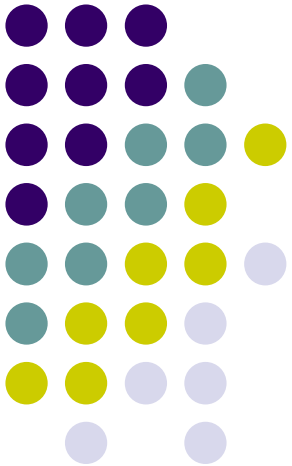
THÉORIE DES LANGAGES ET DES AUTOMATES

CH2: AUTOMATES FINIS ET EXPRESSIONS RÉGULIÈRES



esprit

Ecole Supérieure Privée
d'Ingénierie et de Technologies

ANNÉE UNIVERSITAIRE 2013 / 2014

Plan

- Les expressions régulières
- Les automates finis non déterministes
- Les automates finis déterministes
- Les automates avec ε-transitions
- L'équivalence des modèles

EXPRESSIONS RÉGULIÈRES – PRINCIPE

- L'ensemble ∅ représente le langage vide.
- Le mot vide ε représente le langage $\{\varepsilon\}$
- Un caractère c de Σ représente le langage $\{c\}$
- L'alternative $p_1 \mid p_2$ représente l'union des langages représentés par p_1 et p_2.
- La concaténation $p_1.p_2$ représente la concaténation des langages représentés par p_1 et p_2.
- La répétition $p*$ représente l'itération du langage représenté par p.

Dans les Expressions régulières (ER) on va distinguer :

- La syntaxe (le modèle)
- La sémantique (le langage correspondant)

EXPRESSIONS RÉGULIÈRES – LA SYNTAXE

- Le mot vide ε est une expression régulière.
- Un caractère c est une expression régulière.
- Si p_1 et p_2 sont deux expressions régulières alors l'alternative $p_1 \mid p_2$ est une expression régulière.
- Si p_1 et p_2 sont deux expressions régulières alors la concaténation $p_1.p_2$ est une expression régulière.
- Si p est une expression régulière alors $p*$ est une expression régulière.

EXPRESSIONS RÉGULIÈRES – PRIORITÉS

- Analogie avec les expressions arithmétiques, en interprétant :
  - '|' par '+'
  - '.' par '×'
  - '*' par une mise à la puissance
- Formellement et par ordre décroissant de priorités nous avons :
  - ()
  - '*'
  - '.'
  - '|'
- Le symbole de concaténation peut être omis.
- Exemple :
  - $ab*$ correspond à $a.(b*)$.
  - $a.b\,|\,c$ correspond à $(a.b)\,|\,c$.
  - $ab*\,|\,c$ correspond à $(a.(b*))\,|\,c$.

LES EXPRESSIONS RÉGULIÈRES

Soit un alphabet Σ, les règles définissant les expressions régulières sur Σ :

- ε est expression régulière qui dénote $\{\varepsilon\}$ l'ensemble constitué de la chaîne vide.
- Si a est un symbole de Σ, alors a est une expressions régulière qui dénote $\{a\}$, l'ensemble constitué de la chaîne a.
- Si r et s sont des ER dénotant les langages $L(r)$ et $L(s)$, alors :
  - $(r)|(s)$ est une expression régulière dénotant $L(r) \cup L(s)$.
  - $(r)(s)$ est une expression régulière dénotant $L(r)L(s)$.
  - $(r)^*$ est une expression régulière dénotant $(L(r))^*$
  - (r) est une expression régulière dénotant $L(r)$
- Si deux ER r et s dénotent le même langage, elles sont équivalentes et note $r = s$. Par exemple $(a|b) = (b|a)$.

LES EXPRESSIONS RÉGULIÈRES

EXEMPLE

Soit l'alphabet $\Sigma = \{a, b\}$

- L'expression $a \mid b$ dénote l'ensemble $\{a, b\}$.
- L'expression $(a \mid b)(a \mid b)$ dénote l'ensemble $\{aa, ab, ba, bb\}$. Ce même ensemble peut être dénoté par l'ER $aa \mid ab \mid ba \mid bb$.
- L'expression a^* dénote l'ensemble de toutes les chaînes constituées d'un nombre quelconque (éventuellement nul) de a, c'est-à-dire $\{\varepsilon, a, aa, aaa, \ldots\}$.
- L'expression régulière $(a \mid b)^*$ dénote l'ensemble de toutes les chaînes constituées d'un nombre quelconque (éventuellement nul) de a ou de b. Ce même ensemble peut être dénoté par l'ER $(a^* b^*)^*$.
- L'expression $a \mid a^*b$ contenant la chaîne a ou toutes les chaînes constituées d'un ensemble quelconque (éventuellement vide) de a suivi d'un b.

LES LANGAGES RÉGULIERS

- Un langage L est dit régulier s'il existe une expression régulière qui le génère.
- Une expression régulière r dénote un langage régulier $L(r)$
- Tout langage fini est régulier.
- Ø est un langage régulier ;
- {ε} est un langage régulier ;
- Pour tout $a \in \Sigma$, $\{a\}$ est un langage régulier ;
- Si L_1 et L_2 sont réguliers, alors $L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1 \text{ et } w_2 \in L_2\}$ est régulier.
- Si L est régulier, alors :

  $L^* = \{\varepsilon\} \cup \{w_1 w_2 w_3 \cdots w_n \mid w_i \in L\}$ est régulier ;
- Si L_1 et L_2 sont réguliers, alors $L_1 \cup L_2$ est régulier.
- Si L est un langage régulier, $L^c = \sum^* \setminus L$ est un langage régulier

LES AUTOMATES FINIS

- Validateur pour langages : programme prenant une chaîne x en entrée et retournant « *oui* » si c'est une phrase du langage et « *non* » sinon.
- Expression régulière $\rightarrow$ Validateur : construction d'un diagramme de transitions généralisé appelé « automate fini ».
- Un automate fini peut être déterministe ou non déterministe.
- Automate fini non déterministe (AFN) : on peut trouver plus d'une transition sortant d'un état sur le même symbole d'entrée.
- Les deux types d'automates finis reconnaissent précisément les ensembles réguliers.
- Les automates finis déterministes conduisent à des validateurs plus rapides.
- L'implémentation d'un automate fini déterministe est plus volumineuse que celle d'un automate fini non déterministe.

