Modèles de calculs

Les automates finis

Florent Madelaine

Fondements de l’informatique

Plan

Introduction

Nous nous penchons sur un premier modèle de calcul, celui des automates finis.
On travaille ici sur des problèmes de décisions, c'est-à-dire qu'on calcule des fonctions indicatrices de $f : \mathbb{N} \to \{\text{oui}, \text{non}\}$.
On code l'entier en entrée sous forme d'un mot. En jargon, on parle d'un langage.
On verra en chemin :

- qu'on peut spécifier un langage de manière non ambiguë avec les expressions régulières ;
- que le non-déterminisme n'apporte pas plus de pouvoir d'expressivité au modèle ;
- qu'on peut tester si deux automates reconnaissent le même langage.

Un peu de vocabulaire

- Alphabet : Σ ensemble fini fixé de symboles.
- Entrée : un mot comportant un nombre fini de symboles de Σ.
- Problème de décision (langage) : ensemble de mots.

Exemple

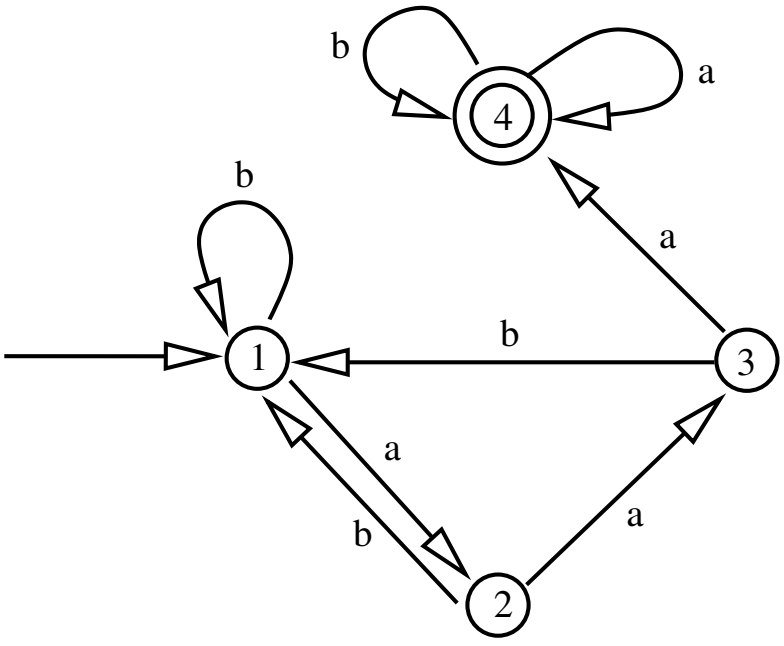
- $\Sigma = \{0, 1\}$
- Problème des mots qui sont des palindromes (qui sont identiques quel que soit le sens de lecture).
- 101101 est un mot du langage des palindromes.
- 100 n'est pas un mot de ce langage.

Autres exemples de langages

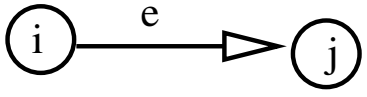
- L'ensemble des mots sur $\{0, 1\}$ représentant un entier pair.
- L'ensemble des mots sur $0, 1$ dont la première et dernière lettre sont identiques.
- L'ensemble des mots sur $0, 1$ dont le nombre de 0 et de 1 est égal.
- L'ensemble des mots sur $\{0, 1\}$ représentant un nombre premier.

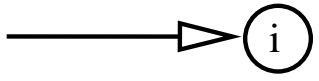
Démo JFLAP

L'automate



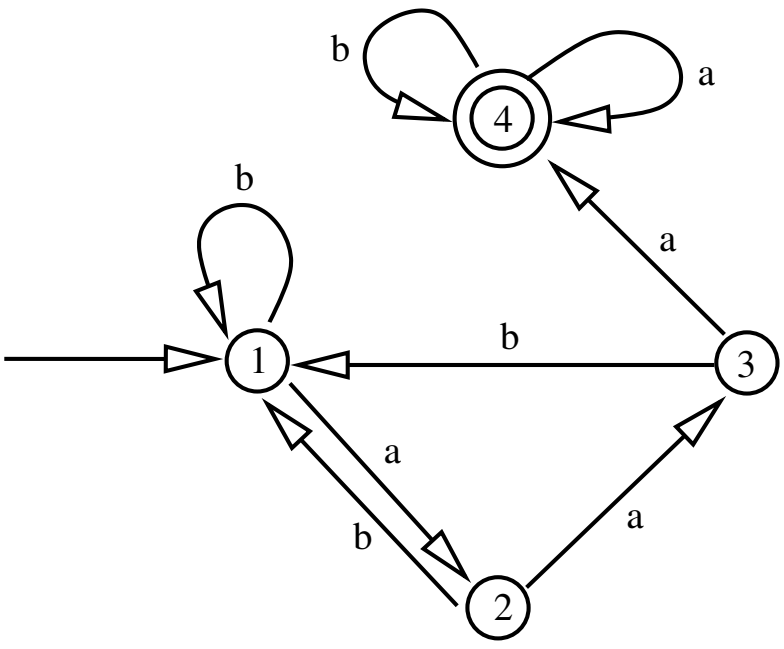
- **État** 

- **Transition** 

- **État initial** 

- **État(s) acceptant(s)**


Le calcul



- Donnée : w
- Début : état initial
- Lire w de gauche à droite lettre par lettre en suivant les transitions
- Fin :
  - Mot terminé
  - Pas de transition possible
- Réponse :
  - Accepté
  - Refusé

Vocabulaire

- état
- état acceptant
- état initial
- transition
- mot accepté par l'automate
- mot rejeté par l'automate

Les mots

- Nos automates manipulent des mots.
- On se fixe un alphabet fini Σ, par exemple, $\Sigma = \{a, b\}$.
- Les mots sont des séquences finies de lettres, par exemple, $abba$ ou encore a ou encore aaa.
- Il y a un mot spécial qu'on appelle le mot vide et qu'on note ϵ (ou encore λ dans certains livres). C'est le mot spécial qui n'a aucune lettre.

Les langages

- On appelle langage un ensemble, fini ou infinis, de mots (sur le même alphabet Σ).
- On note $\Sigma^\star$ le langage de tous les mots (dont le mot vide ϵ).
- On note $\varnothing$ le langage qui ne contient aucun mot.
- Attention à ne pas confondre $\varnothing$ avec $\{\epsilon\}$ (le langage qui contient un seul mot : le mot vide)

Définition

Le langage reconnu par un automate est l'ensemble des mots acceptés par cet automate. On parle aussi du langage de l'automate.

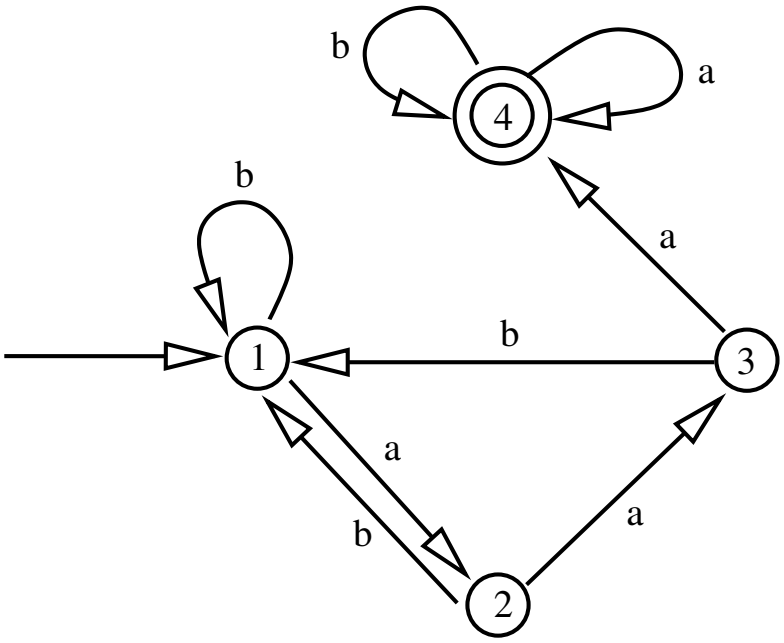
Les langages

- On appelle langage un ensemble, fini ou infinis, de mots (sur le même alphabet Σ).
- On note $\Sigma^\star$ le langage de tous les mots (dont le mot vide ϵ).
- On note $\varnothing$ le langage qui ne contient aucun mot.
- Attention à ne pas confondre $\varnothing$ avec $\{\epsilon\}$ (le langage qui contient un seul mot : le mot vide)

Définition

Le langage reconnu par un automate est l'ensemble des mots acceptés par cet automate. On parle aussi du langage de l'automate.

Décrire un automate



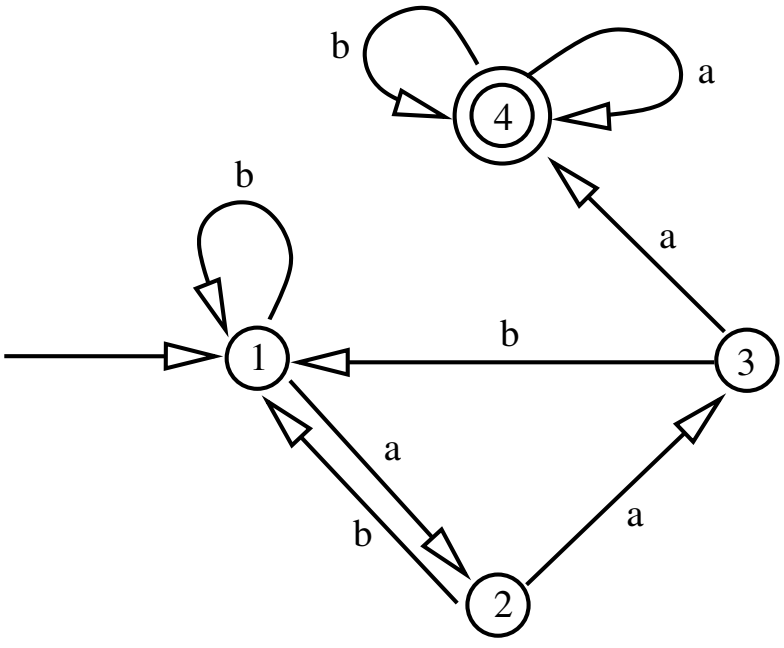
- Alphabet $\Sigma = \{a, b\}$
- Ensemble (fini) d'états $Q = \{1, 2, 3, 4\}$
  - État initial $q_0 = 1$
  - État(s) acceptant(s) $Q_A = \{4\}$
- Table de transition δ

	a	b
1	2	1
2	3	1
3	4	1
4	4	4

Intuition

- états = mémoire finie de l'automate
- tableau de transition = programme de l'automate

Utilisations



- Modélisation des automates industriels
- Spécification de protocoles
- Recherche de mots dans un texte (compilation, génomique, etc.)
- Preuve de programmes
- Informatique théorique (« modèle de calcul »)

Problème : comment décrire un langage autrement qu'en donnant un automate ?

- La description d'un langage en Français est souvent ambigüe
- On a besoin d'une façon plus mathématique de décrire les langages des automates, appelée « les expressions régulières »
- On utilise pour cela trois opérations : la concaténation, la réunion et l'étoile

Expressions régulières

- La concaténation L_1L_2 :
  un mot du langage L_1 suivi d'un mot du langage L_2

Exemple

$$L_1 = \{aa, bb\} \text{ et } L_2 = \{ab, bba, b\}$$
$$\Downarrow$$
$$L_1L_2 = \{aaab, aabba, aab, bbab, bbbba, bbb\}$$

Expressions régulières

- La réunion $L_1 + L_2$:
  un mot du langage L_1 ou bien du langage L_2

Exemple

$$L_1 = \{aa, bb\} \text{ et } L_2 = \{ab, bba, b\}$$
$$\Downarrow$$
$$L_1 + L_2 = \{aa, bb, ab, bba, b\}$$

Expressions régulières

- L'étoile L^* : concaténation d'un nombre quelconque (peut-être nul) de mots du langage L

Exemple

$$L = \{aa, bb\}$$
$$\Downarrow$$
$$L^* = \left\{ \begin{array}{l} \varepsilon, \\ aa, bb, \\ aaaa, aabb, bbaa, bbbb, \\ aaaaaa, aaaabb, aabbaa, \ldots \\ \ldots \end{array} \right\}$$

- Formellement, $L^\star = \{\epsilon\} + L + LL + LLL + \ldots$.

