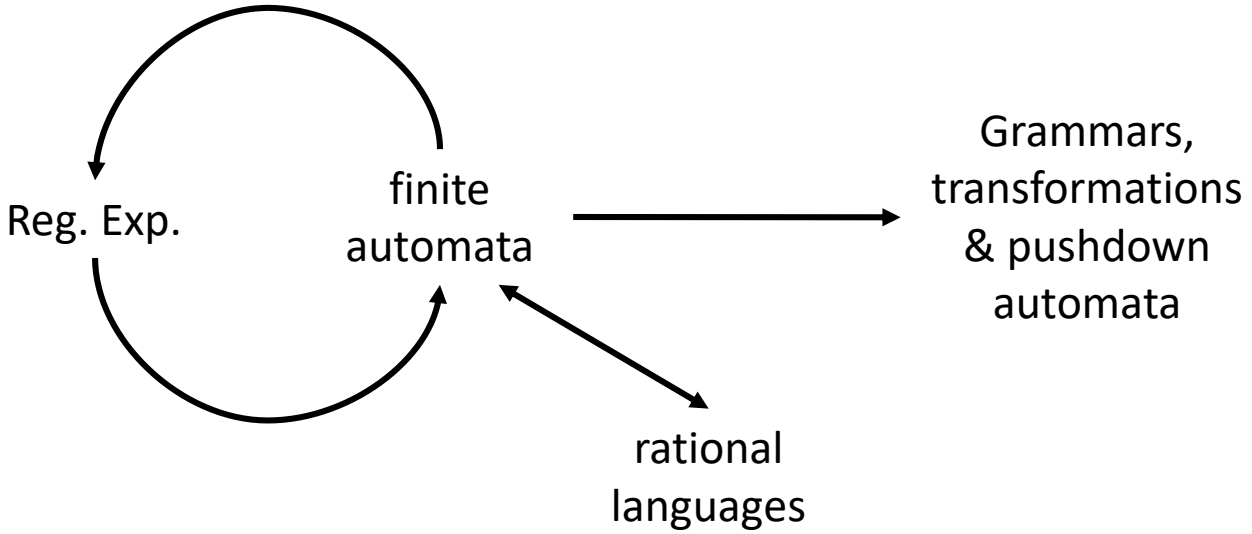
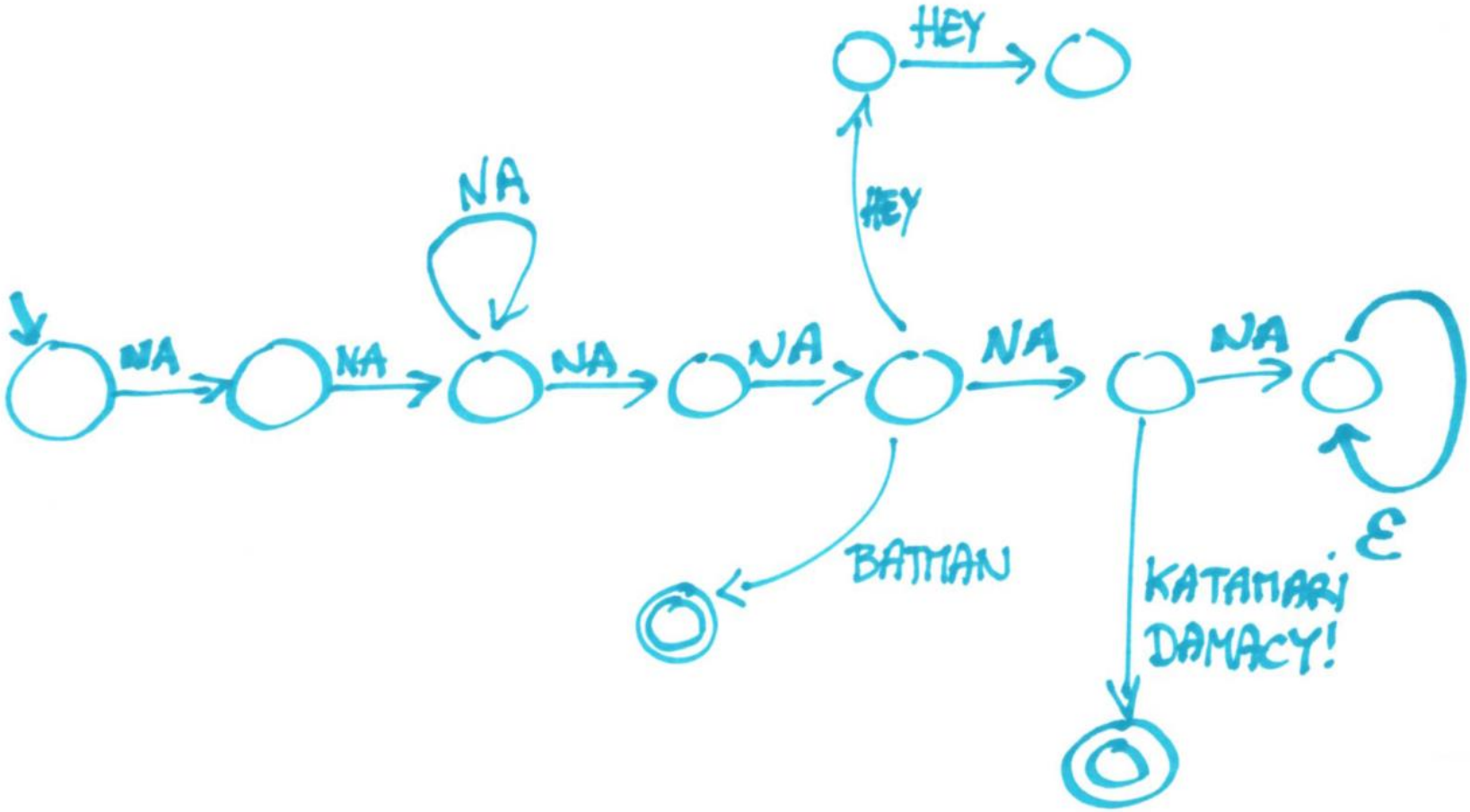
Automates et Langages



Automates

101

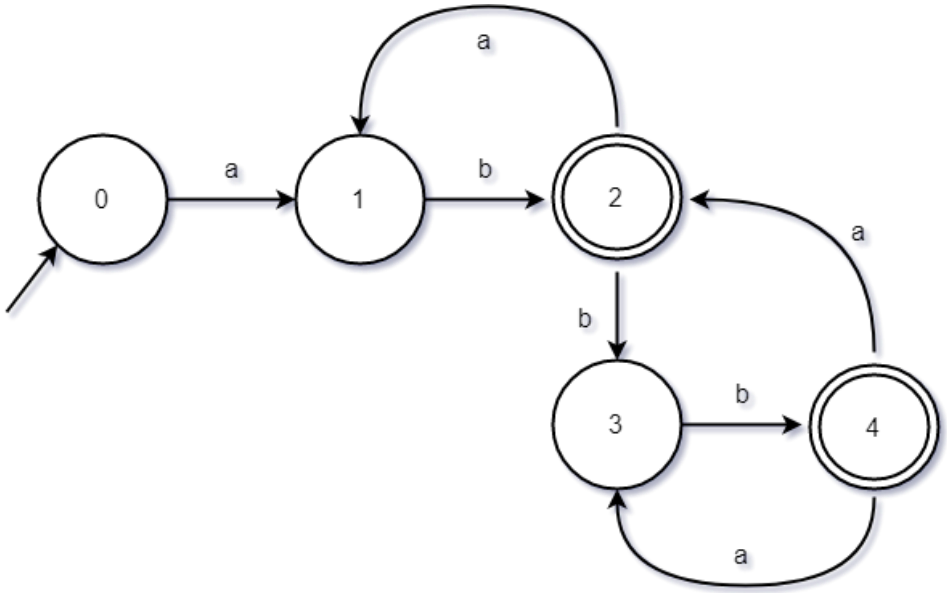
Automate à états finis (exemple)



≈ XKCD 851

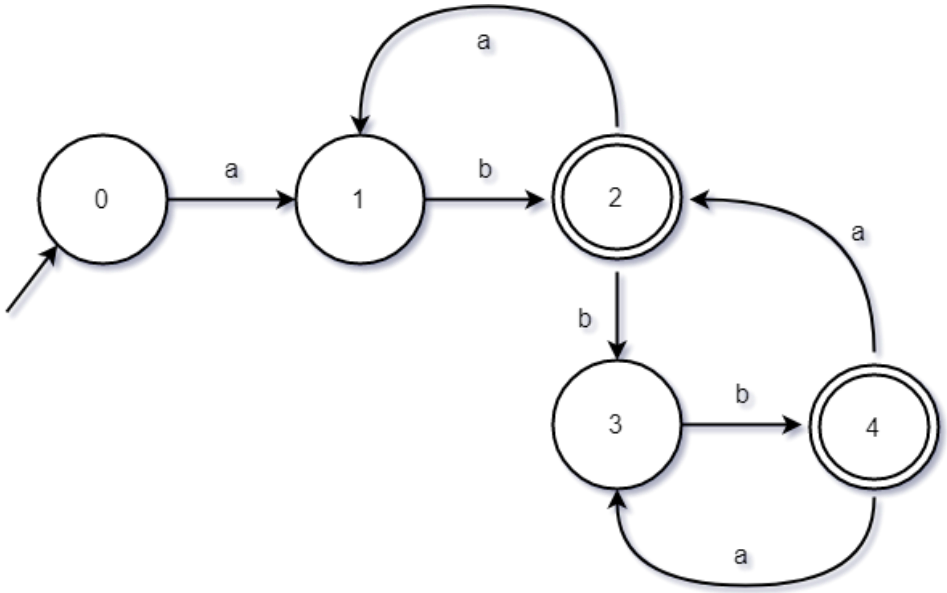
Automate à états finis (exemple)

- un automate est un *graphe orienté*
- il a des *états* (⊙) et des *transitions* (⟶)
- certains états peuvent être *initiaux* ou *finaux*



Automate à états finis (exemple)

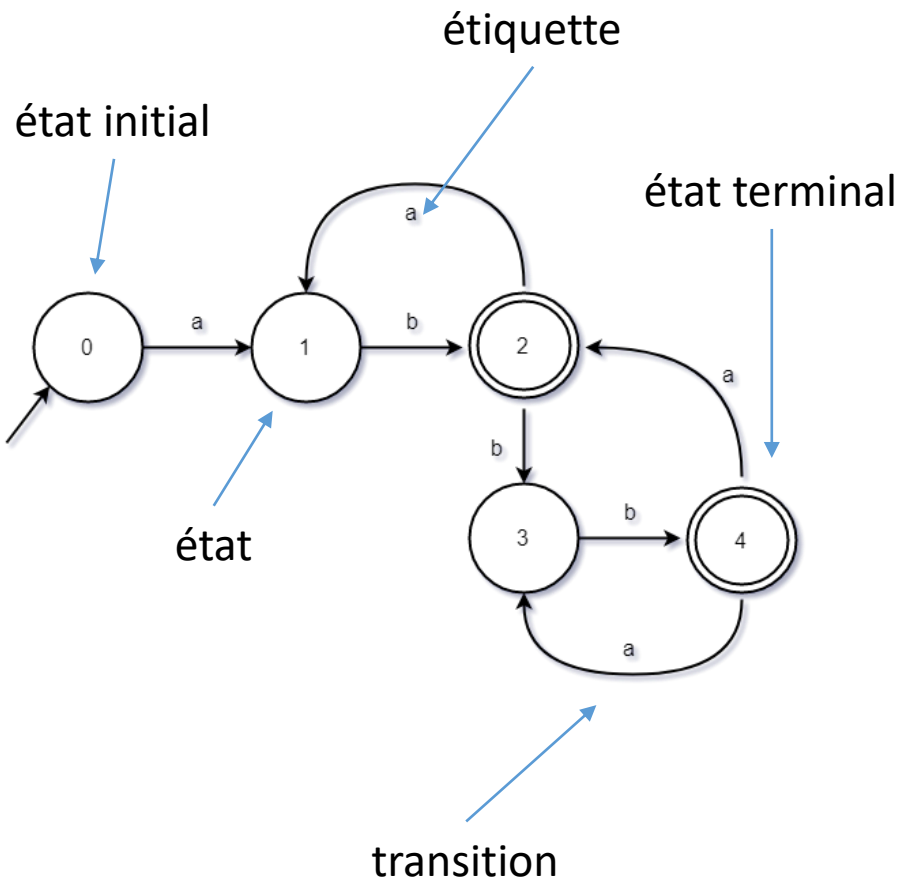
- les étiquettes des transitions sont les *symboles* d'un alphabet, Σ
- les *mots* sont des séquences, dans $\Sigma^{\star}$, e.g. $a\ b\ a\ b\ b$



Automate à états finis (glossaire)

autre noms:

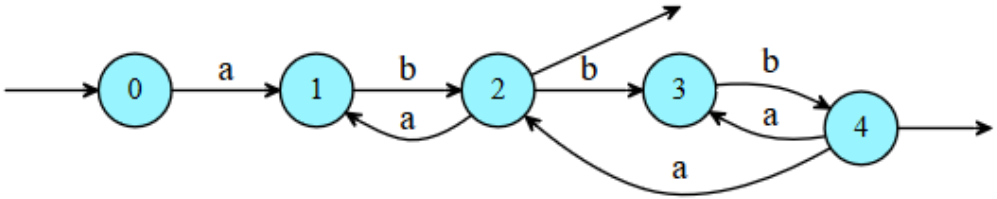
- finite state automata (FSA)
- DFA / NFA (dét. et undét.)
- machines à états
- finite state machine (FSM)
- ≈ finite state transducer
- …

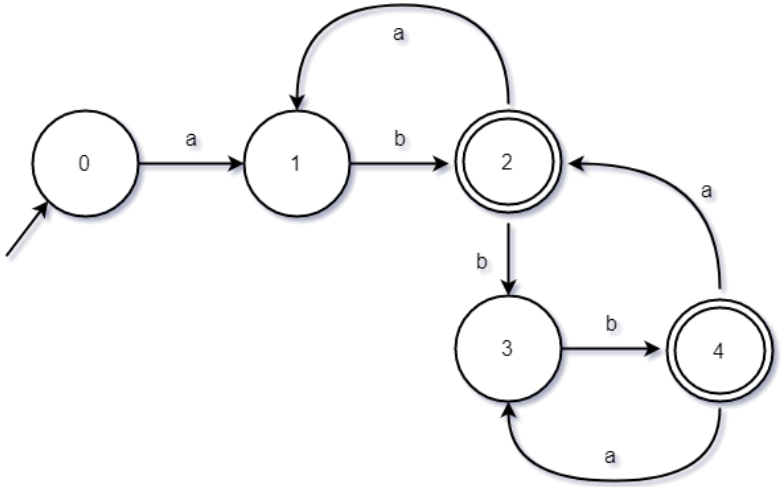


Automate à états finis (syntaxe)

Le même automate dans l'outil VCSN

```
aut1 = vcsn.automaton('''
context = lan(abc), b
$ -> 0
0 -> 1 a
1 -> 2 b
2 -> 1 a
2 -> 3 b
2 -> $
3 -> 4 b
4 -> 3 a
4 -> 2 a
4 -> $
''')
```



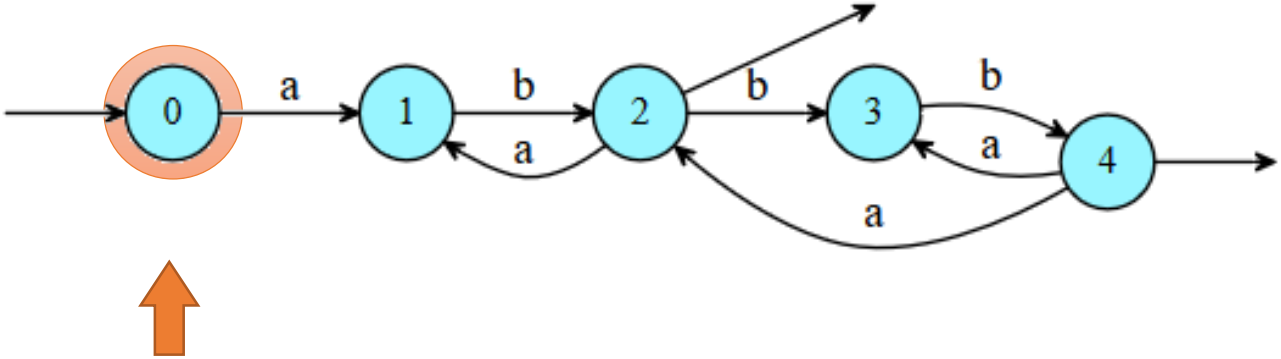


https://www.lrde.epita.fr/wiki/Vcsn

Automates

sémantique

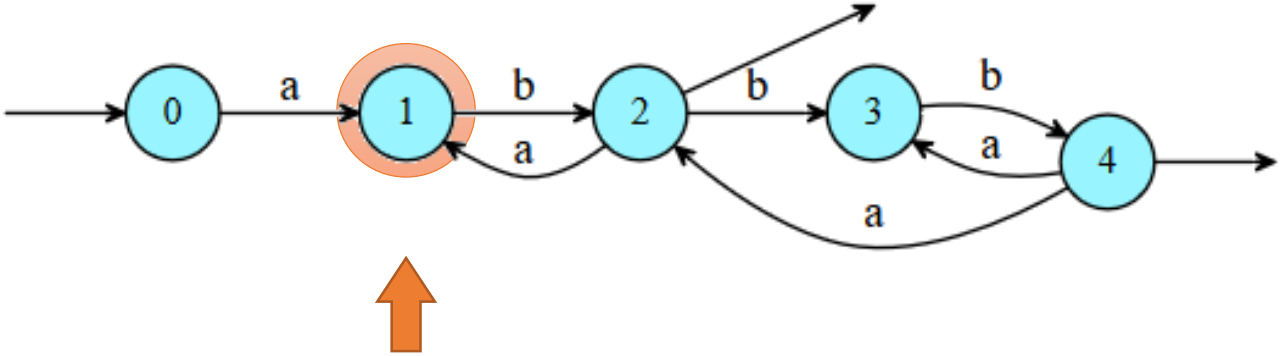
FSA: machine à accepter des mots



Entrée:

$a\ b\ a\ b$

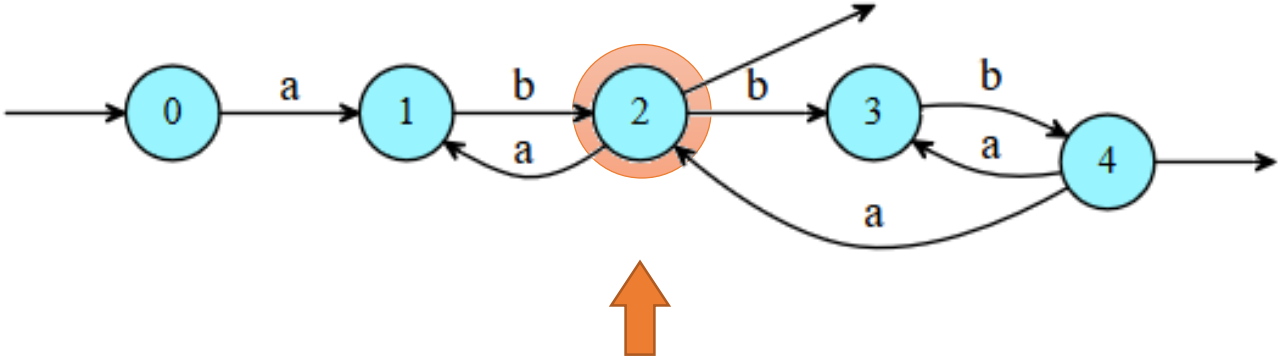
FSA: machine à accepter des mots



Entrée:

$a\ b\ a\ b$

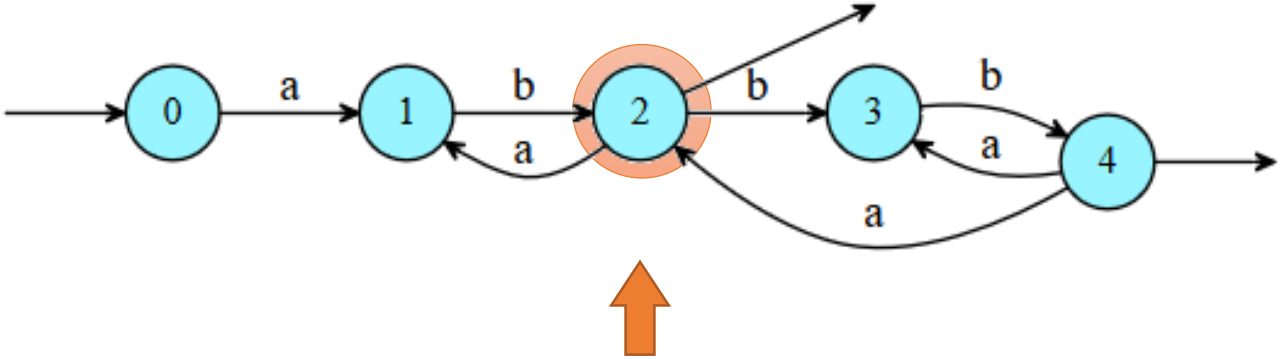
FSA: machine à accepter des mots



Entrée:

$a\ b\ a\ b$

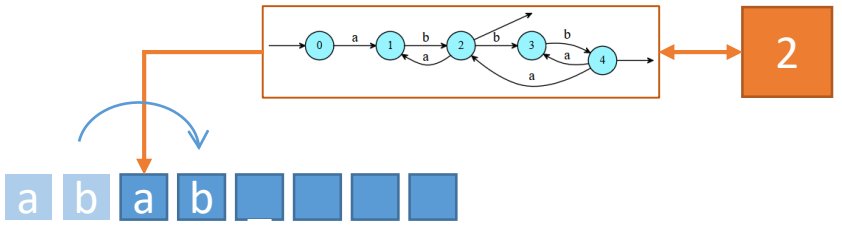
FSA: machine à accepter des mots



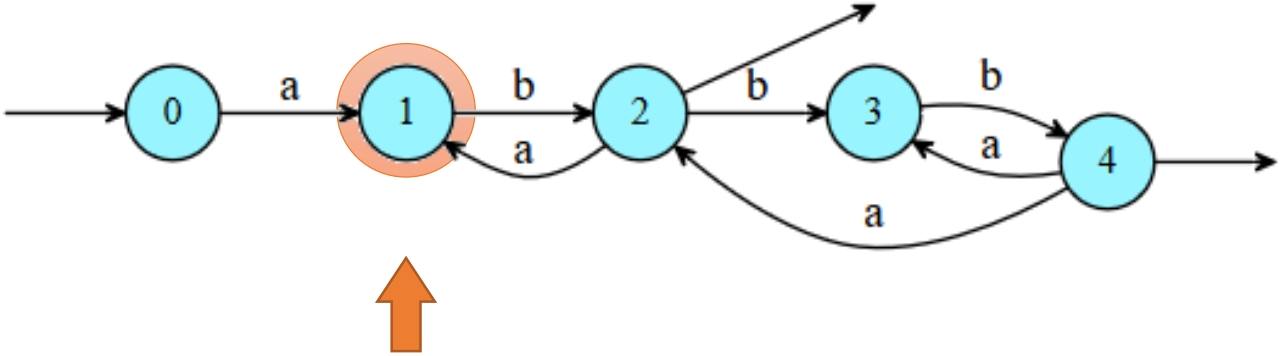
Entrée:

$a\ b\ a\ b$

on peut imaginer une machine lisant ses entrées sur *un ruban* et possédant *un registre* interne



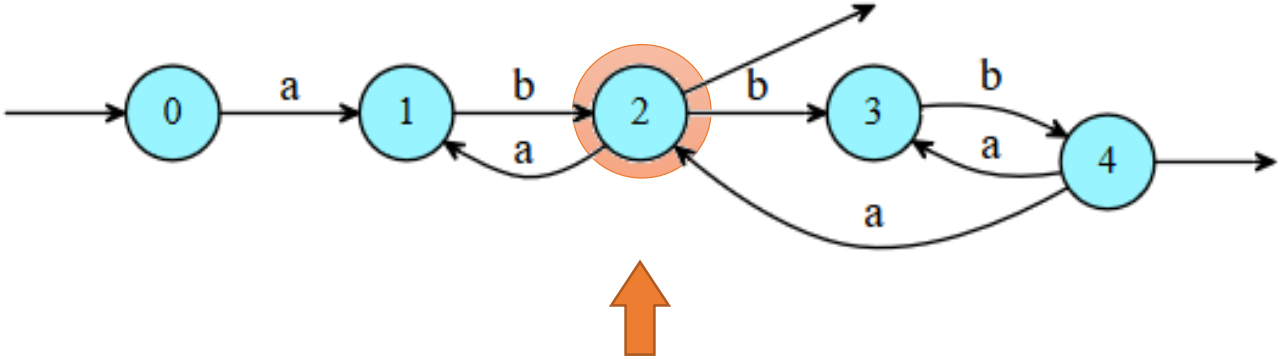
FSA: machine à accepter des mots



Entrée:

$a\ b\ a\ b$

FSA: machine à accepter des mots



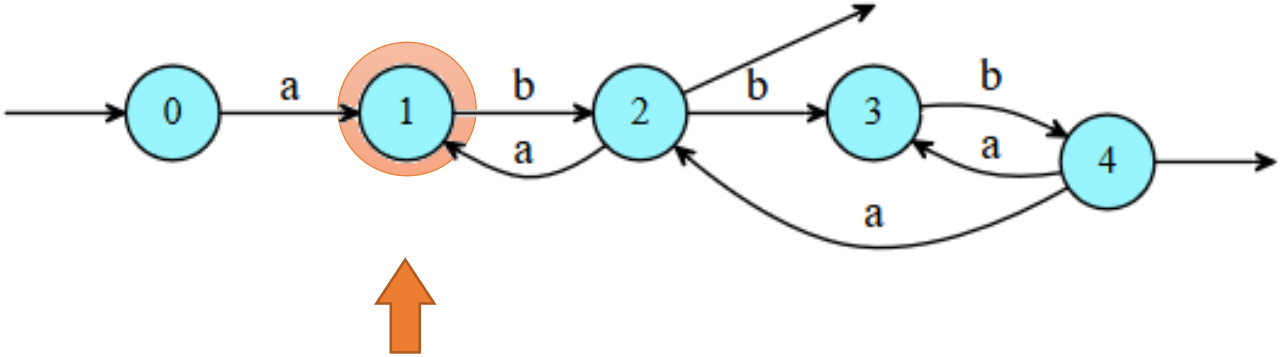
Entrée:

$a\ b\ a\ b$

OK

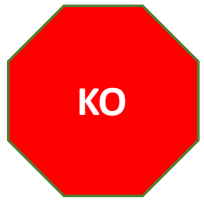
on stoppe sur un état final !
$a\ b\ a\ b \in \mathcal{L}(\mathcal{A})$

FSA: machine à accepter des mots



Entrée:

$a\ b\ a$

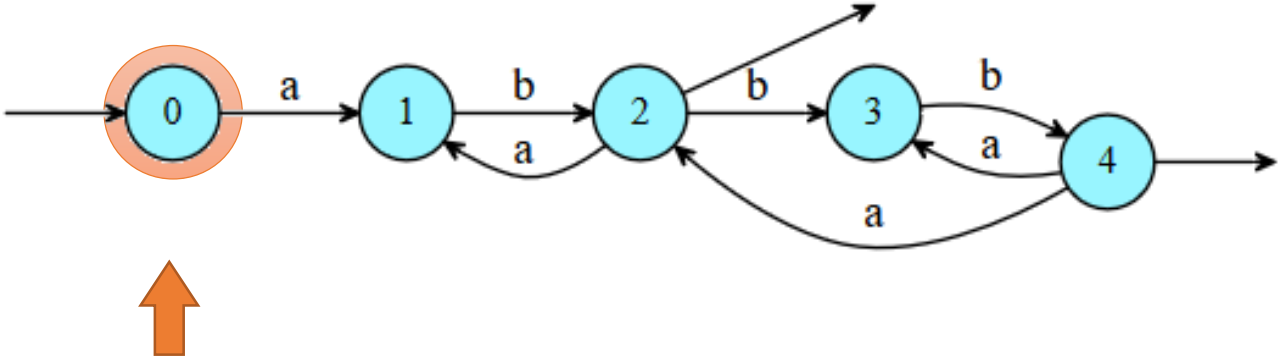


on stoppe sur un état non final !

$a\ b\ a \notin \mathcal{L}(\mathcal{A})$

- run acceptant: $a\ b\ a\ b$ → OK
- run non-acceptant: $a\ b\ a$ → KO
- run bloquant: $a\ b\ a\ b\ b\ a$ → KO

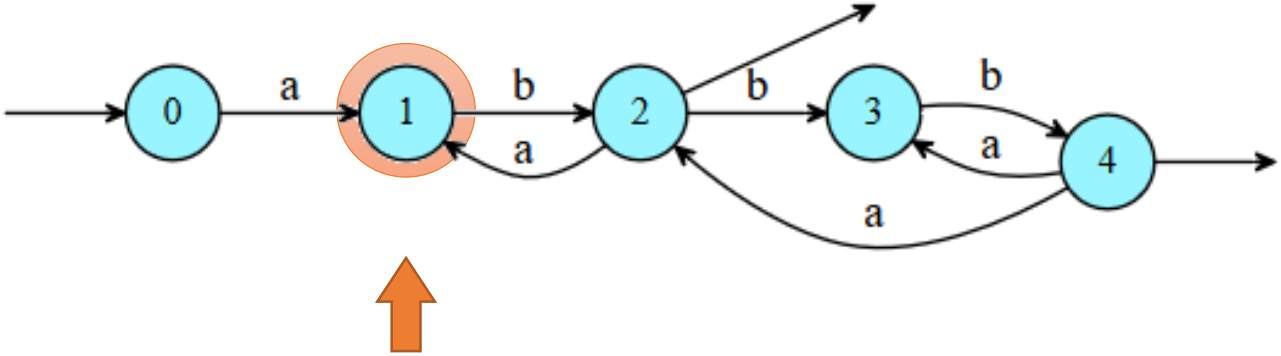
FSA: machine à accepter des mots



Entrée:

$a\ b\ a\ b\ b\ a$

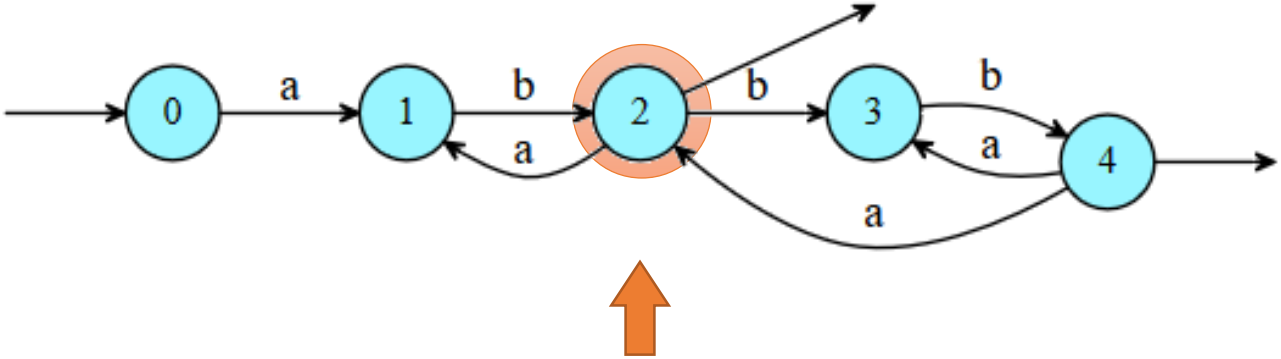
FSA: machine à accepter des mots



Entrée:

$a\ b\ a\ b\ b\ a$

FSA: machine à accepter des mots

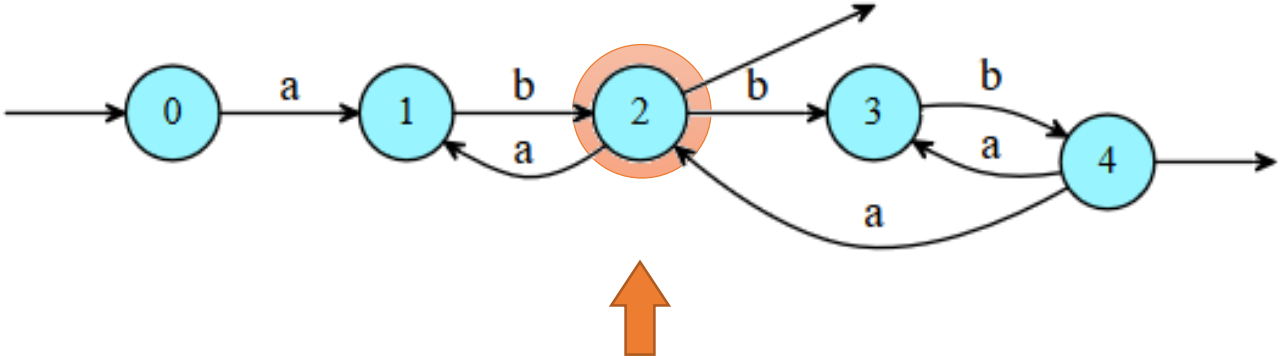


Entrée:

$a\ b\ a\ b\ b\ a$

- run acceptant: $a\ b\ a\ b$ → OK
- run non-acceptant: $a\ b\ a$ → KO
- run bloquant: $a\ b\ a\ b\ b\ a$ → KO

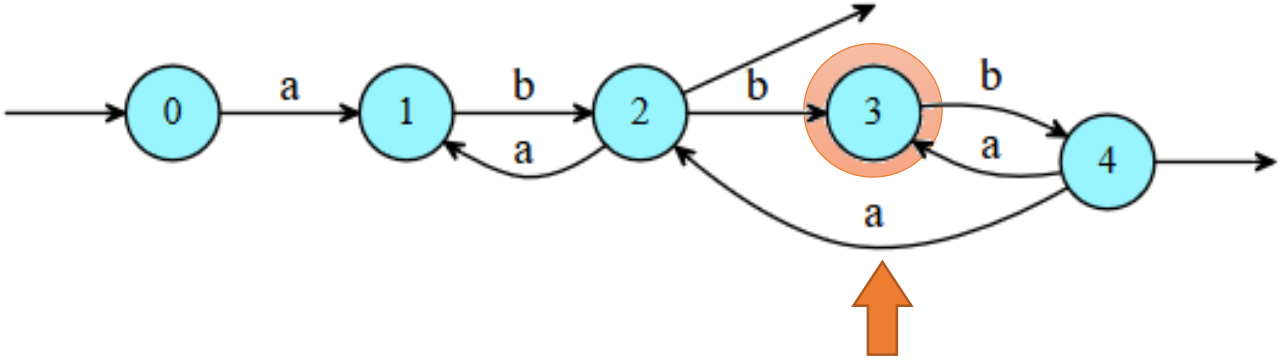
FSA: machine à accepter des mots



Entrée:

$a\ b\ a\ b\ b\ a$

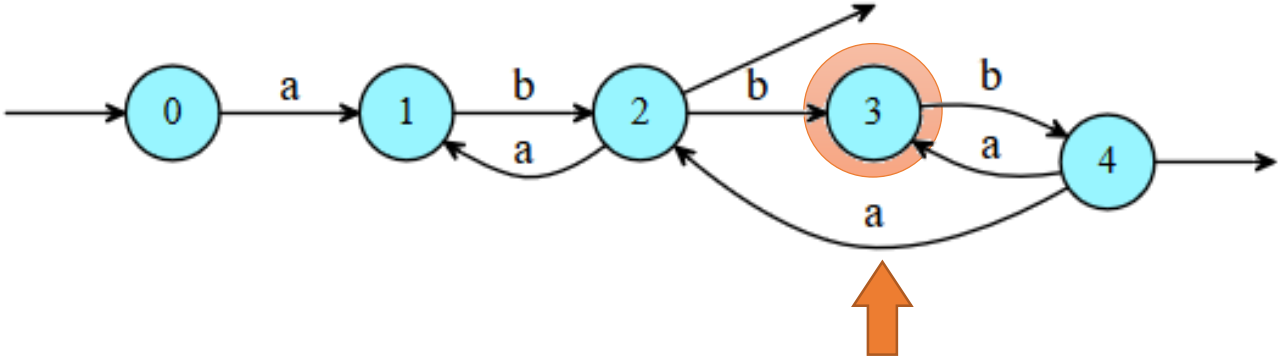
FSA: machine à accepter des mots



Entrée:

$a\ b\ a\ b\ b\ a$

FSA: machine à accepter des mots



Entrée:

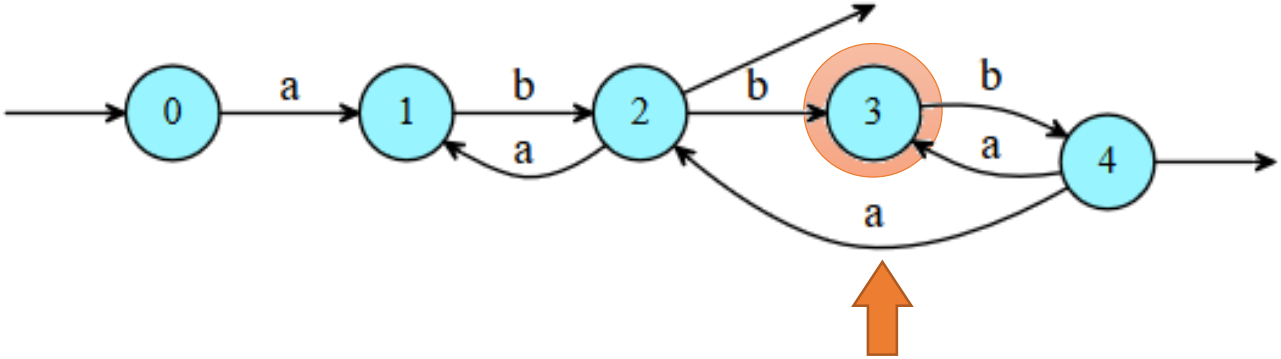
$a\ b\ a\ b\ b\ a$

KO

blocage: l'automate n'est pas complet

- run acceptant: $a\ b\ a\ b$ → OK
- run non-acceptant: $a\ b\ a$ → KO
- run bloquant: $a\ b\ a\ b\ b\ a$ → KO
- run ambigu: $a\ b\ a\ b\ b\ b\ a$ → OK / KO (!?)

