Automates et systèmes de transitions

Gérard Berry

Collège de France
Chaire Informatique et sciences numériques

Cours 4, 16 décembre 2009

Agenda

1. Systèmes d'états finis
2. Expressions régulières
3. Dérivées et traduction en automates
4. Algorithme de Berry-Sethi
5. Déterminisation et minimisation
6. Des automates aux circuits Booléens
7. Suites automatiques et nombres transcendants

Systèmes d'états finis

- Principes généraux :
  - le système n'a accès qu'à des ressources finies
  - il procède par suite de transitions élémentaires
- Formalismes multiples
  - machines : automates finis, circuits booléens
  - expressions régulières (shell, lex, perl, etc).
  - langages de haut niveau : automates hiérarchiques (Statecharts), langages synchrones (Esterel, SyncCharts, Lustre, Signal)
  - sémantique : langages rationnels

Champs d'applications

- Analyse de langages
  - langues naturelles
  - langages de programmation
- Circuits électroniques
  - chemins de données / chemins de contrôle
  - gestion mémoire, gestion des caches et de la cohérence
  - réseaux sur puce, USB, SATA, etc.
- Protocoles de communication
  - établissement et maintien des liens (Link Layer)
  - gestion des erreurs et retransmissions
- Automatismes industriels
  - automates programmables

Champs d'applications

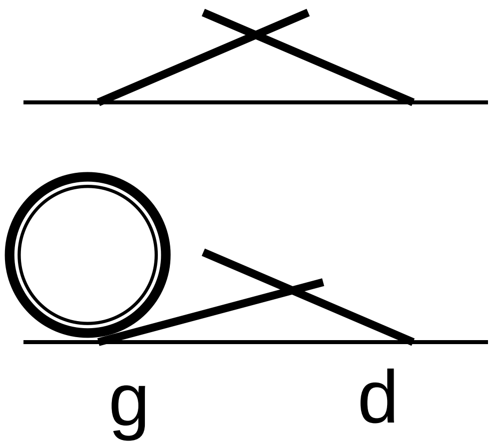
- Systèmes embarqués
  - téléphones
  - ABS, climatisation, contrôle train d'atterrissage
  - signalisation ferroviaire, aiguillages, etc.
  - contrôle de tâches, de robots, etc.

- Interfaces homme-machine
  - interacteurs d'entrée (menus, joysticks d'avions, etc.)
  - gestion d'écrans (cockpits d'avions)

- Mathématiques pures : théorie des nombres
  - suites automatiques et nombres transcendants

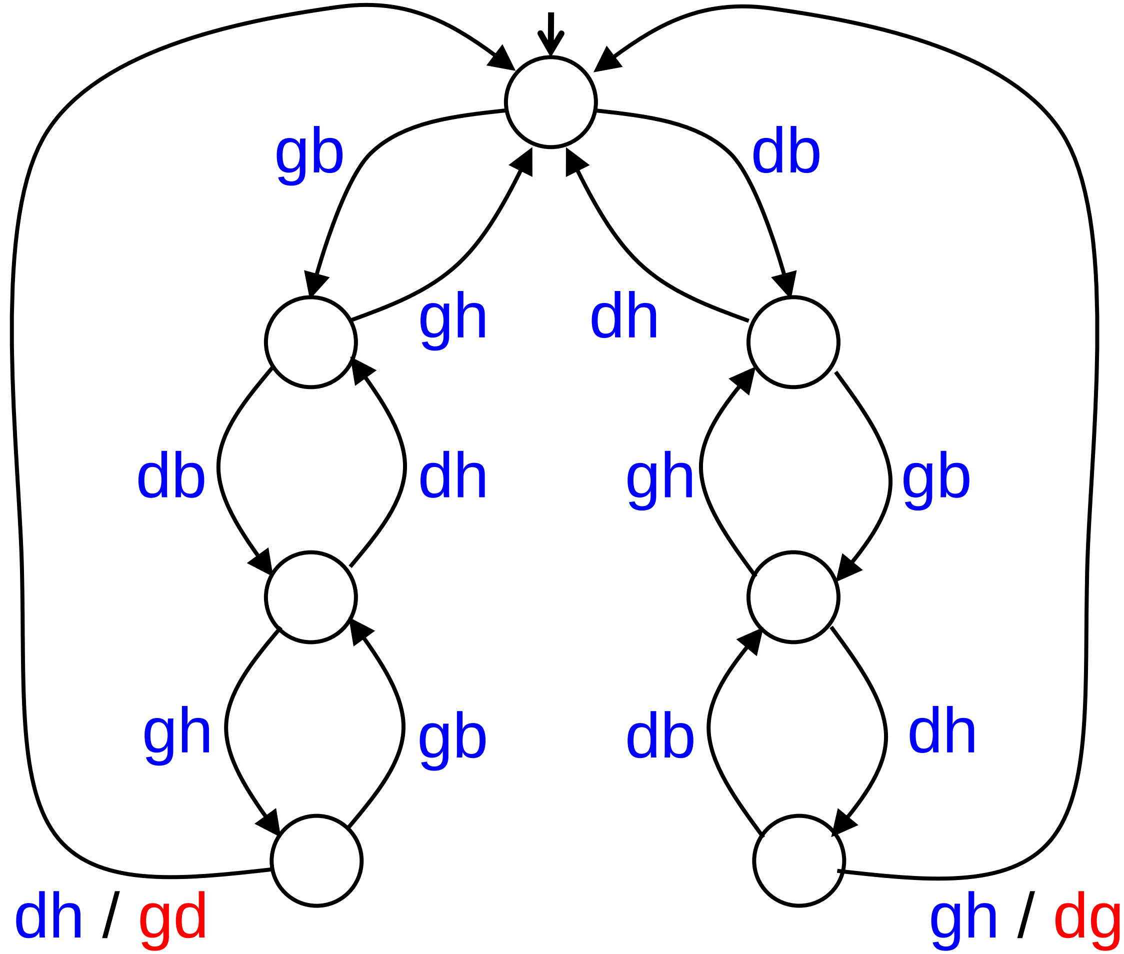
Automates : machines explicites

compteur d'essieus

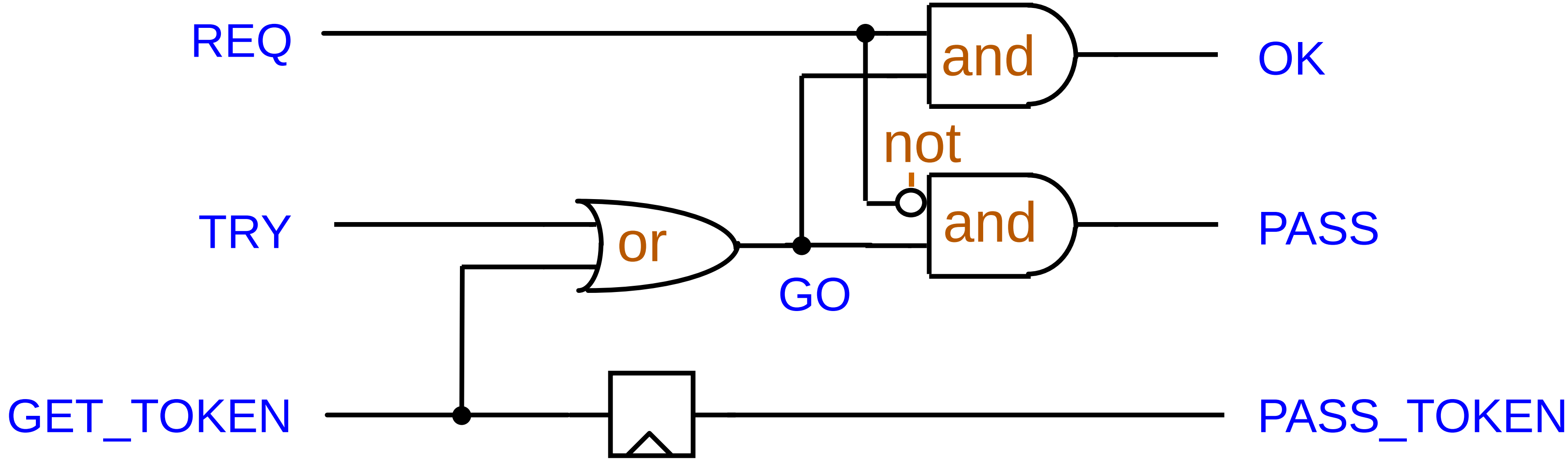


- entrées :
  gb, gh, db, dh
- sorties :

→ gd, ← dg



Circuits booléens



OK = REQ and GO

PASS = not REQ and GO

GO = TRY or GET_TOKEN

PASS_TOKEN = reg(GET_TOKEN)

Agenda

1. Systèmes d’états finis
2. Expressions régulières
3. Dérivées et traduction en automates
4. Algorithme de Berry-Sethi
5. Déterminisation et minimisation
6. Des automates aux circuits Booléens
7. Suites automatiques et nombres transcendants

Langages rationnels

- Alphabet : ensemble (fini) de lettres a, b, c, x, y...
- Mot : suite finie u, v, w de lettres ; ε = mot vide
- Concaténer deux mots : les mettre bout à bout

  $ab \cdot cd = abcd$

- Langage L : ensemble de mots
- Langages rationnels : la plus petite famille de langages contenant les langages finis et fermée par union et concaténation

Expression régulières

- 0 $\quad L(0) = \varnothing$
- 1 $\quad L(1) = \{\varepsilon\}$
- a $\quad L(a) = \{a\}$
- $e+e'$ $\quad L(e+e') = L(e) \cup L(e')$
- $e \cdot e'$ ou $e\,e'$ $\quad L(e \cdot e') = \{uu' \mid u \in L(e),\ u' \in L(e')\}$
- e^* $\quad L(e^*) = \{u_0 u_1 \ldots u_n \mid n \geq 0,\ u_i \in L(e)\}$

$$e+e' = e'+e$$
$$e+(e'+e'') = (e+e')+e''$$
$$e \cdot (e'+e'') = e \cdot e' + e \cdot e''$$
$$(e+e') \cdot e'' = e \cdot e'' + e' \cdot e''$$

$$0+e = e+0 = e$$
$$0 \cdot e = e \cdot 0 = 0$$
$$1 \cdot e = e \cdot 1 = 1$$

Expression test : (ab+b)* ba

ba, abba, bba, abbabba, bbbbababbbbba,...

Test du mot vide

- $\varepsilon(L) = 1$ si $\varepsilon \in L$

  $= 0$ sinon

- $\varepsilon(0) = 0$
- $\varepsilon(1) = 1$
- $\varepsilon(a) = 0$
- $\varepsilon(e+e') = \varepsilon(e) + \varepsilon(e')$
- $\varepsilon(e \cdot e') = \varepsilon(e) \cdot \varepsilon(e')$
- $\varepsilon(e^*) = 1$

Agenda

1. Systèmes d'états finis
2. Expressions régulières
3. Dérivées et traduction en automates
4. Algorithme de Berry-Sethi
5. Déterminisation
6. Des automates aux circuits Booléens
7. Suites automatiques et nombres transcendants

Dérivées d'expressions régulières

- $u^{-1}(L) = \{ v \mid uv \in L \}$ $\qquad ua^{-1}(L) = a^{-1}(u^{-1}(L))$

  ce qui reste à écrire quand on a déjà écrit u
- $a^{-1}(e)$: expression régulière engendrant $a^{-1}(L(e))$

- $a^{-1}(0) = 0$
- $a^{-1}(1) = 0$
- $a^{-1}(a) = 1$
- $a^{-1}(b) = 0$ si $b \neq a$
- $a^{-1}(e+e') = a^{-1}(e) + a^{-1}(e')$
- $a^{-1}(e \cdot e') = a^{-1}(e) \cdot e' + \varepsilon(e) \cdot a^{-1}(e')$
- $a^{-1}(e^*) = a^{-1}(e) \cdot e^*$

Expression test : $(ab+b)^*\,ba$

ba, abba, bba, abbabba, bbbbababbbbba,...