Expressions régulières

Remarques

- Langage composé d'un seul mot : $\{abba\}$ est noté $abba$
- Langage fini : $\{ab, bba, b\}$ est noté $ab + bba + b$
- Langage composé de tous les mots : Σ^* est aussi noté $(a + b)^*$

Expression régulière représentant un langage donné

- Sur l'alphabet $\{a, b\}$, le langage constitué des mots qui contiennent aa ou bb

Expression régulière représentant un langage donné

- Sur l'alphabet $\{a, b\}$, le langage constitué des mots qui contiennent aa ou bb

$$(a+b)^*(aa+bb)(a+b)^*$$

Expression régulière représentant un langage donné

- Sur l'alphabet $\{a, b\}$, le langage constitué des mots qui contiennent aa ou bb

$$(a+b)^*(aa+bb)(a+b)^*$$

ou

$$(a+b)^*aa(a+b)^* + (a+b)^*bb(a+b)^*$$

Expression régulière représentant un langage donné

- Sur l'alphabet $\{a, b\}$, le langage constitué des mots qui commencent par aa et finissent par bb

Expression régulière représentant un langage donné

- Sur l'alphabet $\{a, b\}$, le langage constitué des mots qui commencent par aa et finissent par bb

$$aa(a+b)^*bb$$

Expression régulière représentant un langage donné

- Sur l'alphabet $\{a, b, c\}$, le langage constitué des mots qui commencent par aa et finissent par bb

Expression régulière représentant un langage donné

- Sur l'alphabet $\{a, b, c\}$, le langage constitué des mots qui commencent par aa et finissent par bb

$$aa(a + b + c)^* bb$$

Expression régulière représentant un langage donné

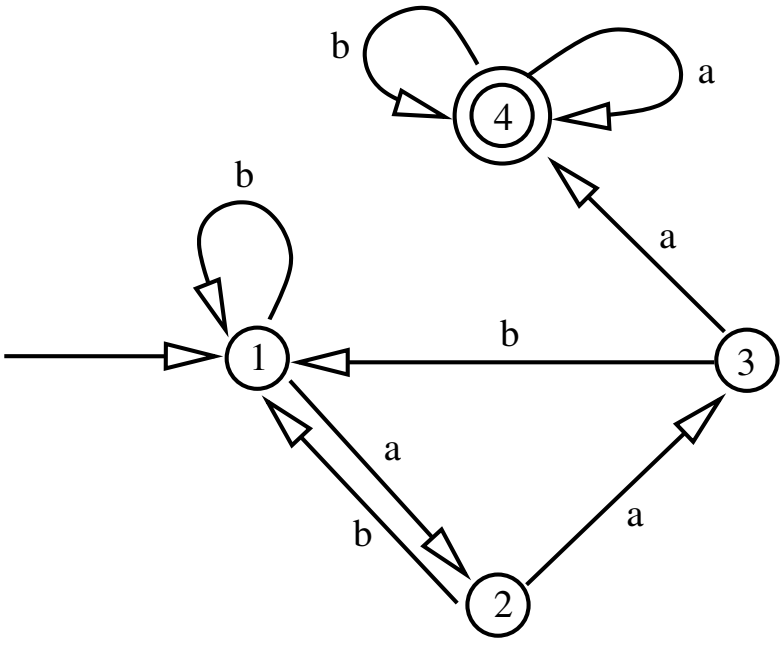
- Sur l'alphabet $\{a, b\}$, le langage constitué des mots qui commencent par aa ou finissent par bb

Expression régulière représentant un langage donné

- Sur l'alphabet $\{a, b\}$, le langage constitué des mots qui commencent par aa ou finissent par bb

$$aa(a+b)^* + (a+b)^*bb$$

Utilisations



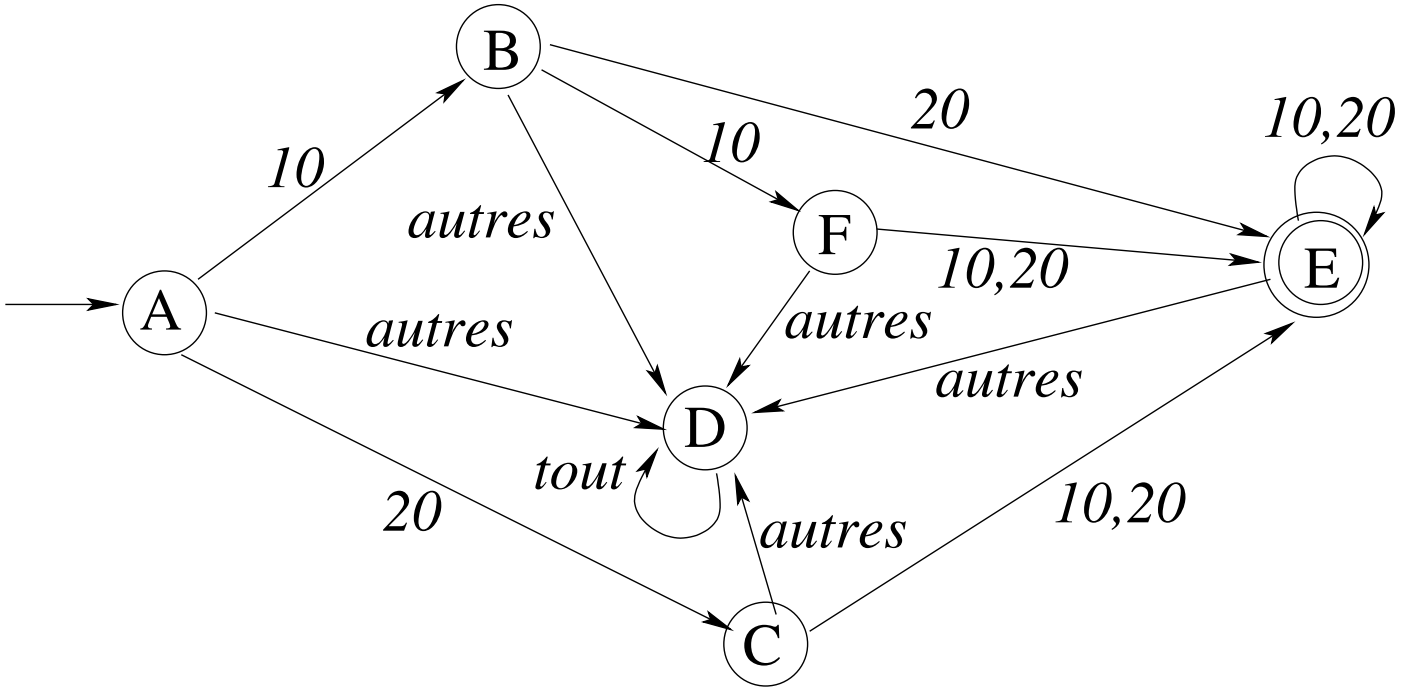
- Modélisation des automates industriels
- Spécification de protocoles
- Recherche de mots dans un texte (compilation, génomique, etc.)
- Preuve de programmes
- Informatique théorique (« modèle de calcul »)

Machine à café

- Une machine à café simple accepte les pièces de 10 centimes et 20 centimes
- Un café coûte 30 centimes
- Quand le montant est suffisant, elle libère un bouton permettant d’obtenir le café
- Elle ne rend pas la monnaie

Machine à café

- Une machine à café simple accepte les pièces de 10 centimes et 20 centimes
- Un café coûte 30 centimes
- Quand le montant est suffisant, elle libère un bouton permettant d'obtenir le café
- Elle ne rend pas la monnaie

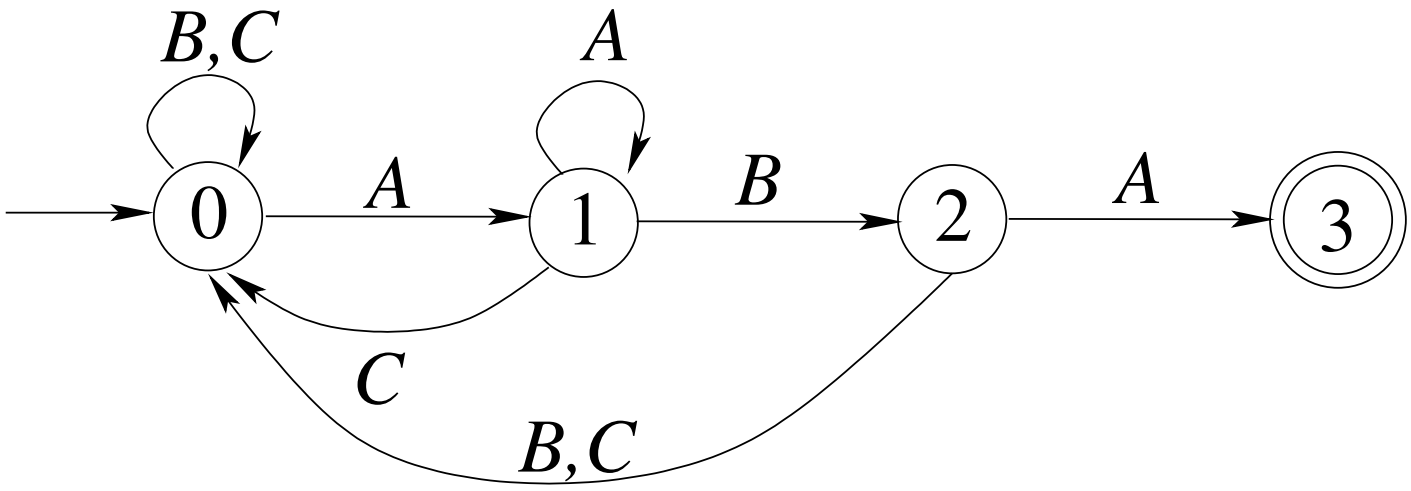


Digicode

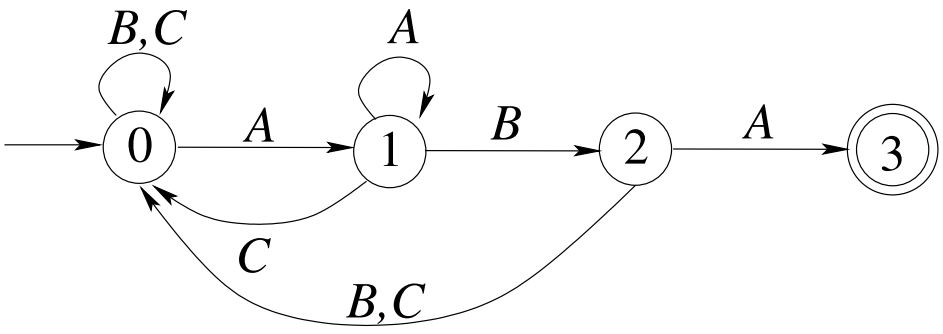
- Un digicode avec trois touches A, B, C ouvre la porte dès qu'on a tapé ABA

Digicode

- Un digicode avec trois touches A, B, C ouvre la porte dés qu'on a tapé ABA



Digicode



- Spécification : porte ouverte en 3 et fermée sinon

- Propriétés structurelles :
  - On vient de taper *A* (vrai en 1 et 3, faux sinon)
  - On vient de taper *B* (toujours vrai en 2, parfois vrai ailleurs)
  - L'état précédent est 2 (vrai en 3 seulement)
  - Etc.

- À vérifier
  - Si la porte s'ouvre, c'est que les 3 dernières lettres tapées sont *ABA*
  - Toute suite de lettres tapées finissant par *ABA* ouvre la porte

