurs)

Entrée : une expression rationnelle

Sortie : un automate non déterministe équivalent

Complexité d'algorithmes pour les automates

(iii) Il existe un algorithme exponentiel (en fonction du nombre d'états)

Entrée : un automate non déterministe

Sortie : une expression rationnelle équivalente

(la taille des R(i, j, k) est multipliée par 3 à chaque incrément de k)

$$R(i, j, k) = R(i, j, k-1) \cup R(i, k, k-1)\, R(k, k, k-1)^*\, R(k, j, k-1)$$

(iv) Il existe un algorithme polynomial (en fonction du nombre d'états)

Entrée : un automate déterministe

Sortie : l'automate déterministe minimal (standard) équivalent

Complexité d'algorithmes pour les automates

(v) il existe un algorithme polynomial pour décider si deux automates déterministes sont équivalents

(Passe par l'automate standard)

(vi) Il existe un algorithme exponentiel pour déterminer si deux automates non déterministes son équivalents

Complexité d'algorithmes pour les automates

- Théorème

  L = langage rationnel (donné par un automate ou une expression rationnelle) et $w \in \Sigma^*$

  Il existe un algorithme qui teste si $w \in L$ *avec une complexité en temps de* $O(|w|)$

- Théorème

  $M = (K, \Sigma, \Delta, s, F)$ *non déterministe et* $w \in \Sigma^*$

  Il existe un algorithme qui teste si $w \in L(M)$ *avec une complexité en temps de* $O(|K|^2 |w|)$

LIF15 – Théorie des langages formels

Sylvain Brandel
2015 – 2016
sylvain.brandel@univ-lyon1.fr

Chapitre 3

LANGAGES ALGÉBRIQUES

Grammaires algébriques

- Exemple
  - L = a(ab ∪ ba)*b. Mots générés : un *a* suivi de *ab* ou *ba* un certain nombre de fois suivi d'un *b*.
  - Un mot de L peut naturellement être décomposé en un début, un milieu et une fin.

$$S \rightarrow aE$$
$$E \rightarrow abE$$
$$E \rightarrow baE$$
$$E \rightarrow b$$

  - Génération :

$$S \rightarrow aE \rightarrow aabE \rightarrow aabbaE \rightarrow aabbab$$
$$S \rightarrow aE \rightarrow abaE \rightarrow ababaE \rightarrow ababaabE \rightarrow ababaabb$$

Grammaires

- Une grammaire algébrique est composée :
  - d'un alphabet Σ de symboles terminaux à partir desquels sont construits les mots du langage (*a* et *b* dans l'exemple),
  - d'un ensemble de symboles non terminaux (S et E dans l'exemple) parmi lesquels on distingue un symbole particulier (souvent S pour *Start*),
  - d'un ensemble fini de règles ou production de la forme

    symbole non terminal → suite finie de symboles terminaux et / ou non terminaux.

Grammaires

- Définition

  Une grammaire algébrique est définie par un quadruplet

  $G = (V, \Sigma, R, S)$ où :

  – Σ est un ensemble fini de symboles terminaux appelé alphabet,

  – V est un ensemble fini de symboles non terminaux tels que

  $V \cap \Sigma = \varnothing$,

  – $S \in V$ est le symbole initial

  – R est un sous-ensemble fini de $V \times (V \cup \Sigma)^*$

  Les éléments de R sont appelés règles ou productions.

Grammaires

- Définition

  Soient u et v $\in (V \cup \Sigma)^*$.

  On dit que v dérive directement de u, et on note $u \Rightarrow_G v$,

  ssi $\exists$ x, y, w $\in (V \cup \Sigma)^*$, $\exists$ A $\in$ V tels que

  $u = xAy$ et $v = xwy$ et $A \rightarrow w \in R$

- Exemple

  En utilisant la grammaire G définie par les règles suivantes :

  $S \rightarrow aE$

  $E \rightarrow abE \mid baE \mid b$

  on obtient $aabE \Rightarrow_G aabbaE$ par application de la règle $E \rightarrow baE$.

- Définition

  La relation $\Rightarrow_G^*$ est la fermeture réflexive transitive de la relation $\Rightarrow_G$.

Grammaires

- **Définition**

  Soient u et v $\in (V \cup \Sigma)^*$.