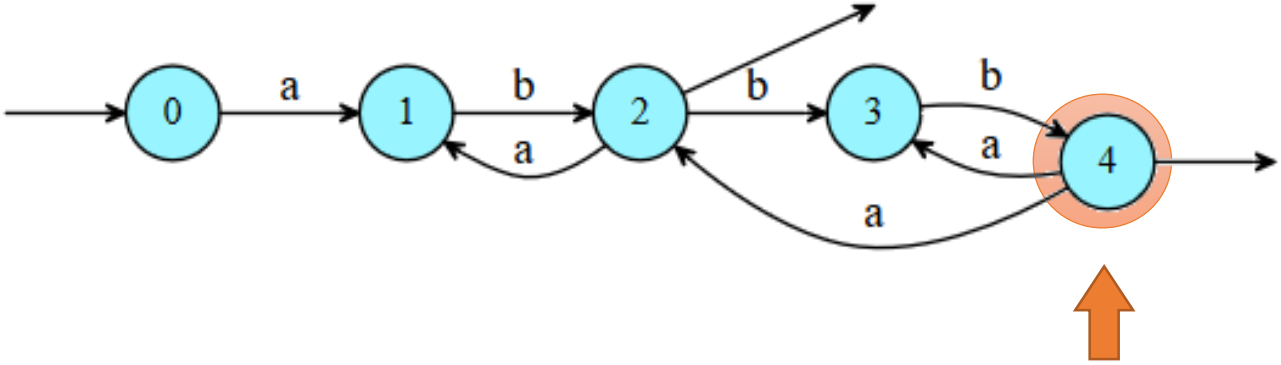
FSA: machine à accepter des mots



Entrée:

$a\ b\ a\ b\ b\ b\ a$

FSA: machine à accepter des mots



Entrée:

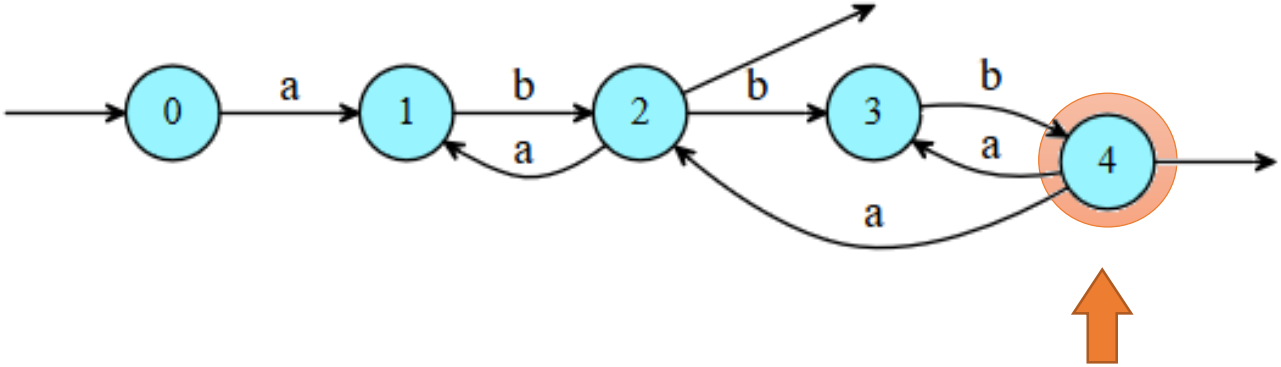
$a\ b\ a\ b\ b\ b\ a$

KO

deux choix possible depuis 4:
l'automate est non déterministe

si on choisit 3, on stoppe, mais sur un état non final

FSA: machine à accepter des mots



Entrée:

$a\ b\ a\ b\ b\ b\ a$

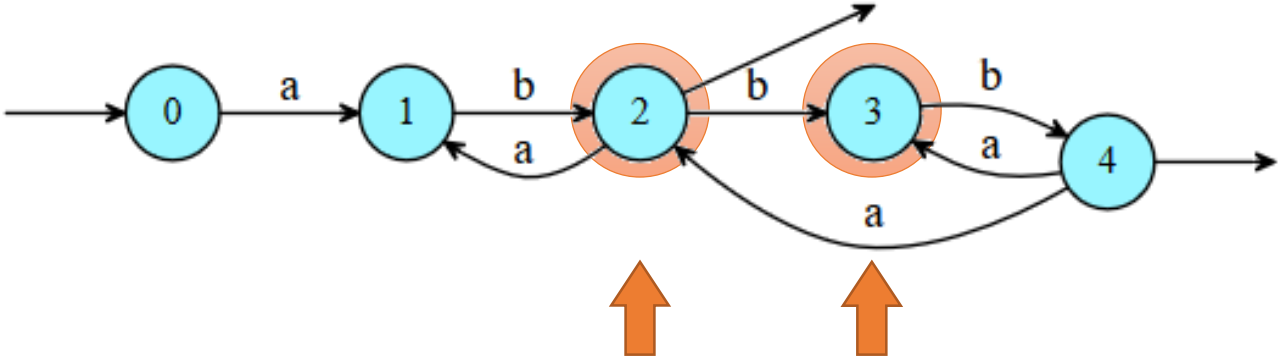
OK

si on choisit 2, on stoppe sur un état final

- run acceptant: $a\ b\ a\ b$ → OK
- run bloquant: $a\ b\ a\ b\ b\ a$ → KO
- run non-acceptant: $a\ b\ a\ b\ b\ b\ a$ → OK / KO (!?)

- run concurrent (multi-têtes) et non-dét.

FSA: machine à accepter des mots

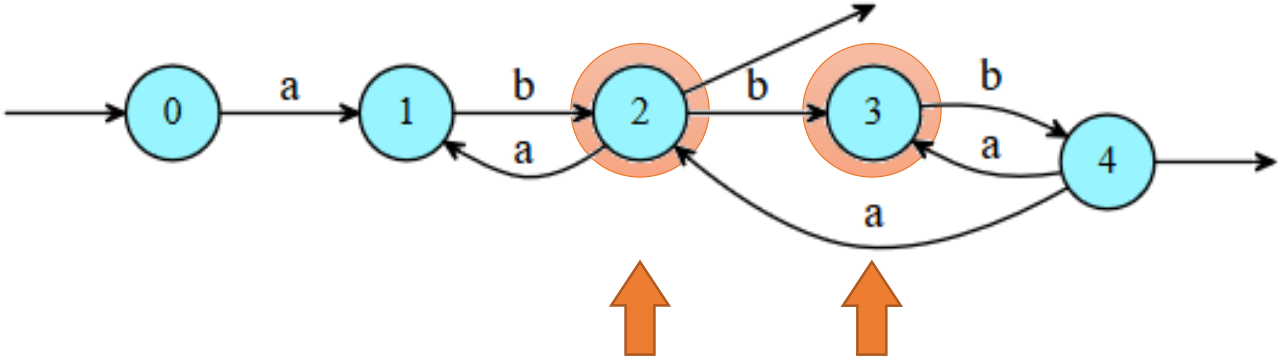


Entrée:

$a\ b\ a\ b\ b\ b\ a$

on peut aussi decider d'avoir plusieures "têtes de recherches" ($\leq |Q|$)

FSA: run de l'automate



Entrée:

$a\ b\ a\ b\ b\ b\ a$

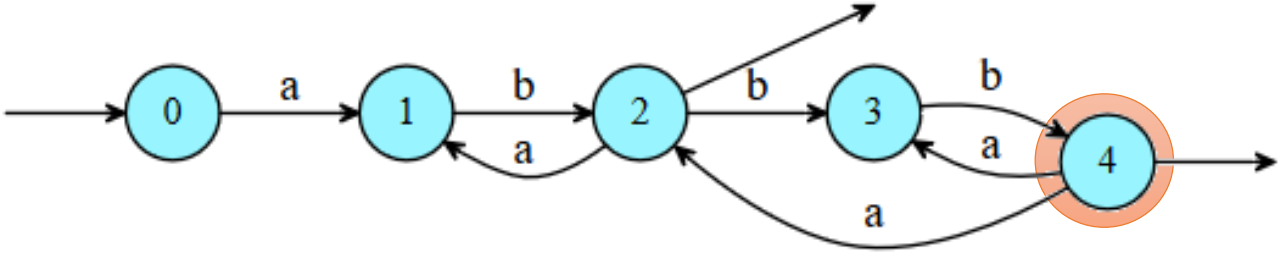
Runs:

0 1 2 1 2 3 4 2

0 1 2 1 2 3 4 3

{0} {1} {2} … {4} {2,3}

FSA: dét. ⊂ unamb ⊂ undét.



Entrée:

$a\ b\ a\ b\ b\ b\ a$

Runs:

0 1 2 1 2 3 4 2 ✓

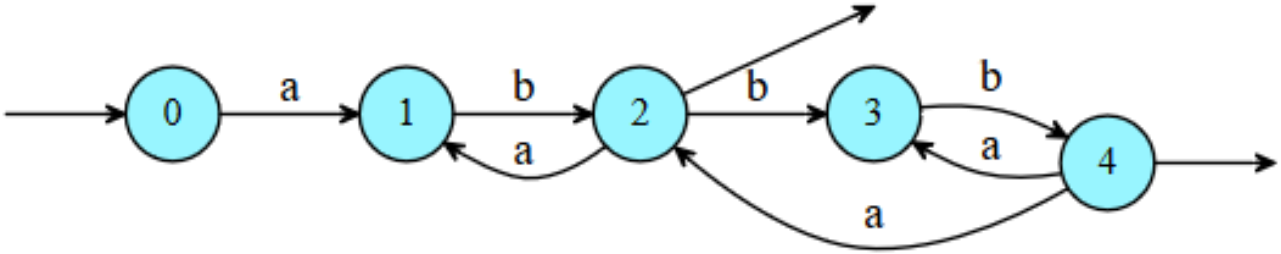
0 1 2 1 2 3 4 3 ✗

l'automate n'est pas ambigu
≡ au plus un run est acceptant

FSA: language

FSA = machine à accepter des mots

FSA = machine à *générer* des mots




```
In [44]: aut1.shortest(6)
```

Out[44]: $ab \oplus abab \oplus abbb \oplus abbba \oplus ababab \oplus ababbb$

Automates

l’introduction qu’il vous manquait

Un domaine …

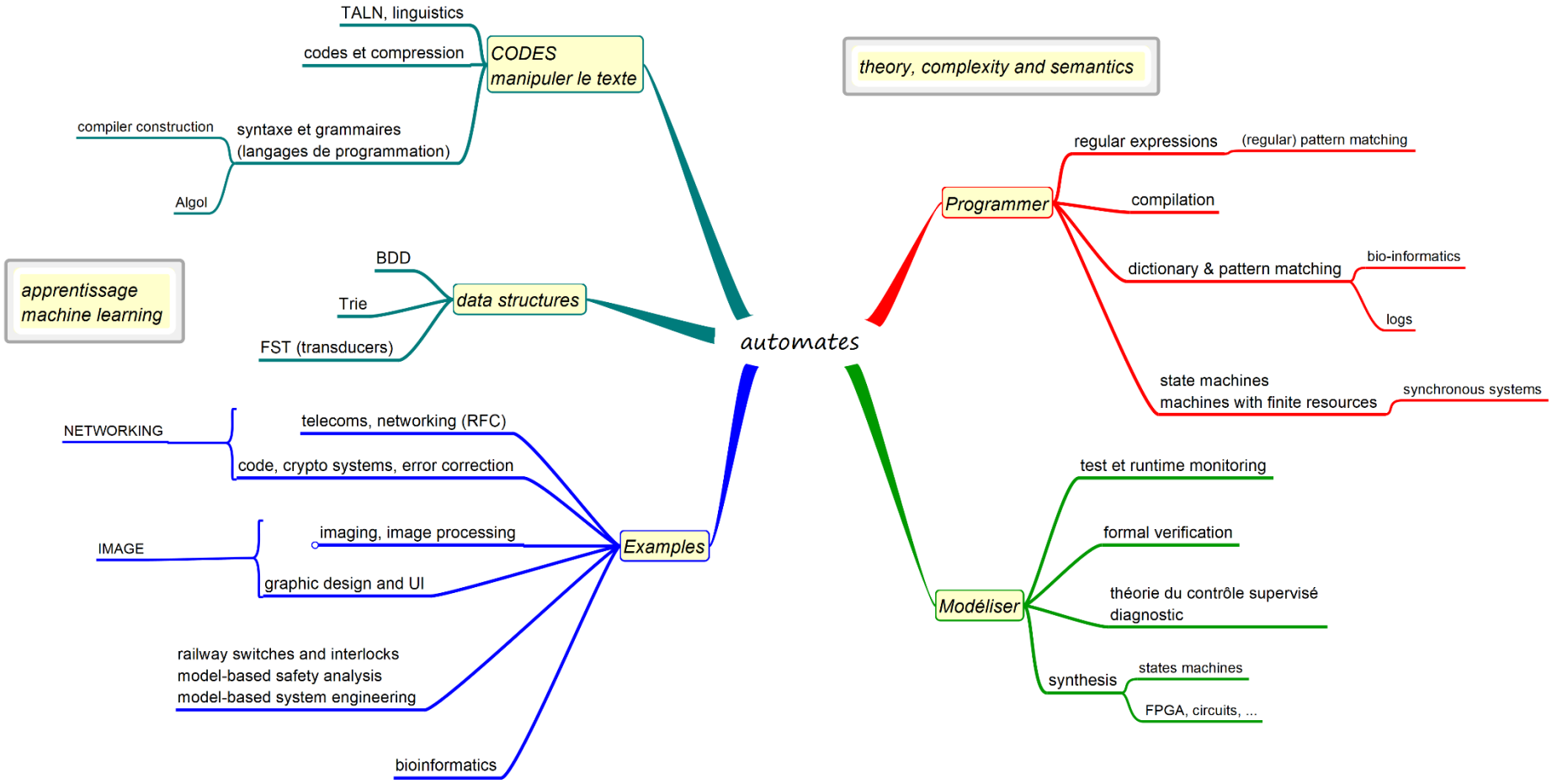
- riche en applications
- riche en histoire
- toujours actif

- fait partie naturelle d’un cours de compilation

un domaine riche en littérature

contient 2 exercices pour les élèves de primaires

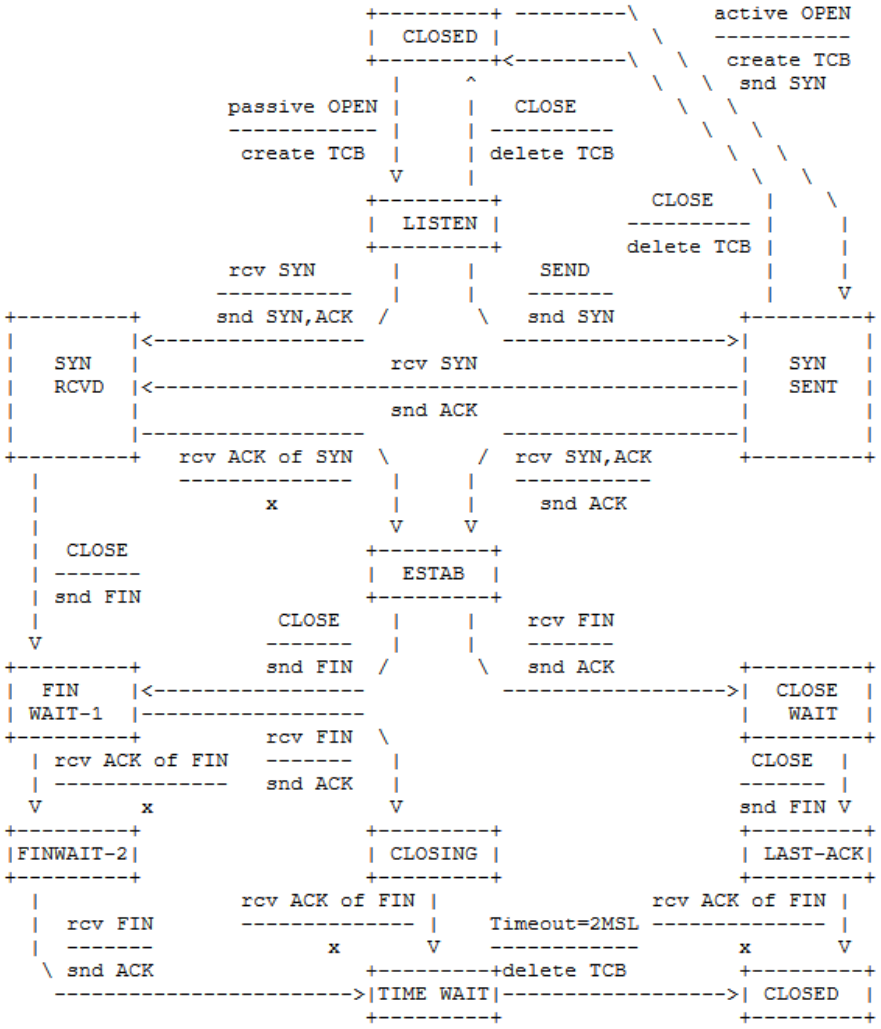
contient ~ 2500 pages



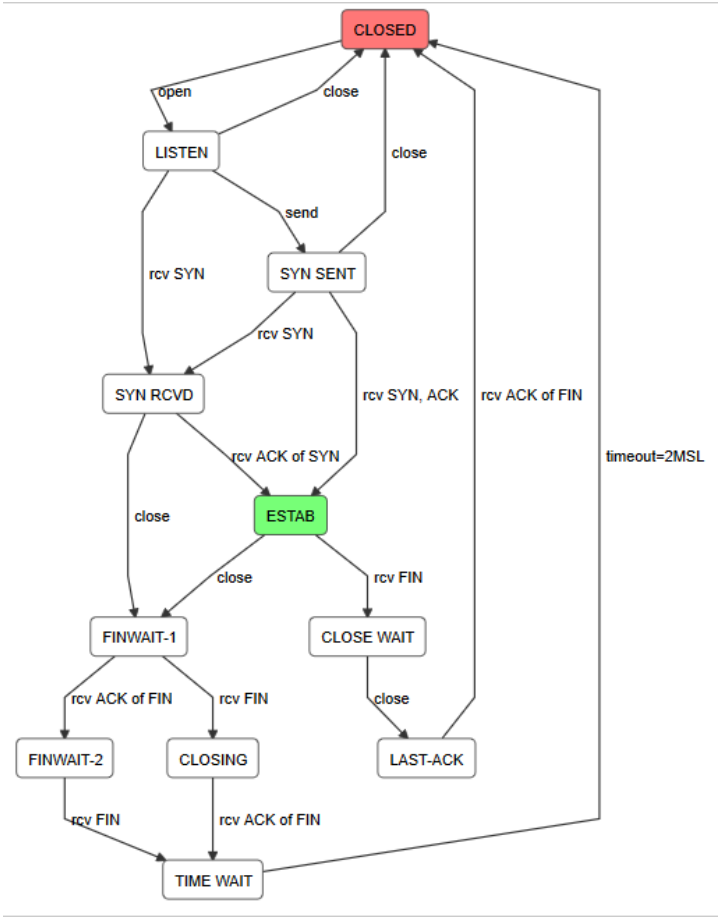
“ le rôle des automates en informatique est ∝ à celui de l’étude des fonctions en mathématiques. ” [moi, 2019]