Langages de programmation

Contraintes syntaxiques

- Tous les programmes doivent commencer par `begin` et finir par end

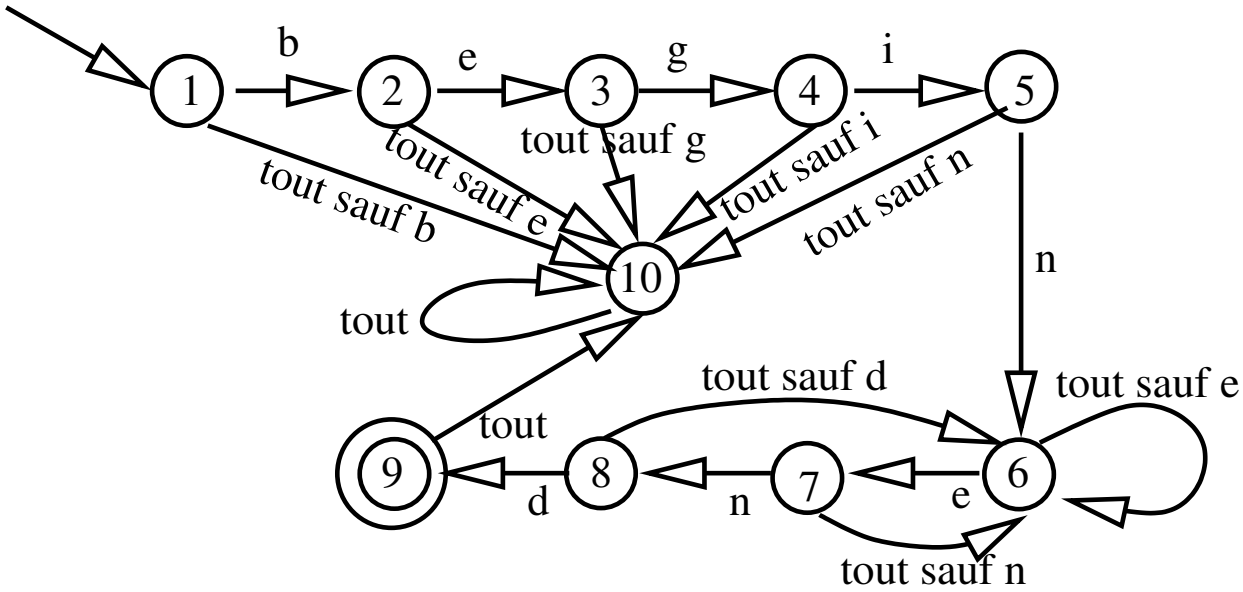
Compilation

Langages de programmation

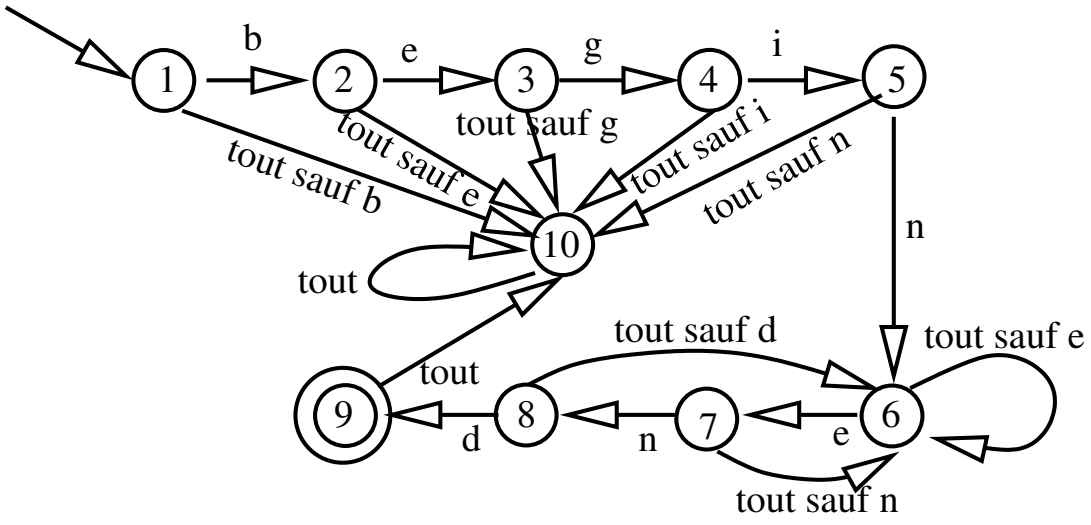
Contraintes syntaxiques

- Tous les programmes doivent commencer par `begin` et finir par end

Compilation

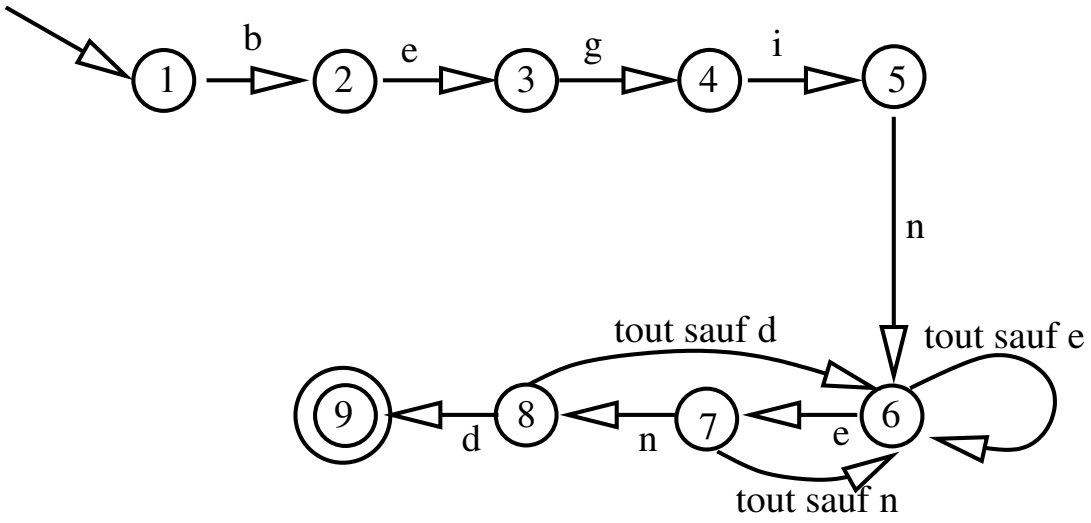


État de rebut



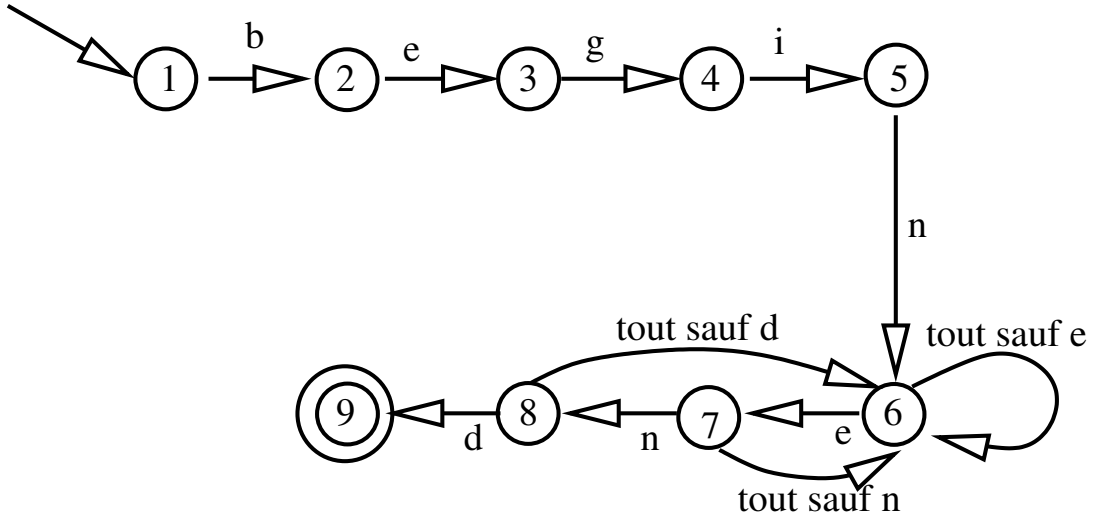
- Rebut pas obligatoire
- Si pas de rebut, le calcul peut se bloquer avant la fin
  - Le mot est alors REFUSÉ
- Si pas de rebut, la table de transition peut contenir des cases vides

État de rebut



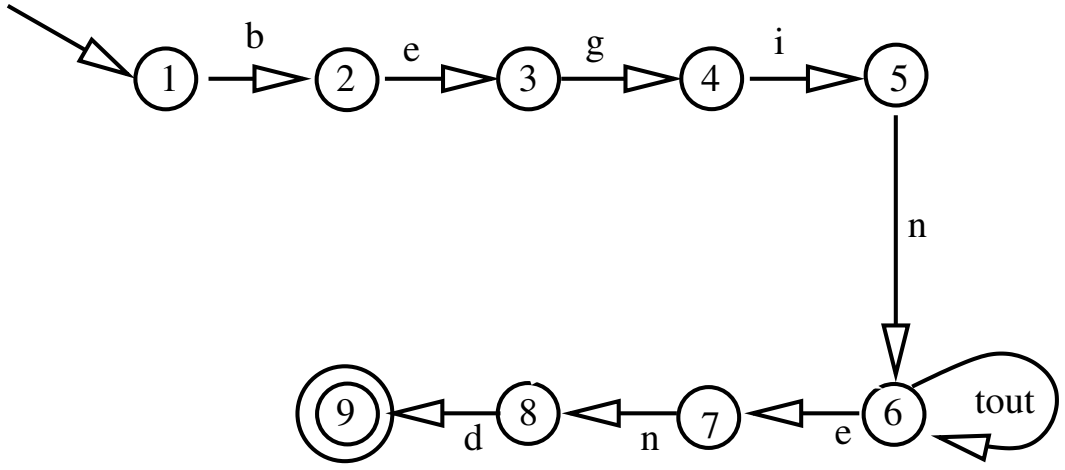
- Rebut pas obligatoire
- Si pas de rebut, le calcul peut se bloquer avant la fin
  - Le mot est alors REFUSÉ
- Si pas de rebut, la table de transition peut contenir des cases vides

Non-déterminisme



- Plusieurs flèches étiquetées par la même lettre partent d'un même état
- Plusieurs calculs possibles pour un même mot
  - Un mot est accepté quand il est accepté par AU MOINS UN CALCUL
- La table de transition peut contenir plusieurs états par case

Non-déterminisme



- Plusieurs flèches étiquetées par la même lettre partent d'un même état
- Plusieurs calculs possibles pour un même mot
  - Un mot est accepté quand il est accepté par AU MOINS UN CALCUL
- La table de transition peut contenir plusieurs états par case

Langages de progammation

Contraintes syntaxiques

- Tous les programmes doivent commencer par `begin` et finir par end

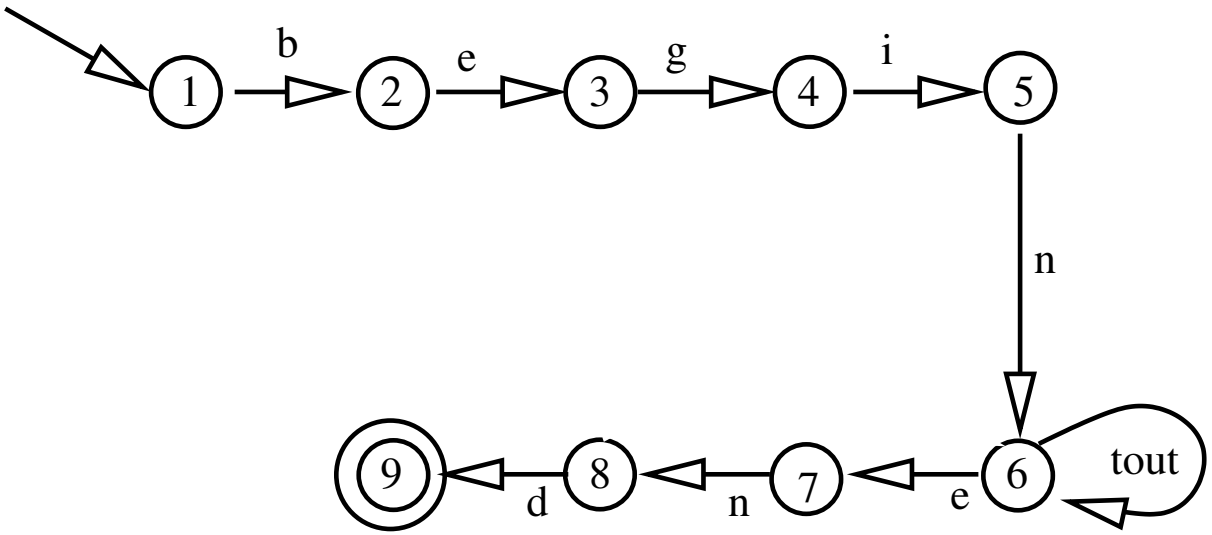
Compilation

Langages de progammation

Contraintes syntaxiques

- Tous les programmes doivent commencer par `begin` et finir par end

Compilation



Langages de progammation

Contraintes syntaxiques

- Alphabet
  $\Sigma := \{r, w, i, n, a, A, B, C, \cdots, X, Y, Z, <, =, :, ;, \sqcup, \hookleftarrow\}$.

Compilation

r	$\rightarrow$	read	$<=$	$\rightarrow$	less or equal
w	$\rightarrow$	write	$:=$	$\rightarrow$	variable assignment
i	$\rightarrow$	if	$;$	$\rightarrow$	sequentiation
n	$\rightarrow$	not	$\sqcup$	$\rightarrow$	blank space
a	$\rightarrow$	and	$\hookleftarrow$	$\rightarrow$	carriage return

Langages de progammation

Contraintes syntaxiques

- Alphabet
  $\Sigma := \{r, w, i, n, a, A, B, C, \cdots, X, Y, Z, <, =, :, ;, \sqcup, \hookleftarrow\}$.