  On dit que v dérive de u, et on note $u \Rightarrow_G^* v$,

  ssi $\exists\ w_0, \ldots, w_n \in (V \cup \Sigma)^*$ tels que

  $u = w_0$ et $v = w_n$ et $w_i \Rightarrow_G w_{i+1}\ \forall\ i < n$.

  La suite $w_0 \Rightarrow_G w_1 \Rightarrow_G \ldots \Rightarrow_G w_n$ est appelée une dérivation (lorsqu'il n'y a pas d'ambiguïté, on peut omettre la référence à la grammaire G dans les symboles $\Rightarrow_G$ et $\Rightarrow_G^*$).

  La valeur de n ($n \geq 0$) est la longueur de la dérivation.

Grammaires

- **Définition**

  Soit G = (V, Σ, R, S) une grammaire algébrique.

  Le langage engendré par G, noté L(G), est :

  $$L(G) = \{ w \in \Sigma^* \mid S \Rightarrow_G^* w \}$$

- **Définition**
  - Un langage est dit algébrique s'il peut être engendré par une grammaire algébrique.

Grammaires

- Exemple

  G = (V, Σ, R, S) avec :
  - V = { S, E }
  - Σ = { a, b }
  - R = { S → EE, E → EEE | bE | Eb | a }

- La chaîne ababaa peut être dérivée de plusieurs façons :

(a)	(b)	(c)	(d)
S ⇒ EE	S ⇒ EE	S ⇒ EE	S ⇒ EE
⇒ EEEE	⇒ aE	⇒ Ea	⇒ aE
⇒ aEEE	⇒ aEEE	⇒ EEEa	⇒ aEEE
⇒ abEEE	⇒ abEEE	⇒ EEbEa	⇒ aEEa
⇒ abaEE	⇒ abaEE	⇒ EEbaa	⇒ abEEa
⇒ ababEE	⇒ ababEE	⇒ EbEbaa	⇒ abEbEa
⇒ ababaE	⇒ ababaE	⇒ Ebabaa	⇒ ababEa
⇒ ababaa	⇒ ababaa	⇒ ababaa	⇒ ababaa

Grammaires

- Types de dérivations
  - (a) et (b) : chaque étape de la dérivation consiste à transformer le non-terminal le plus à gauche. On appelle ce genre de dérivation une dérivation la-plus-à-gauche (*left-most derivation*).
  - (c) : dérivation la-plus-à-droite (*right-most derivation*) où le symbole transformé à chaque étape est le non-terminal le plus à droite.
  - (d) : dérivation ni plus-à-droite, ni plus-à-gauche.

- Une suite de dérivations peut être visualisée par un arbre de dérivation ou arbre syntaxique (ang. *parse tree*).
  - Un tel arbre indique quelles sont les règles appliquées aux non-terminaux.
  - Il n'indique pas l'ordre d'application des règles.
  - Les feuilles de l'arbre représentent la chaîne dérivée.

Grammaires

- Lorsqu'une grammaire peut produire plusieurs arbres distincts associés à un même mot, ont dit que la grammaire est ambiguë.

- Définition

  Deux grammaires qui engendrent le même langage sont dites équivalentes.

Langages rationnels et grammaires algébriques

- Il existe des langages algébriques qui ne sont pas rationnels.

- Définition

  Une grammaire algébrique $G = (V, \Sigma, R, S)$ telle que

  $$R \subseteq V \times \Sigma^*(V \cup \{e\})$$

  est appelée grammaire régulière (ou encore linéaire à droite).

  (rappel : dans une grammaire algébrique (non régulière),

  $$R \subseteq V \times (V \cup \Sigma)^*.)$$

Langages rationnels et grammaires algébriques

- Exemple

  G = (V, Σ, R, S) avec :

  – V = { S }

  – Σ = { a, b }

  – R = { S → aaS | bbS | e }

  grammaire régulière : L(G) = (aa ∪ bb)*

Langages rationnels et grammaires algébriques

- Théorème

  Un langage est rationnel si et seulement si il est engendré par une grammaire régulière.

  Ce théorème exprime que tout langage rationnel est algébrique (et non l'inverse).

- On peut utiliser la preuve de ce théorème pour :
  - passer d'un automate (déterministe ou non) à une grammaire,
  - passer d'une grammaire à un automate.