Networking

RFC: 793
TRANSMISSION CONTROL PROTOCOL
DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION
September 1981



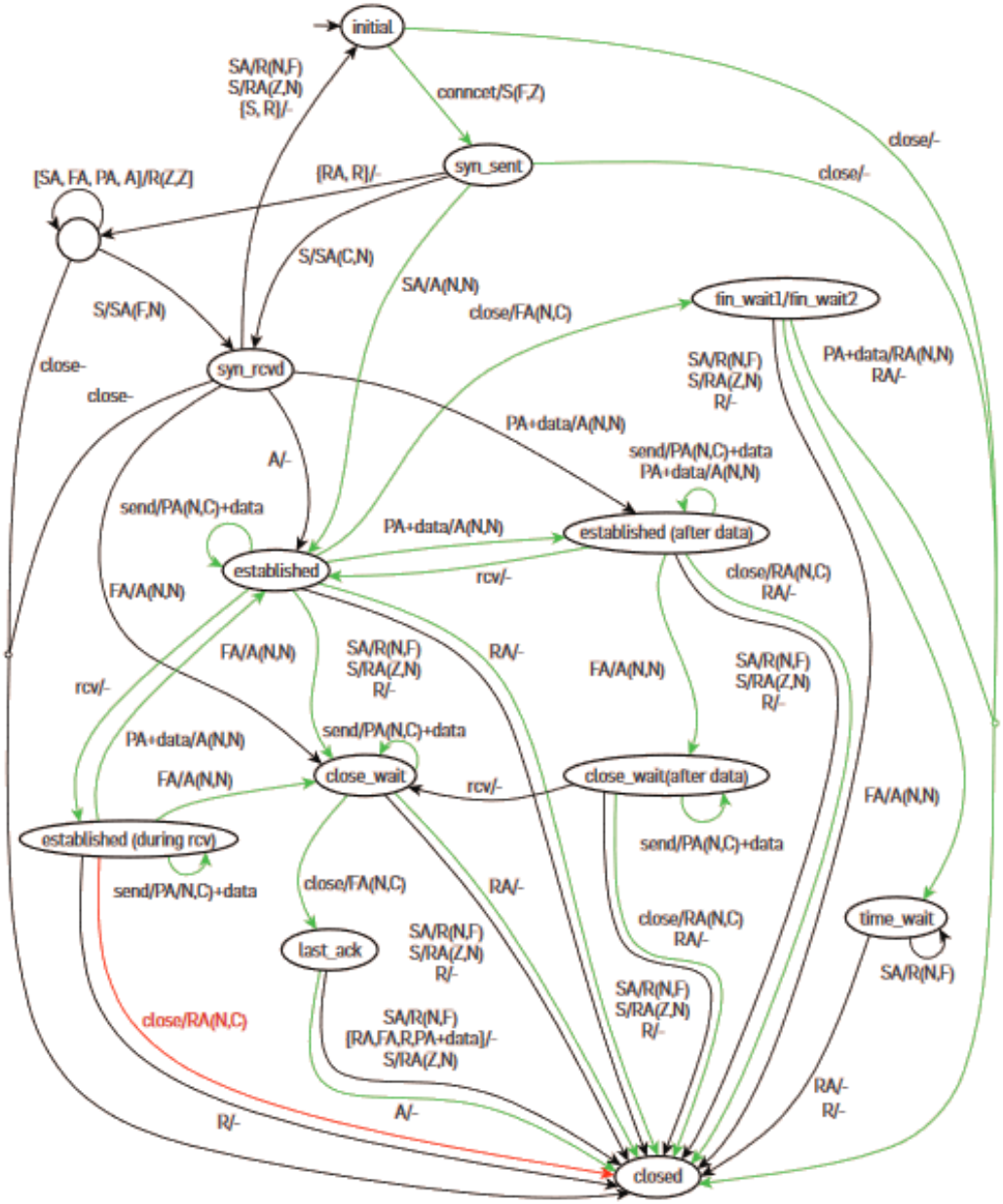
TCP Connection State Diagram
Figure 6.



Learning

Learning Regular Languages (DFA) using Membership and Equivalence Queries

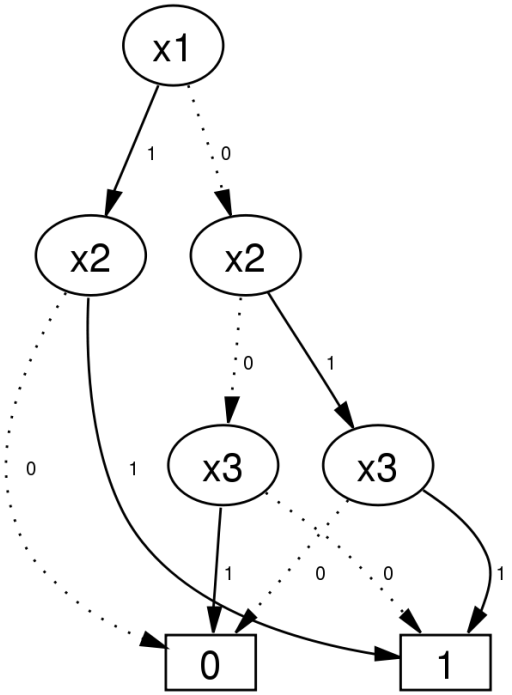
Angluin's L∗ Algorithm



Learned state machine for Windows8 TCP Client*

[*] Fiterau-Brostean, P., Janssen, R., Vaandrager, F. Combining model learning and model checking to analyze TCP implementations. In CAV'16, LNCS 9780 (2016). Springer, 454–471

Data Structures



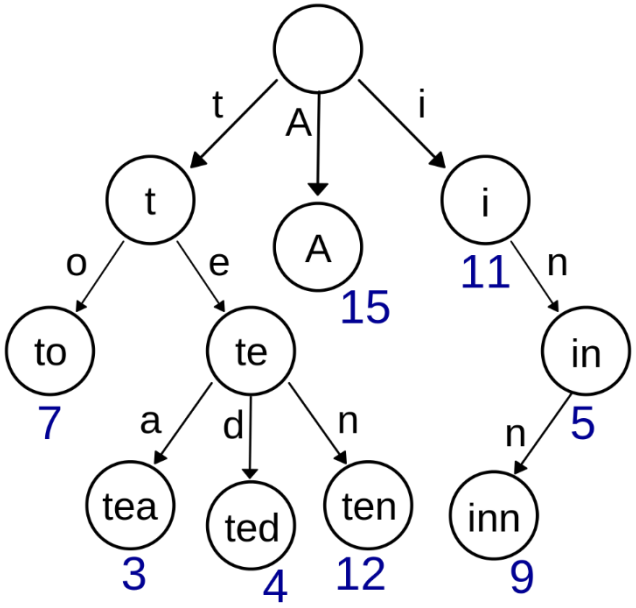
Binary Decision Diagram for the function

$f(x_1, x_2, x_3) = \overline{x_1}\ \overline{x_2}\ \overline{x_3} + x_1\ x_2 + x_2\ x_3$

[Wikipedia]

A trie for keys "A", "to", "tea", "ted", "ten", "i", "in", and "inn".

[Wikipedia]



Computer Imaging

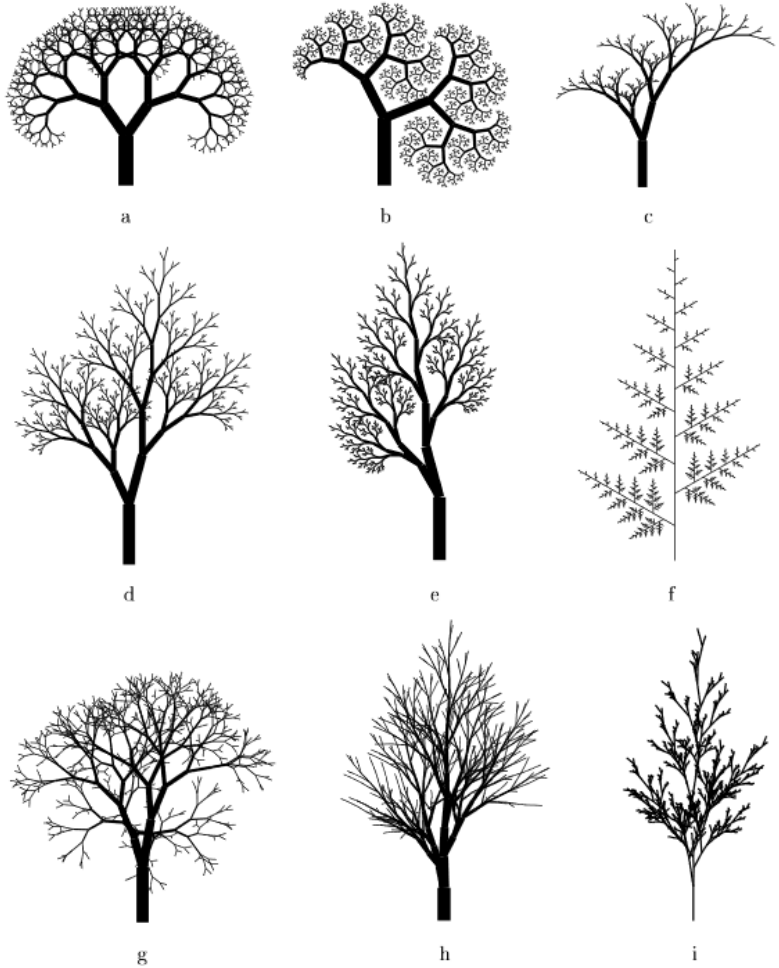


Fig. 4.4. Sample structures generated by a parametric D0L-system with different values of constants

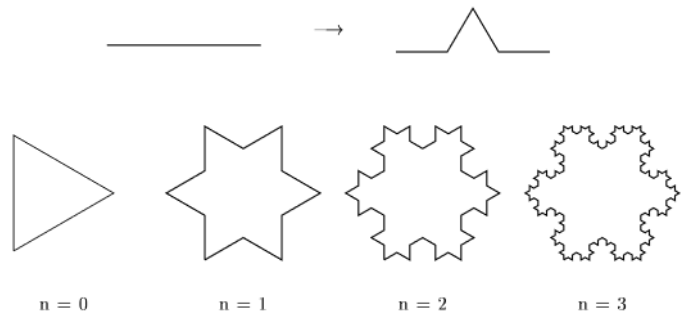
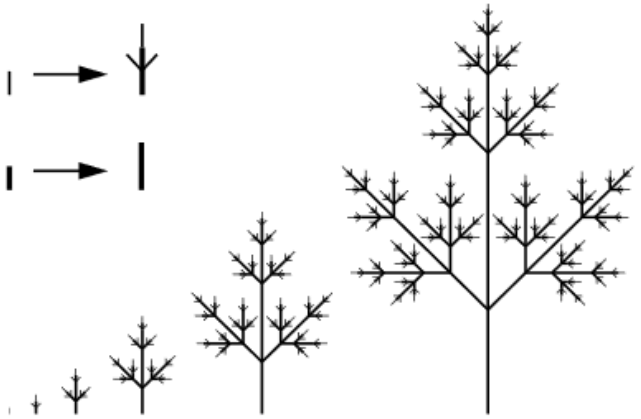
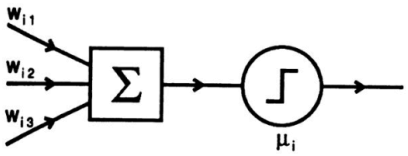


Fig. 4.2. Visual interpretation of the production for the snowflake curve, and the curve after $n = 0, 1, 2$, and 3 derivation steps



Visual models of plant development
Przemyslaw Prusinkiewicz[1], Mark Hammel[1], Jim Hanan[2], and Radomir Mech[1]

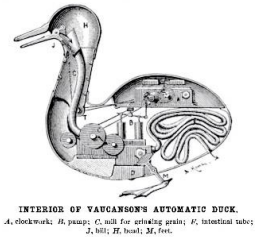
People, history and geography



Warren McCulloch and Walter Pitts

1940s

Andrey Markov



Norbert Wiener

V. M. Glushkov

John von Neumann and Stephen Cole Kleene

Alan Turing

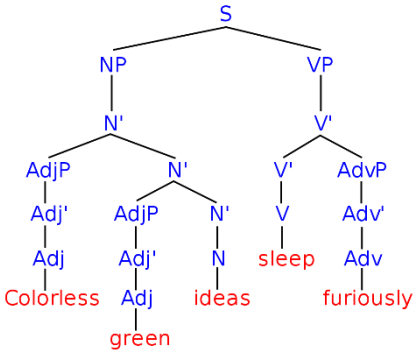
Brzozowski et McCluskey

G.H. Mealy and E.F. Moore

1960s

Büchi and Elgot

John Myhill and Anil Nerode



Marcel-Paul Schützenberger

Noam Chomsky

1970s

Turing award 1976
Rabin, Michael O.
Scott, Dana Stewart

Alexandra Silva (2019)

On étudie quoi ici ?

- Le cours se concentre sur l'aspect acceptation / generation de langages
- On répond ≈ à quelques questions:
  - à quoi peut ressembler une machine *abstraite* et *uniforme.*
  - c'est quoi la puissance d'un *instrument de calcul* qui à des ressources finies ?
  - c'est quoi la puissance du non-déterminisme ?
- On aborde des premieres questions de complexité

Puissance du non-déterminisme ?

En général, le non-déterminisme augmente la puissance d'un système ⇒ sur un ordinateur (une machine de Turing), il permettrait de:

- tester tous les mots de passe d'un seul coup.

⇔ trouver un mot de passe inconnu en un seul essai

- trouver rapidement des séquences génétiques particulières
- trouver rapidement la forme d'une protéine
- factoriser rapidement les grands nombres

⇒ casser les protocoles cryptographiques

Extensions

plus de bandes

Streaming (online)

Counters, FIFO

Tapes, Turing machines

∞ et $\mathbb{N}$

Weights, rational series

Transducers, I/O and values

ω-languages (mots infinis)

d'autres types de mondes

Tree automata

Two-dimensional automata

Cellular automata (Conway)

probabilité, temps, concurrence

Bayesian networks

Markov chains

Time Automata

Petri nets

synchronous languages

Quelques sujets d'études

- Le problème de l'$\star$
- Les mots synchronisant et la conjecture de Černý
- Quelques propriétés des langages unaires ($\Sigma = \{a\}$)
- Complémentation pour les automates non-ambigu

Automata theory

lorsque les choses deviennent formelle

Le monoïde $\Sigma^\star$

- un mot w est une séquence de symboles de Σ
- on peut concaténer deux mots $w_1.w_2$
  - on écrira plus simplement $w_1\ w_2$
  - c'est une operation de produit, associative
- $|w|$, la taille de w, est le nombre de symboles de w
- on note ϵ la sequence vide, $|\epsilon| = 0$
  - ϵ est l'élément neutre pour le produit, $w.\epsilon = \epsilon.w = w$

Exemple: $\Sigma = \{0,1\}$ (binaire), $\{1\}$ (unaire),

ASCII, UTF-8 (texte), 𝅘𝅥𝅮 (musique), …

Propriétés du monoïde $\Sigma^\star$ (ex.)

- si Σ est ordonné, $a \leq b$, alors on obtient un ordre (dit lexicographique) sur $\Sigma^\star$, $\epsilon \leq a\ a\ b \leq a\ b$

- le mot u est dit *préfixe* de w si il existe v tel que $u\ v = w$; de la même manière v est un *suffixe de w*

  on peut noter $u \sqsubseteq w$ cet ordre préfixe*

Lemme (ex.): si $u \sqsubseteq w$ et $v \sqsubseteq w$ (u et v sont deux prefixes de w), alors soit $u \sqsubseteq v$, soit $v \sqsubseteq u$.