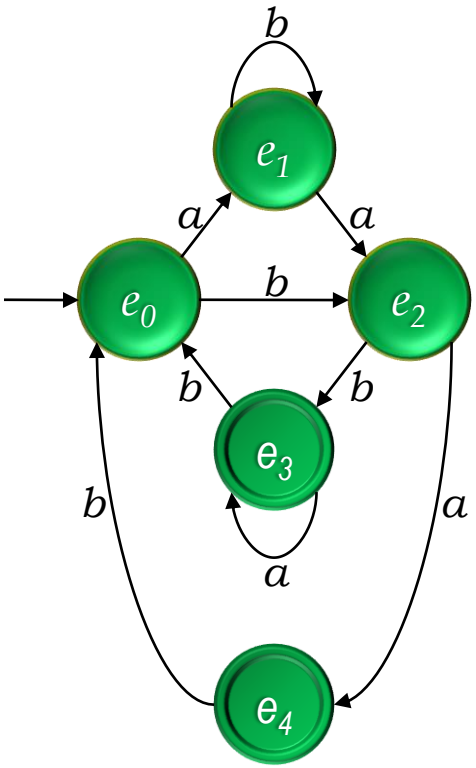
LES AUTOMATES FINIS

Un automate fini $\mathcal{A} = \langle E, \Sigma, \delta, e_0, F \rangle$

- E un ensemble d'états.
- Σ un ensemble de symboles (l'alphabet des symboles d'entrée).
- $\delta : E \times \Sigma \rightarrow E$, une fonction de transition faisant correspondre des couples état-symbole à des ensembles d'états.
- Un état e_0 : état de départ ou état initial.
- Un ensemble d'états F connus comme états d'acceptation ou états finaux.

LES AUTOMATES FINIS NON DÉTERMINISTES

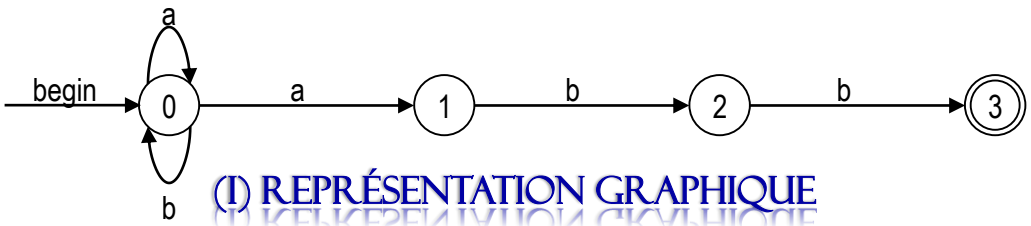
AFN - EXEMPLE



- $E = \{e_0, e_1, e_2, e_3, e_4\}$.
- $\Sigma = \{a, b\}$.
- $\delta : E \times \Sigma \rightarrow E$.
  - $\delta(e_0, a) = e_1$
  - $\delta(e_0, b) = e_2$
  - $\delta(e_1, a) = e_2$
  - $\delta(e_1, b) = e_1$
  - ...
- e_0 : état initial.
- $F = \{e_3, e_4\}$.

REPRÉSENTATION DES AFN

Considérons le langage dénoté par l'ER $(a \mid b)^*abb$ (chaînes de a et de b se terminant par abb). L'automate correspondant peut être représenté comme un graphe orienté étiqueté : les nœuds sont des états, les arcs représentent la fonction de transition (I) ou sous forme de table de transitions(II)



(I) REPRÉSENTATION GRAPHIQUE

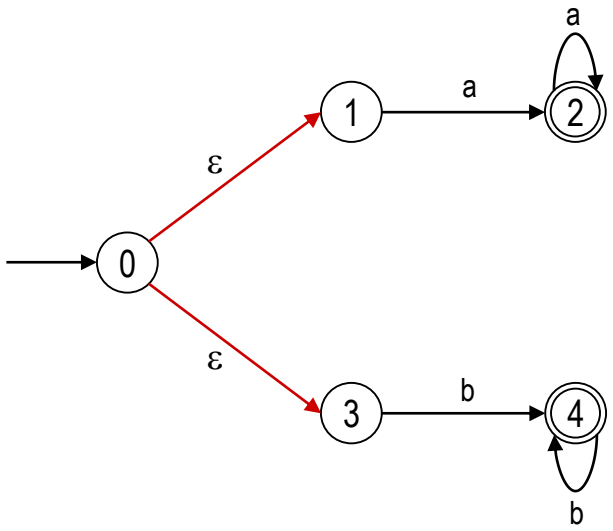
ÉTAT	SYMBOLE D'ENTRÉE	
	a	b
0	{0, 1}	{0}
1	–	{2}
2	–	{3}

(II) REPRÉSENTATION SOUS FORME DE TABLE DE TRANSITIONS

AUTOMATES FINIS NON DÉTERMINISTES

EXEMPLE

L'AFN suivant reconnaît $aa^* \mid bb^*$. La chaîne aaa est acceptée en se déplaçant via les états 0, 1, 2, 2 et 2. Les étiquettes de ces arcs sont ε, a, a et a dont la concaténation est aaa (ε disparaît dans la concaténation).