Langages rationnels et grammaires algébriques

- Exemple ($G \Rightarrow M$)

  $G = (V, \Sigma, R, S)$ une grammaire régulière avec :

  – $V = \{ S, X, Y \}$
  – $\Sigma = \{ a, b \}$
  – $R = \{ S \rightarrow aX \mid bY, X \rightarrow aS \mid a, Y \rightarrow bS \mid b \}$

  Construisons l'automate M acceptant L(G), en utilisant la preuve du théorème précédent.

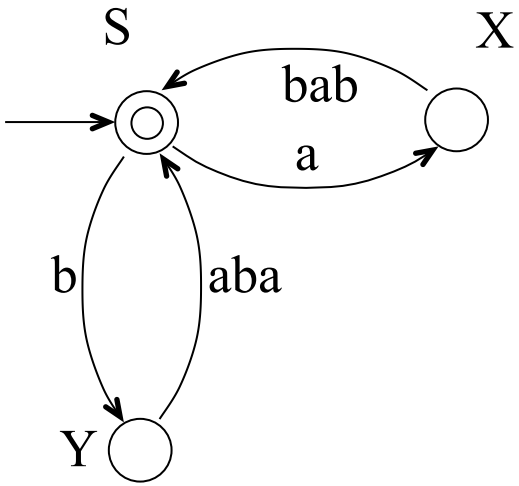
  Pour cela on associe à chaque non-terminal de G un état dans M de la façon suivante :

  – si $A \rightarrow wB \in R$ alors on crée dans M une transition de l'état A vers l'état B étiquetée par w,
  – si $A \rightarrow w \in R$ alors on crée dans M une transition de l'état A vers l'état F, où F est le seul état introduit dans M qui ne correspond à aucun non-terminal de G.

Langages rationnels et grammaires algébriques

- Exemple (M ⇒ G)

  Soit l'automate :



Construisons une grammaire régulière G engendrant le même langage, en utilisant la preuve du théorème.

Pour cela on associe à chaque transition de M une règle dans G de la façon suivante :

– pour toute transition de l'état p vers l'état q étiquetée par w, on crée la règle correspondante dans G : P → wQ,
– pour tout état final f de M, on crée dans G une règle d'effacement : F → e.

Automates à pile

- Définition

  Un automate à pile est un sextuplet

  $M = (K, \Sigma, \Gamma, \Delta, s, F)$ où :

  – K est un ensemble fini d'états,

  – Σ est un ensemble fini de symboles d'entrée appelé alphabet,

  – Γ est un ensemble fini de symboles de la pile,

  – $s \in K$ est l'état initial,

  – $F \subseteq K$ est l'ensemble des états finaux,

  – Δ est un sous-ensemble fini de

  $$\big(K \times (\Sigma \cup \{e\}) \times (\Gamma \cup \{e\}) \big) \times \big(K \times (\Gamma \cup \{e\}) \big)$$

  appelé fonction de transition.

Automates à pile

- Une transition $((p, a, A), (q, B)) \in \Delta$ où :
  - p est l'état courant,
  - a est le symbole d'entrée courant,
  - A est le symbole sommet de la pile,
  - q est le nouvel état,
  - B est le nouveau symbole en sommet de pile,

a pour effet :

(1) de passer de l'état p à l'état q,

(2) d'avancer la tête de lecture après a,

(3) de dépiler A du sommet de la pile,

(4) d'empiler B sur la pile.

Automates à pile

- **Définition**

  Soit M = (K, Σ, Γ, Δ, s, F) un automate à pile. Une configuration de M est définie par un triplet

  $(q_i, w, \alpha) \in K \times \Sigma^* \times \Gamma^*$ où :

  – q_i est l'état courant de M,

  – w est la partie de la chaîne restant à analyser,

  – α est le contenu de la pile.

- **Définition**

  Soient (q_i, u, α) et (q_j, v, β) deux configurations d'un automate à pile M = (K, Σ, Γ, Δ, s, F). On dit que (q_i, u, α) conduit à (q_j, v, β) en une étape

  ssi $\exists\, \sigma \in (\Sigma \cup \{e\}), \exists\, A, B \in (\Gamma \cup \{e\})$ tels que :

  $u = \sigma v$ et $\alpha = \alpha' A$ et $\beta = \beta' B$ et $((q_i, \sigma, A), (q_j, B)) \in \Delta$.