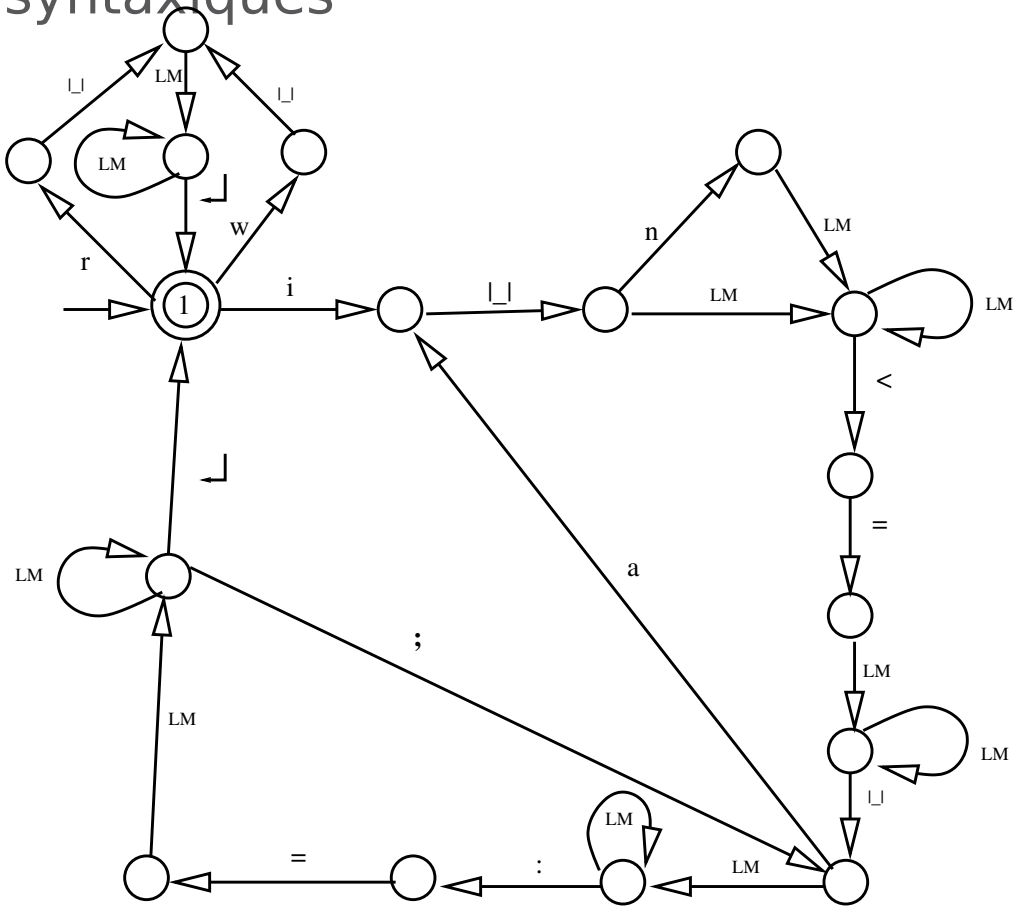
Compilation

Exemple de programme à compiler : $r \sqcup A \hookleftarrow r \sqcup B \hookleftarrow i \sqcup A <= B \hookleftarrow$
$C := A; D := B \hookleftarrow i \sqcup n \sqcup A <= B \hookleftarrow C := B; D := A \hookleftarrow$

Langages de progammation

Contraintes syntaxiques

Compilation



Langages de progammation

Contraintes syntaxiques

- Nombreuses
- Parfois compliquées

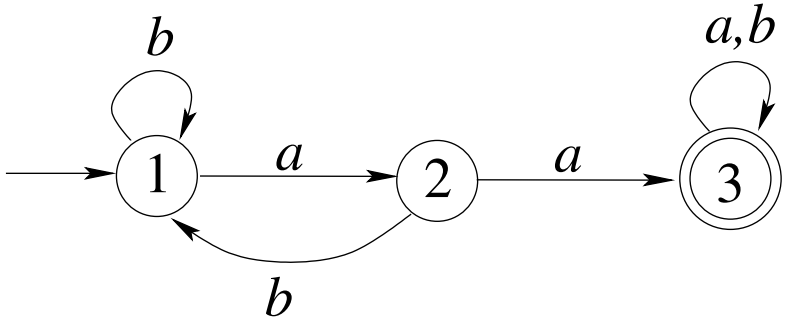
Compilation

- Analyse syntaxique à l'aide d'automates finis
- Mais pas seulement (parenthésage)

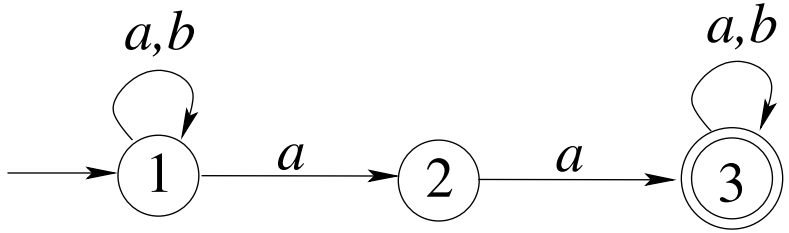
Non-déterminisme

Définition

Un automate est déterministe si pour tout état et toute lettre de l'alphabet, il y a au plus une transition étiquetée par cette lettre qui part de cet état.
Sinon, c'est un automate non-déterministe.

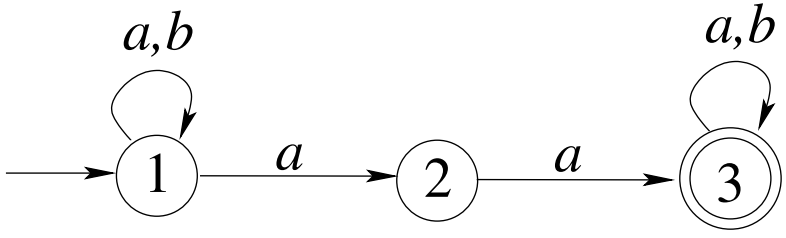


déterministe



non-déterministe

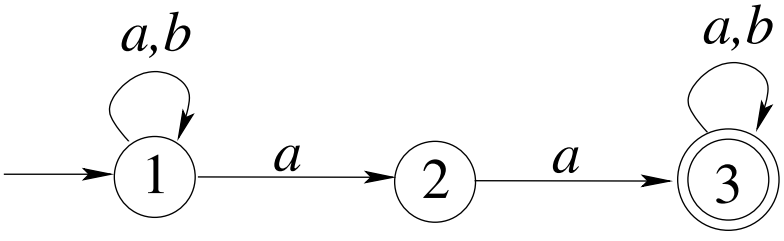
Table de transition d'un automate non-déterministe



	a	b
1	1, 2	1
2	3	
3	3	3

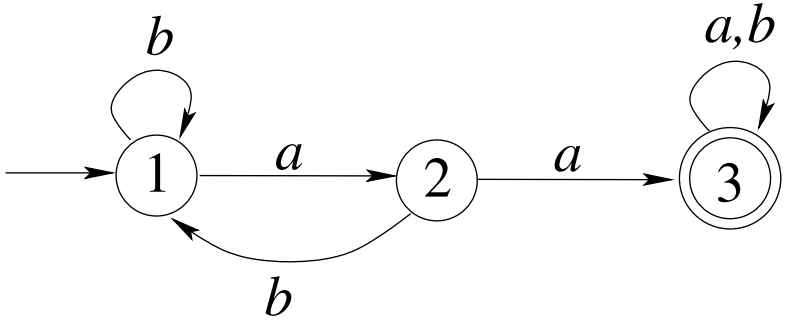
- À partir de l'état 1, en lisant la lettre a, on peut atteindre l'état 1 OU l'état 2

Exemple



Cet automate non-déterministe reconnaît le langage L = {mots contenant deux a consécutifs}

Cet automate déterministe reconnaît le même langage L



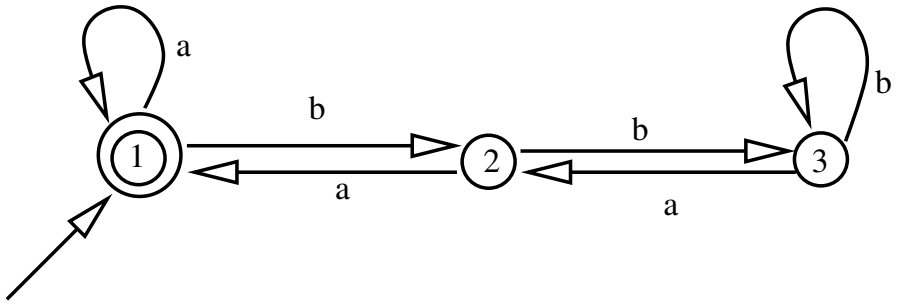
Une question

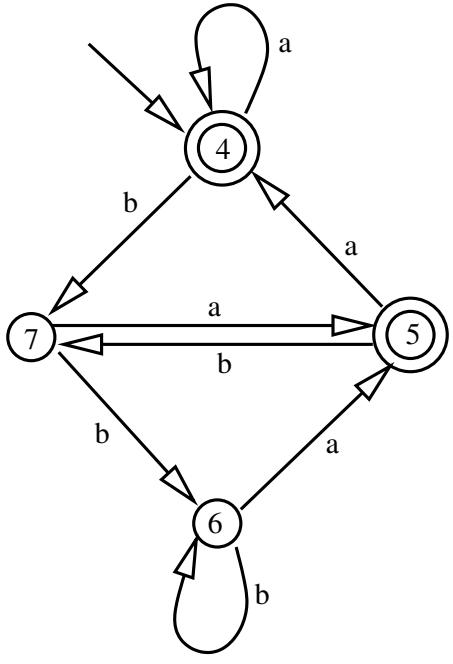
Est-ce que le non déterminisme est plus puissant ? Peut-on rendre un automate déterministe ?

Exemple

Question

Les automates $\mathcal{A}$ et $\mathcal{A}'$ acceptent-ils le même langage ?

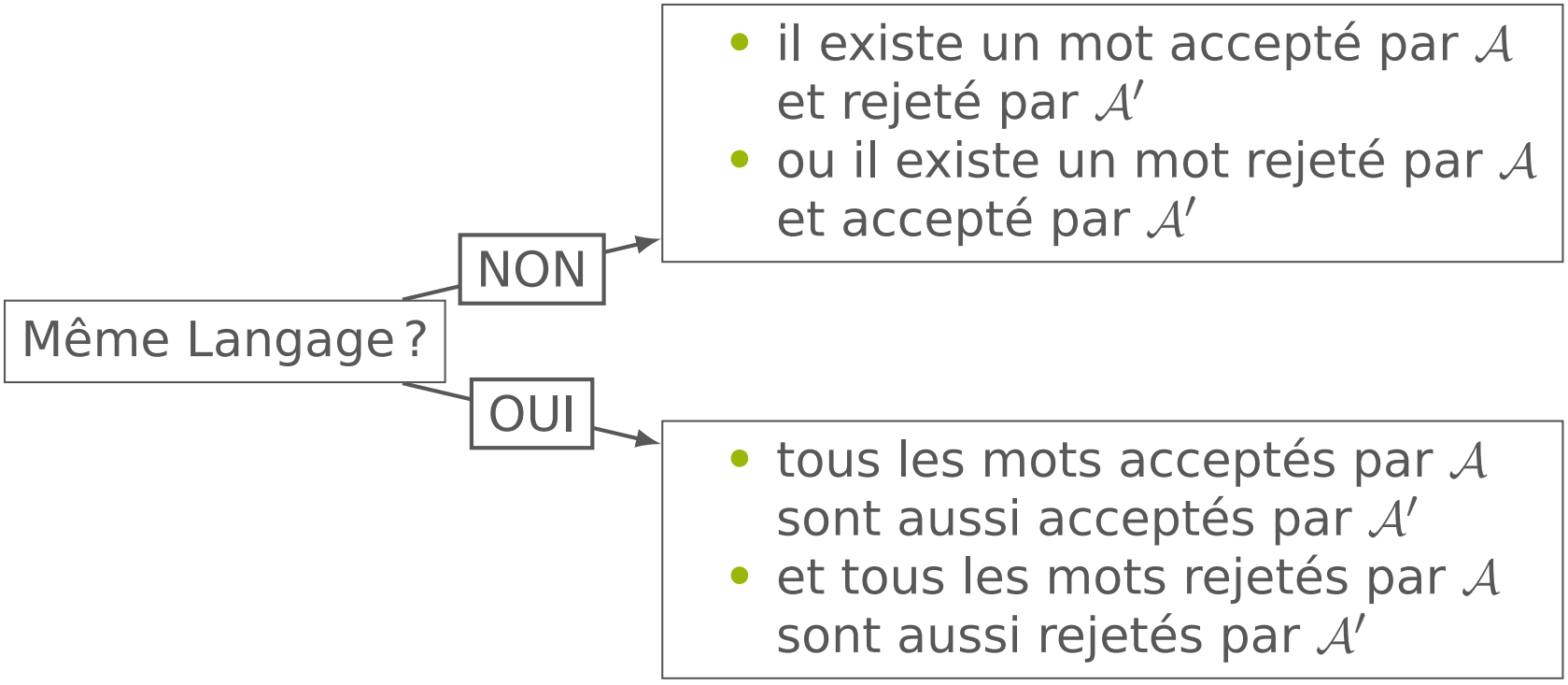




Exemple

Question

Les automates $\mathcal{A}$ et $\mathcal{A}'$ acceptent-ils le même langage ?