⇒ il y a quand même un peu de structure sur $\Sigma^\star$

[*]: pas de condition sur Σ

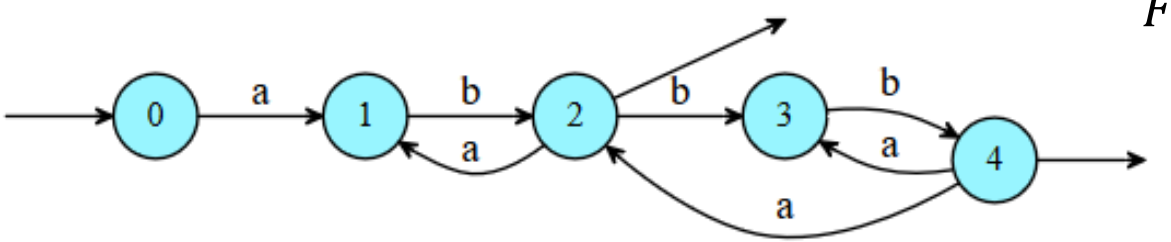
Automate fini, $\mathcal{A}$

$\mathcal{A}$ est un quintuplet $(Q,\ \Sigma,\ \delta,\ q_I, F)$ où :

- Q : ensemble (fini) d'états
- Σ : alphabet
- $q_I \in Q$: état initial de l'automate (parfois $I \subseteq Q$)
- $F \subseteq Q$: états finaux (ou terminaux)
- $\delta \in (Q \times \Sigma) \to Q$: fonction de transition

$Q = \{0\,..\,4\}$

$F = \{2, 4\}$



$q_I = 0$

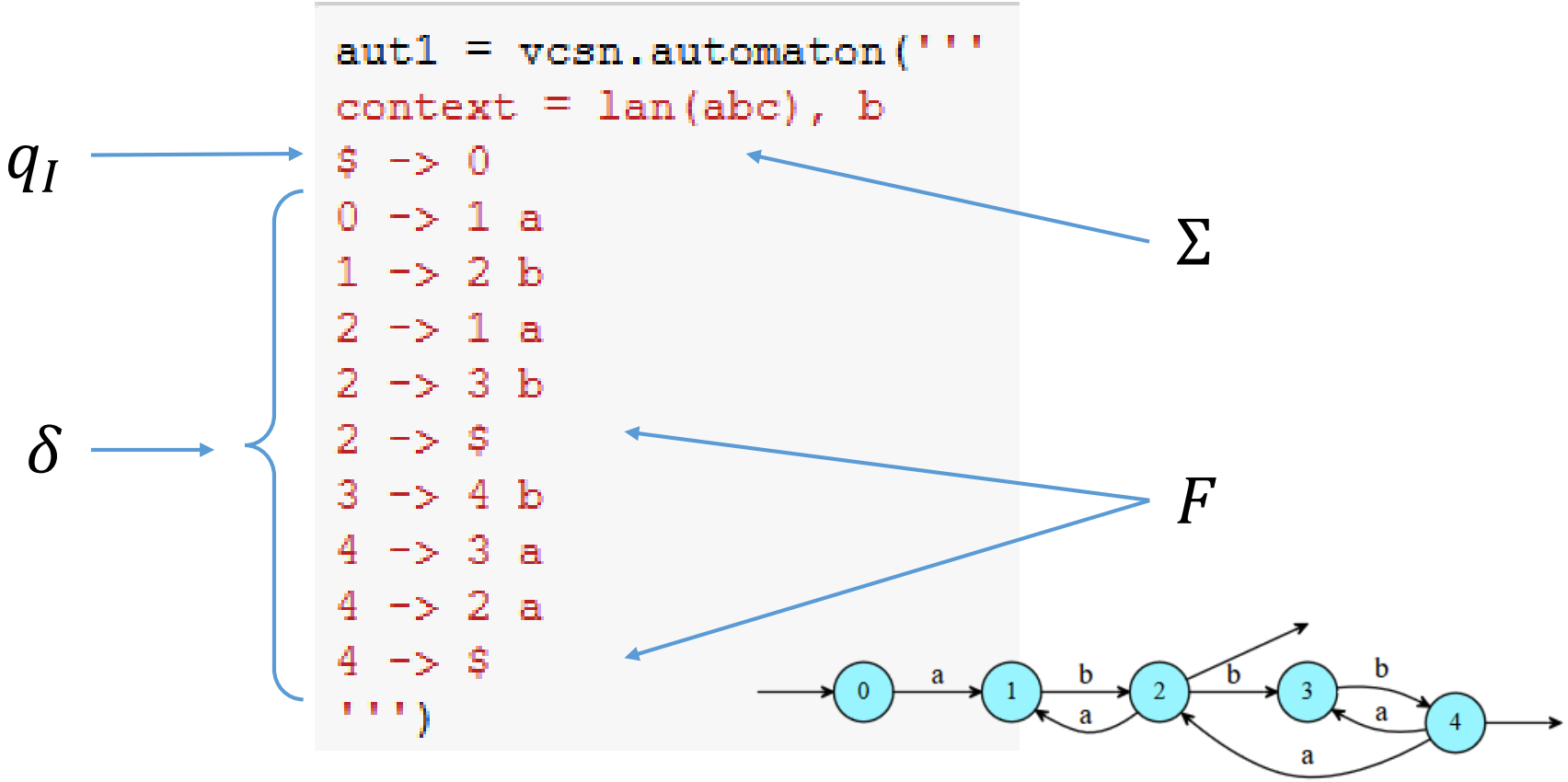
$\Sigma \supseteq \{a, b\}$

$\delta(2, b) = 3, \ldots$

$\delta(4, a) = \{2, 3\}$!!

Syntaxe concrète: graphe

$\mathcal{A}$ est un quintuplet $(Q,\ \Sigma,\ \delta,\ q_I, F)$



Syntaxe concrète: matricielle

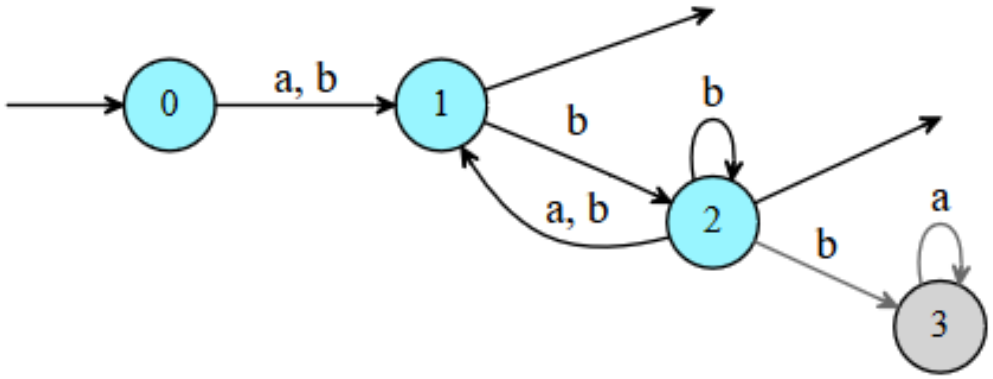
$\mathcal{A}$ est un quintuplet $(Q,\ \Sigma,\ \delta,\ q_I, F)$

	$	0	1	2	3
$	ϵ	T	-	-	-
0	-	ϵ	a, b	-	-
1	T	-	ϵ	b	-
2	T	-	a, b	ϵ, b	b
3	-	-	-	-	ϵ, a

quel est le sens de $\mathcal{A} \times \mathcal{A}$?

$\det(\mathcal{A})$?

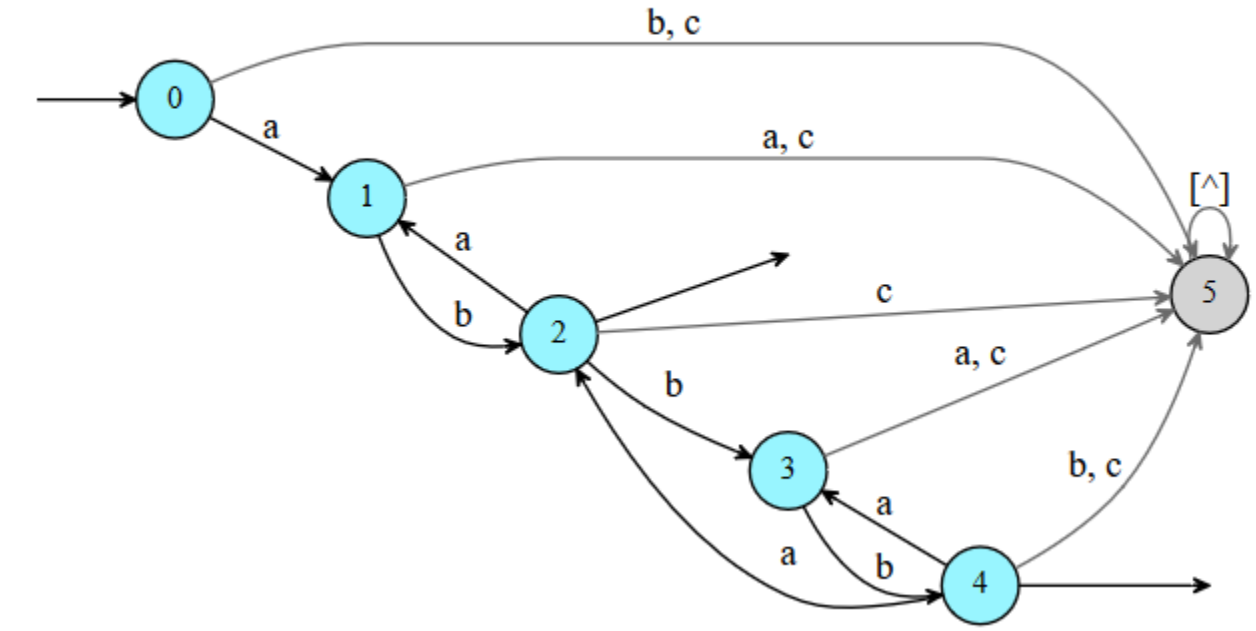
$\mathcal{A}^{-1}$?



Automate complet, $\mathcal{A}$

$\mathcal{A}$ est complet si la fonction $\delta \in (Q \times \Sigma) \to Q$ est totale

```
aut1.complete()
```

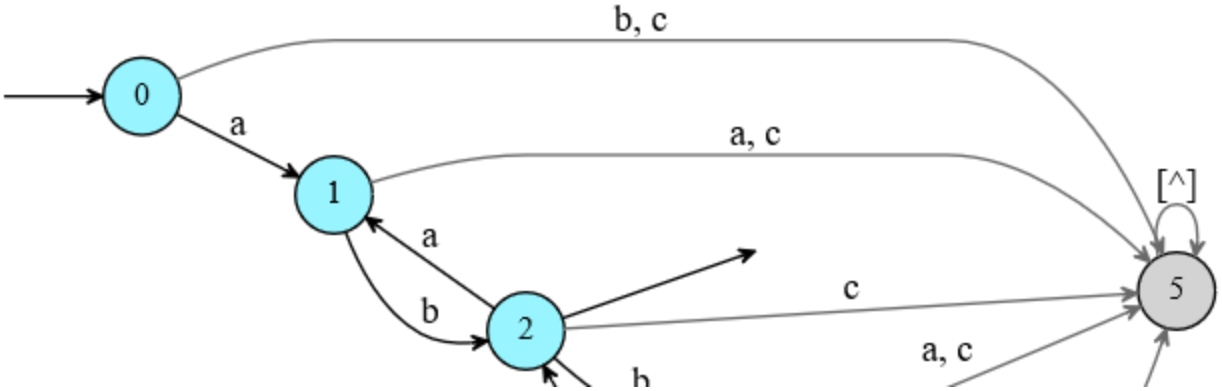


Automate complet, $\mathcal{A}$

$\mathcal{A}$ est complet si la fonction $\delta \in (Q \times \Sigma) \rightarrow Q$ est totale

On peut toujours compléter un automate en ajoutant un état piège/puit

Un automate complet ne peut pas bloquer.



Langage $\mathcal{L}(\mathcal{A})$

On peut étendre δ à $\Sigma^\star$ simplement:

$$\hat{\delta}(q, \epsilon) = q$$
$$\hat{\delta}(q, a\, w) = \hat{\delta}(\delta(q, a), w)$$

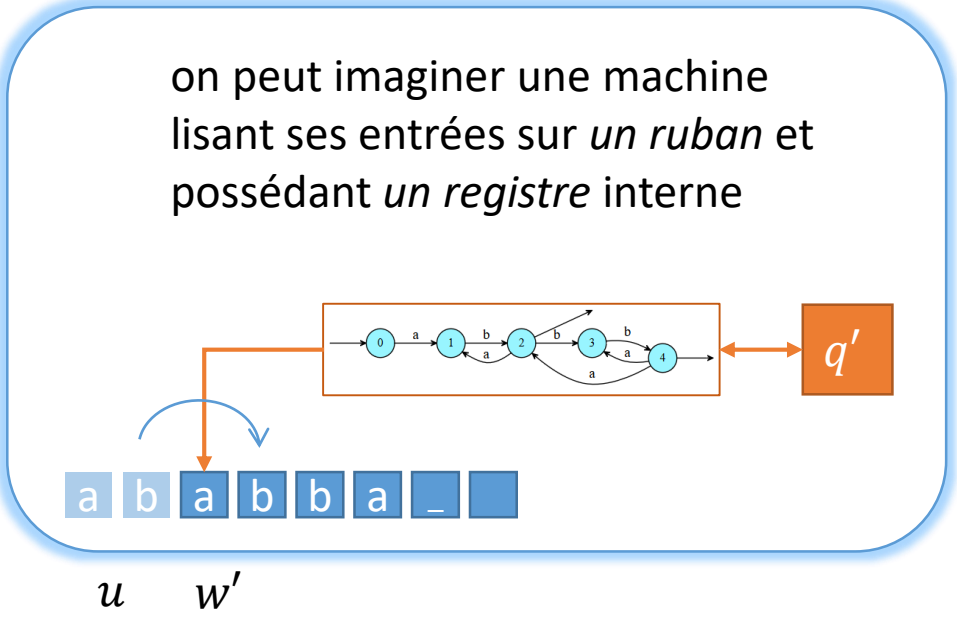
et donc:

$$\hat{\delta}(q, w_1 w_2) = \hat{\delta}(\hat{\delta}(q, w_1), w_2)$$

Langage $\mathcal{L}(\mathcal{A})$

Plus simplement on écrit $(q, w) \Rightarrow (q', w')$ si

$w = u\,w'$ et $\hat{\delta}(q, u) = q'$



Langage $\mathcal{L}(\mathcal{A})$

Plus simplement on écrit $(q, w) \Rightarrow (q', w')$ si

$w = u\,w'$ et $\hat{\delta}(q, u) = q'$

Avec ces notations, le langage de l'automate est:

$$\mathcal{L}(\mathcal{A}) = \{\, w \in \Sigma^\star \mid \hat{\delta}(q_I, w) \in F \,\}$$
$$\mathcal{L}(\mathcal{A}) = \{\, w \in \Sigma^\star \mid (q_I, w) \Rightarrow (q', \epsilon),\ q' \in F \,\}$$

ϵ-transitions

On peut étendre δ aux ϵ-transitions (*spontaneous transition*)

$\Delta(q, w) \supseteq \Delta(q', w)$ si $\delta(q, \epsilon) \supseteq q'$

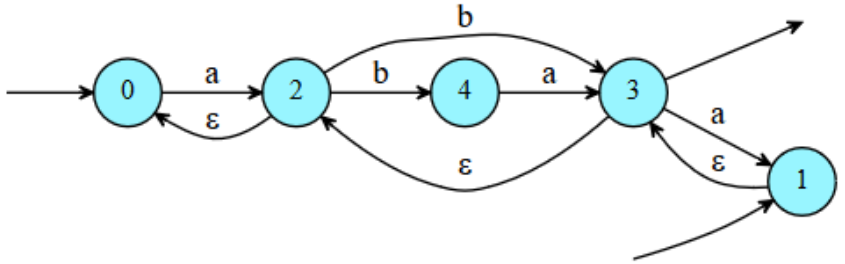
Par exemple:

$(0, a) \Rightarrow 0$

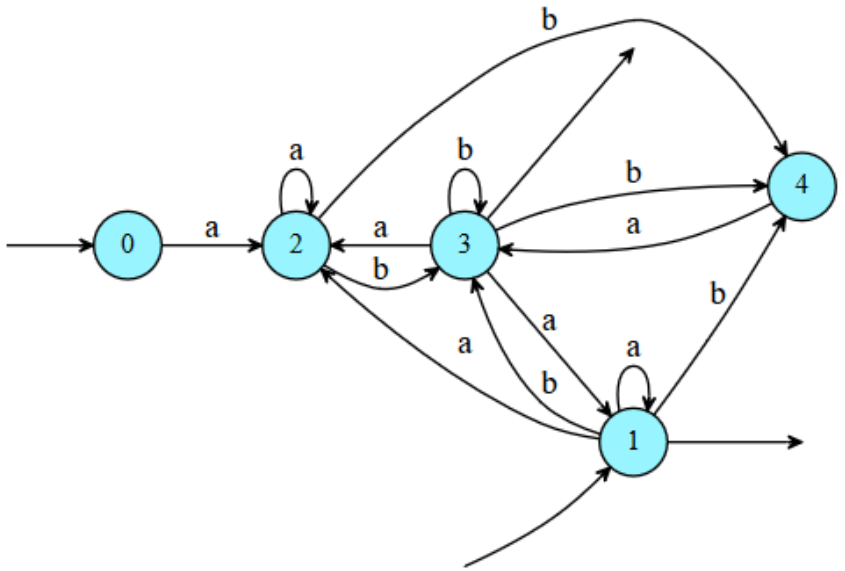
$(2, a) \Rightarrow 2$

$(3, a) \Rightarrow 2$

```
$ -> 0
$ -> 1
0 -> 2 a
1 -> 3 \e
2 -> 0 \e
2 -> 3 b
2 -> 4 b
3 -> 1 a
3 -> 2 \e
4 -> 3 a
3 -> $
''')
a
```