  On note $(q_i, u, \alpha) \vdash_M (q_j, v, \beta)$.

Automates à pile

- **Définition**

  La relation $\vdash_M^*$ est la fermeture réflexive transitive de $\vdash_M$.

- **Définition**

  Soit M = (K, Σ, Γ, Δ, s, F) un automate à pile. Une chaîne w $\in \Sigma^*$ est acceptée par M

  ssi (s, w, e) $\vdash_M^*$ (f, e, e) avec f $\in$ F.

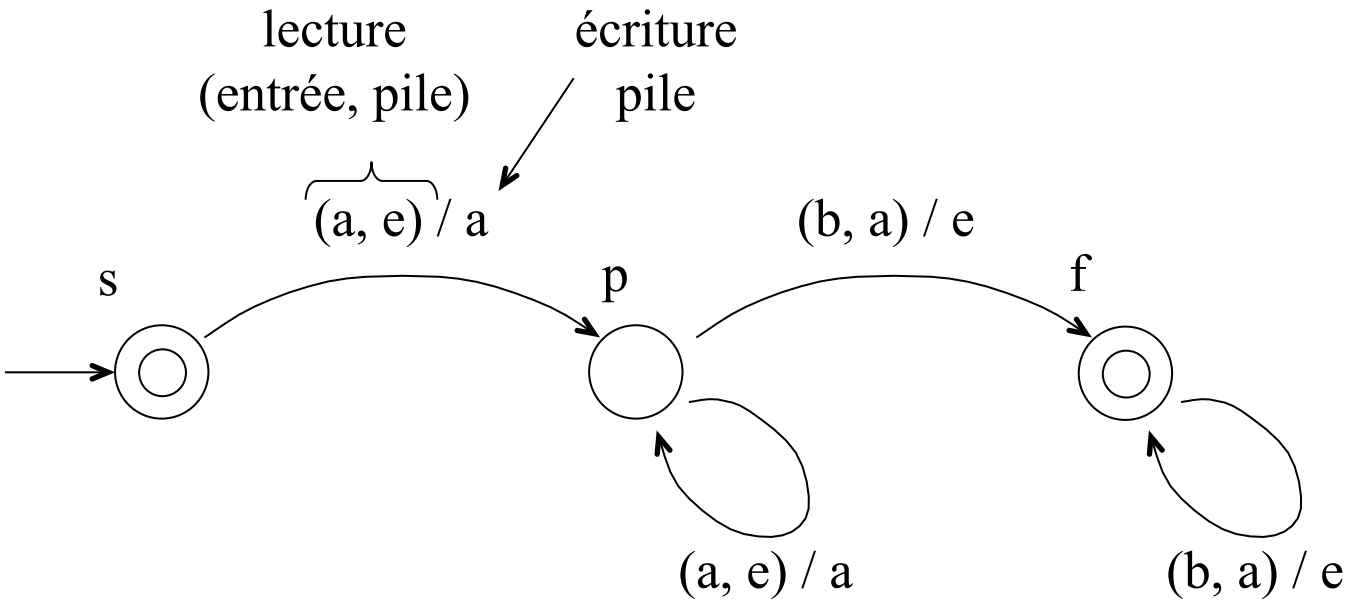
- **Définition**

  Le langage accepté par M, noté L(M), est l'ensemble des chaînes acceptées par M.

Automates à pile

- Soit l'automate à pile M = (K, Σ, Γ, Δ, s, F) avec :
  - K = {s, p, f}
  - Σ = {a, b}
  - Γ = {a, b}
  - F = {s, f}

Δ = { ((s, a, e), (p, a)),
((p, a, e), (p, a)),
((p, b, a), (f, e)),
((f, b, a), (f, e)) }



Automates à pile

- Un automate à pile est déterministe s'il y a au plus une transition applicable pour tout triplet de la forme

  (État courant, symbole d'entrée, sommet de pile).

Automates à pile et grammaires algébriques

- Théorème

  La classe des langages acceptés par les automates à pile est égale à la classe des langages engendrés par les grammaires algébriques.