- $a^{-1}((ab+b)^*ba) = \underline{a^{-1}((ab+b)^*)} \cdot ba$

  $+\ \varepsilon((ab+b)^*) \cdot \underline{a^{-1}(ba)}$

  $= \underline{a^{-1}(ab+b)} \cdot (ab+b)^* \cdot ba + 0$

  $= b \cdot (ab+b)^* \cdot ba$

- $b^{-1}((ab+b)^*ba) = \underline{b^{-1}((ab+b)^*)} \cdot ba$

  $+\ \varepsilon((ab+b)^*) \cdot \underline{b^{-1}(ba)}$

  $= \underline{b^{-1}(ab+b)} \cdot (ab+b)^* \cdot ba$

  $+\ a$

  $= (ab+b)^*ba + a$

Des dérivées aux automates

Théorème (Brzozowski) : une expression régulière e n'a qu'un nombre fini de dérivées (simplifiées) distictes $u^{-1}(e)$

Corollaire : on construit un automate déterministe reconnaissant L(e) en prenant pour états les dérivées distinctes, avec une flèche étiquetée a de $u^{-1}(e)$ à $ua^{-1}(e)$ si les deux sont non vides; un état e' est final si $\boldsymbol{\varepsilon}(e') = 1$.

Itération jusqu'à plus soif

- $e = e_0 = ((ab+b)^* ba)$

- $a^{-1}(e_0) = b(ab+b)^* ba = e_1$
- $b^{-1}(e_0) = (ab+b)^* ba + a = e_2$

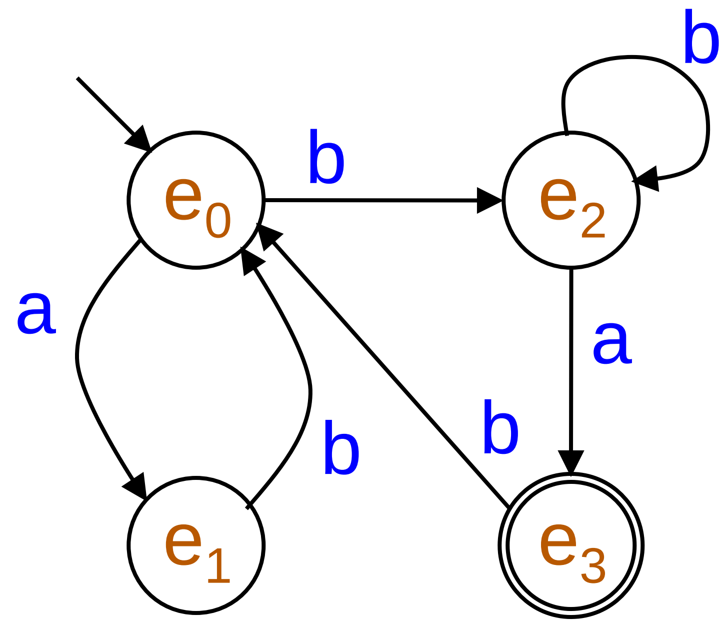
- $a^{-1}(e_1) = 0$
- $b^{-1}(e_1) = (ab+b)^* ba = e_0$

- $a^{-1}(e_2) = b(ab+b)^* ba + 1 = e_3$
- $b^{-1}(e_2) = (ab+b)^* ba + a = e_2$

- $a^{-1}(e_3) = 0$
- $b^{-1}(e_3) = (ab+b)^* ba = e_0$

Construction d'un automate déterministe

- $a^{-1}(e_0) = e_1$
- $b^{-1}(e_0) = e_2$
- $a^{-1}(e_1) = 0$
- $b^{-1}(e_1) = e_0$
- $a^{-1}(e_2) = e_3$
- $b^{-1}(e_2) = e_2$
- $a^{-1}(e_3) = 0$
- $b^{-1}(e_3) = e_0$
- $\boldsymbol{\varepsilon}(e_3) = 1$



automate déterministe
(Brzozoswki)

Autres résultats

Corollaire : la classe des langages rationnels est fermée par complémentation et intersection

preuve : négation par inversion des états terminaux et non-terminaux, intersection puisque union et négation

Remarque : la dérivation marche pareil en incluant la négation et l'intersection dans les expressions :

$$a^{-1}(\neg e) = \neg a^{-1}(e) \qquad a^{-1}(e \cap e') = a^{-1}(e) \cap a^{-1}(e')$$

Théorème réciproque : tout langage reconnu par un automate est rationnel

preuve : par construction itérative d'une expression régulière (pas si facile – essayer le compteur d'essieus !)

Danger d'explosion !

- L'automate ainsi construit peut être exponentiellement plus grand que e

- Pour certaines expressions régulières, c'est vrai pour tout automate déterministe équivalent

- Pour certains automates, les expressions régulières équivalentes peuvent être toutes exponentiellement plus grandes

Agenda

1. Systèmes d'états finis
2. Expressions régulières
3. Dérivées et traduction en automates
4. Algorithme de Berry-Sethi
5. Déterminisation et minimisation
6. Des automates aux circuits Booléens
7. Suites automatiques et nombres transcendants

Expression linéaires

- expression linéaire : chaque lettre n'apparaît qu'une seule fois
- linéarisation d'une expression : indexation des lettres

$$s_0 = (a_0 b_1 + b_2)^* b_3 a_4$$

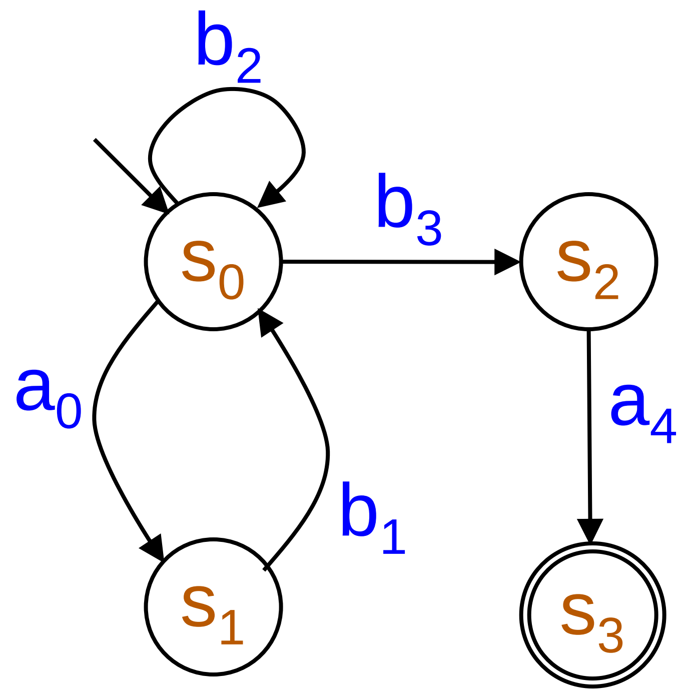
$$a_0^{-1}(s_0) = b_1(a_0 b_1 + b_2)^* b_3 a_4 = s_1$$

$$b_2^{-1}(s_0) = (a_0 b_1 + b_2)^* b_3 a_4 = s_0$$

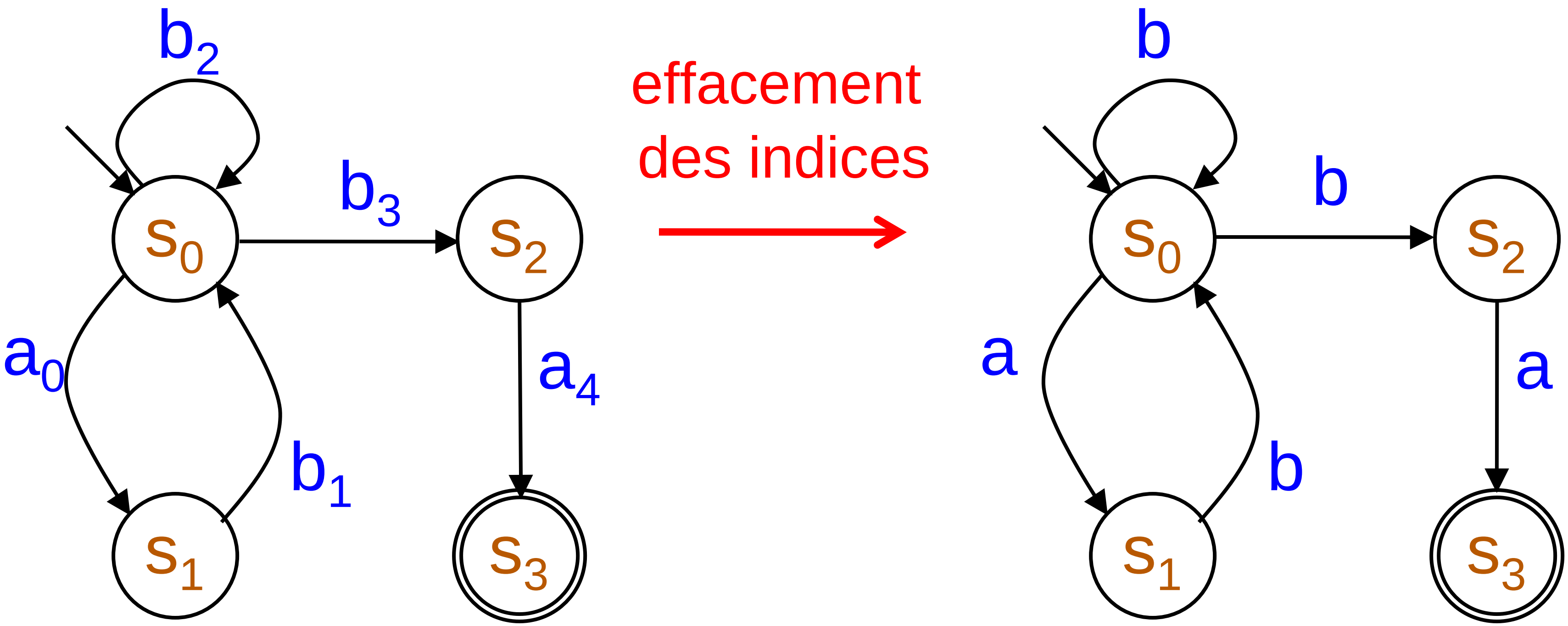
$$b_3^{-1}(s_0) = a_4 = s_2$$

$$b_1^{-1}(s_1) = s_0$$

$$a_4^{-1}(s_0) = \varepsilon = s_3$$



Automate non-déterministe



automate non-déterministe reconnaissant $L(e_0)$

Algorithme de Berry-Sethi

- <u>Théorème</u> : si e est une expression linéaire, alors une dérivée non nulle est caractérisée par sa dernière lettre x : toutes les dérivées de la forme $ux^{-1}(e)$ sont nulles ou égales

- <u>Preuve</u> : par récurrence sur $|e|$ (taille de e). Supposons $ux^{-1}(e)$ non nul.