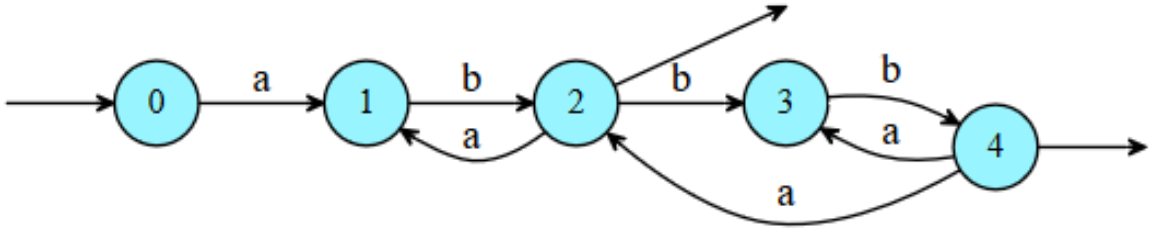

```
a.proper()
```



Automate déterministe, DFA

$\mathcal{A}$ est déterministe si $\delta \in (Q \times \Sigma) \to Q$

$\mathcal{A}$ non-déterministe implique $\delta \in (Q \times \Sigma) \to 2^Q$



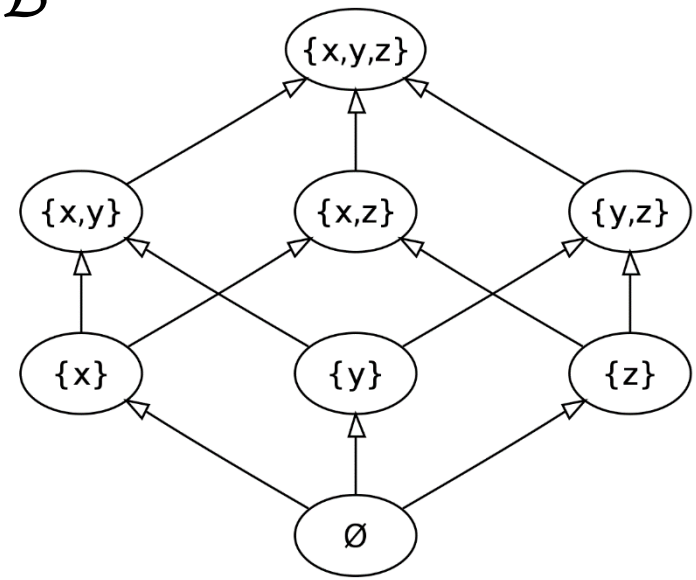
e.g. $\delta(4, a) = \{2, 3\}$

On the powerset construction

- On utilise 2^Q pour dénoter l'ensemble des parties de Q (ou $\mathcal{P}(Q)$ en notation "classique")

- C'est un rappel de la relation entre 2^Q et l'ensemble des fonctions $Q \mapsto \mathcal{B}$

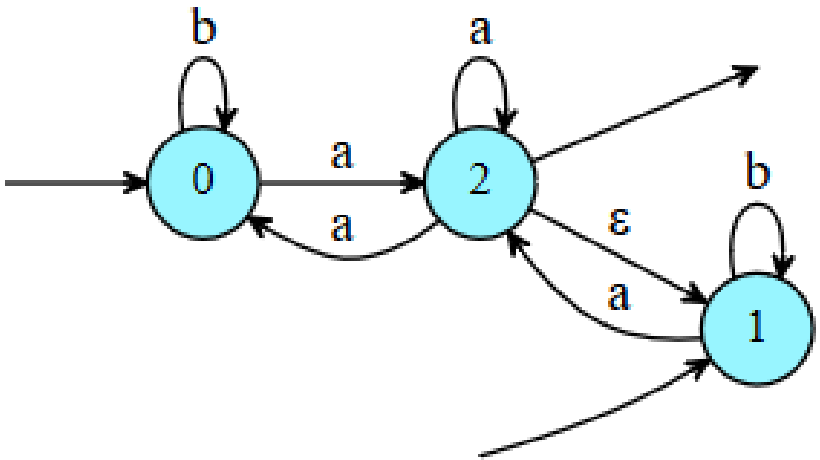
  (où $card(\mathcal{B}) = 2$)



Automate non-déterministe, NFA

Il y a 3 critères pour que $\mathcal{A}$ soit non-déterministe:

- δ n'est pas une fonction $|\delta(q,a)| > 1$
- il y a des ϵ-transitions $\delta(q,\epsilon) \neq \emptyset$
- I n'est pas un singleton $|I| > 1$



Langage $\mathcal{L}(\mathcal{A})$, non-dét.

On peut étendre δ à Q et $\Sigma^\star$ simplement:

$$\hat{\delta}(q, \epsilon) = \{q\}$$
$$\hat{\delta}(q, a\,w) = \bigcup_{q' \in \delta(q,a)} \delta(q', w)$$
$$\Delta(S, w) = \bigcup_{q \in S} \hat{\delta}(q, w)$$

Langage $\mathcal{L}(\mathcal{A})$, non-dét.

Plus simplement on écrit $(q, w) \Rightarrow (q', w')$ si

$w = u\, w'$ et $q' \in \hat{\delta}(q, u)$

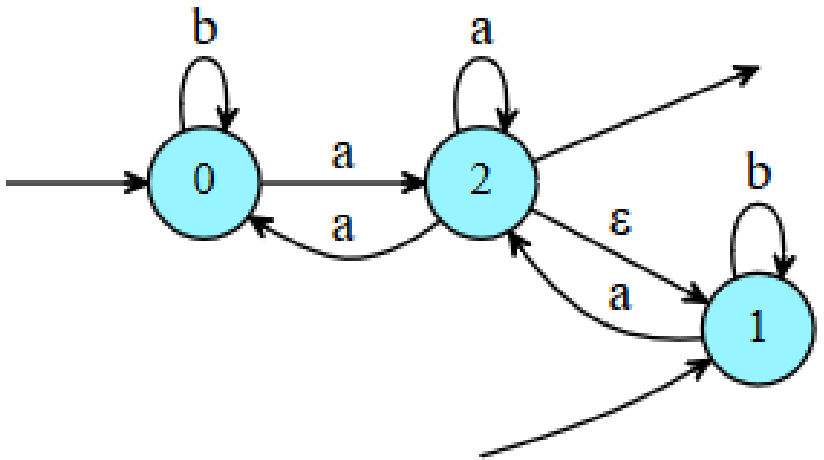
Avec ces notations, le langage de l'automate est:

$$\mathcal{L}(\mathcal{A}) = \{\, w \in \Sigma^\star \mid \Delta(I, w) \cap F \neq \emptyset \,\}$$
$$\mathcal{L}(\mathcal{A}) = \{\, w \in \Sigma^\star \mid (q, w) \Rightarrow (q', \epsilon), q \in I, q' \in F \,\}$$

Caractériser $\mathcal{L}(\mathcal{A})$

(question type)

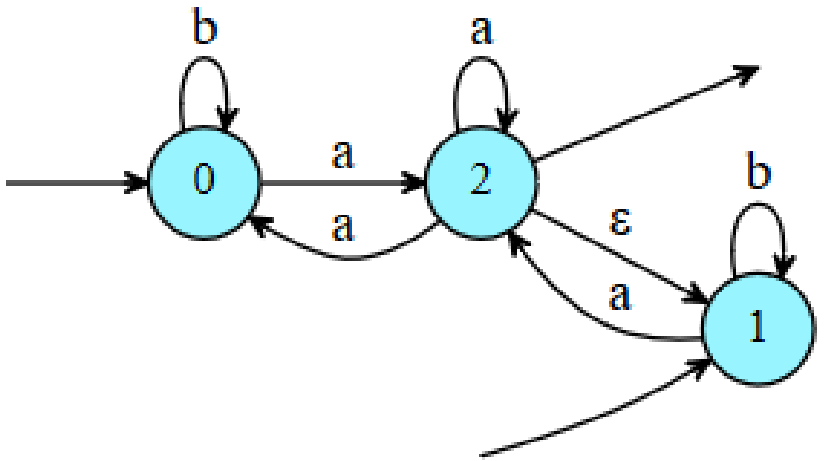
Langage (exemple)

Quel est le langage de 

?

(pour $\Sigma = \{a, b\}$)

Langage (exemple)

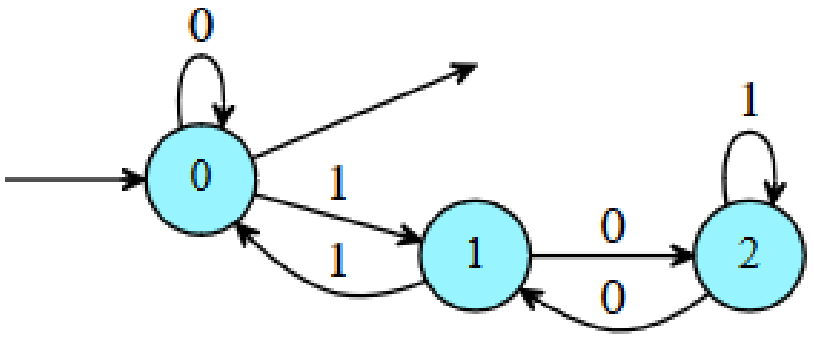
Quel est le langage de 

?

L'ensemble des mots se terminant par a

L'ensemble $\mathcal{L} = \{ w \mid w = w'a \ \wedge \ w' \in \Sigma^\star \}$

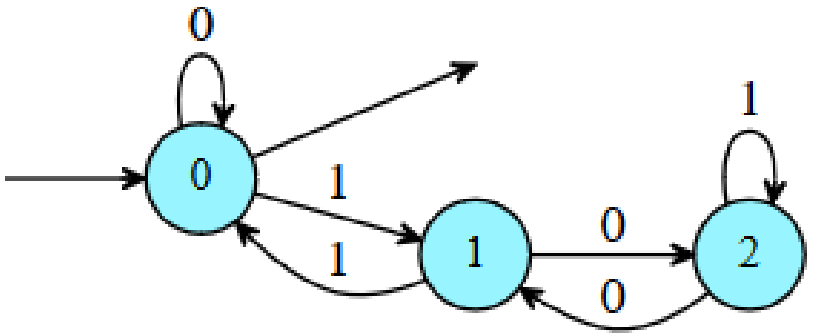
Langage (exemple)

Quel est le langage de 

?

⚠ dans cet exemple le nom des états à une signification utile

Langage (exemple)

Quel est le langage de 

?

L'ensemble $\{ x \in \mathbb{N} \mid x \equiv 0 \ [3] \}$ (x en binaire)

Pressburger arithmetics: $\exists Y . (X = Y + Y + Y) \wedge \ \ldots$

Automates et modélisation

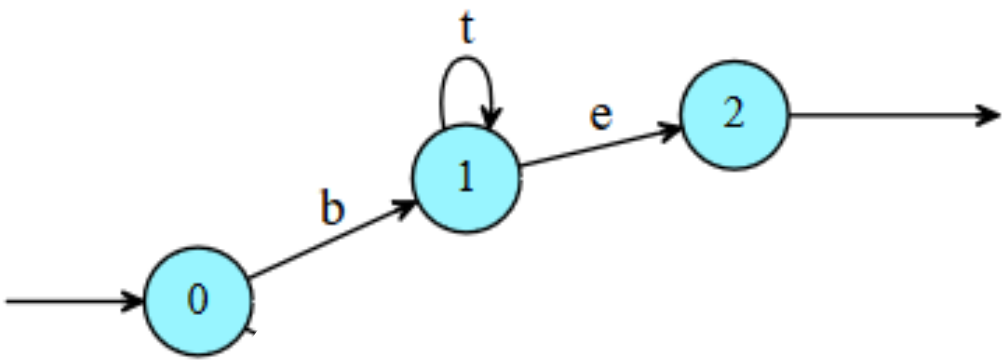
(question type)

Protocole de communication

- le protocole implique un émetteur et un récepteur.
- Les messages (les symboles de Σ) sont :
  - b : début de communication (begin)
  - t : trame de données
  - e : fin des trames (end)
  - c : code correcteur
  - s : somme de contrôle
- Le protocole permet de mélanger
  - des conversations simplifiée: $b\ t\ t\ \ldots\ t\ e$
  - et des conversations complètes: $b\ t\ t\ c\ \ldots t\ t\ c\ e\ s$

courtesy D. Le Botlan

Protocole de communication

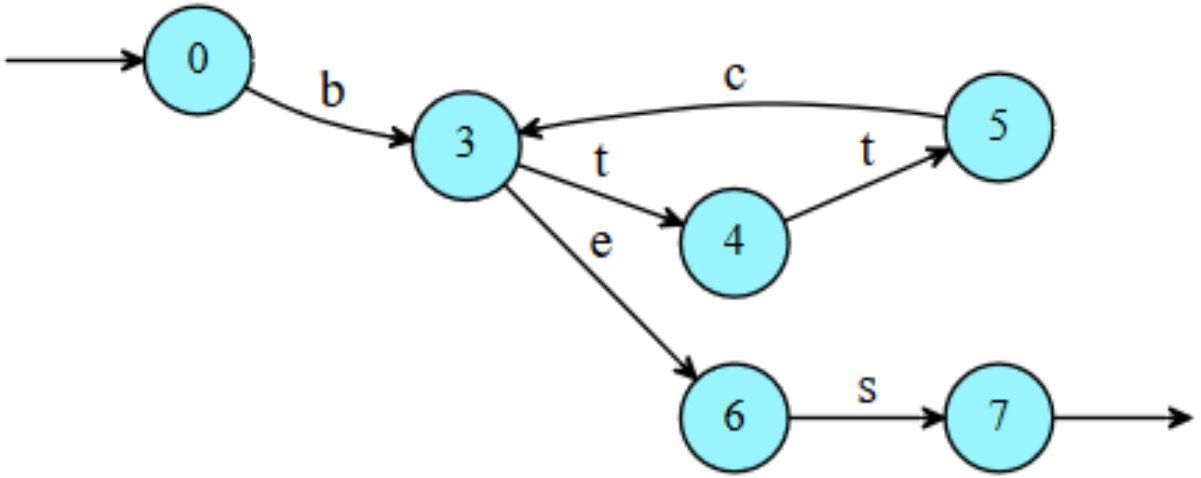


b : début de communication (begin)
t : trame de données
e : fin des trames (end)
c : code correcteur
s : somme de contrôle

conversations simplifiée: $b\ t\ t\ \ldots\ t\ e$

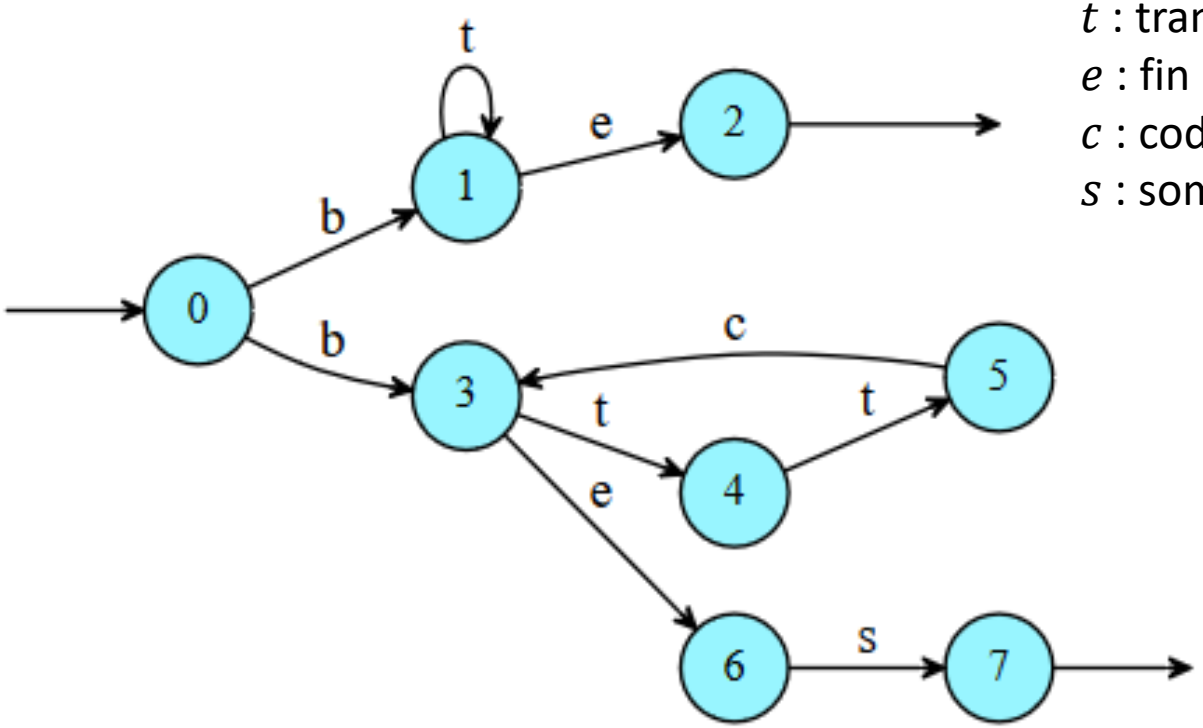
Protocole de communication

b : début de communication (begin)
t : trame de données
e : fin des trames (end)
c : code correcteur
s : somme de contrôle



conversations complètes: $b\ t\ t\ c\ \ldots t\ t\ c\ e\ s$

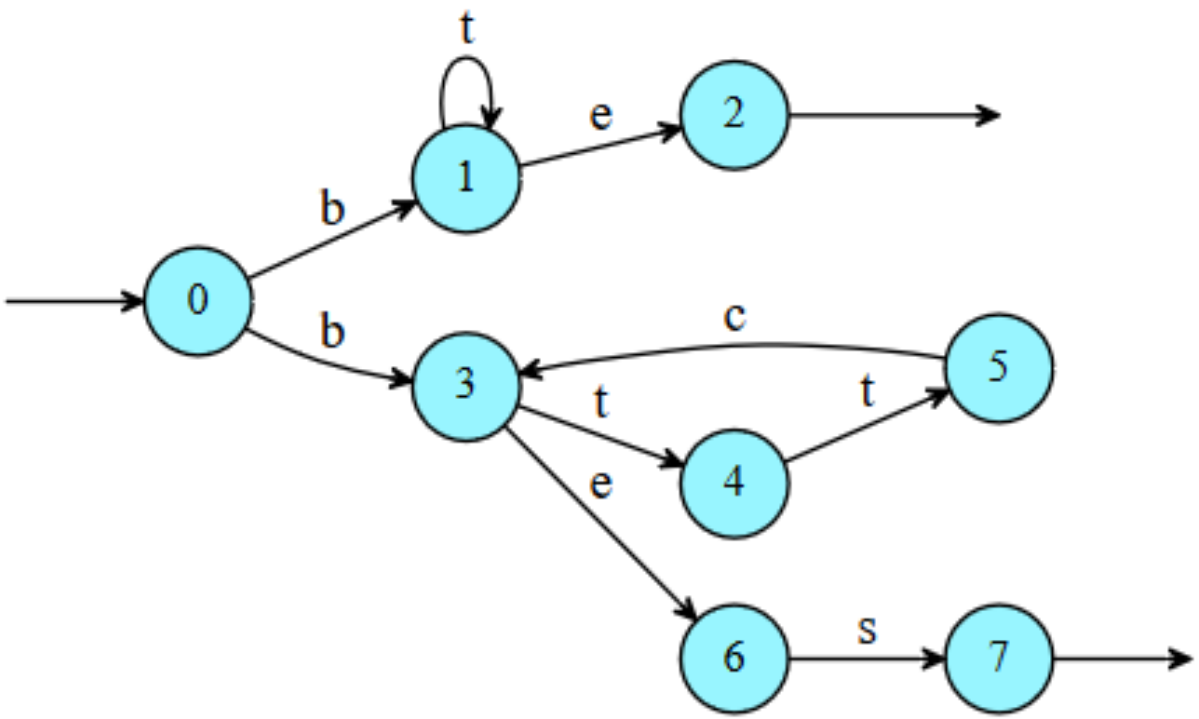
Protocole de communication



b : début de communication (begin)
t : trame de données
e : fin des trames (end)
c : code correcteur
s : somme de contrôle

conversations simplifiée: $b\ t\ t\ \ldots\ t\ e$
conversations complètes: $b\ t\ t\ c\ \ldots t\ t\ c\ e\ s$