AFN AVEC ε-TRANSITIONS

AUTOMATES FINIS DÉTERMINISTES

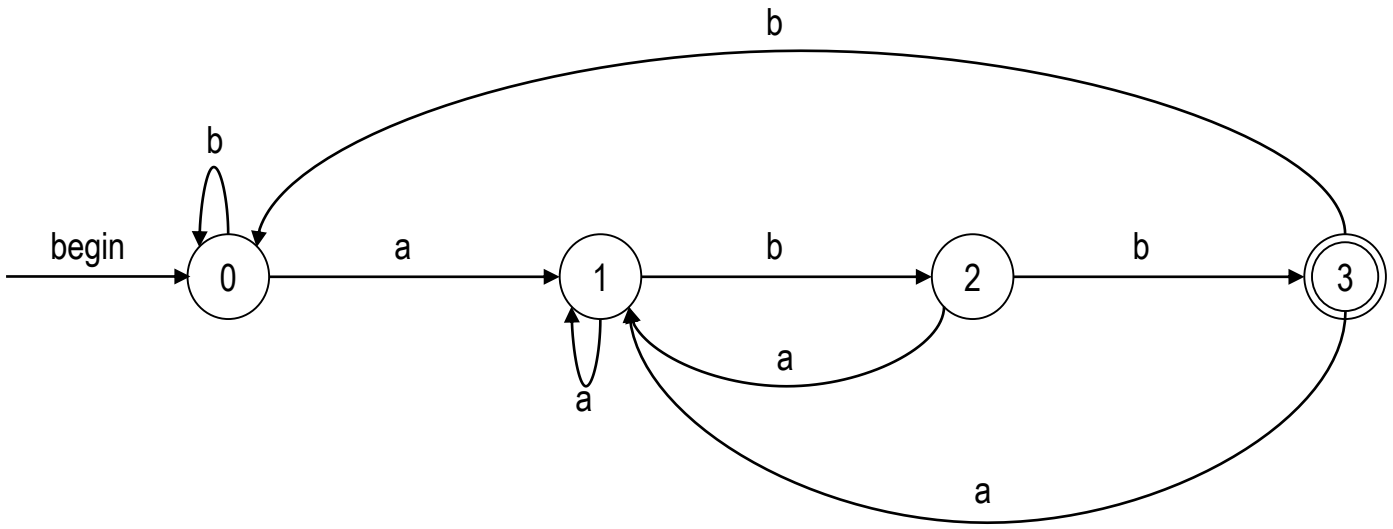
Un automate fini déterministe (AFD) est un cas particulier de l'AFN dans lequel

- Aucun état n'a de ε-transition (transition sur l'entrée ε).
- Pour chaque état e et chaque symbole d'entrée a, il y a au plus un arc étiqueté a qui quitte e

AUTOMATES FINIS DÉTERMINISTES

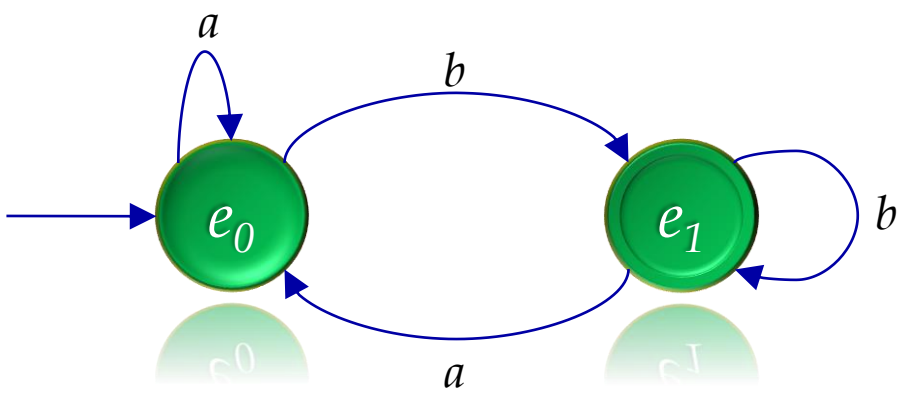
EXEMPLE

L'AFD suivant accepte le même langage $(a \mid b)^*abb$ que celui accepté par l'AFN (page 18)



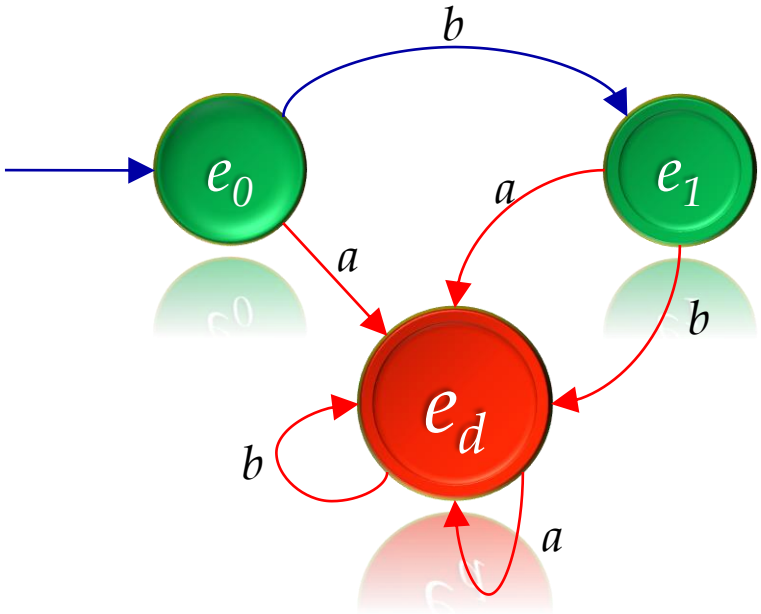
AUTOMATES FINIS COMPLETS

Un automate fini déterministe est dit complet si, $\forall\ e \in E$, il y a exactement une transition pour chaque symbole possible de Σ.