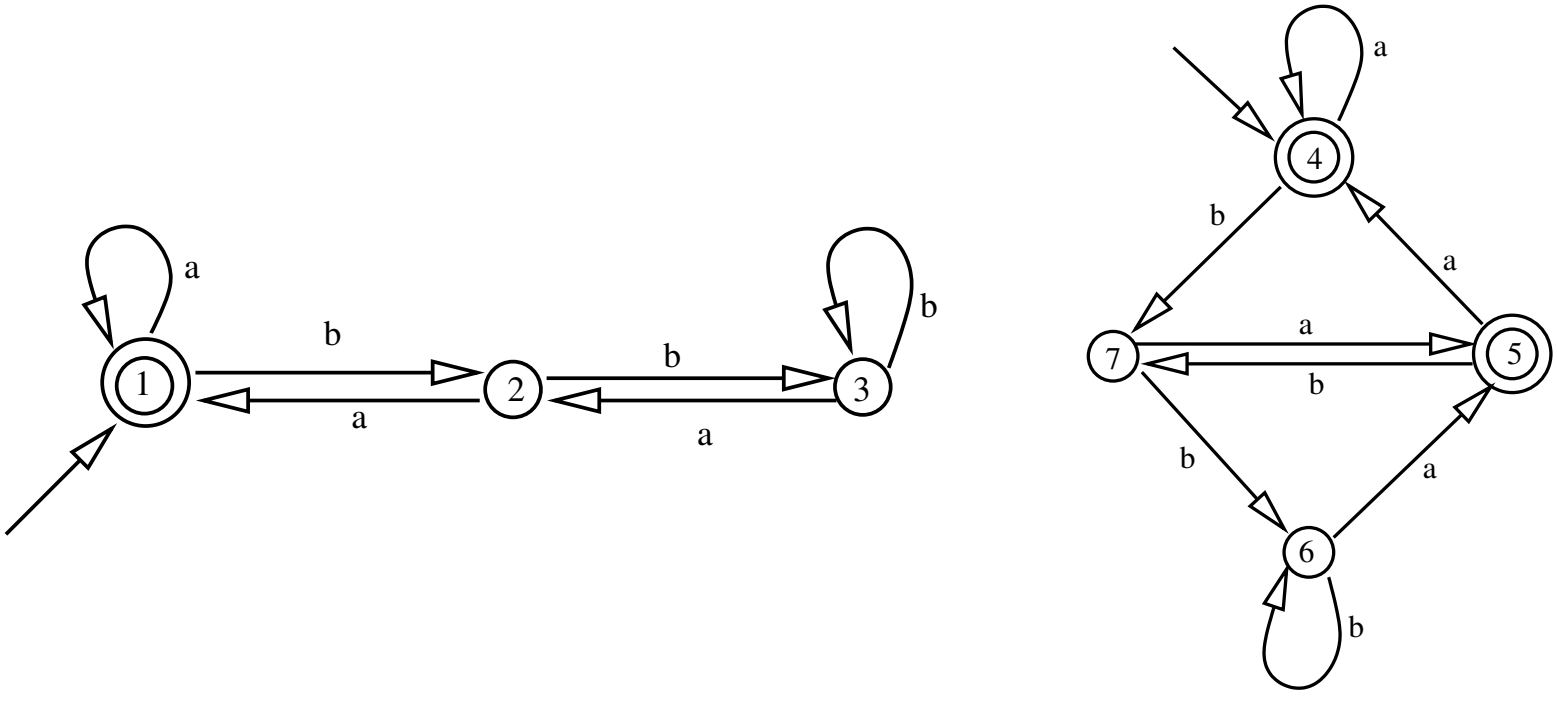
Idée générale

Définition

Deux automates qui acceptent le même langage s'appellent des automates équivalents

- Pour prouver que $\mathcal{A}$ et $\mathcal{A}'$ ne sont pas équivalents, il suffit de trouver un mot accepté par l'un et rejeté par l'autre
- Pour prouver que $\mathcal{A}$ et $\mathcal{A}'$ sont équivalents, on doit montrer qu'ils acceptent exactement les mêmes mots (ce qui paraît plus difficile)

Exemple (2)



Le mot *bba* est rejeté par $\mathcal{A}$ et accepté par $\mathcal{A}'$

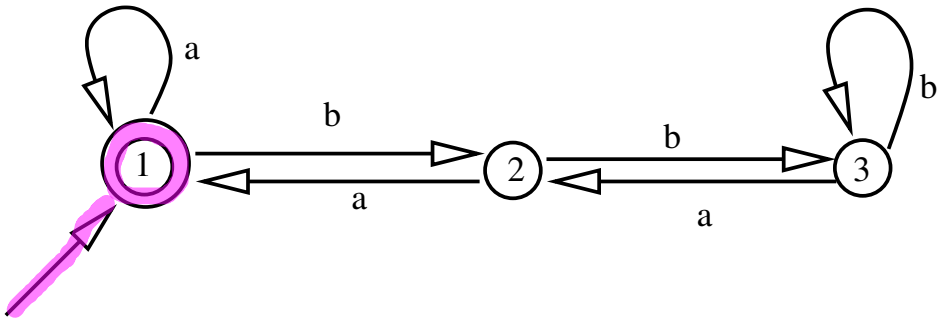
Conclusion : $\mathcal{A}$ et $\mathcal{A}'$ ne sont pas équivalents

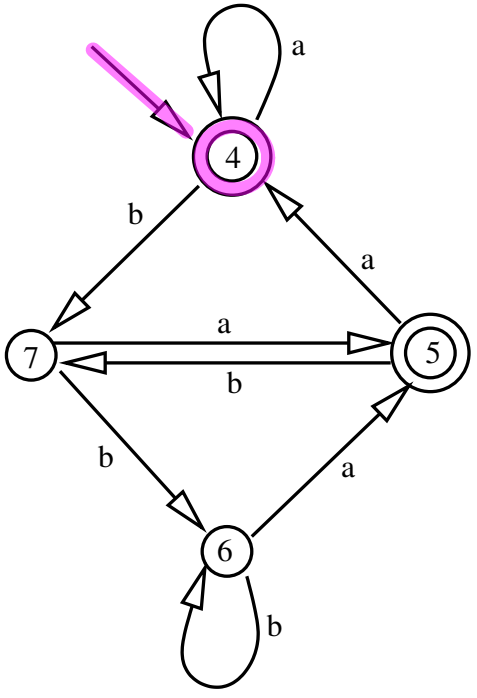
Méthode pour les automates déterministes

On suit des chemins « parallèles »dans les deux automates déterministes $\mathcal{A}$ et $\mathcal{A}'$ à partir de leur état initial respectif.

- Si on rencontre un couple d'états (q, q') avec q acceptant et q' non acceptant (ou le contraire), alors on a trouvé un mot w accepté par $\mathcal{A}$ et rejeté par $\mathcal{A}'$ (ou le contraire). Donc $\mathcal{A}$ et $\mathcal{A}'$ ne sont pas équivalents.
- Sinon, on répond qu'ils sont équivalents.

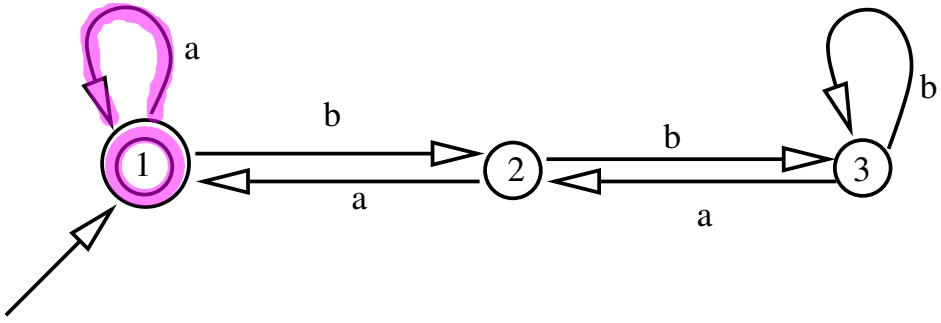
Exemple (3)

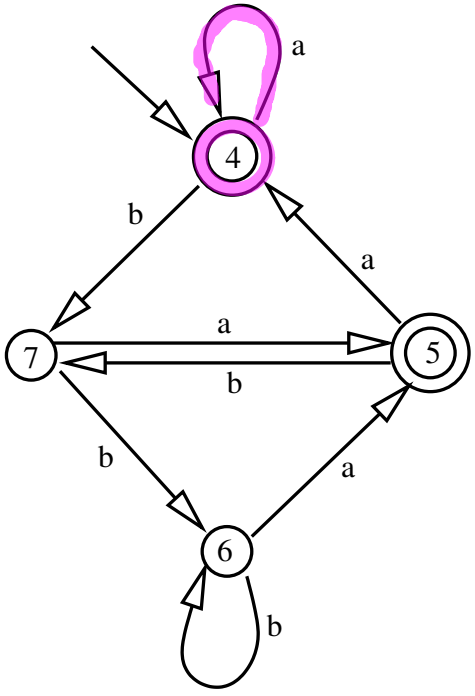




	a	b
$(\underline{1}, \underline{4})$		

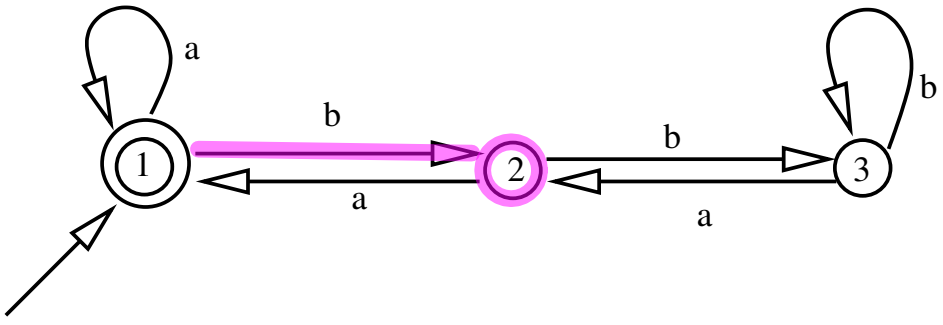
Exemple (3)

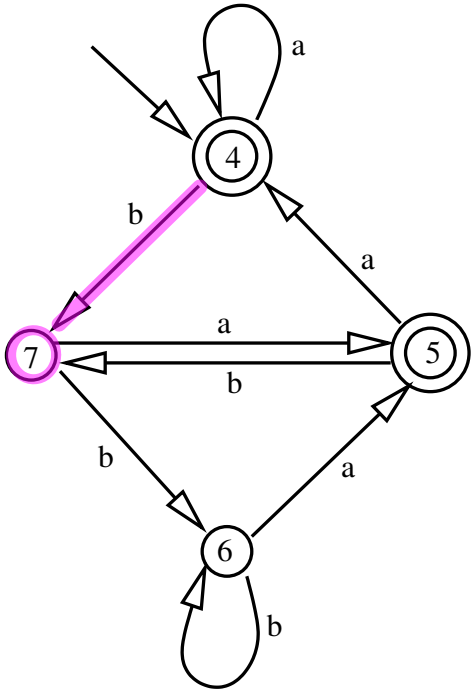




	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	

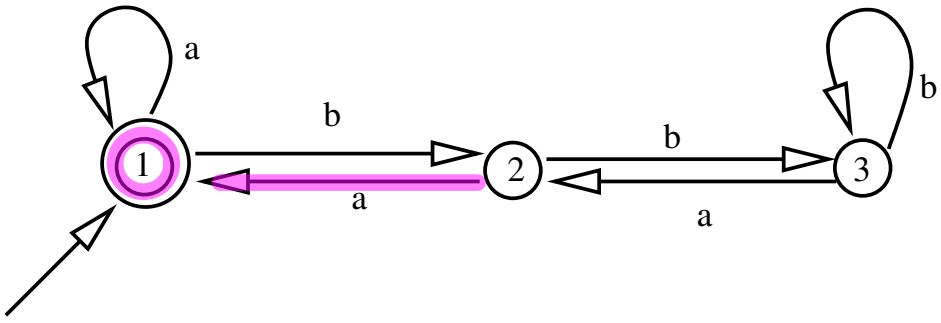
Exemple (3)

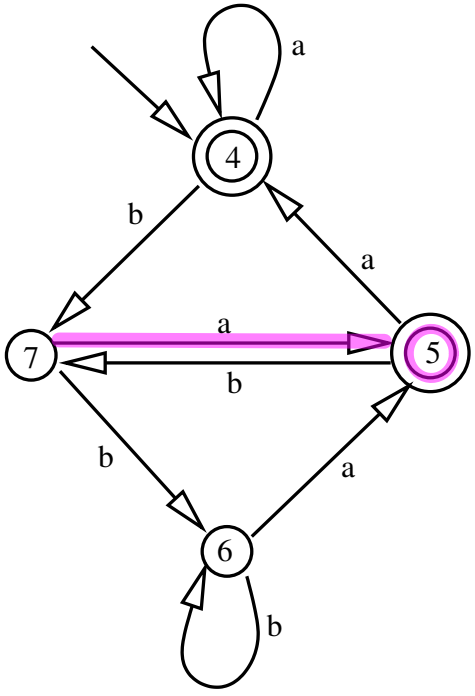




	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	$(2,7)$

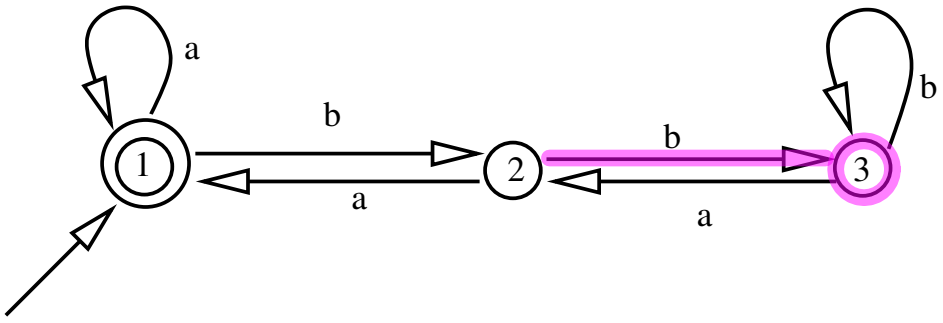
Exemple (3)