- Définition

  Un automate à pile est dit simple ssi quelle que soit la transition $((p, a, \alpha), (q, \beta)) \in \Delta$, on a :

  $\alpha \in \Gamma$ (sauf pour p = S où on ne dépile rien) et $|\beta| \leq 2$

- Proposition

  On peut transformer tout automate à pile en un automate simple équivalent.

Propriétés des langages algébriques

- Pour montrer qu'un langage est algébrique, on peut :
  - soit définir une grammaire algébrique qui engendre ce langage,
  - soit définir un automate à pile qui l'accepte.

- Il est également possible d'utiliser les propriétés de stabilité de la classe des langages algébriques

Propriétés des langages algébriques
Propriétés de stabilité

- **Théorème**

  La classe des langages algébriques est stable par les opérations d'union, de concaténation et d'étoile de Kleene.

- **Preuve**

  Soient deux grammaires $G_1 = (V_1, \sum_1, R_1, S_1)$ et $G_2 = (V_2, \sum_2, R_2, S_2)$, avec $V_1 \cap V_2 = \varnothing$. (On renomme éventuellement les non-terminaux.)

  La preuve (constructive) consiste à :

  - construire une grammaire G à partir de G_1 et G_2 validant les propriétés de stabilité,
  - montrer que $L(G) = L(G_1)$ op $L(G_2)$ (op $\in \{\cup, .\}$) et $L(G) = L(G_1)^*$.

Propriétés des langages algébriques
Propriétés de stabilité

- Preuve

(a) Union

Soit G = (V, $\sum$, R, S) avec :

- V = $V_1 \cup V_2 \cup \{S\}$ où $S \notin V_1 \cup V_2$ (renommage éventuel)
- $\sum = \sum_1 \cup \sum_2$
- R = $R_1 \cup R_2 \cup \{S \rightarrow S_1 \mid S_2\}$

(b) Concaténation

Soit G = (V, $\sum$, R, S) avec :

- V = $V_1 \cup V_2 \cup \{S\}$ où $S \notin V_1 \cup V_2$ (renommage éventuel)
- $\sum = \sum_1 \cup \sum_2$
- R = $R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\}$

(c) Opération étoile

Soit G = (V, $\sum$, R, S) avec :

- V = $V_1 \cup \{S\}$ où $S \notin V_1$ (renommage éventuel)
- $\sum = \sum_1$
- R = $R_1 \cup \{S \rightarrow S_1 S \mid e\}$

Propriétés des langages algébriques
Propriétés de stabilité

- Remarque

  Contrairement à la classe des langages rationnels, la classe des langages algébriques n'est pas stable par intersection et complémentation.

- Théorème

  L'intersection d'un langage rationnel et d'un langage algébrique est algébrique

Propriétés des langages algébriques

Lemme de l'étoile pour les langages algébriques

- Théorème (lemme de la double étoile)

  Soit L un langage algébrique.

  Il existe un nombre k, dépendant de L, tel que tout mot $z \in L$, $|z| \geq k$, peut être décomposé en z = uvwxy avec :

  (*i*) $|vwx| \leq k$

  (*ii*) $|v| + |x| > 0$ (ie. $v \neq e$ ou $x \neq e$)

  (*iii*) $uv^nwx^ny \in L, \forall\ n \geq 0$ (d'où l'appellation de double étoile : v^n et x^n = v* et x*)

Propriétés des langages algébriques

Lemme de l'étoile pour les langages algébriques

- Pour le montrer on utilise la forme normale de Chomsky.

- <u>Définition</u>

  Une grammaire algébrique G = (V, $\sum$, R, S) est sous forme normale de Chomsky si chaque règle est de la forme :

  $A \rightarrow BC$ avec $B, C \in V - \{S\}$

  ou $A \rightarrow \sigma$ avec $\sigma \in \sum$

  ou $A \rightarrow e$

- <u>Théorème</u>

  Pour toute grammaire algébrique, il existe une grammaire sous forme normale de Chomsky équivalente.

Propriétés des langages algébriques

Lemme de l'étoile pour les langages algébriques

- **Lemme**

  Soit $G = (V, \sum, R, S)$ une grammaire algébrique sous forme normale de Chomsky.

  Soit $S \Rightarrow_G^* w$ une dérivation de $w \in \sum^*$ dont l'arbre de dérivation est noté T.