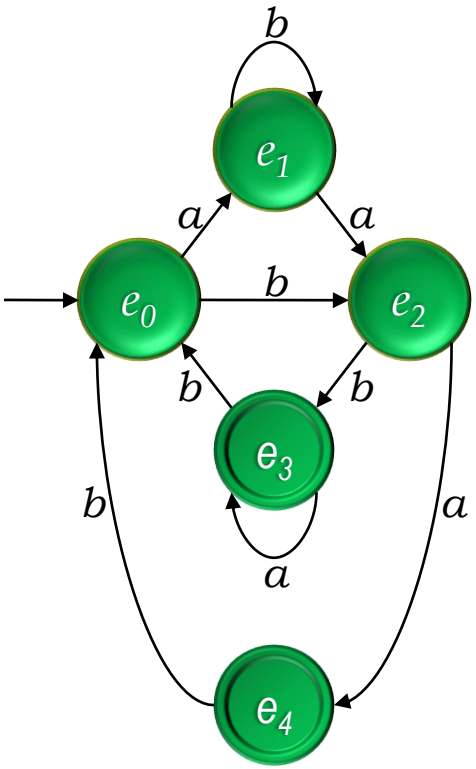


AUTOMATES FINIS COMPLETS

Tout AFD non complet peut être transformé en AFD complet par l'ajout d'un état que nous qualifions de « mort » ou « poubelle » e_d vers lequel arrivent des transitions que nous ajoutons de telle sorte que de chaque état de l'automate parte exactement une transitions sur chaque symbole de l'alphabet.



LANGAGE ACCEPTÉ PAR UN AFN



- Un automate prend en entrée un mot w et l'accepte ou le rejette.
  - On part de l'état e_0.
  - On lit les caractères du mot w un à un en suivant la transition correspondante (si il n'y en a pas, la validation échoue).
  - Lorsque tous les caractères sont lus, on accepte si l'on est sur un état final, on rejette sinon.
- Exemple :
  - Le mot *abbbab* est accepté.
  - Le mot *babba* est accepté.
  - Le mot *ababab* n'est pas accepté.

ÉQUIVALENCE D'AUTOMATES FINIS

Définition : deux automates sont **équivalents** s'ils reconnaissent le même langage.

Cette relation est bien une **équivalence**, car elle est définie à partir de l'égalité des ensembles qui est une équivalence.

CONSTRUCTION D'UN AFD À PARTIR D'UN AFN SANS ε TRANSITIONS

- Pour un AFN sans ε transitions, construire à partir de A un nouveau automate B qui lui est équivalent non ambigu en faisant la fusion de quelques états pour enlever l'ambigüité. L'état initial est le même et les états finaux sont toutes sommes d'états contenant un ancien état final.
- Soit L'AFN M:

	a	b
q0	q0/q1	q1
q1	q1	q1

Etat/alphabet	a	B
{q0} /A	{q0,q1} /B	{q1} /C
{q0,q1} /B	{q0,q1} /B	{q1} /C
{q1} /C	{q1} /C	{q1} /C

AFD M':

	a	b
A	B	C
B	B	C
C	C	C

Les états finaux sont B et C car dans B on a q1 l'état final de l'ancien automate et de même pour C.

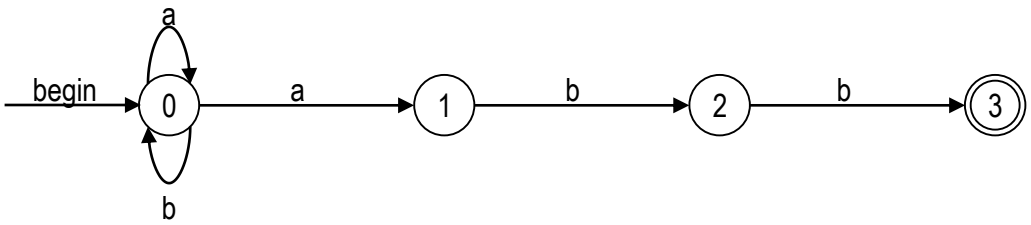
CONSTRUCTION D'UN AFD À PARTIR D'UN AFN

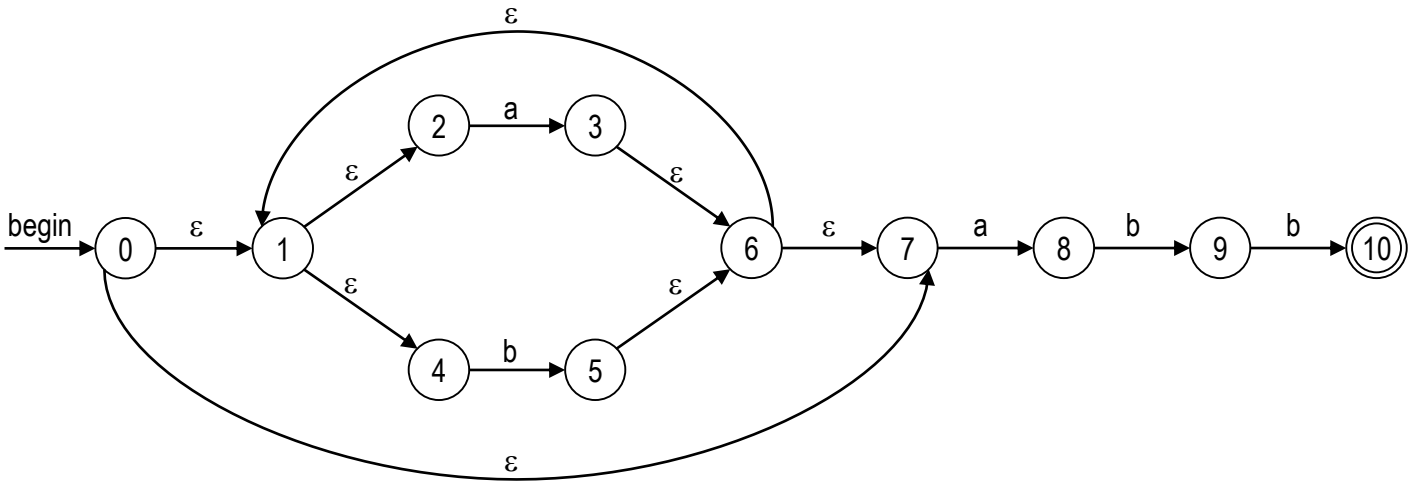
On construit Dstates, l'ensemble des états de D et Dtran, la table de transitions de la manière suivante :