1. évident si $e = x$ (dans ce cas $|u| = 0$)
2. si $e = e_1 + e_2$, alors x apparaît soit dans e_1, soit dans e_2 mais pas dans les deux. Si x apparaît dans e_1, alors

   $ux^{-1}(e) = ux^{-1}(e_1)$ avec $|e_1| < |e|$

1. si $e = e_1 \cdot e_2$, preuve similiaire en calculant $ux^{-1}(e_1 \cdot e_2)$ comme une somme de termes dont un seul peut contenir x
2. si $e = e_1{}^*$, preuve similaire de celle du cas 3

Algorithme de Berry-Sethi

<u>Construction de l'automate pour e linéaire</u> :

1. on construit un automate avec un état initial et un état par lettre x, qui représente $ux^{-1}(e)$ pour tout u

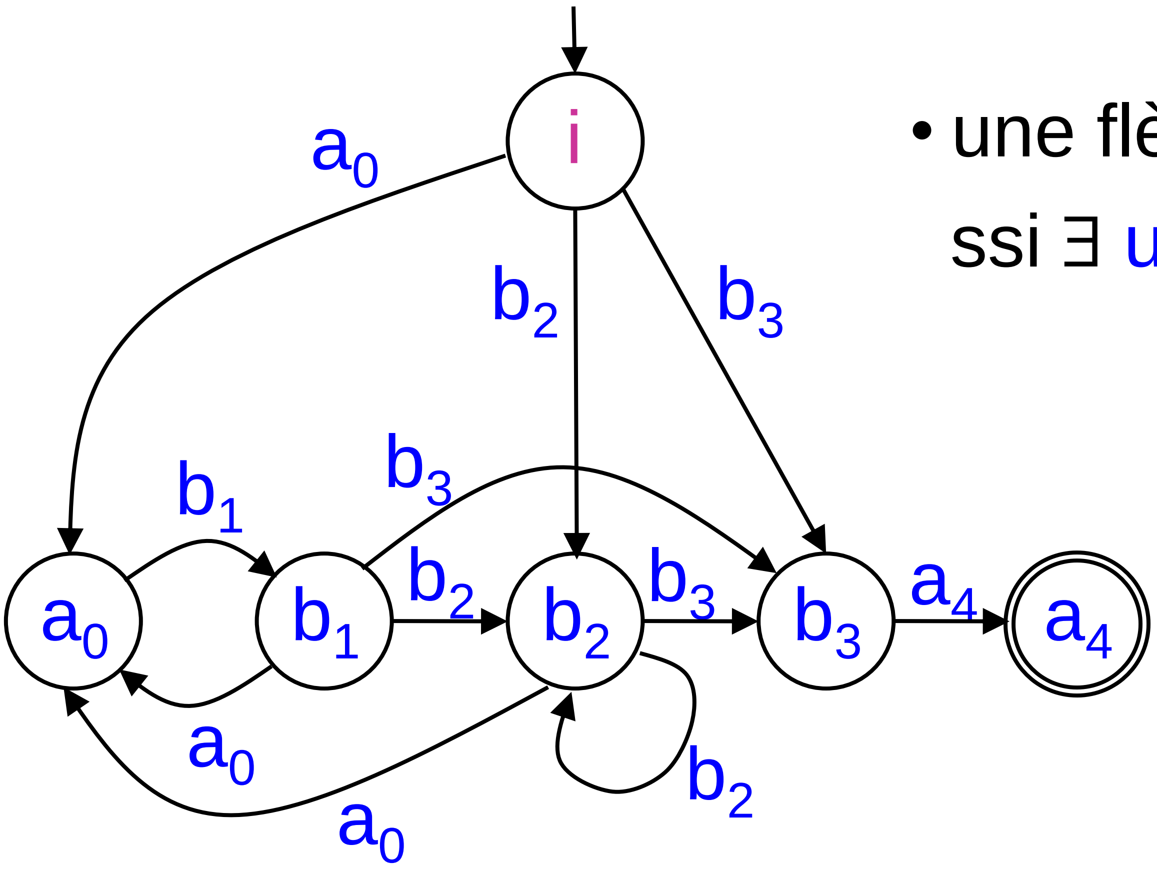
2. on va de l'état initial à l'état de toute lettre x qui peut apparaître en première position : $x^{-1}(e) \neq 0$

3. on va de l'état de x à l'état de y par une flèche y ssi
   $ux^{-1}(e) \neq 0$ et $uxy^{-1}(e) \neq 0$,
   i.e. s'il existe un mot $uxyv \in L(e)$,
   i.e. si y peut suivre x dans $L(e)$

4. l'état de x est terminal s'il existe un mot $ux \in L(e)$

Algorithme de Berry-Sethi

$s_0 = (a_0 b_1 + b_2)^* b_3 a_4$



- une flèche de i vers x
  ssi $\exists$ u = xu $\in$ L(s_0)

- une flèche y de x vers y
  ssi $\exists$ u xy v $\in$ L(s_0)

Calcul de first et last

- first (0) = first(1) = $\varnothing$
- first (a) = $\{a\}$
- first$(e+e')$ = first(e) ∪ first(e')
- first$(e\cdot e')$ = first(e) si $\varepsilon(e) = 0$

  = first(e) ∪ first(e') sinon
- first(e^*) = first(e)

premières lettres que e peut écrire

- last (0) = last(1) = $\varnothing$
- last (a) = $\{a\}$
- last$(e+e')$ = last(e) ∪ last(e')
- last$(e\cdot e')$ = last(e') si $\varepsilon(e')=0$

  = last(e) ∪ last(e') sinon
- last(e^*) = last(e)

dernières lettres que e peut écrire

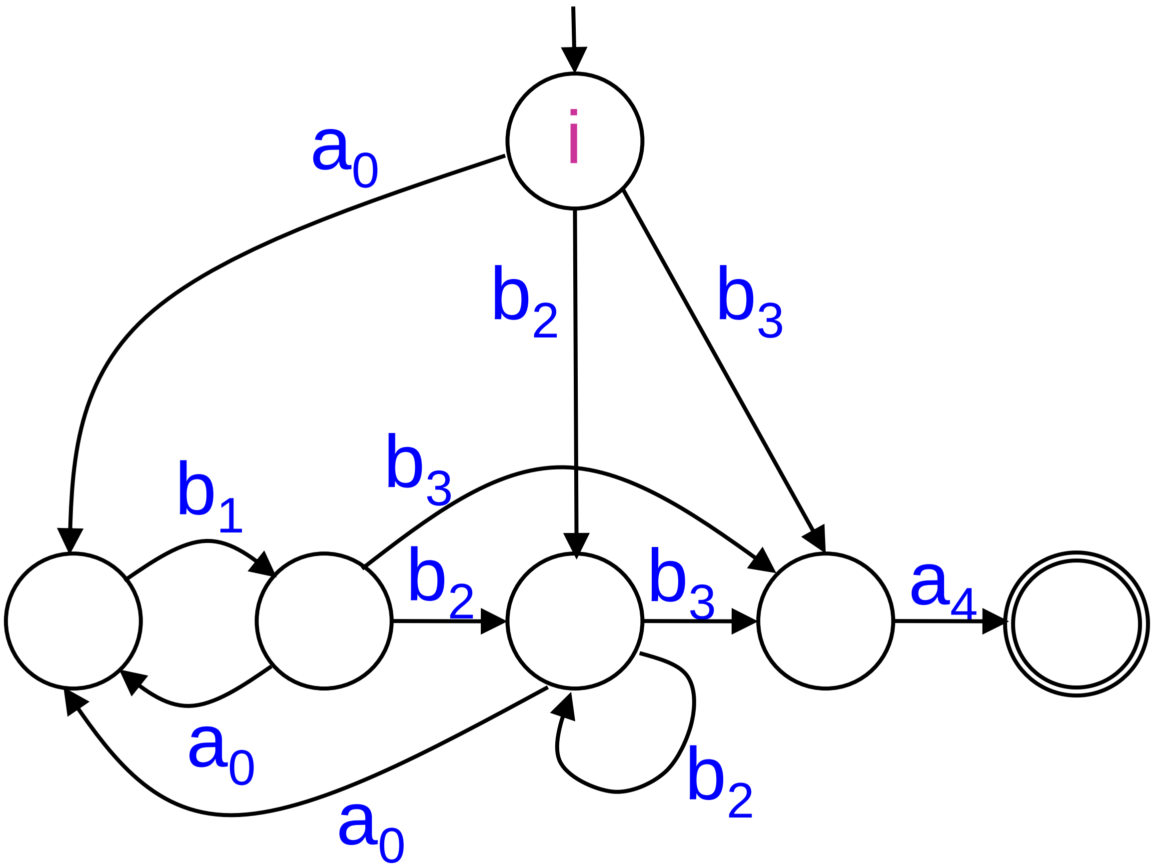
Calcul de y peut suivre x dans $L(e)$

- follow(e, x, y) = vrai si $\exists\, w x y w' \in L(e)$
- follow(0, x, y) = follow(1, x, y) = follow(a, x, y) = faux
- follow($e+e'$, x, y) = follow(e, x, y) $\vee$ follow(e', x, y)
- follow($e \cdot e'$, x, y) = follow(e, x, y)
  $\vee$ follow(e', x, y)
  $\vee$ ($x \in$ last(e) $\wedge$ $y \in$ first(e'))
- follow(e*, a) = follow(e, x, y)
  $\vee$ ($x \in$ last(e) $\wedge$ $y \in$ first(e'))

$\boxed{\text{Exercice : calculer plus efficacement !}}$

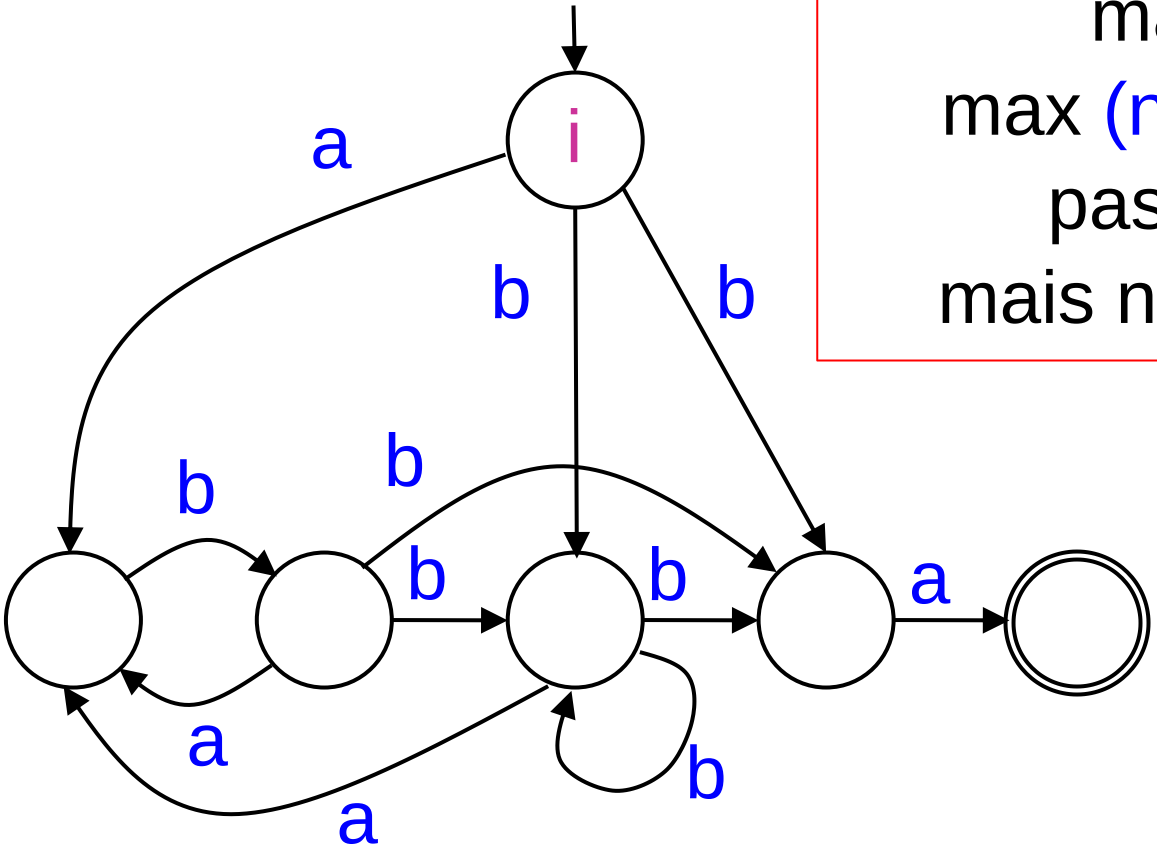
Algorithme de Berry-Sethi

$s_0 = (a_0 b_1 + b_2)^* b_3 a_4$



... et hop, on efface les indices !