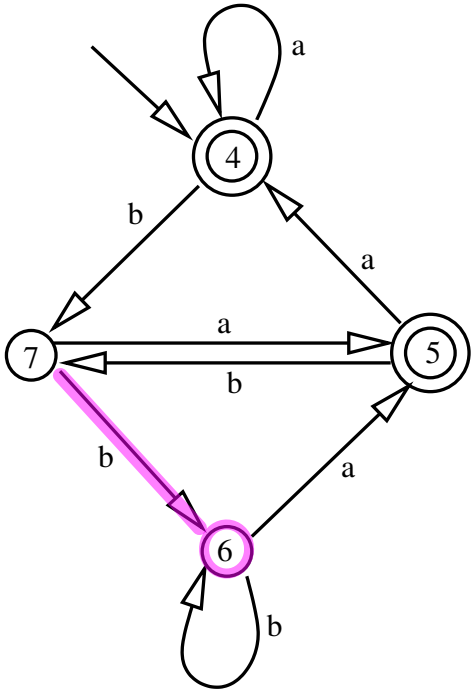




	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	$(2,7)$
$(2,7)$	$(\underline{1},\underline{5})$	

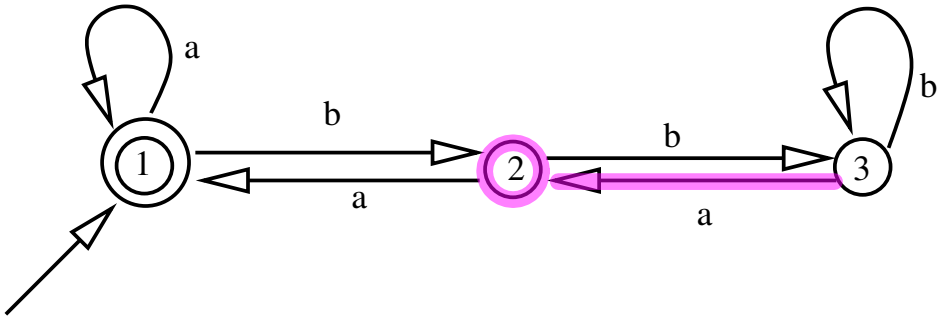
Exemple (3)

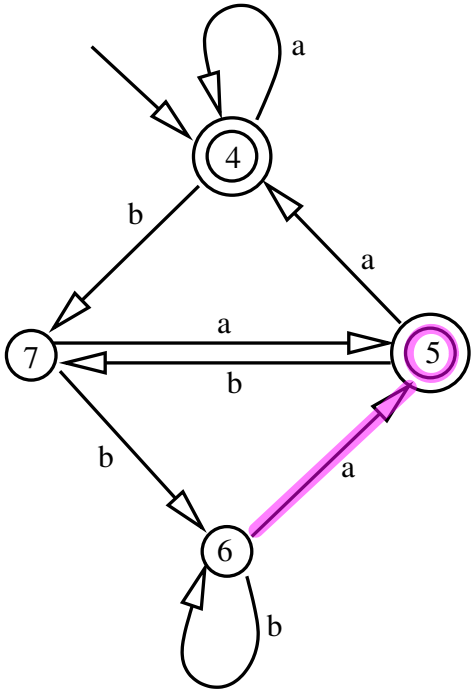




	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	$(2,7)$
$(2,7)$	$(\underline{1},\underline{5})$	$(3,6)$

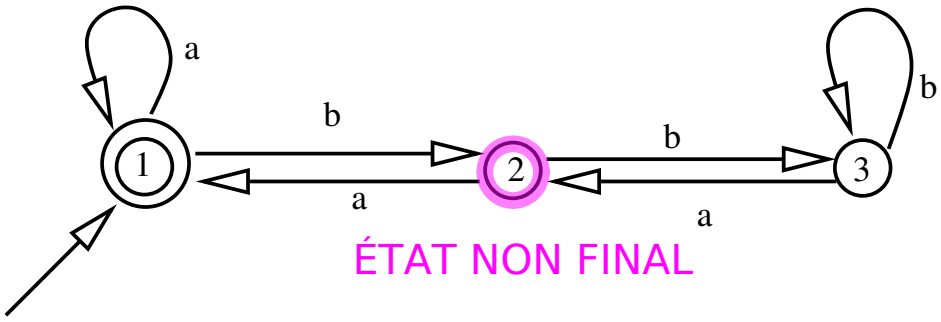
Exemple (3)

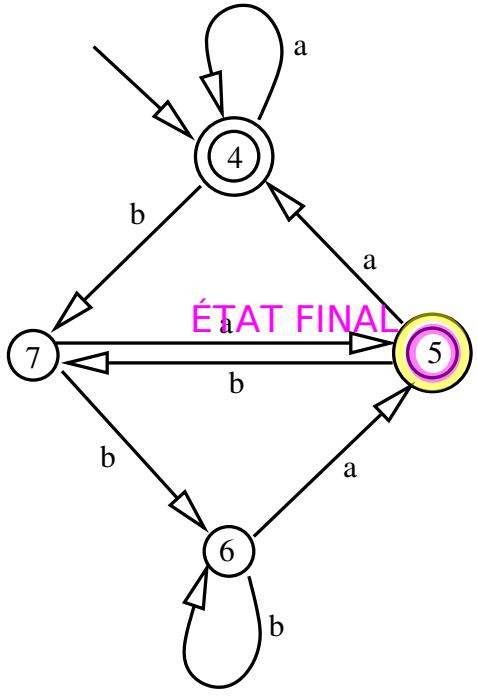




	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	$(2,7)$
$(2,7)$	$(\underline{1},\underline{5})$	$(3,6)$
$(\underline{1},\underline{5})$	$(\underline{1},\underline{4})$	$(2,7)$
$(3,6)$	$(2,\underline{5})$	...

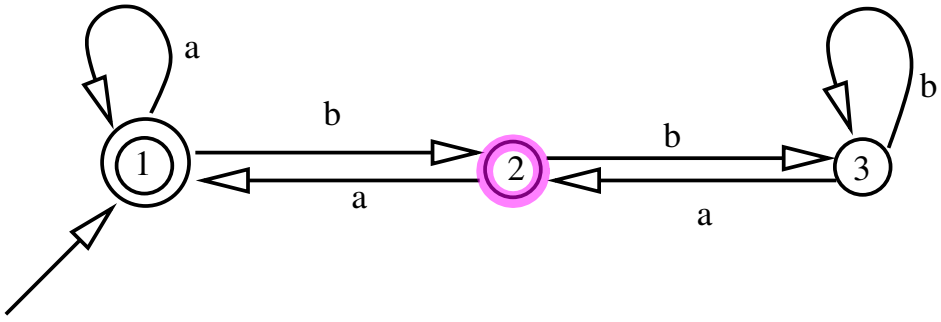
Exemple (3)

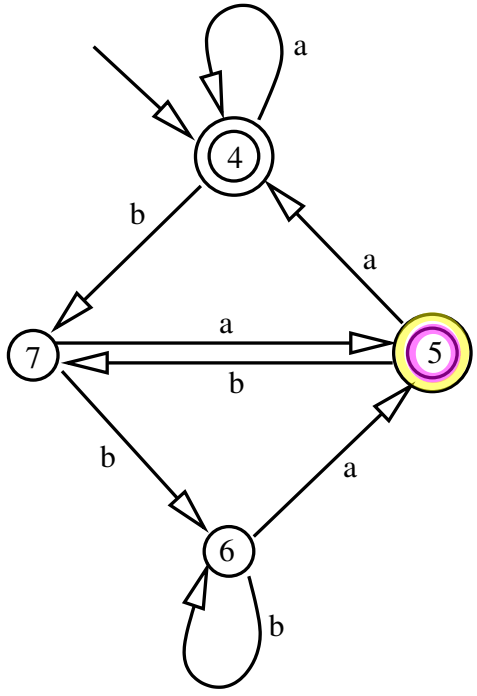




	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	$(2,7)$
$(2,7)$	$(\underline{1},\underline{5})$	$(3,6)$
$(\underline{1},\underline{5})$	$(\underline{1},\underline{4})$	$(2,7)$
$(3,6)$	$(2,\underline{5})$	...

Exemple (3)

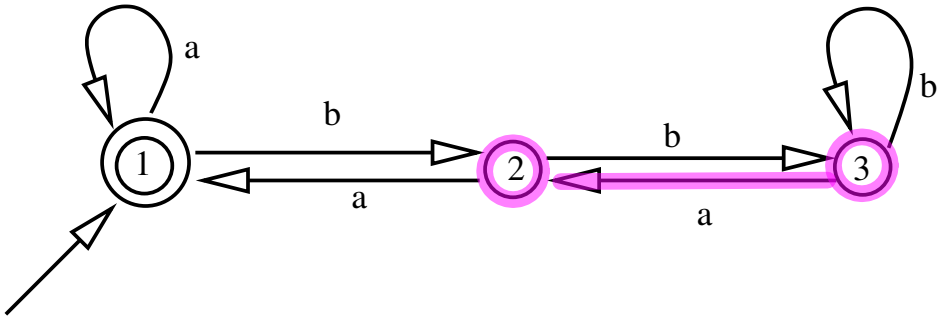


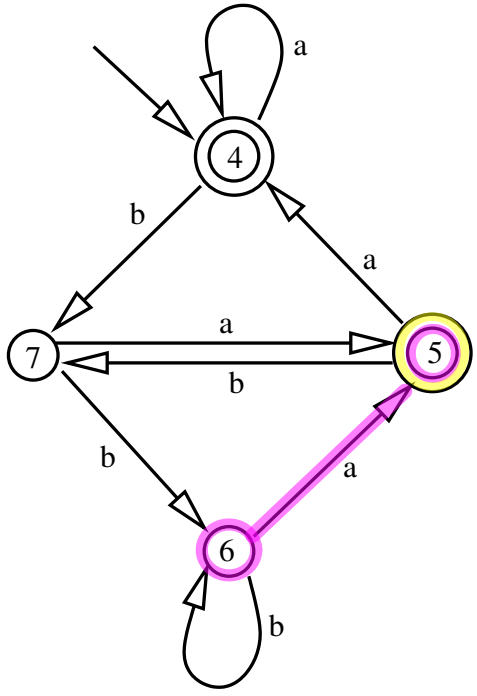


	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	$(2,7)$
$(2,7)$	$(\underline{1},\underline{5})$	$(3,6)$
$(\underline{1},\underline{5})$	$(\underline{1},\underline{4})$	$(2,7)$
$(3,6)$	$(2,\underline{5})$	…

En remontant le calcul, on s'aperçoit que le mot bba est rejeté par $\mathcal{A}$ et accepté par $\mathcal{A}'$

Exemple (3)

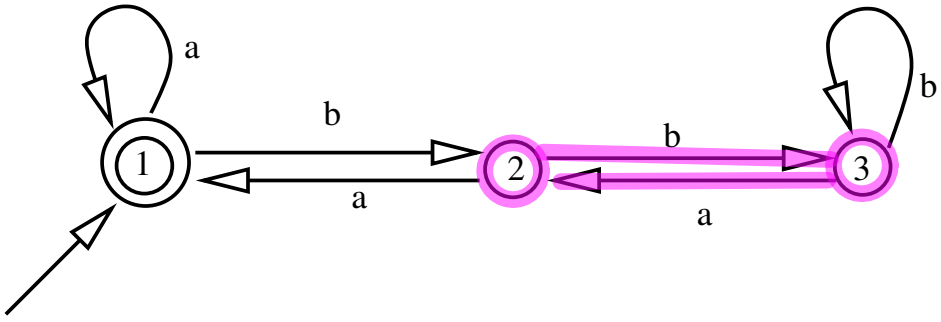


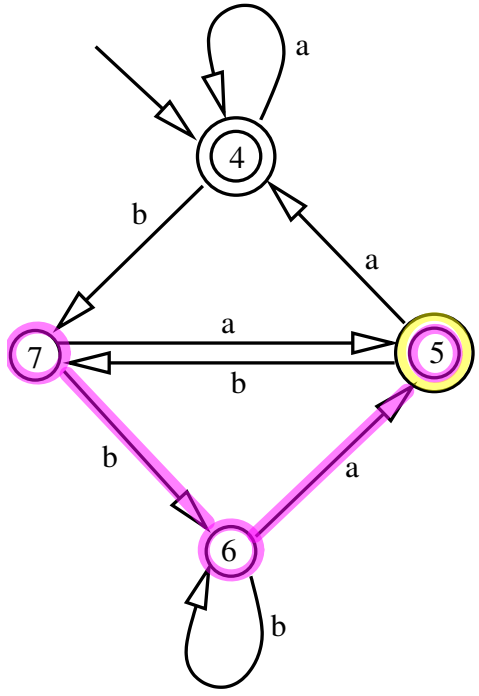


	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	$(2,7)$
$(2,7)$	$(\underline{1},\underline{5})$	$(3,6)$
$(\underline{1},\underline{5})$	$(\underline{1},\underline{4})$	$(2,7)$
$(3,6)$	$(2,\underline{5})$	…

En remontant le calcul, on s'aperçoit que le mot bb**a** est rejeté par $\mathcal{A}$ et accepté par $\mathcal{A}'$

Exemple (3)