- Chaque état de D correspond à un ensemble d'états de l'AFN,
- L'état de départ de D est la ε-fermeture (e_0) .
- On ajoute des états et des transitions. Un état de D est un état d'acceptation s'il contient au moins un état d'acceptation de l'AFN

CONSTRUCTION D'UN AFD À PARTIR D'UN AFN

EXEMPLE

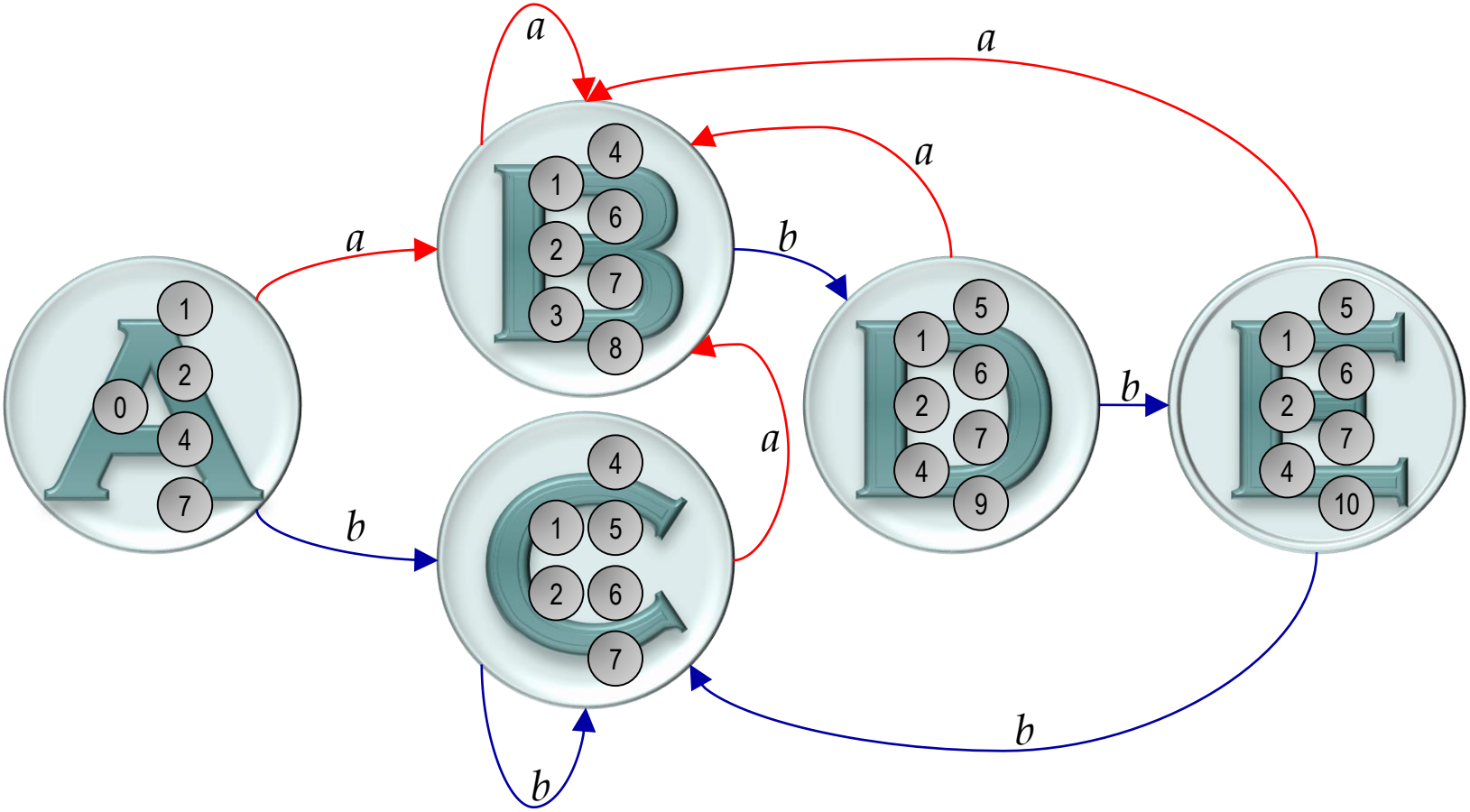




AFN NPOUR (*a* | *b*)**abb*

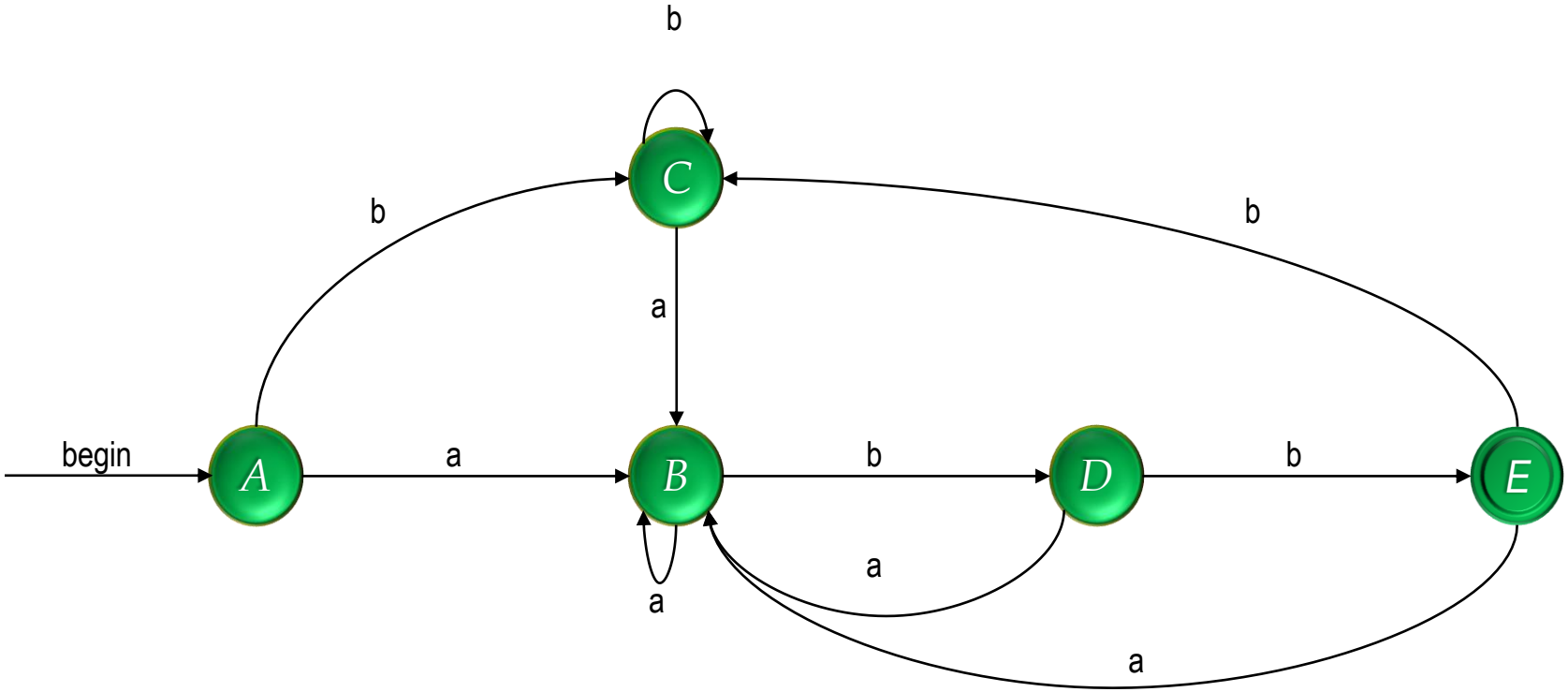
esprit

CONSTRUCTION D'UN AFD À PARTIR D'UN AFN