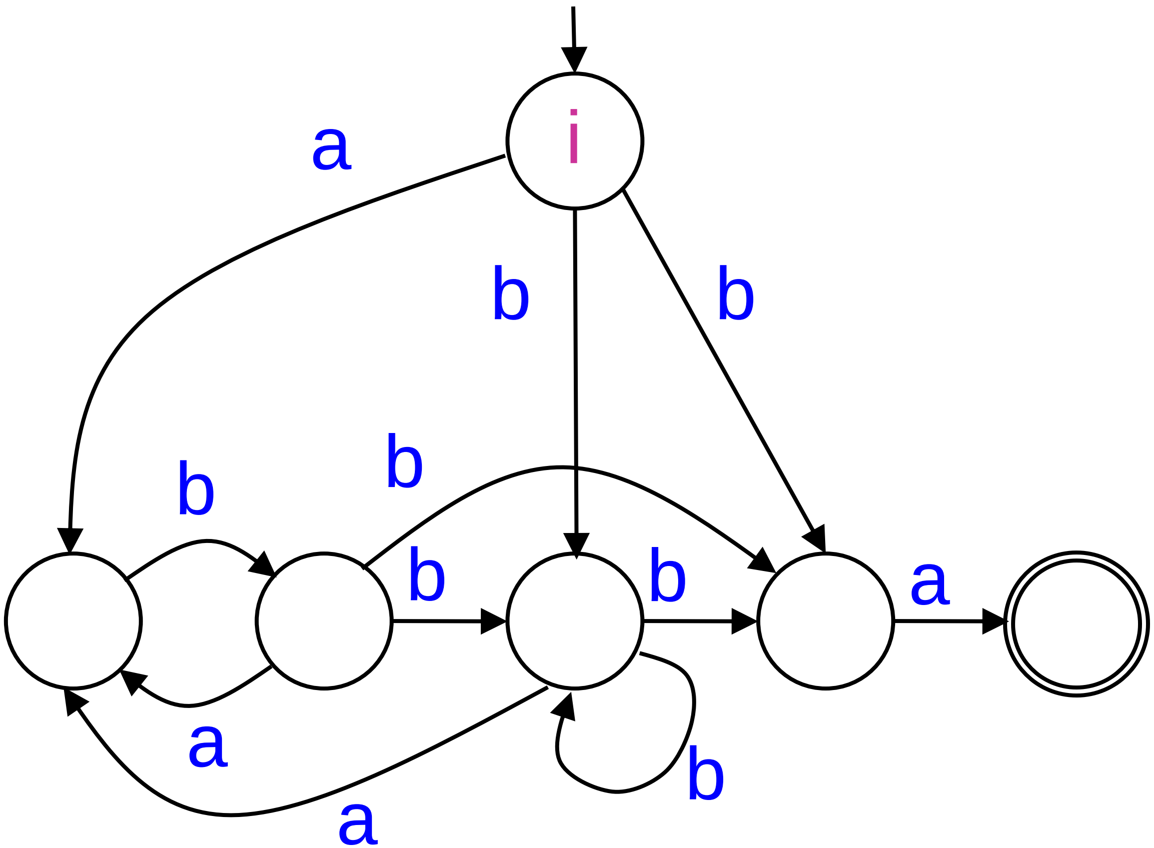
$e = (ab+b)^* ba$



pour n occurrences de lettres
max $n+1$ états
max $(n+1)^2$ transitions !
pas d'explosion !
mais non-déterministe..

... et on optimise l'état initial !

$e = (ab+b)^* ba$



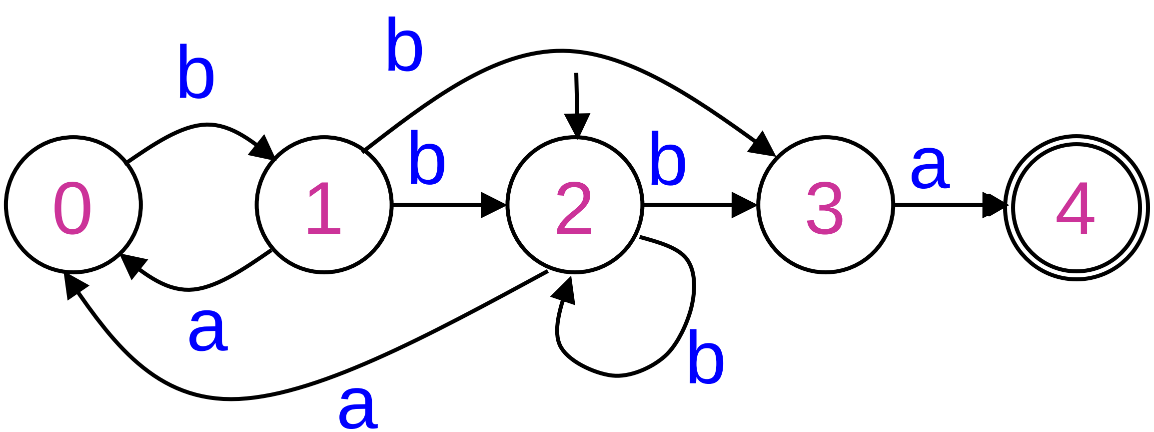
Agenda

1. Systèmes d'états finis
2. Expressions régulières
3. Dérivées et traduction en automates
4. Algorithme de Berry-Sethi
5. Déterminisation et minimisation
6. Des automates aux circuits Booléens
7. Suites automatiques et nombres transcendants