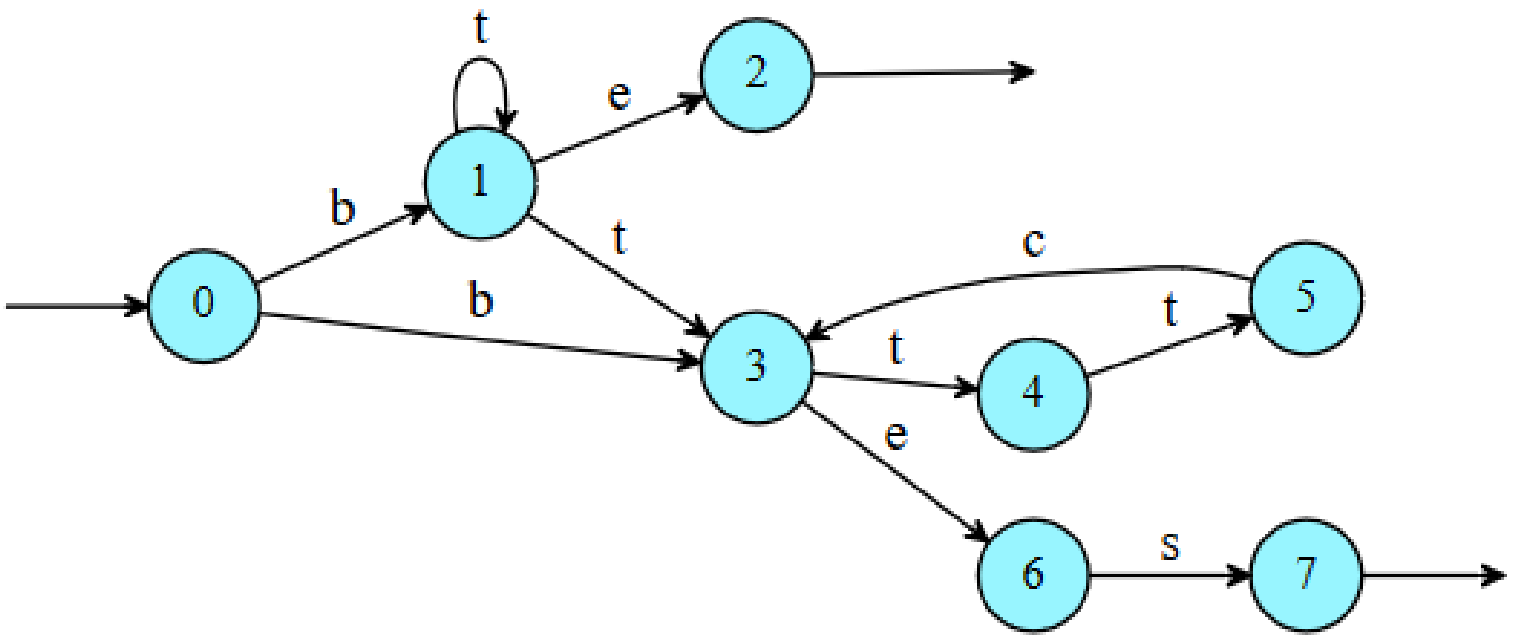
Protocole de communication



J'oubliais, on peut passer de *simple* à *complexe* en cours d'execution !

$b\ t\ t\ t\ t\ t\ c\ e\ s$

Protocole de communication



J'oubliais, on peut passer de *simple* à *complexe* en cours d'execution !

Déterminiser un NFA

ϵ-fermeture + NFA $\Rightarrow$ DFA

On prouve que undét. $\subseteq$ dét.

Théorèmes

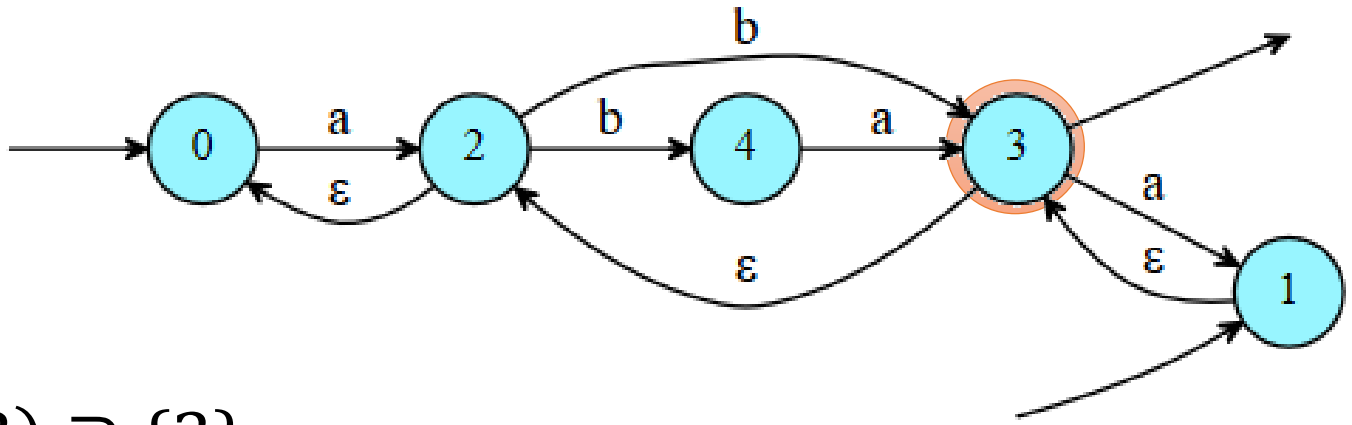
1. Pour tout automate $\mathcal{A}$, il existe un automate sans ϵ-transitions qui accepte le même langage.

2. Pour tout automate (NFA) $\mathcal{A}$, il existe un DFA qui accepte le même langage.

ϵ-fermeture

Pour un automate, on peut calculer l'ϵ-fermeture d'un état, q, c'est-à-dire l'ensemble des états accessible en suivant les ϵ-trans.

$\epsilon\mathrm{F}(3)$ = états accessible instantanément depuis 3

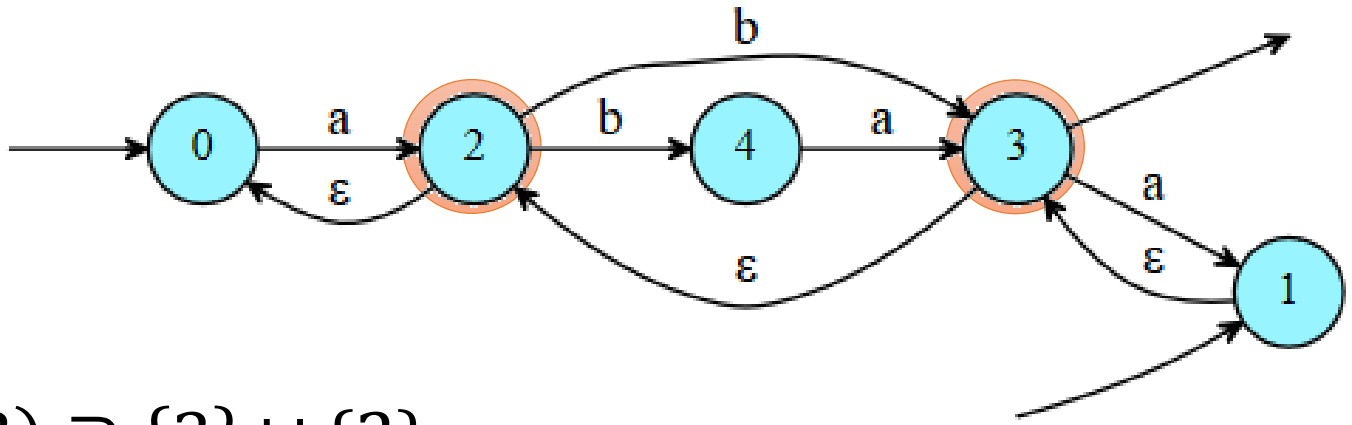


$\epsilon\mathrm{F}(3) \supseteq \{3\}$

ϵ-fermeture

Pour un automate, on peut calculer l'ϵ-fermeture d'un état, q, c'est-à-dire l'ensemble des états accessible en suivant les ϵ-trans.

$\epsilon\mathrm{F}(3)$ = états accessible instantanément depuis 3

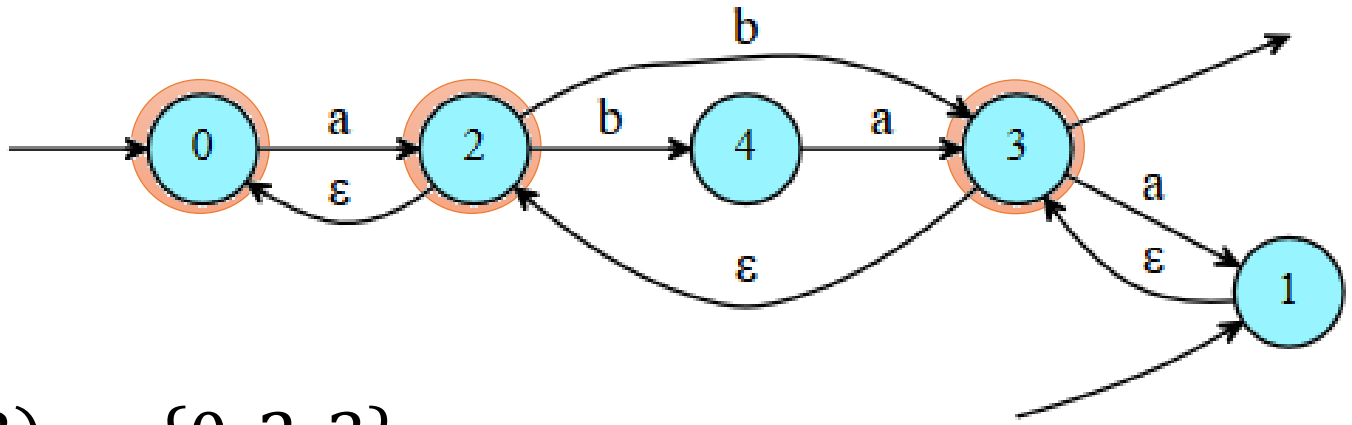


$\epsilon\mathrm{F}(3) \supseteq \{3\} \cup \{2\}$

ϵ-fermeture

Pour un automate, on peut calculer l'ϵ-fermeture d'un état, q, c'est-à-dire l'ensemble des états accessible en suivant les ϵ-trans.

$\epsilon\mathrm{F}(3)$ = états accessible instantanément depuis 3



$\epsilon\mathrm{F}(3) = \{0, 2, 3\}$

ϵ-fermeture

- Pour un automate, on peut calculer l'ϵ-fermeture d'un état, q, c'est-à-dire l'ensemble des états accessible en suivant les ϵ-trans.
- c'est le plus petit ensemble $\epsilon\mathrm{F}(q)$ tel que:

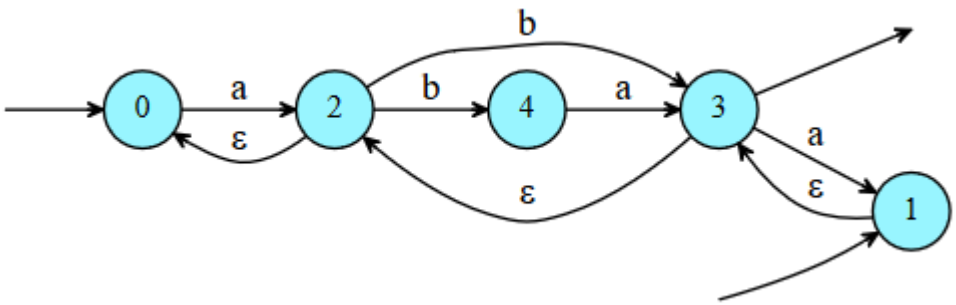
$$\epsilon\mathrm{F}(q) \supseteq \{ q \}$$
$$\epsilon\mathrm{F}(q) \supseteq \{ q'' \mid \exists q' \in \epsilon\mathrm{F}(\mathrm{q}),\ \delta(q', \epsilon) = q'' \}$$

- la definition se généralise à $\epsilon\mathrm{F}(S)$, pour $S \subseteq Q$

ϵ-fermeture

peut créer un automates avec plus de transitions, mais autant d'états

$$\epsilon\mathrm{F}(q) \supseteq \{ q \}$$
$$\epsilon\mathrm{F}(q) \supseteq \{ q'' \mid \exists q' \in \epsilon\mathrm{Fq},\ \delta(q', \epsilon) = q'' \}$$



$$\epsilon\mathrm{F}(A) = \{ A \}$$
$$\epsilon\mathrm{F}(B) = \{ B \} \cup \epsilon\mathrm{F}(D)$$
$$\epsilon\mathrm{F}(C) = \{ C \} \cup \epsilon\mathrm{F}(A)$$
$$\epsilon\mathrm{F}(D) = \{ D \} \cup \epsilon\mathrm{F}(C)$$
$$\epsilon\mathrm{F}(E) = \{ E \}$$

$$\epsilon\mathrm{F}(\{A, B\}) = \{ A, B \} \cup \{ D \} \cup \{ \mathrm{C} \}$$

Déterminisation

Idée: pour calculer le *run* dans un NFA on peut utiliser “plusieures têtes” (au plus $n = |Q|$)

- il suffit de se rappeler, à chaque instant, de l’ensemble des états sur lesquels on pointe: $S \subseteq Q$
- il y a un nombre fini de configuration possible $\leq 2^n$
- on peut calculer “statiquement” la fonction de transition δ' sur ces “états quotients”

$$\delta'(S, a) = \bigcup_{q \in S, q' \in \epsilon F(q)} \delta(q', a)$$

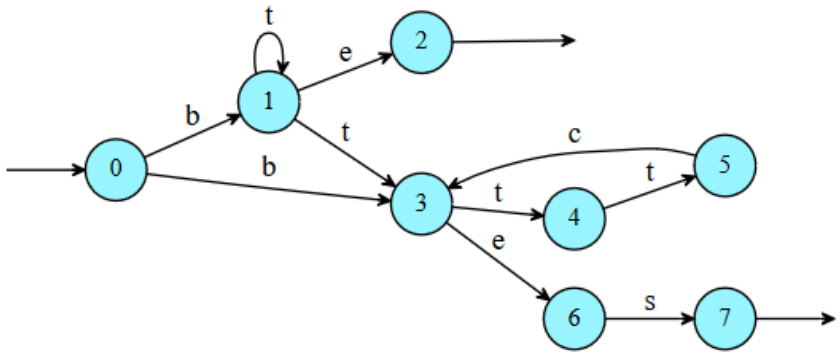
Déterminisation

- le successeur de S est l'ensemble S' qui contient les états q' tels que $q \in S$ et $(q, a) \Rightarrow q'$ (en suivant les ϵ-transitions)

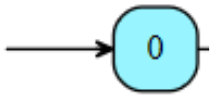
$$\delta'(S, a) = \bigcup_{q \in S, q' \in \epsilon F(q)} \delta(q', a)$$

- l'état quotient S est final si $S \cap F \neq \emptyset$
- l'état initial est I (plus précisément $\epsilon F(I)$)

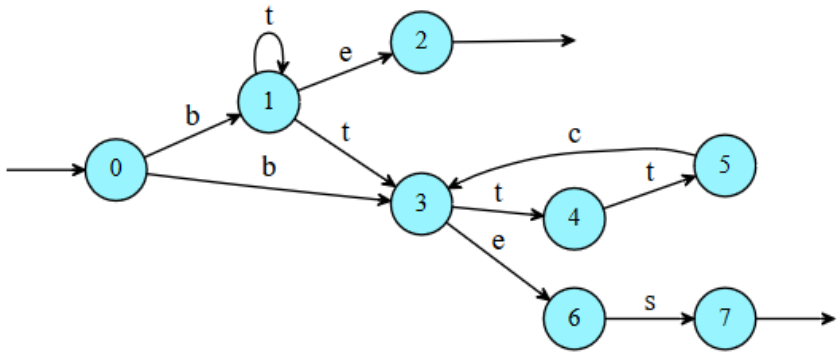
Déterminisation



$\epsilon F(I)$ = {0} $\delta(0, a) = \emptyset$ $\delta(0, b)$ = {1,3}

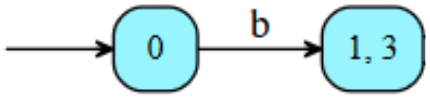


Déterminisation

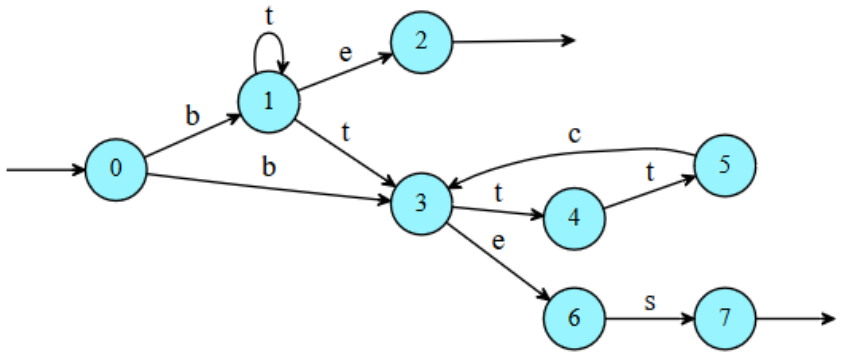


$\delta(1, t) = \{1,3\}$ $\delta(3, t) = \{4\}$

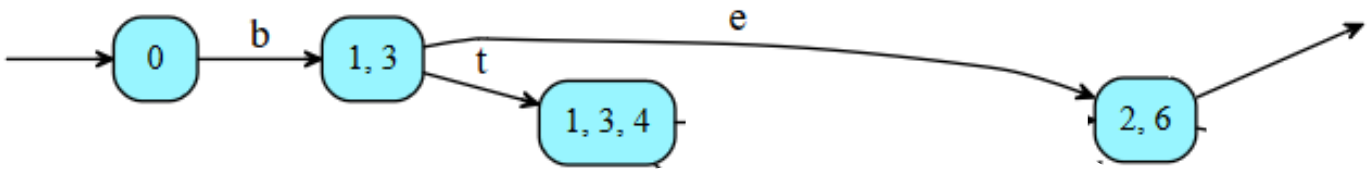
$\delta(1, e) = \{2\}$ $\delta(3, e) = \{6\}$ $6 \in F$