esprit

CONSTRUCTION D'UN AFD À PARTIR D'UN AFN



esprit

CONSTRUCTION D'UN AFD À PARTIR D'UN AFN

ÉTAT	SYMBOLE D'ENTRÉE	
	a	b
A	*B*	*C*
B	*B*	*D*
C	*B*	*C*
D	*B*	*E*
E	*B*	*C*

ALGORITHME DE MINIMISATION D'UN AFD

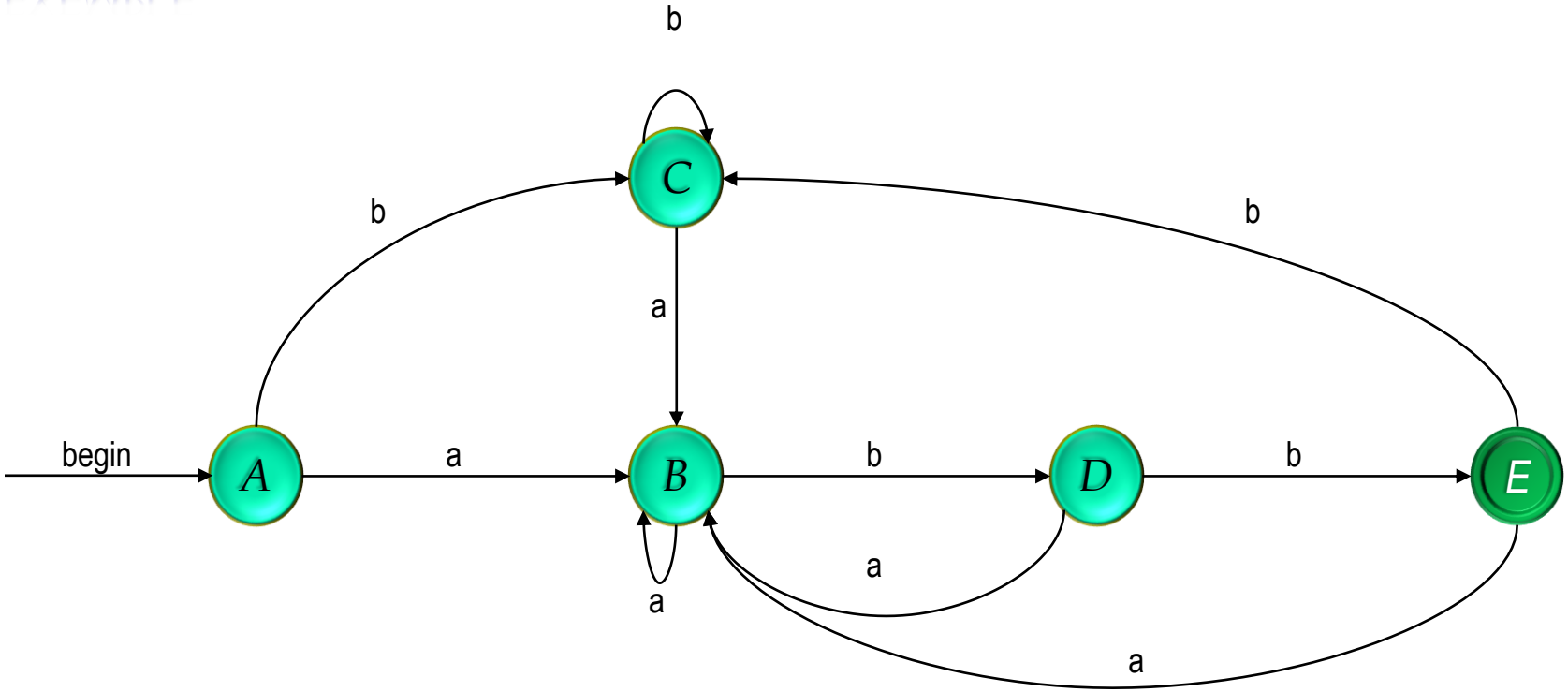
Données : un AFD $\mathcal{A}$ avec un ensemble d'états E, un alphabet d'entrée Σ, des transitions définies pour tous les états, un état initial e_0 et un ensemble d'états d'acceptation F.