Déterminisation

$e = (ab+b)^{*}ba$

00100 : initial

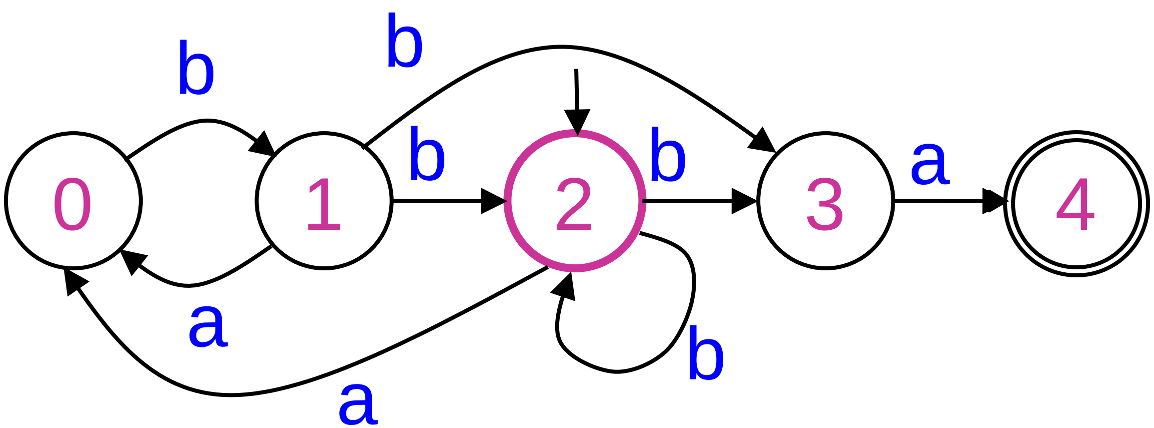


Déterminisation

00100 : a →

e = (ab+b)*ba

00100 : initial

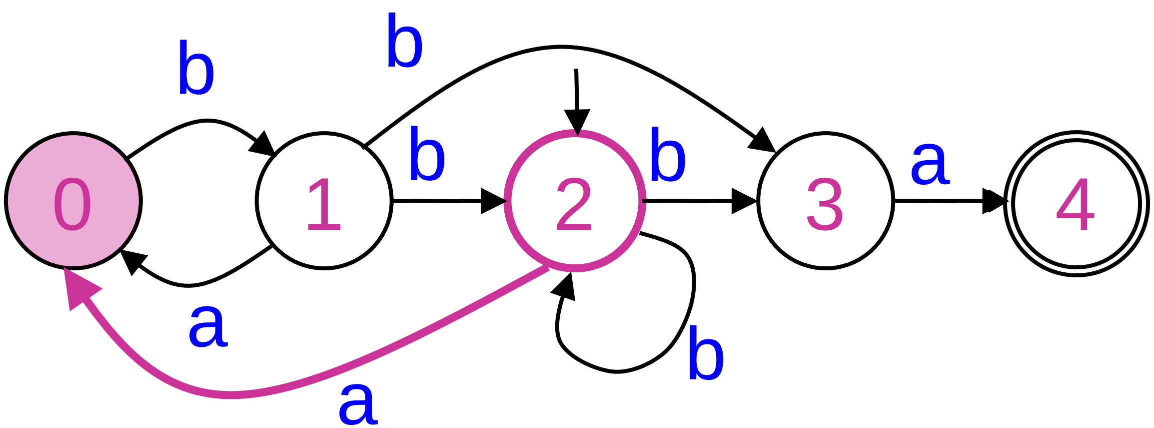


Déterminisation

00100 : a → 10000

$e = (ab+b)^*ba$

00100 : initial



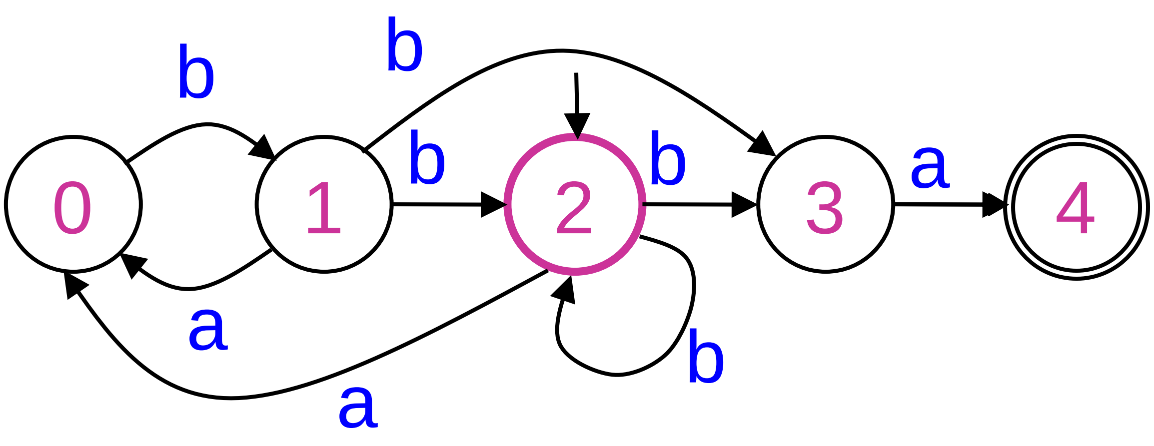
Déterminisation

e = (ab+b)*ba

00100 : initial

00100 : a → 10000

b →

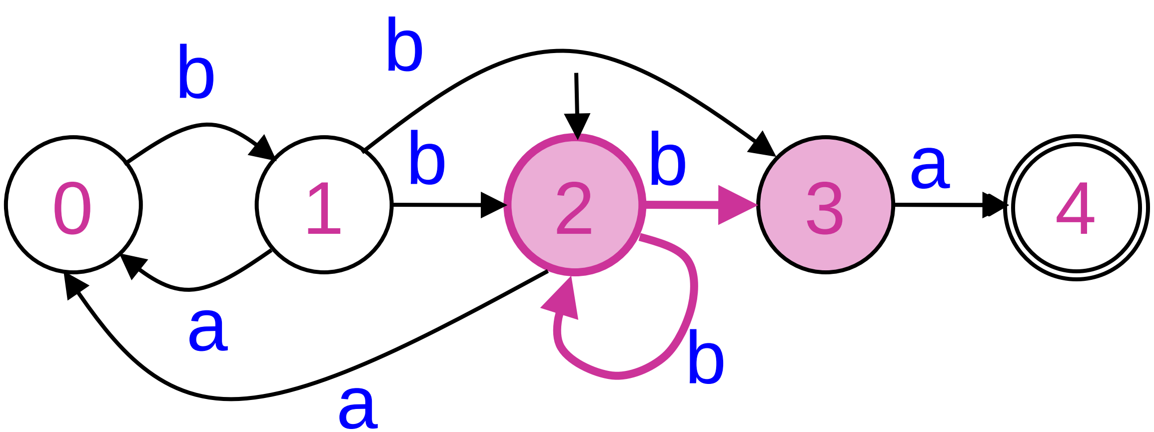


Déterminisation

$e = (ab+b)^*ba$

00100 : initial

00100 : a → 10000
b → 00110



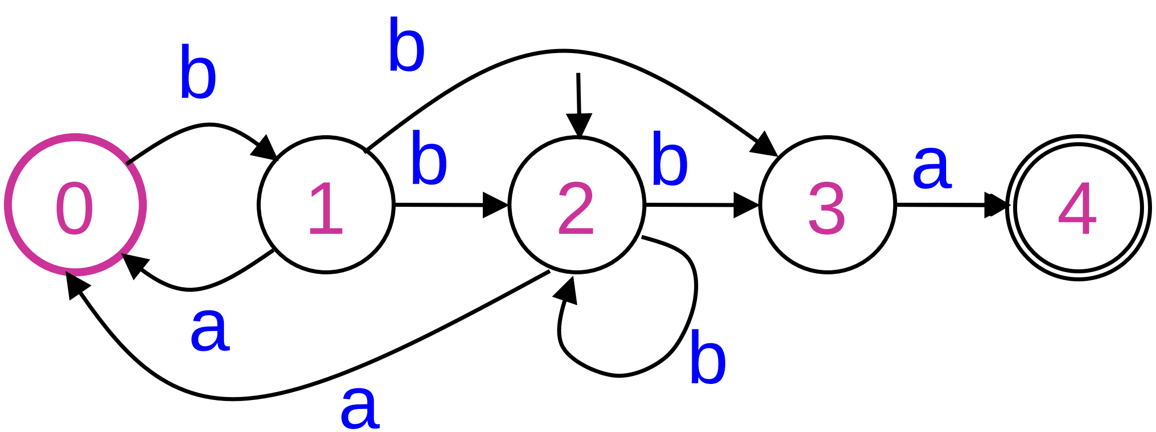
Déterminisation

$e = (ab+b)^*ba$

00100 : initial

00100 : a → 10000
b → 00110

10000 : b →



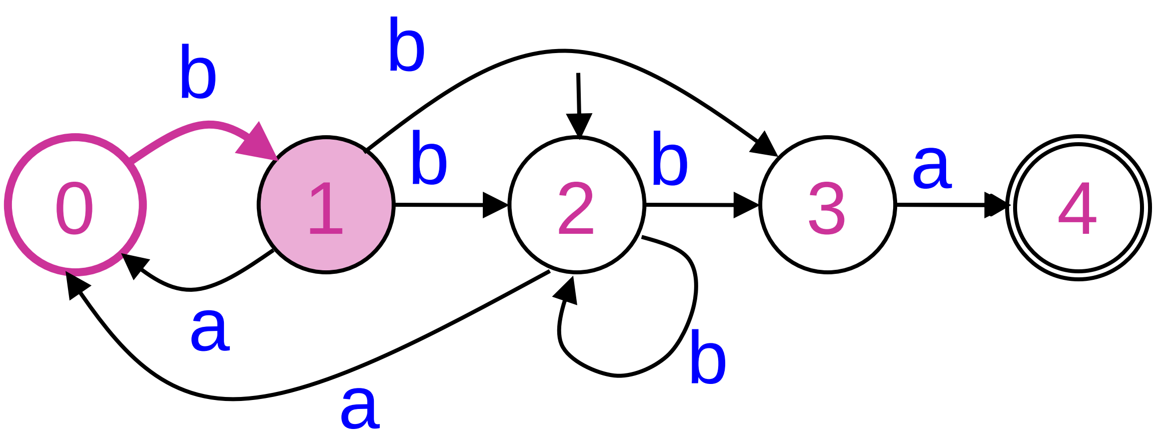
Déterminisation

$e = (ab+b)^* ba$

00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000



Déterminisation

$e = (ab+b)^*ba$

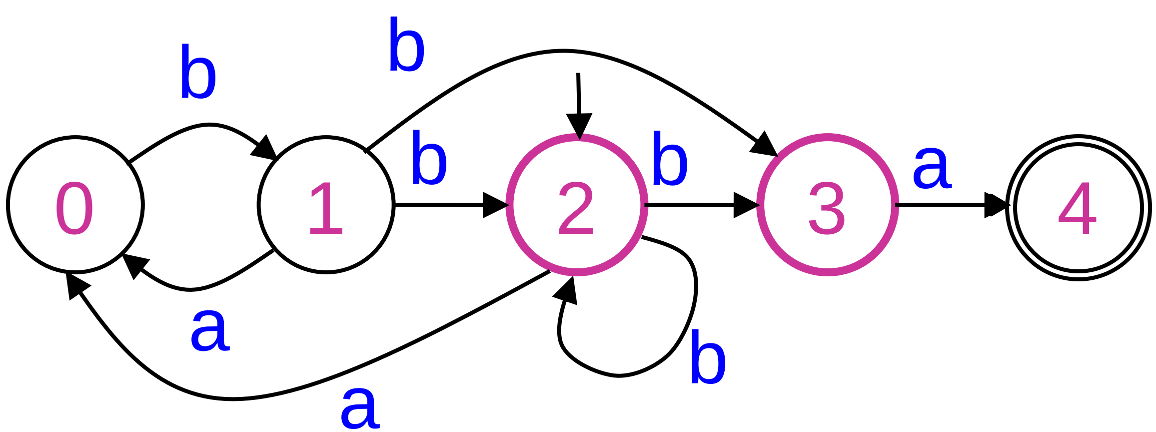
00100 : initial

00100 : a → 10000

b → 00110

10000 : b → 01000

00110 : a →



Déterminisation

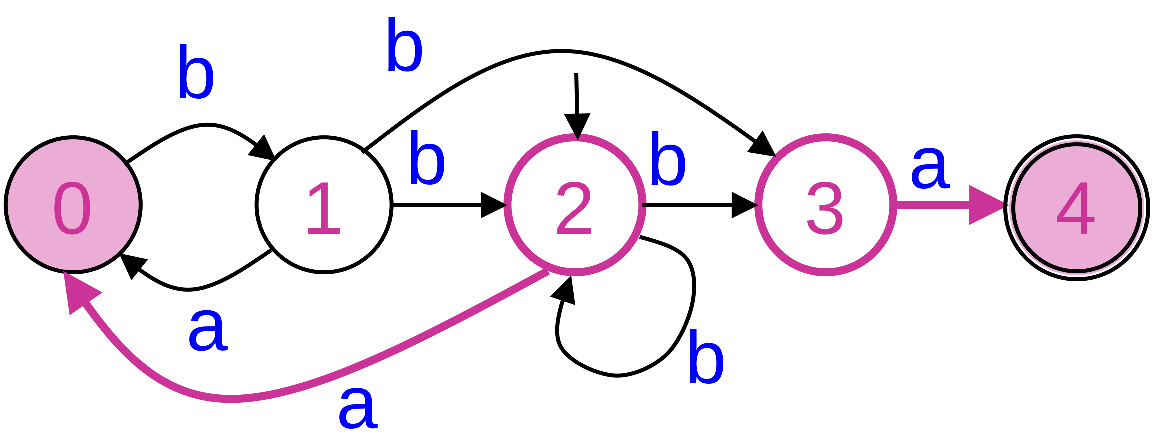
e = (ab+b)*ba

00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → <u>10001</u>



Déterminisation

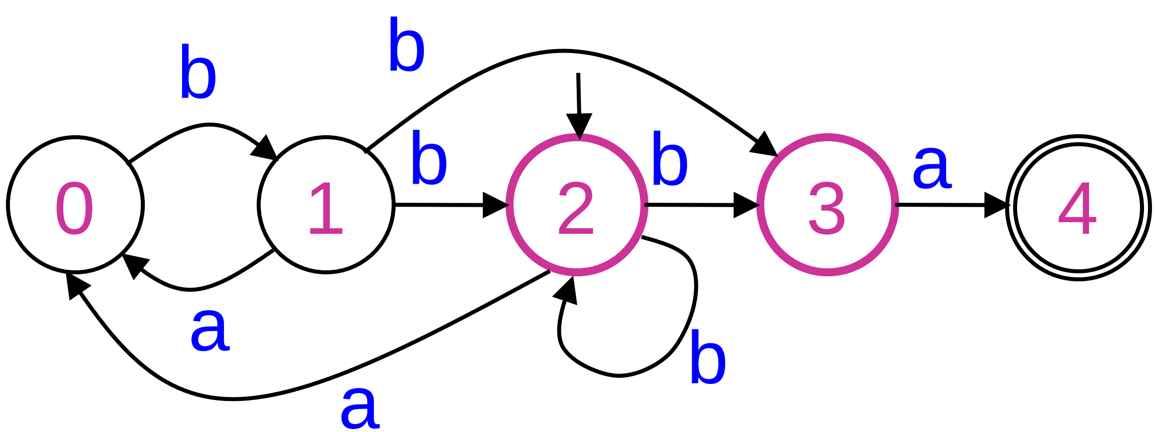
e = (ab+b)*ba

00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → <u>10001</u>
b →



Déterminisation

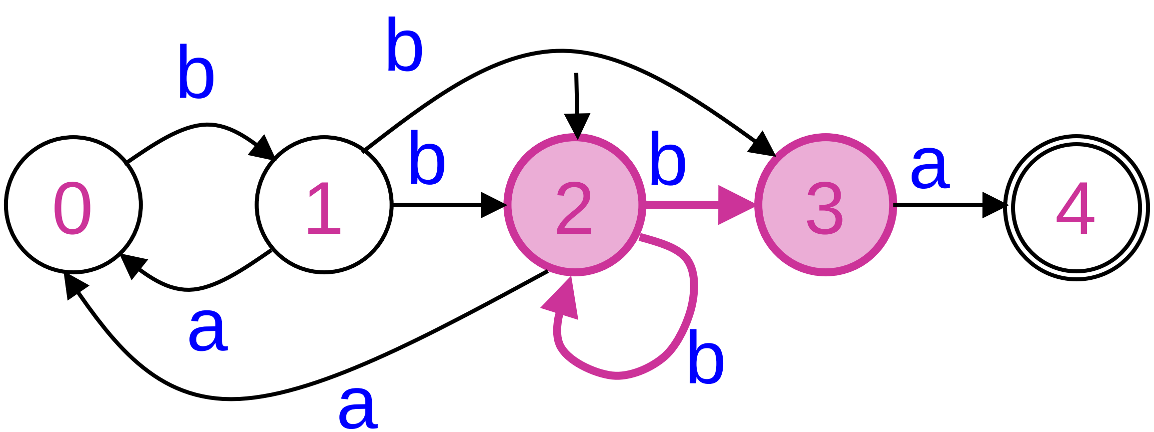
$e = (ab+b)^*ba$

00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → <u>10001</u>
b → 00110



Déterminisation

$e = (ab+b)^*ba$

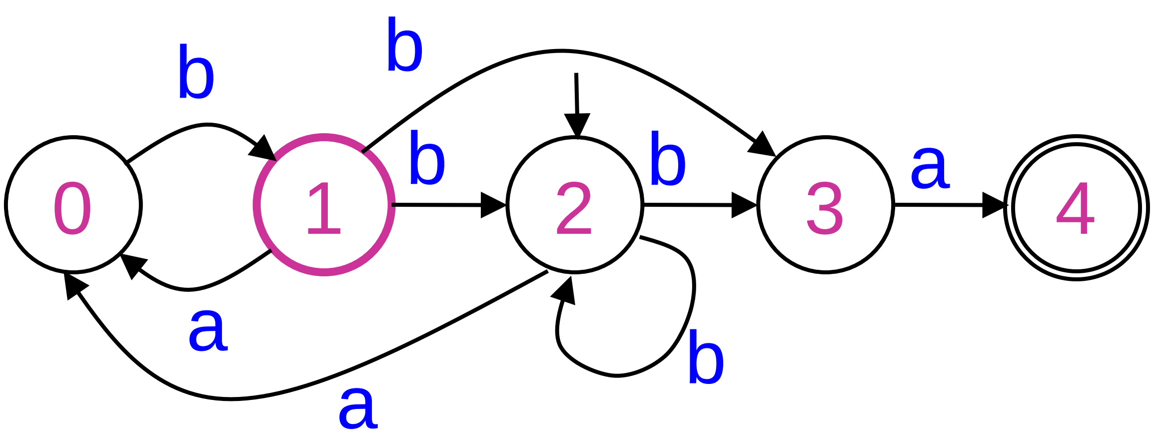
00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → <u>10001</u>
b → 00110