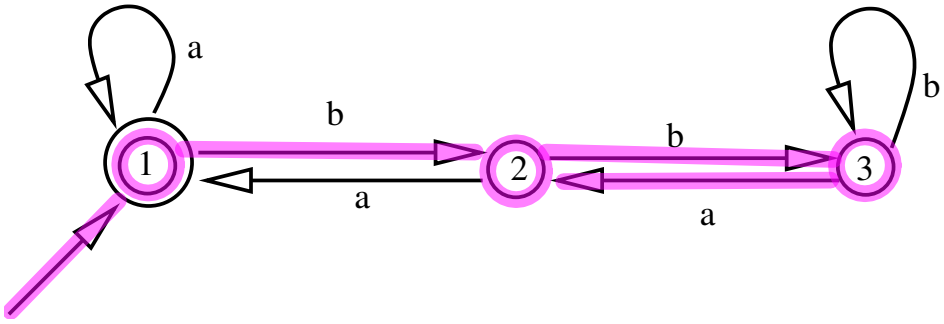


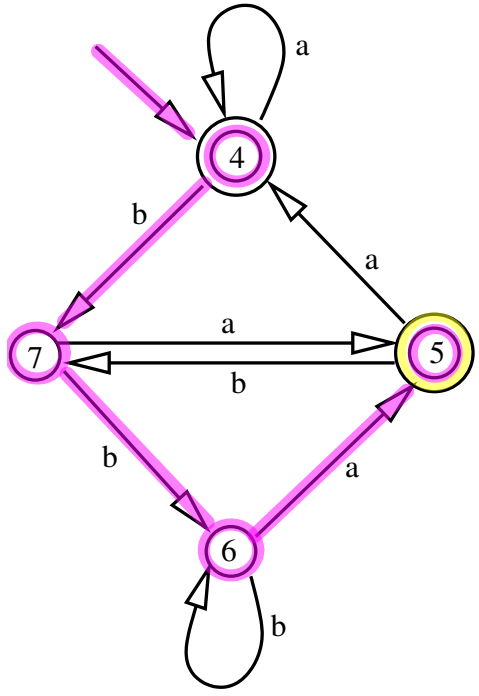


	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	$(2,7)$
$(2,7)$	$(\underline{1},\underline{5})$	$(3,6)$
$(\underline{1},\underline{5})$	$(\underline{1},\underline{4})$	$(2,7)$
$(3,6)$	$(2,\underline{5})$	…

En remontant le calcul, on s'aperçoit que le mot b**b**a est rejeté par $\mathcal{A}$ et accepté par $\mathcal{A}'$

Exemple (3)

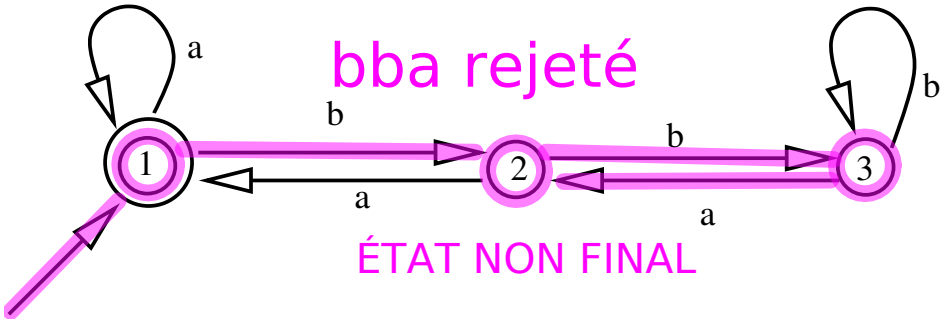


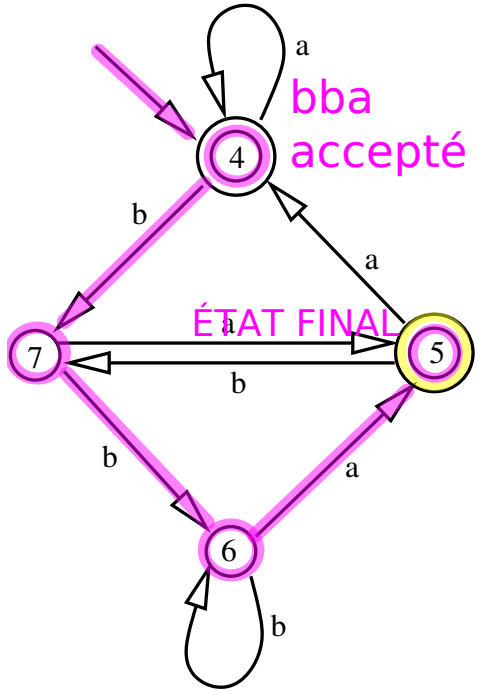


	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	$(2,7)$
$(2,7)$	$(\underline{1},\underline{5})$	$(3,6)$
$(\underline{1},\underline{5})$	$(\underline{1},\underline{4})$	$(2,7)$
$(3,6)$	$(2,\underline{5})$	…

En remontant le calcul, on s'aperçoit que le mot **b**ba est rejeté par $\mathcal{A}$ et accepté par $\mathcal{A}'$

Exemple (3)





	a	b
$(\underline{1},\underline{4})$	$(\underline{1},\underline{4})$	$(2,7)$
$(2,7)$	$(\underline{1},\underline{5})$	$(3,6)$
$(\underline{1},\underline{5})$	$(\underline{1},\underline{4})$	$(2,7)$
$(3,6)$	$(2,\underline{5})$	...

En remontant le calcul, on s'aperçoit que le mot bba est rejeté par $\mathcal{A}$ et accepté par $\mathcal{A}'$

Pourquoi cette méthode fonctionne ?

1. La méthode termine
2. La méthode donne bien la bonne réponse
   - quand la réponse est NON.
   - quand la réponse est OUI.

car il n'y a qu'un nombre fini de couples d'états (q, q'), donc le calcul se termine au bout d'un temps fini.

Pourquoi cette méthode fonctionne ?

1. La méthode termine
2. La méthode donne bien la bonne réponse
   - quand la réponse est NON.
   - quand la réponse est OUI.

Pourquoi cette méthode fonctionne ?

1. La méthode termine
2. La méthode donne bien la bonne réponse
   - quand la réponse est NON.
   - quand la réponse est OUI.

Si la méthode trouve un mot w accepté par $\mathcal{A}$ et rejeté par $\mathcal{A}'$ (ou le contraire), alors les deux automates $\mathcal{A}$ et $\mathcal{A}'$ sont bien non-équivalents.

Pourquoi cette méthode fonctionne ?

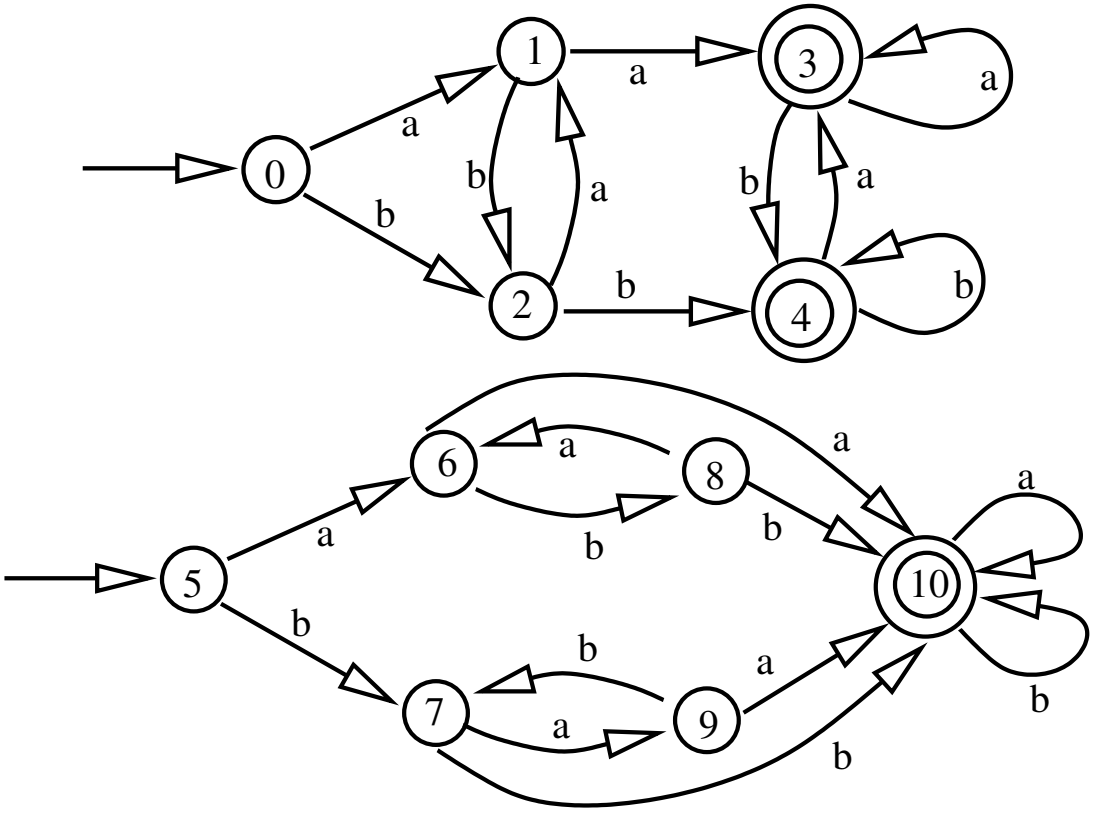
1. La méthode termine
2. La méthode donne bien la bonne réponse
   - quand la réponse est NON.
   - quand la réponse est OUI.

Sinon, tout chemin d'exécution dans l'automate $\mathcal{A}$ montrant qu'un mot w est accepté/rejeté par $\mathcal{A}$ correspond à un chemin d'exécution pour l'automate $\mathcal{A}'$ donnant le même résultat (on trouve ce chemin dans la table de transition de l'automate « bi-processeur »que nous avons construit)

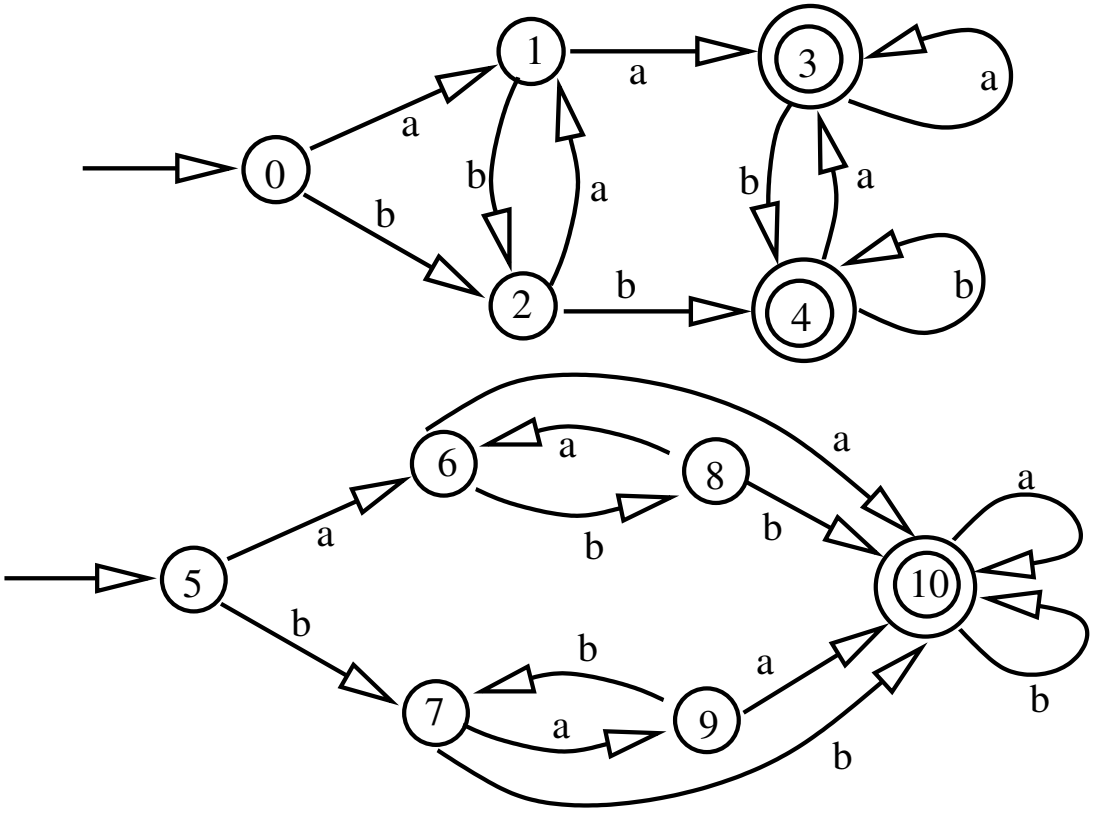
Pourquoi cette méthode fonctionne ?

1. La méthode termine
2. La méthode donne bien la bonne réponse
   - quand la réponse est NON.
   - quand la réponse est OUI.