Résultat : un AFD $\mathcal{A}'$ qui reconnait le même langage que $\mathcal{A}$ qui a le minimum d'états possible.

1. Construire une partition initiale P de l'ensemble des états avec deux groupes :
   - Les états d'acceptation (S).
   - Les états de non acceptation (F).
2. Appliquer la procédure suivante à P pour construire une nouvelle partition P_n.

ALGORITHME DE MINIMISATION D'UN AFD

EXEMPLE

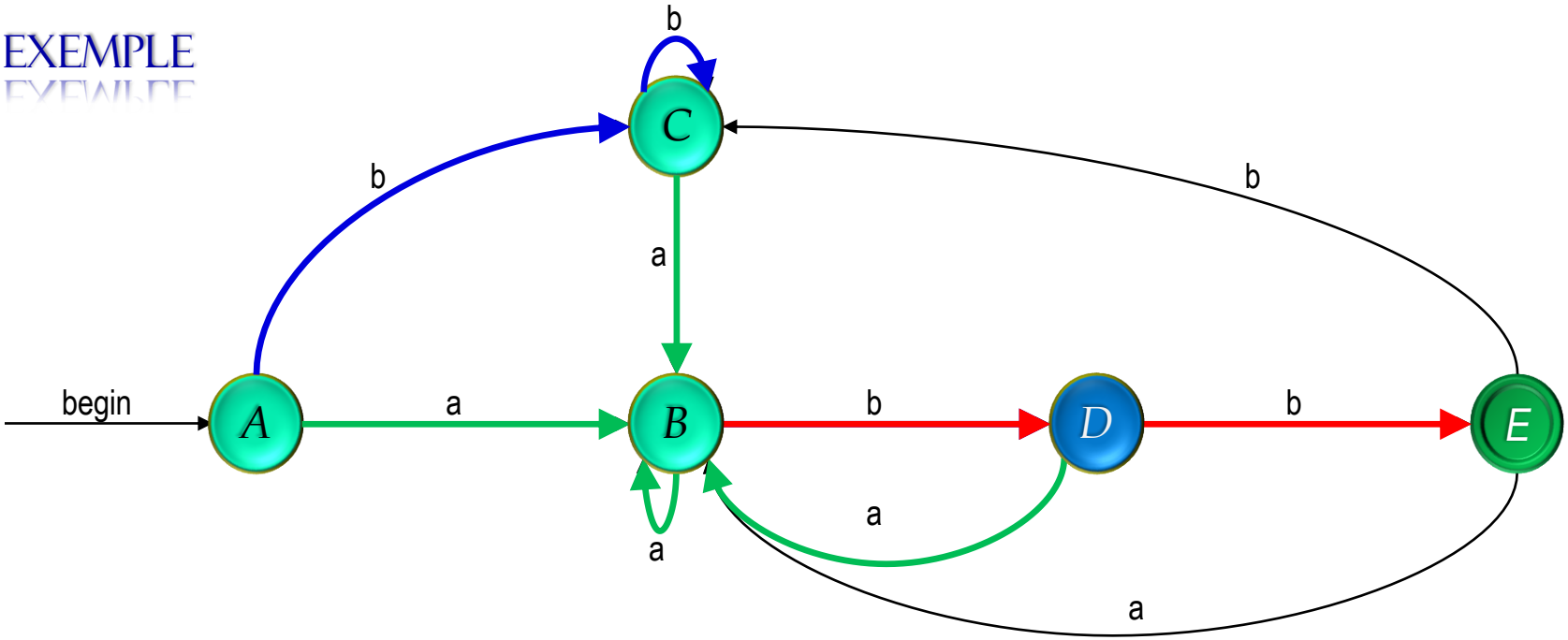


ALGORITHME DE MINIMISATION D'UN AFD

- La partition initiale P consiste en deux groupes :
  - E, l'état d'acceptation.
  - A, B, C et D, les états de non acceptation.
- Construction de P_n :
  - Considérons E constitué d'un seul état et ne peut être découpé, on l'ajoute à P_n.
  - Considérons le groupe constitué de A, B, C et D.
    - ✓ Sur le symbole a, chacun de ces états a une transition vers B, ce qui ne les distingue pas.
    - ✓ Sur le symbole b, A, B et C ont des transitions vers le groupe (A B C D) tandis que D a une ransitions vers E.
    - ✓ Dans P_n, le groupe (A B C D) doit être découpé en (A B C) et (D). P_n = (A B C) (D) (E).
  - Dans la phase suivante nous n'avons aucun découpage de (A B C) sur le symbole a par contre il doit être découpé en (A C) et (B) sur le symbole b. P_n = (A C) (B) (D) (E).
  - Dans la passe suivante on ne peut plus découper aucun groupe ni sur a ni sur b.
- P_f = (AC) (B) (D) (E).