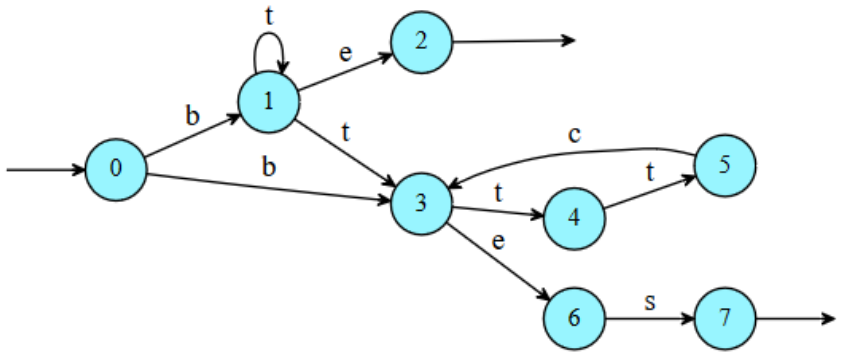
Déterminisation

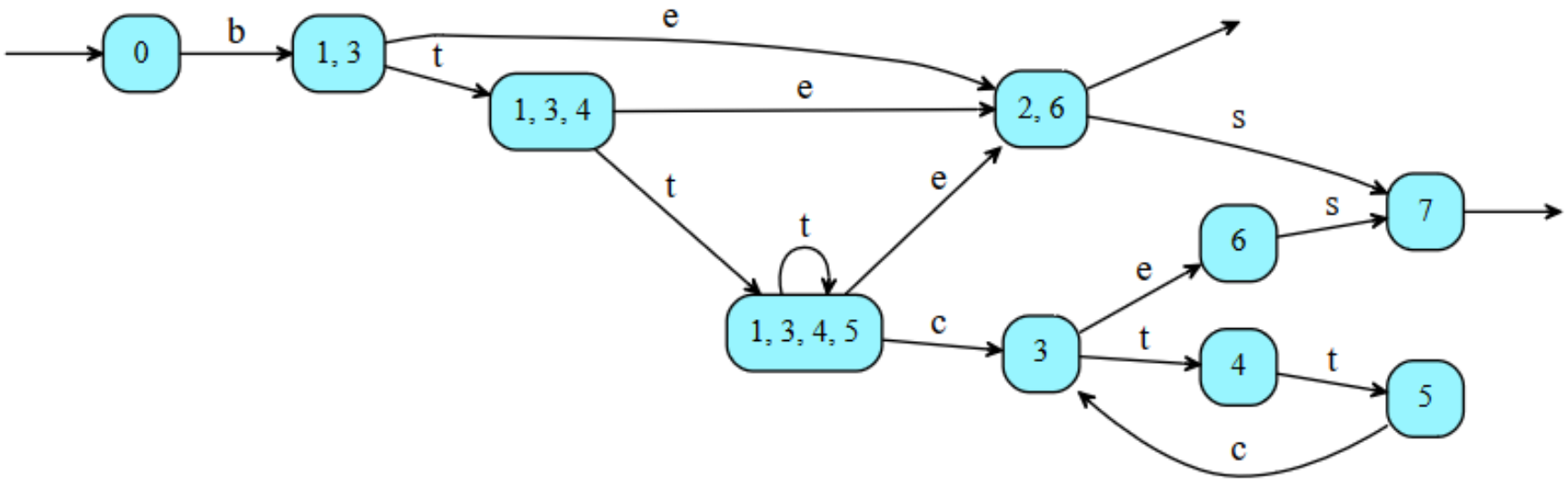


$\delta(1,t) = \{1,3\}$ $\delta(3,t) = \{4\}$ $\delta(4,t) = \{5\}$ $\delta(5,t) = \emptyset$

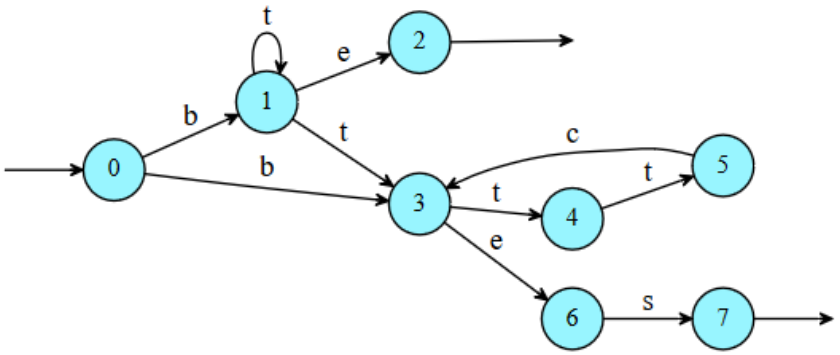


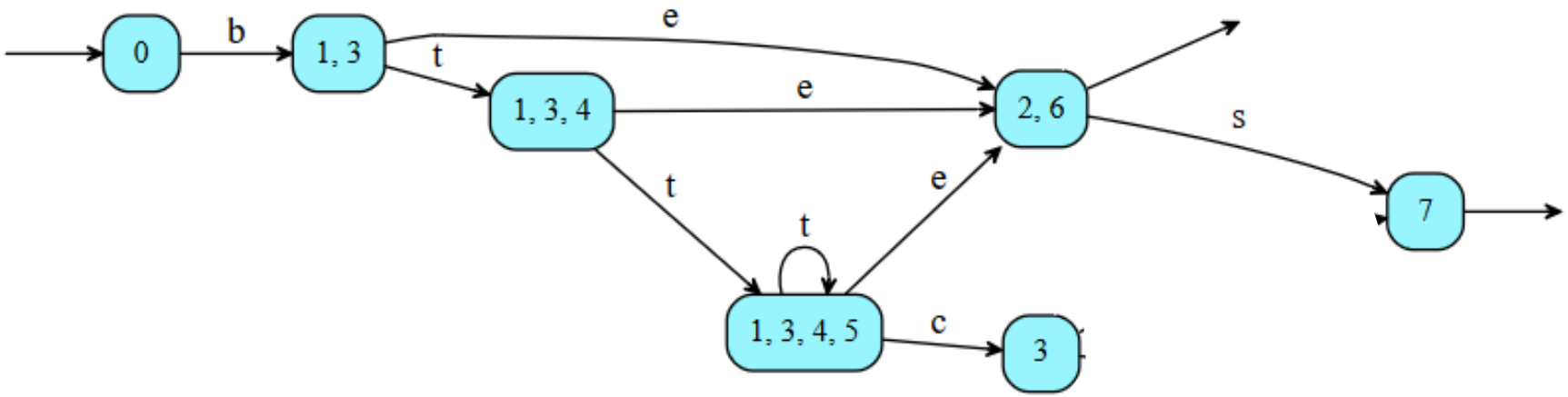
Déterminisation



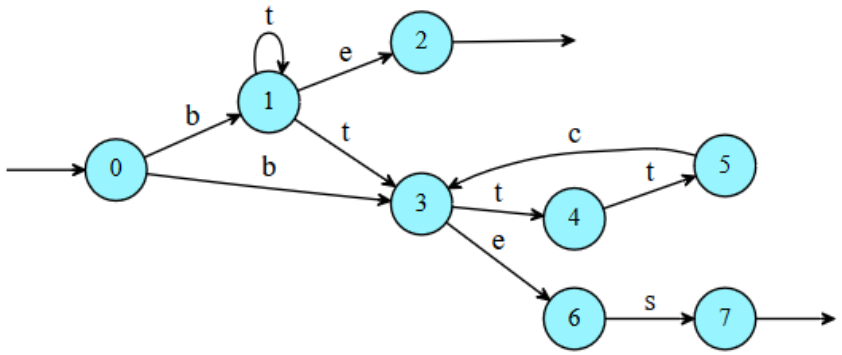


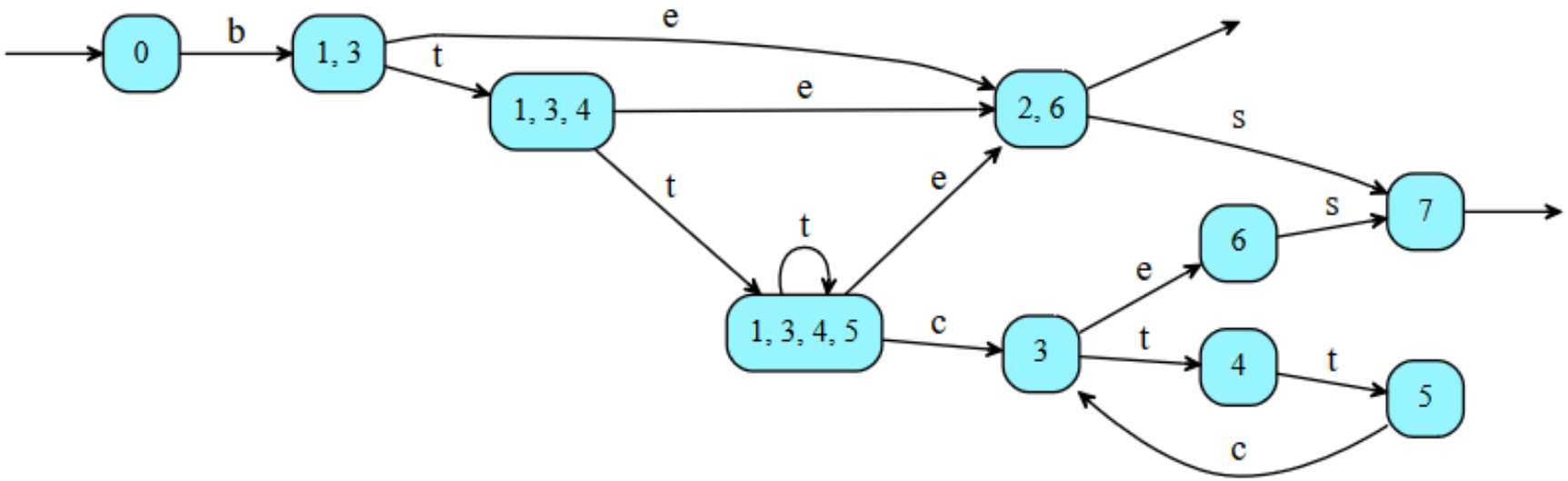
Déterminisation



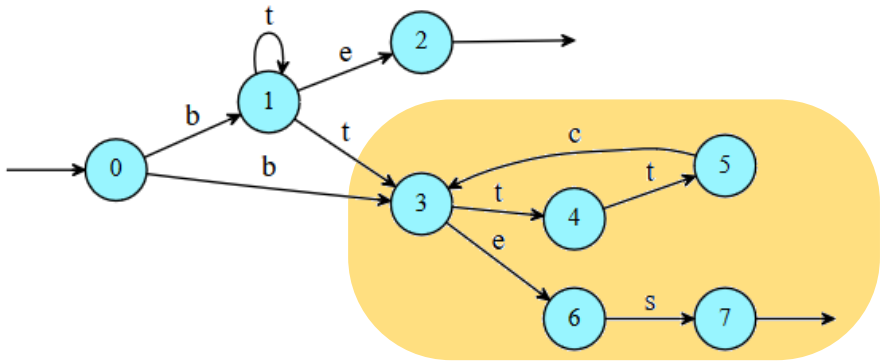


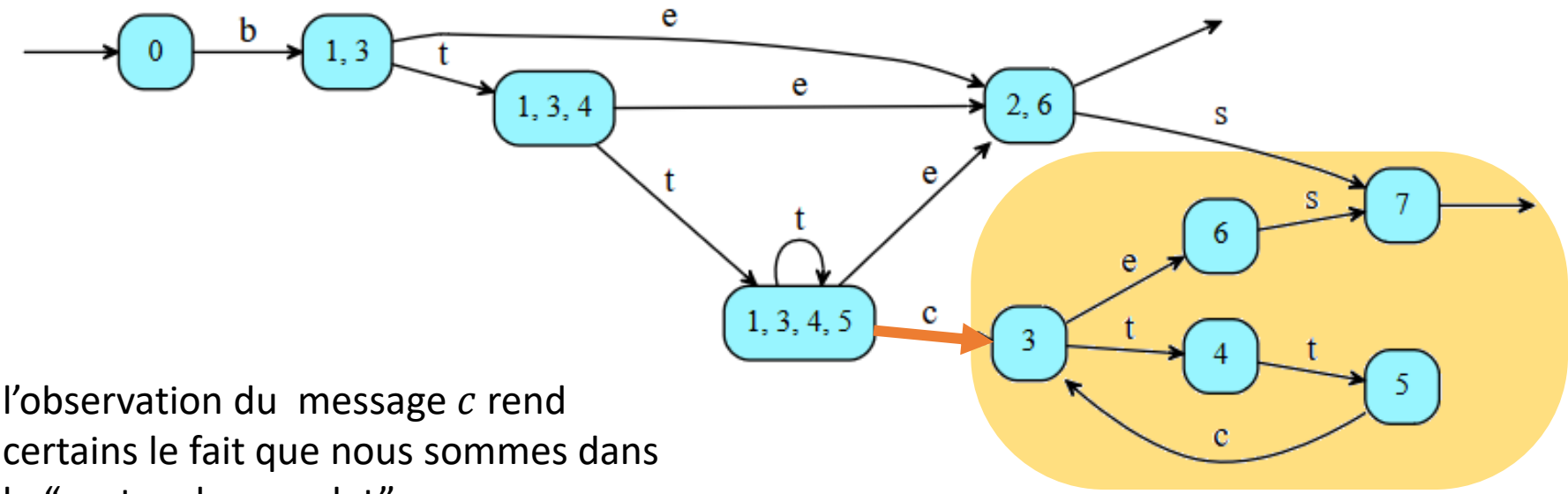
Déterminisation





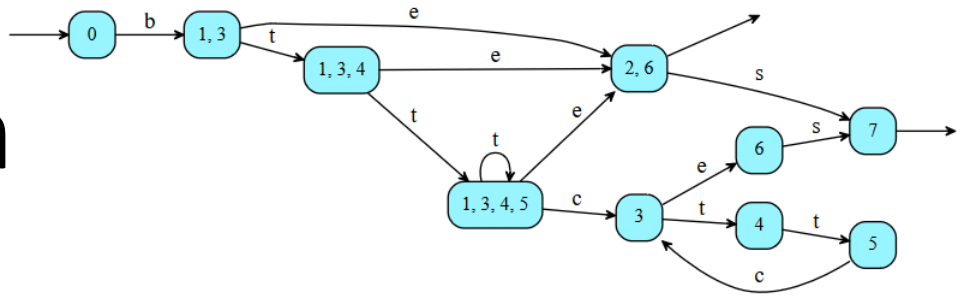
Déterminisation





l'observation du message *c* rend certains le fait que nous sommes dans le "protocole complet"

Déterminisation



$\det(\mathcal{A})$ est un tuple $(Q',\ \Sigma,\ \delta',\ q'_I, F')$ où :

- $Q' = 2^Q = \mathcal{P}(Q)$ (powerset)
- $\Sigma =$ même alphabet que $\mathcal{A}$
- $q'_I = \epsilon F(I)$
- $F' = \{ S \mid S \cap F \neq \emptyset \}$
- $\delta' \in (Q' \times \Sigma) \rightarrow Q'$: fonction de transition

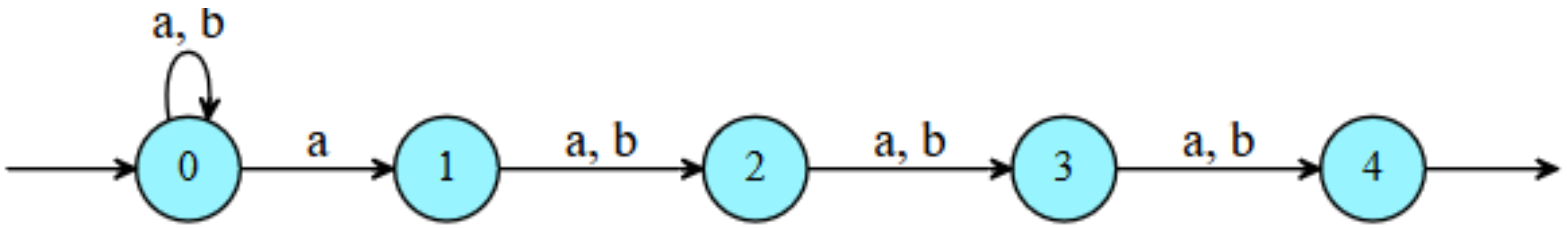
⚠ $\det(\mathcal{A})$ est complet, car ∅ est un état valide et c'est un état puit.

⚠ tel quel $\det(\mathcal{A})$ peut contenir des états non accessibles (inutiles);
≠ d'une construction à la volée

Explosion du nombre d'états

Si l'automate $\mathcal{A}$ *à* n états, l'automate $\det(\mathcal{A})$ a potentiellement 2^n états.

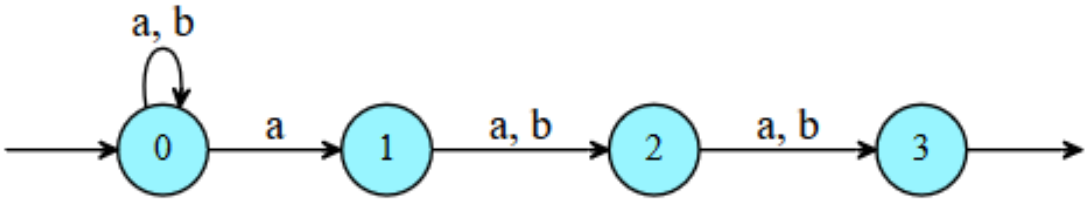
La famille d'automates *De Bruijn(n)* fournie un exemple avec ≈ le pire cas en complexité (i.e. $\det(\mathcal{A})$ *a* 2^{n-1} états)



accepte les mots de la forme $\{a, b\}^\star\ a\ \{a, b\}^3$

Explosion du nombre d'états

Si l'automate $\mathcal{A}$ *à* n états, l'automate $\det(\mathcal{A})$ a potentiellement 2^n états



De Bruijn(2)