Autre exemple

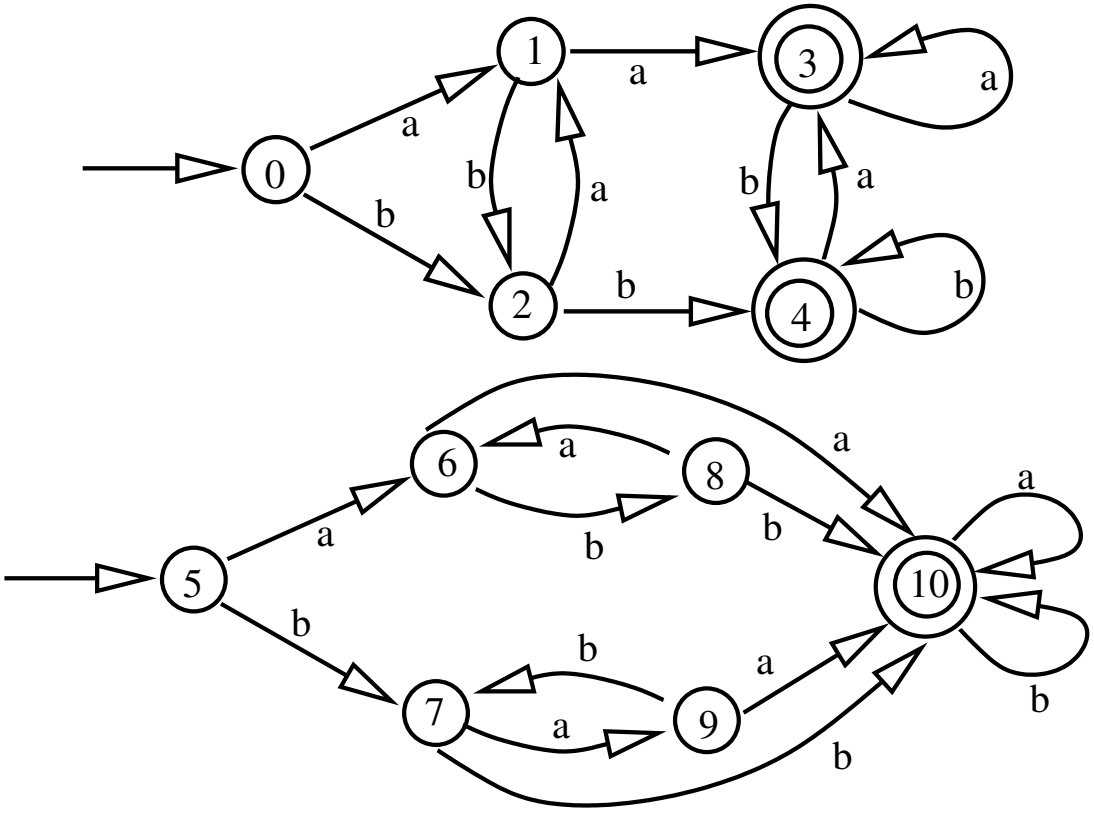


Autre exemple



	a	b
$(0, 5)$	$(1, 6)$	$(2, 7)$
$(1, 6)$	$(\underline{3}, \underline{10})$	$(2, 8)$
$(2, 7)$	$(1, 9)$	$((\underline{4}, \underline{10}))$
$(\underline{3}, \underline{10})$	$(\underline{3}, \underline{10})$	$(\underline{4}, \underline{10})$
$(2, 8)$	$(1, 6)$	$(\underline{4}, \underline{10})$
$(1, 9)$	$(\underline{3}, \underline{10})$	$(2, 7)$
$(\underline{4}, \underline{10})$	$(\underline{3}, \underline{10})$	$(\underline{4}, \underline{10})$

Autre exemple



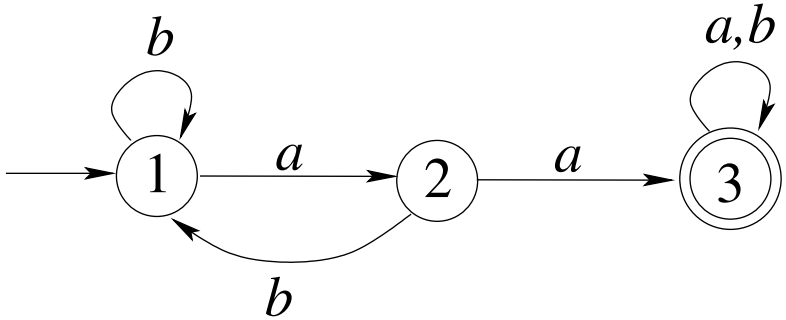
	a	b
$(0, 5)$	$(1, 6)$	$(2, 7)$
$(1, 6)$	$(\underline{3}, \underline{10})$	$(2, 8)$
$(2, 7)$	$(1, 9)$	$((\underline{4}, \underline{10}))$
$(\underline{3}, \underline{10})$	$(\underline{3}, \underline{10})$	$(\underline{4}, \underline{10})$
$(2, 8)$	$(1, 6)$	$(\underline{4}, \underline{10})$
$(1, 9)$	$(\underline{3}, \underline{10})$	$(2, 7)$
$(\underline{4}, \underline{10})$	$(\underline{3}, \underline{10})$	$(\underline{4}, \underline{10})$

Le calcul se termine sans trouver de mot qui sépare les deux automates, donc ils sont équivalents.

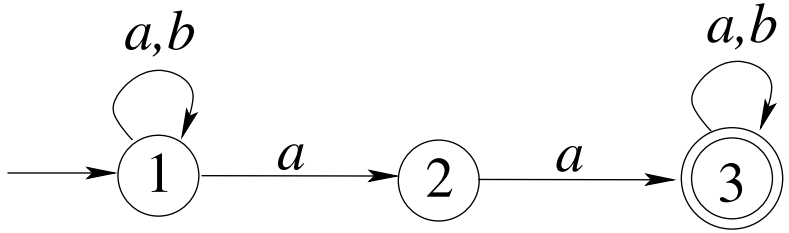
Non-déterminisme

Définition

Un automate est déterministe si pour tout état et toute lettre de l'alphabet, il y a au plus une transition étiquetée par cette lettre qui part de cet état.
Sinon, c'est un automate non-déterministe.

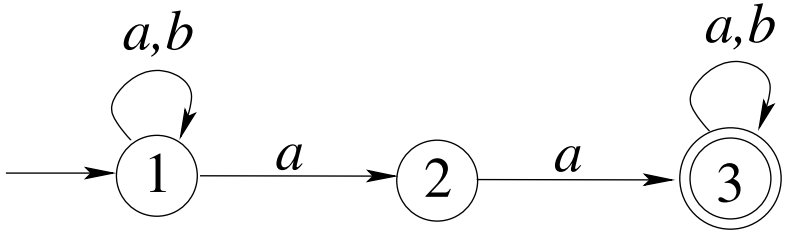


déterministe



non-déterministe

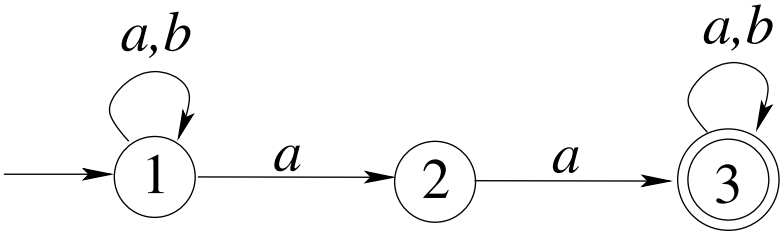
Table de transition d'un automate non-déterministe



	a	b
1	1, 2	1
2	3	
3	3	3

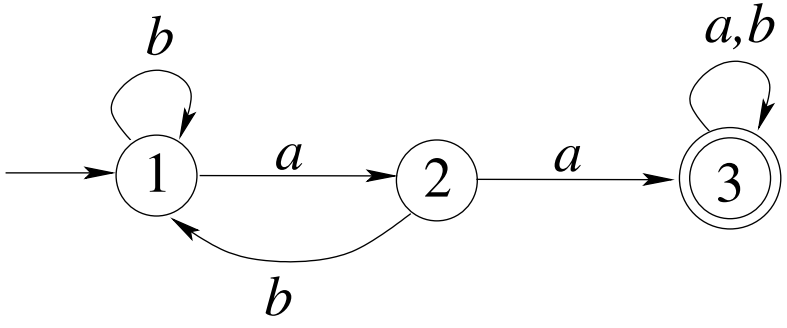
- À partir de l'état 1, en lisant la lettre a, on peut atteindre l'état 1 OU l'état 2

Exemple



Cet automate non-déterministe reconnaît le langage L = {mots contenant deux a consécutifs}

Cet automate déterministe reconnaît le même langage L



Déterminisation

Question

Existe-t-il toujours un automate déterministe qui accepte le même langage qu'un automate non-déterministe donné ?

Réponse

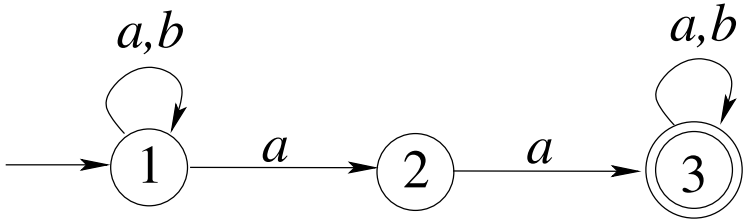
À partir d'un automate non-déterministe qui reconnaît un langage L, on peut toujours calculer un automate déterministe qui reconnaît le même langage L.

Idée générale

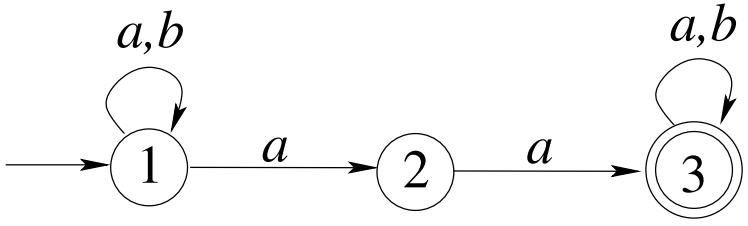
L'automate déterministe que l'on va construire sera composé de *super-états* qui vont *simuler* des ensembles d'états de l'automate initial. La méthode utilisée est la suivante :

- le *super-état* initial est formé de l'état initial ;
- le *super-état* S' issu d'un *super-état* S en appliquant une transition t est formé de l'ensemble des états obtenus en appliquant la transition t à partir de chacun des états formant S ;
- le ou les *super-états* acceptants sont les *super-états* contenant au moins un état final.

Mise en œuvre

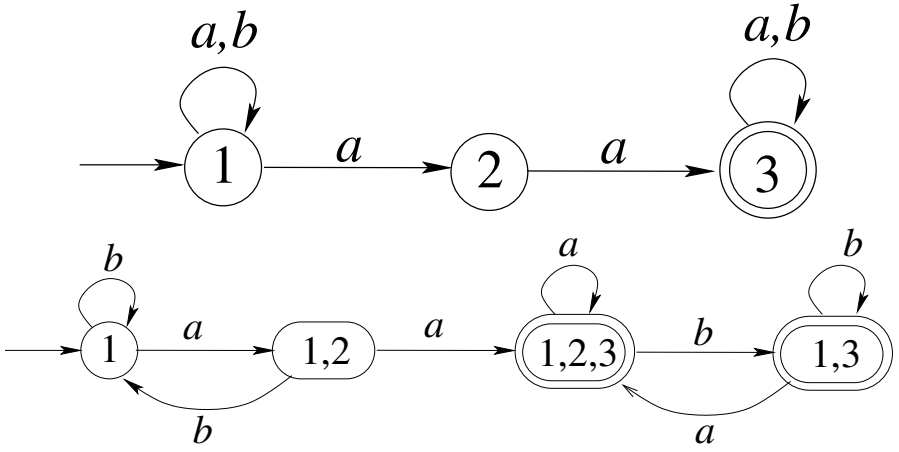


Mise en œuvre



super-états	*a*	*b*
→{1}	{1,2}	{1}
{1,2}	{1,2,3}	{1}
{1,2,3}	{1,2,3}	{1,3}
{1,3}	{1,2,3}	{1,3}

Mise en œuvre



super-états	*a*	*b*
→{1}	{1,2}	{1}
{1,2}	{1,2,3}	{1}
{1,2,3}	{1,2,3}	{1,3}
{1,3}	{1,2,3}	{1,3}

Remarques

- Les automates non-déterministes sont aussi naturels et acceptables que les automates déterministes
- Mais il arrive qu'on soit obligé de déterminiser un automate (par exemple avant d'utiliser la méthode « automates équivalents »), c'est pourquoi vous devez savoir le faire
- Les états de l'automate déterministe obtenu correspondent à des ensembles d'états de l'automate de départ, il peut donc arriver qu'ils soient très nombreux (jusqu'à 2^n, si n est le nombre d'états de l'automate non-déterministe)

Au delà de ce cours

Nous avons entrevu le modèles des automates.
Nous allons évoquer brièvement de manière informelle comment montrer avec le lemme de la pompe que certains langages ne sont pas réguliers.
Il existe une hiérarchie dite de Chomsky pour laquelle on augmente le modèle des automates pour le rendre strictement plus expressif (par exemple les automates à piles).