01000 : a



Déterminisation

$e = (ab+b)^*ba$

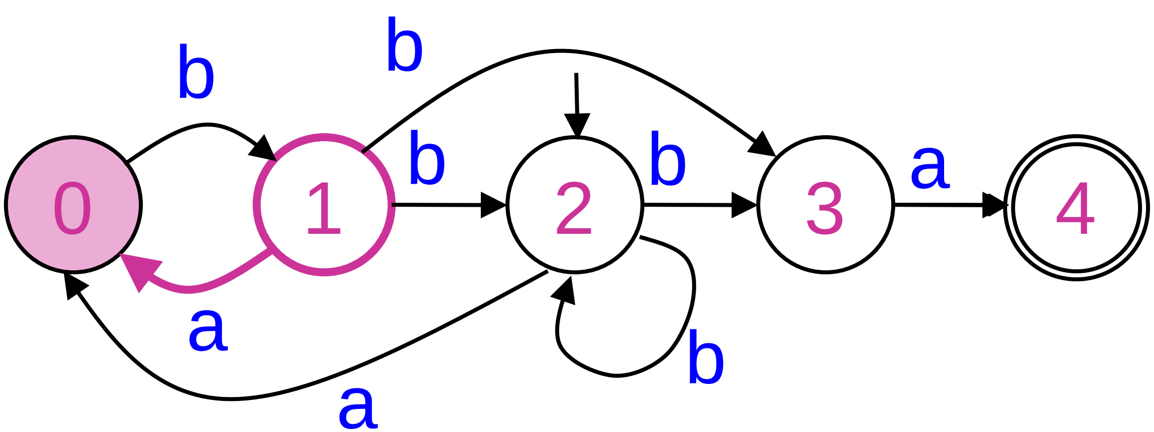
00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → <u>10001</u>
b → 00110

01000 : a → 10000



Déterminisation

$e = (ab+b)^*ba$

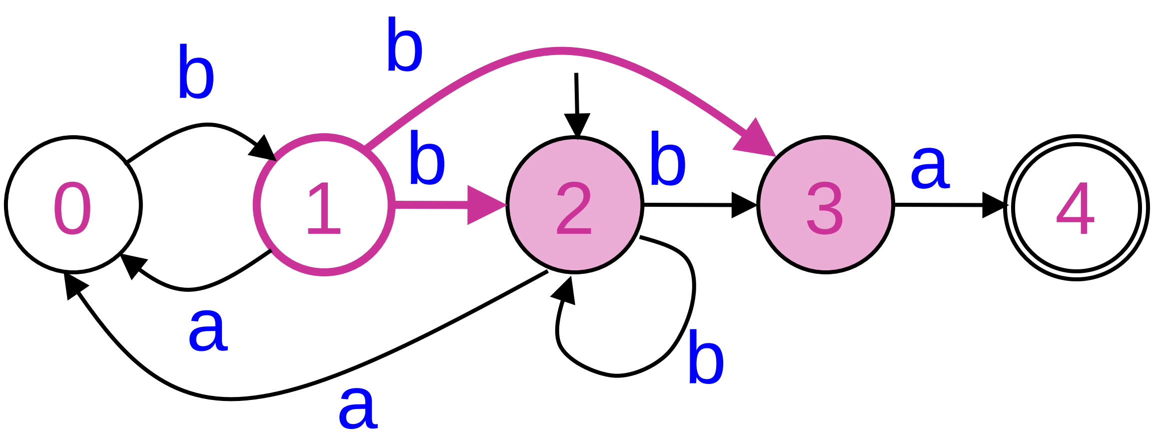
00100 : initial

00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → <u>10001</u>
b → 00110

01000 : a → 10000
b → 00110



Déterminisation

$e = (ab + b)^* ba$

00100 : initial

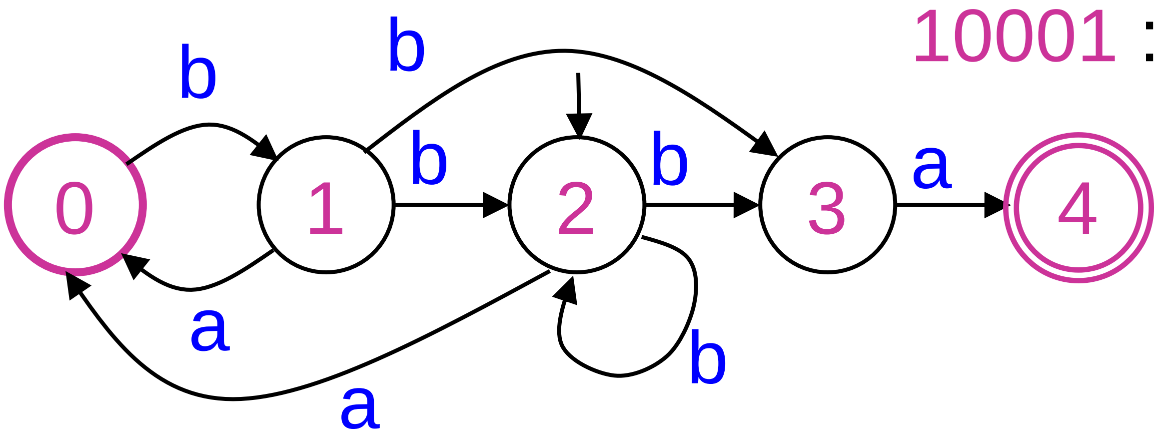
00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → <u>10001</u>
b → 00110