  Si la hauteur de T est n alors $|w| \leq 2^{n-1}$.

- **Corollaire**

  Soit $G = (V, \sum, R, S)$ une grammaire algébrique sous forme normale de Chomsky.

  Soit $S \Rightarrow_G^* w$ une dérivation de $w \in L(G)$.

  Si $|w| \geq 2^n$ alors l'arbre de dérivation est de hauteur $\geq n+1$.

Propriétés des langages algébriques

Lemme de l'étoile pour les langages algébriques

- Exemple

  Montrons que L = { $a^i b^i c^i$ | i ≥ 0} est non algébrique.

  Supposons que L est algébrique.

  D'après le lemme de la double étoile, il existe une constante k, dépendant de L, telle que :

  ∀ z ∈ L, |z| ≥ k, z peut être décomposé en z = uvwxy avec :

  (*i*) |vwx| ≤ k

  (*ii*) |v| + |x| > 0

  (*iii*) $uv^n wx^n y \in L$, ∀ n ≥ 0

Propriétés des langages algébriques

Lemme de l'étoile pour les langages algébriques

- Exemple

  Considérons la chaîne particulière $z_0 = a^k b^k c^k$.

  On a bien $z_0 \in L$ et $|z_0| = 3k \geq k$.

  Les décompositions de $z_0 = uvwxy$ satisfaisant $|vwx| \leq k$ et $|v| + |x| > 0$ sont telles que :

  – soit l'une des sous-chaînes v ou x contient plus d'un type de symbole, c-à-d de la forme a^+b^+ ou b^+c^+.

  → uv^nwx^ny avec $n > 1$ contient un a après un b ou un b après un c.

  (par exemple uv^2wx^2y = u aabb aabb w x x y, si v = aabb)

  donc la chaîne uv^nwx^ny n'est plus de la forme $a^pb^pc^p$ avec $p \geq 0$,

  donc $uv^nwx^ny \notin L$ pour $n > 1$.

  – soit v et x sont des sous-chaînes de a^k ou de b^k ou de c^k.

  Comme au plus une des chaînes v ou x est vide, toute chaîne de la forme uv^nwx^ny avec $n > 1$ est caractérisée par une augmentation de un (v = e ou x = e) ou deux (v ≠ e et x ≠ e) des trois types de terminaux.

  donc pour $n > 1$, la chaîne uv^nwx^ny est de la forme $a^pb^qc^r$ mais avec $p \neq q$ ou $q \neq r$.

  donc $uv^nwx^ny \notin L$ pour $n > 1$.

  Pour toutes les décompositions possibles de la chaîne z_0 il y a une contradiction.

  Donc l'hypothèse est fausse.

  ⇒ L non algébrique.

Propriétés des langages algébriques

Lemme de l'étoile pour les langages algébriques

- Preuve de non algébricité

  Pour montrer qu'un langage est non algébrique, on peut utiliser :

  – le lemme de la double étoile,

  – les propriétés de stabilité de la classe des langages algébriques,

  – le théorème qui dit que l'intersection d'un langage algébrique et d'un langage rationnel est algébrique.

Problèmes indécidables pour les langages algébriques

- Notion de problème indécidable

  Une question est décidable s'il existe un algorithme (c'est-à-dire un processus déterministe) qui s'arrête avec une réponse (oui ou non) pour chaque entrée.

  Une question est indécidable si un tel algorithme n'existe pas.

Problèmes indécidables pour les langages algébriques

- Théorème

  Les questions suivantes sont décidables :

  – Étant donnés une grammaire algébrique G et un mot w, est-ce que $w \in L(G)$?

  – Étant donnée une grammaire algébrique G, est-ce que $L(G) = \varnothing$?

  Les questions suivantes sont indécidables :

  – *Soit G une grammaire algébrique. Est-ce que $L(G) = \Sigma^*$?*

  – *Soient G_1 et G_2 deux grammaires algébriques. Est-ce que $L(G_1) = L(G_2)$?*

  – *Soient M_1 et M_2 deux automates à pile. Est-ce que $L(M_1) = L(M_2)$?*

  – *Soit M un automate à pile. Trouver un automate à pile équivalent minimal en nombre d'états.*

À suivre …