01000 : a → 10000
b → 00110

10001 : b



Déterminisation

$e = (ab+b)^*ba$

00100 : initial

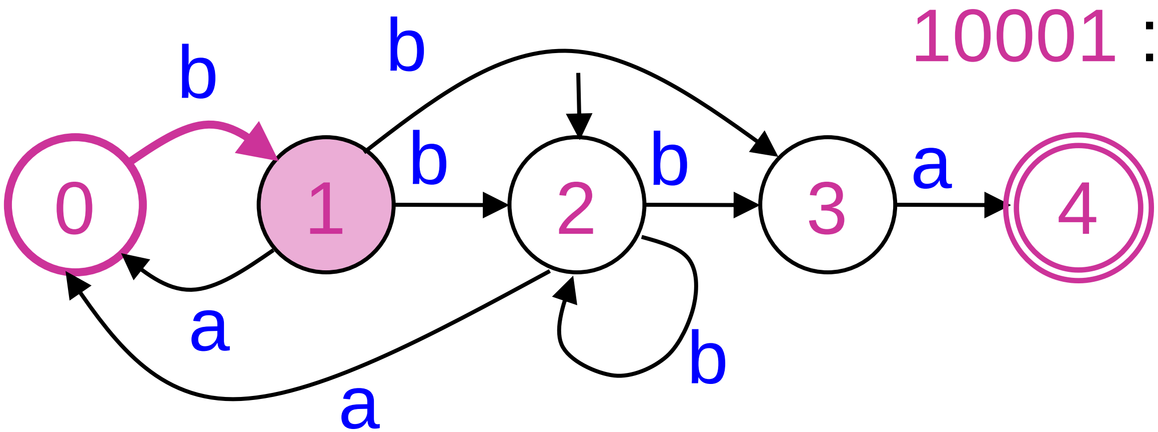
00100 : a → 10000
b → 00110

10000 : b → 01000

00110 : a → <u>10001</u>
b → 00110

01000 : a → 10000
b → 00110

10001 : b → 01000



Déterminisation

$e = (ab + b)^* ba$

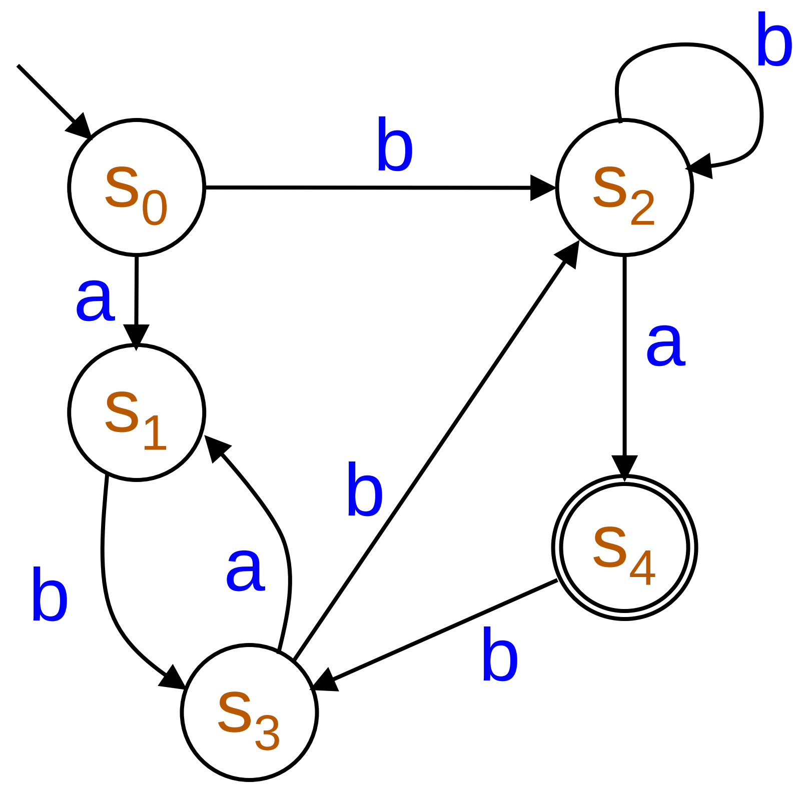
$s_0 = 00100$

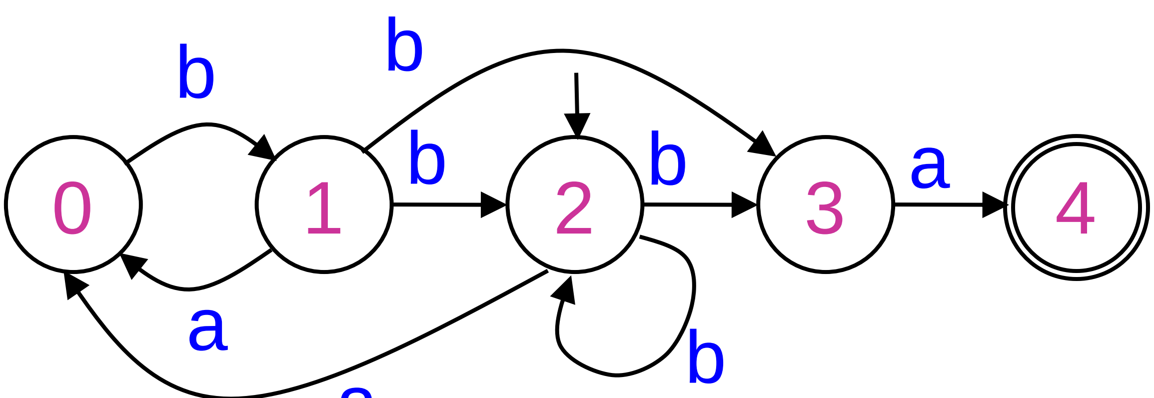
$s_1 = 10000$

$s_2 = 00110$

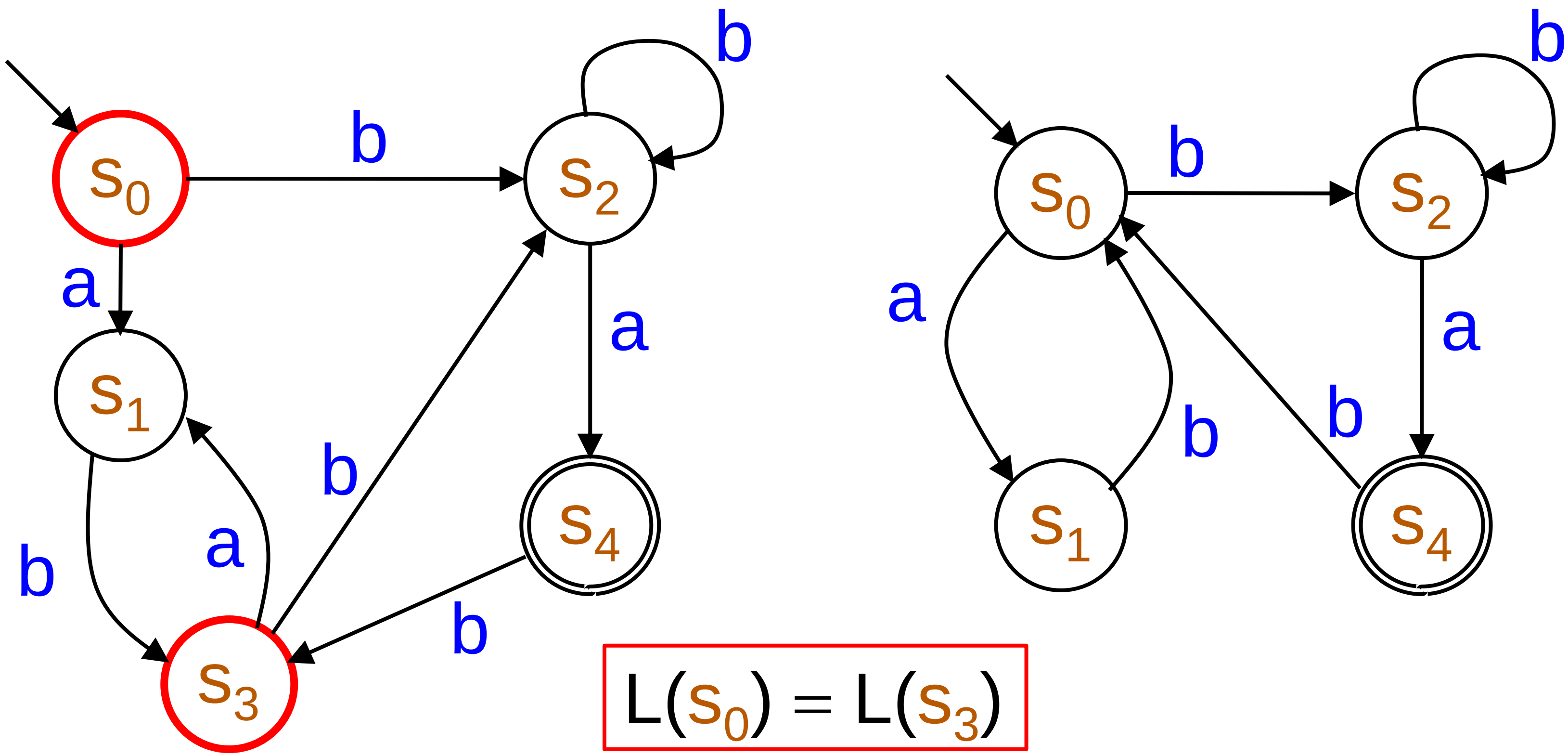
$s_3 = 01000$

$s_4 = 10001$





Minimisation d'automates déterministes



- Théorème : pour tout langage rationnel L, il existe un automate déterministe minimal reconnaissant L.
  Idée : états = $u^{-1}(L)$ non vides
- Bonus : il existe un algorithme rapide pour minimiser un automate déterministe

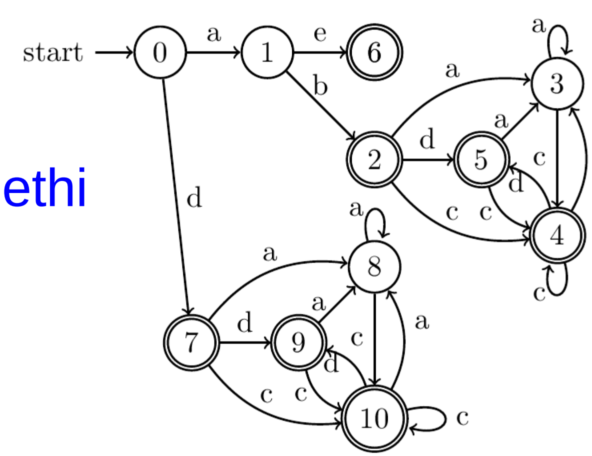
Faisons le point !

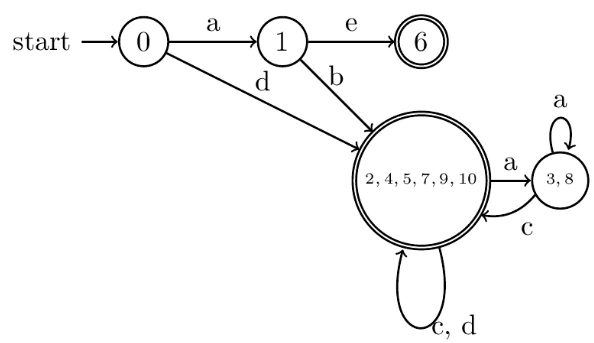
- Un langage rationnel est engendré (ou reconnu) par une expression régulière ou par un automate fini
- On peut construire directement un automate déterministe à partir d'une expression régulière, mais avec explosion exponentielle possible
- On peut construire efficacement un automate non-déterministe de taille maximale n^2 si l'expression est de taille n
- On peut déterminiser cet automate, mais avec explosion exponentielle possible
- On peut minimiser efficacement un automate déterministe (si on n'a pas explosé avant)

Autres résultats

- Il n’y a en général pas d’automate non-déterministe minimal reconnaissant un langage rationnel
- On peut nettement améliorer la construction de Berry-Sethi (cf. Antimirov, Champarnaud, Razet)

Berry-Sethi





Antimirov

Champarnaud-

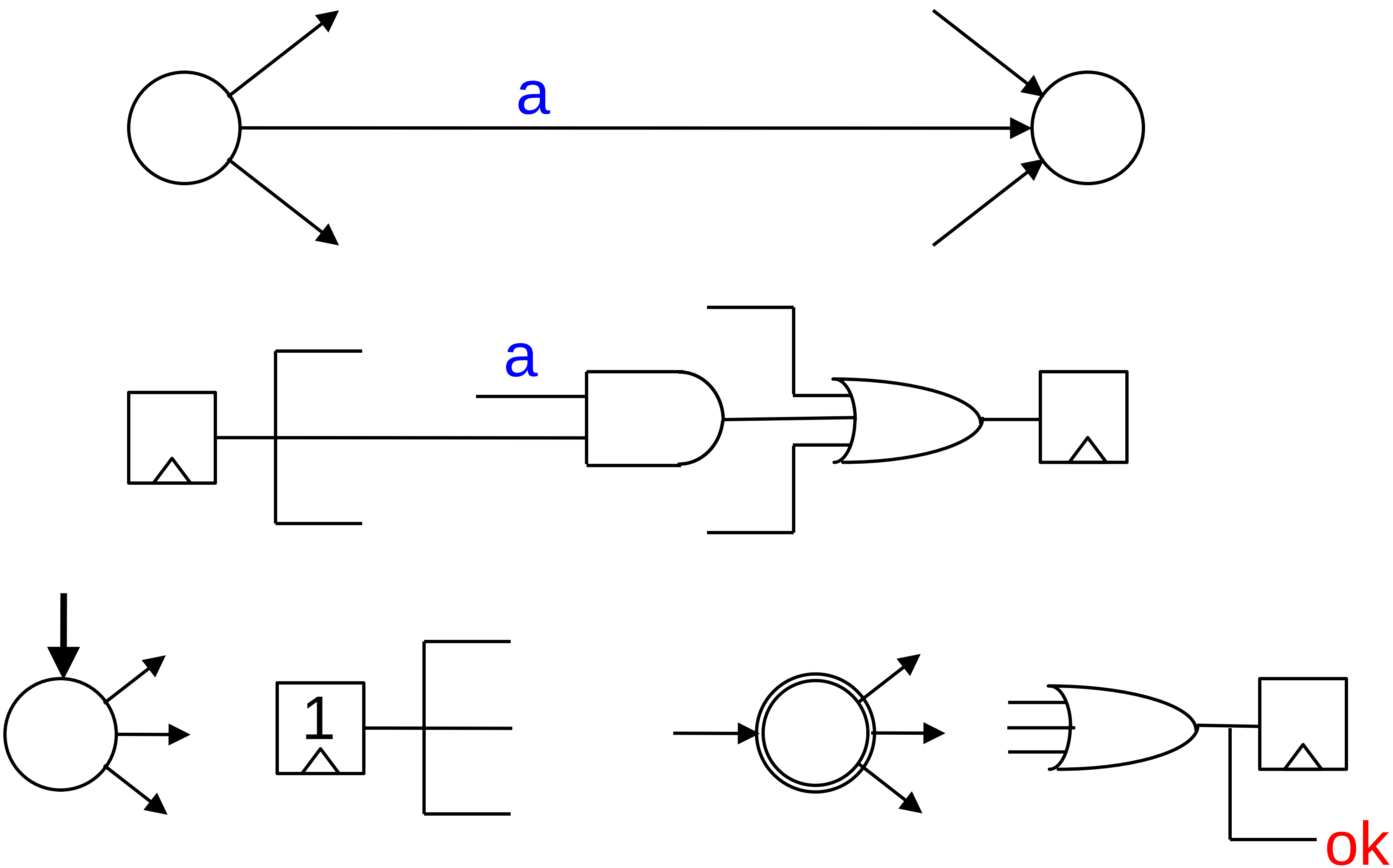
Ziadi

$$E = a(b(a^*c + d)^* + e) + d(a^*c + d)^*$$

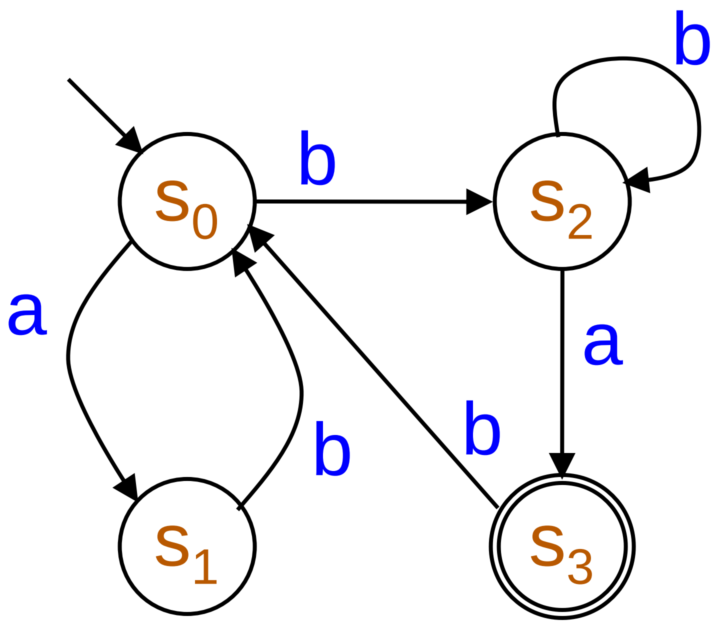
Agenda

1. Systèmes d'états finis
2. Expressions régulières
3. Dérivées et traduction en automates
4. Algorithme de Berry-Sethi
5. Déterminisation et minimisation
6. Des automates aux circuits Booléens
7. Suites automatiques et nombres transcendants

Implémentation par circuit booléen



Implémentation en Esterel v7

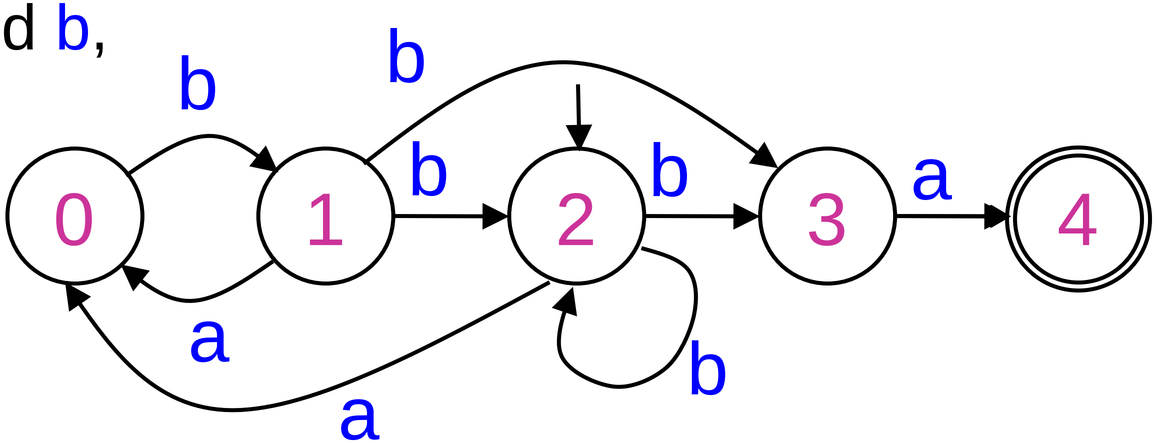



```
module Autom :
input a, b ;
output ok;
local {r0, r1, r2, r3} : reg;
refine r0 : init true;
sustain {
   next r0 <= (r1 or r3) and b,
   next r1 <= r0 and a,
   next r2 <= (r0 or r2) and b,
   next r3 <= r2 and a,
   ok <= next r3 }
end module
```

Compilé en C, C++, VHDL, Verilog, etc.

Déterminisation électrique au vol !

```
module NonDet :
input a, b ;
output ok;
local  {r0, r1, r2, r3 , r4} : reg,;
refine r2 : init true;
sustain {
   next r0 <= (r1 or r2) and a,
   next r1 <= r0 and b,
   next r2 <= (r1 or r2) and b,
   next r3 <= (r1 or r2) and b,
   next r4 <= r3 and a
   ok <= next r4
}
end module
```



Déterminisation électrique au vol !

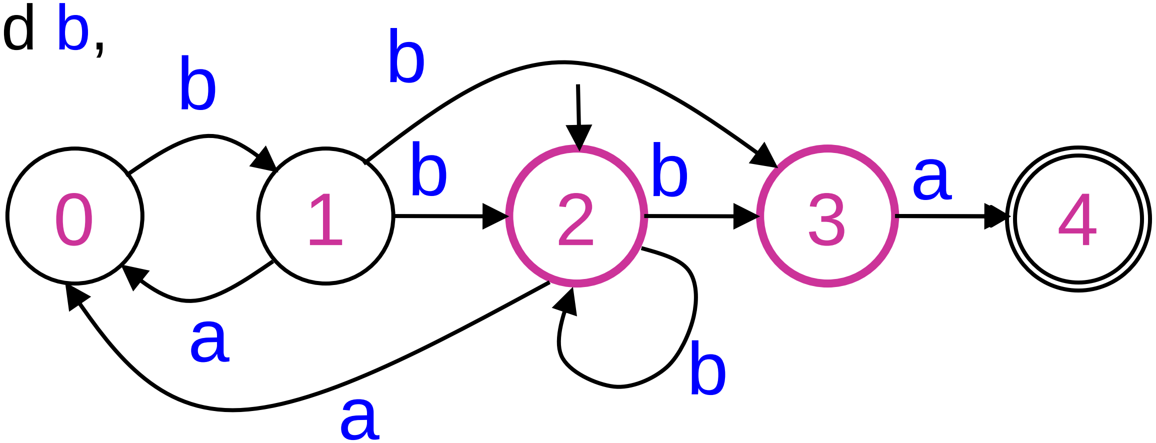
```
module NonDet :
input a, b ;
output ok;
local  {r0, r1, r2, r3 , r4} : reg;
refine r2 : init true;
sustain {
   next r0 <= (r1 or r2) and a,
   next r1 <= r0 and b,
   next r2 <= (r1 or r2) and b,
   next r3 <= (r1 or r2) and b,
   next r4 <= r3 and a
   ok <= next r4
}
end module
```

00110 : a → <u>10001</u>

r2 = r3 = 1

r0 = r3 = r4 = 0

a = 1, b = 0



Détermininisation électrique au vol !

module NonDet :
input a, b ;
output ok;
local {r0, r1, r2, r3 , r4} : reg;
refine r2 : init true;

sustain {
  next r0 <= (r1 or r2) and a,
  next r1 <= r0 and b,
  next r2 <= (r1 or r2) and b,
  next r3 <= (r1 or r2) and b,
  next r4 <= r3 and a
  ok <= next r4
}
end module

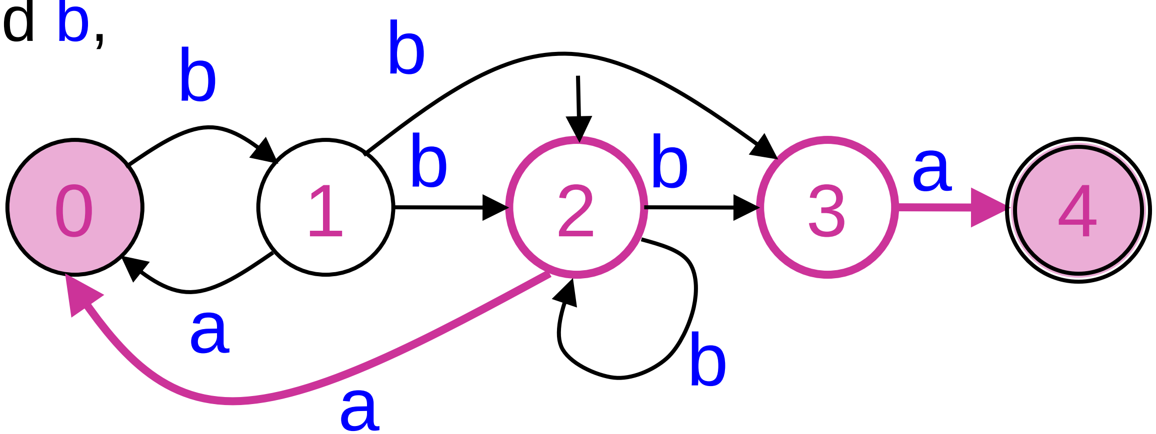
00110 : a $\rightarrow$ <u>10001</u>

r2 = r3 = 1

r0 = r3 = r4 = 0

a = 1, b = 0

next r0 = next r4 = 1
next r1 = next r2 = next r3 = 0
ok = 1



Résultat fondamental en pratique

Toute expression régulière de taille n est reconnue par un circuit à $n+1$ registres et au plus n^2 portes

- Garantit le passage à la grande échelle
- Presque toujours préférable à la déterminisation
- S'étend aux automates hiérarchiques et aux langages synchrones, cf cours 6 (13/01/2010)
- Prochaine question : comment optimiser le circuit ?

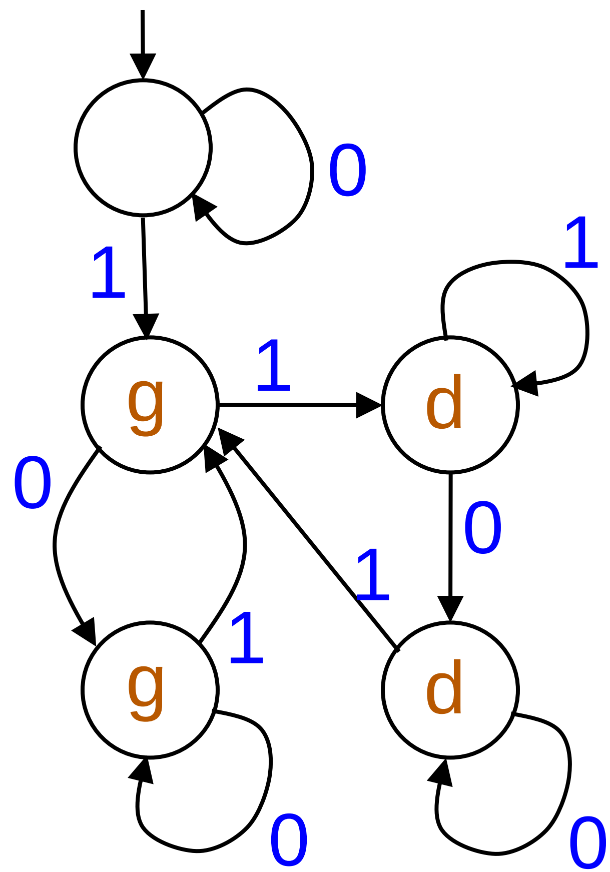
Agenda

1. Systèmes d'états finis
2. Expressions régulières
3. Dérivées et traduction en automates
4. Algorithme de Berry-Sethi
5. Déterminisation et minimisation
6. Des automates aux circuits Booléens
7. Suites automatiques et nombres transcendants

Suite automatique du pliage de papier

- Plier un papier, toujours dans le même sens
- Le déplier et compter la suite des virages
- Jouer les décimales de n de gauche à droite

	1000 : g
1 : g	1001 : g
10 : g	1010 : g
11 : d	1011 : d
100 : g	1100 : d
101 : g	1101 : g
110 : d	1110 : d
111 : d	1111 : d



suite automatique

Transcendance du pliage de papier

- Si une suite est automatique, construire le réel

  $0,s_1s_2...s_n...$

- Théorème (Van der Poorten) : ce nombre est soit rationnel soit transcendant

	1000 : g
1 : g	1001 : g
10 : g	1010 : g
11 : d	1011 : d
100 : g	1100 : d
101 : g	1101 : g
110 : d	1110 : d
111 : d	1111 : d

g=1, d=0

$\boxed{\text{PP} = 0{,}110110011100100... \text{ est transcendant !}}$

cf. Allouche, Mendès-France, Rauzy, etc.

Références

Eléments de théorie des automates

Jacques Sakarovitch

Vuibert Informatique, 2003

From Regular Expressions to Deterministic Automata

Gérard Berry et Ravi Sethi

Theoretical Computer Science 48 (1986) 117-126.

Automatic Sequences: Theory, Applications, Generalizations

Jean-Paul Allouche et Jeffrey Shallit

Cambridge University Press (21 juillet 2003)