ALGORITHME DE MINIMISATION D'UN AFD

EXEMPLE



esprit

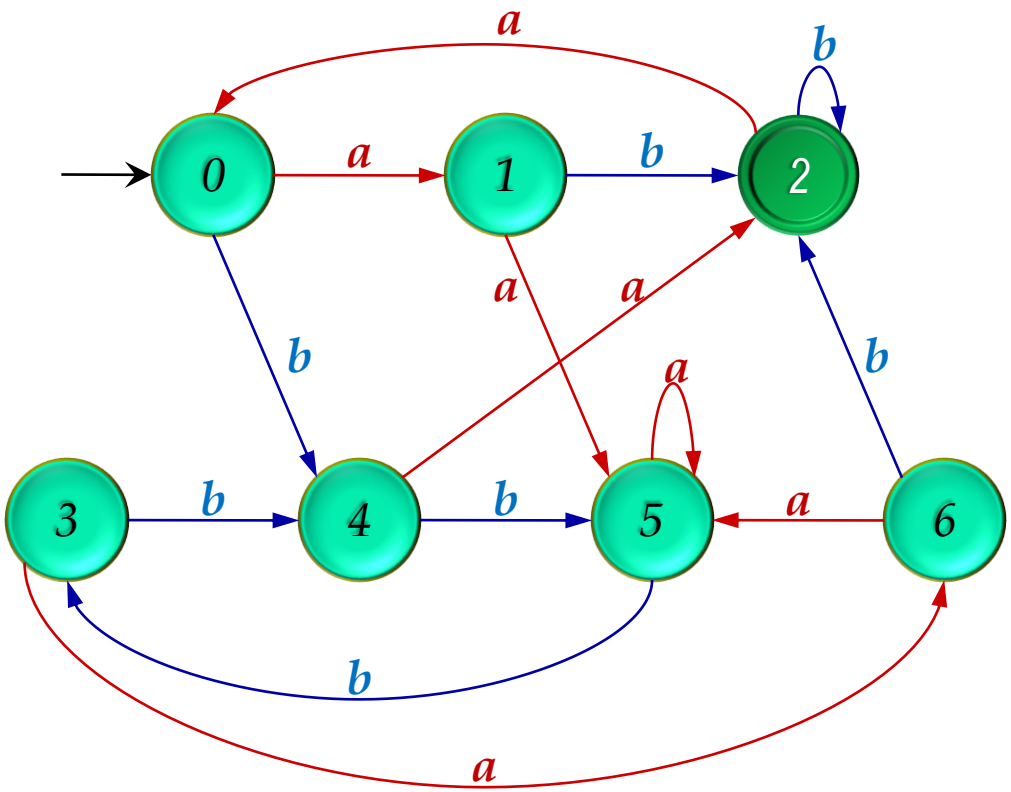
ALGORITHME DE MINIMISATION D'UN AFD

Si on choisit A comme représentant de (AC), et B, C et E pour les groupes singletons, on obtient alors l'automate réduit dont la table de transitions est la suivante:

ÉTAT	SYMBOLE D'ENTRÉE	
	a	b
A,C	B	A,C
B	B	D
D	B	E
E	B	A,C

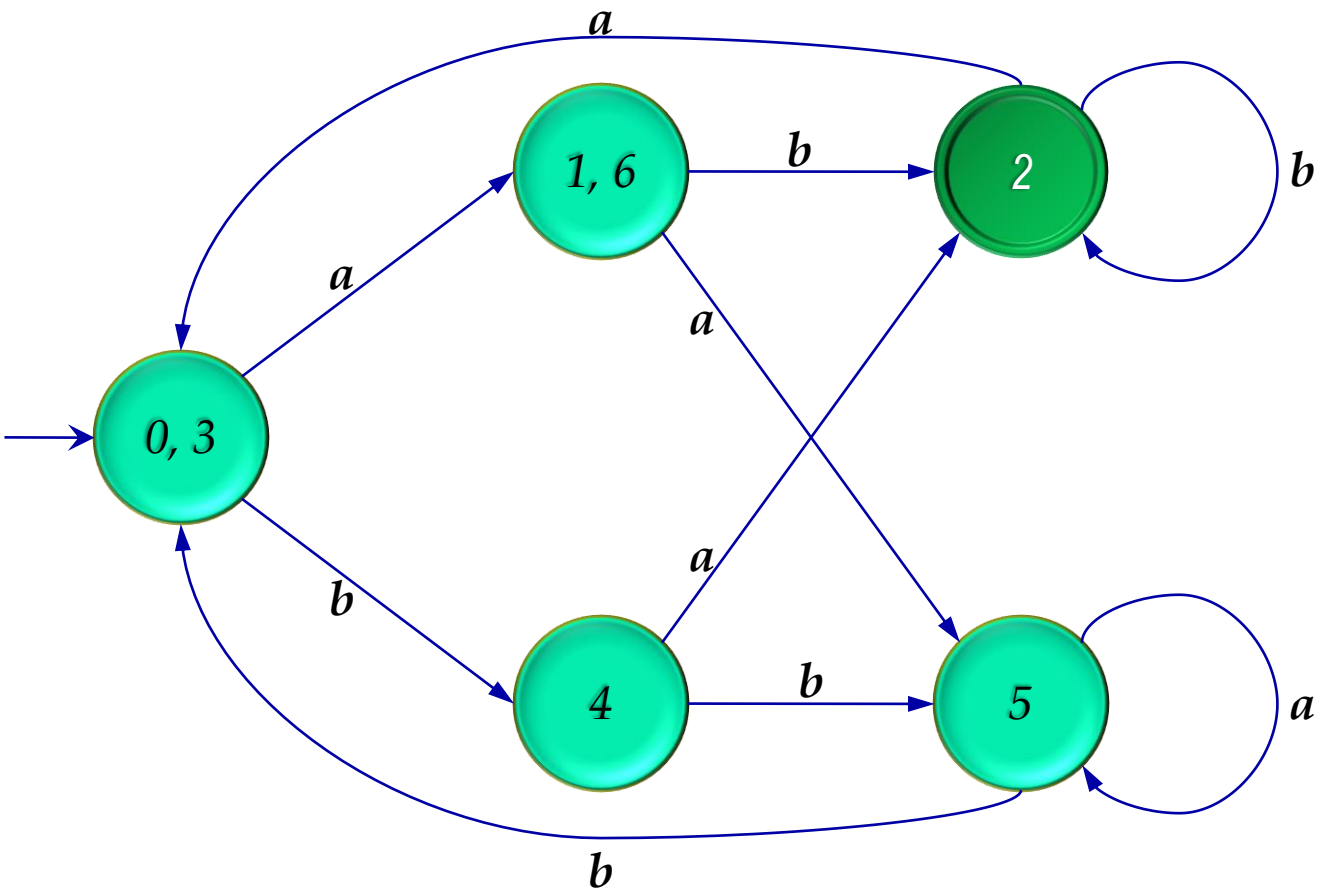
ALGORITHME DE MINIMISATION D'UN AFD

EXERCICE – MINIMISATION DE L'AUTOMATE SUIVANT :

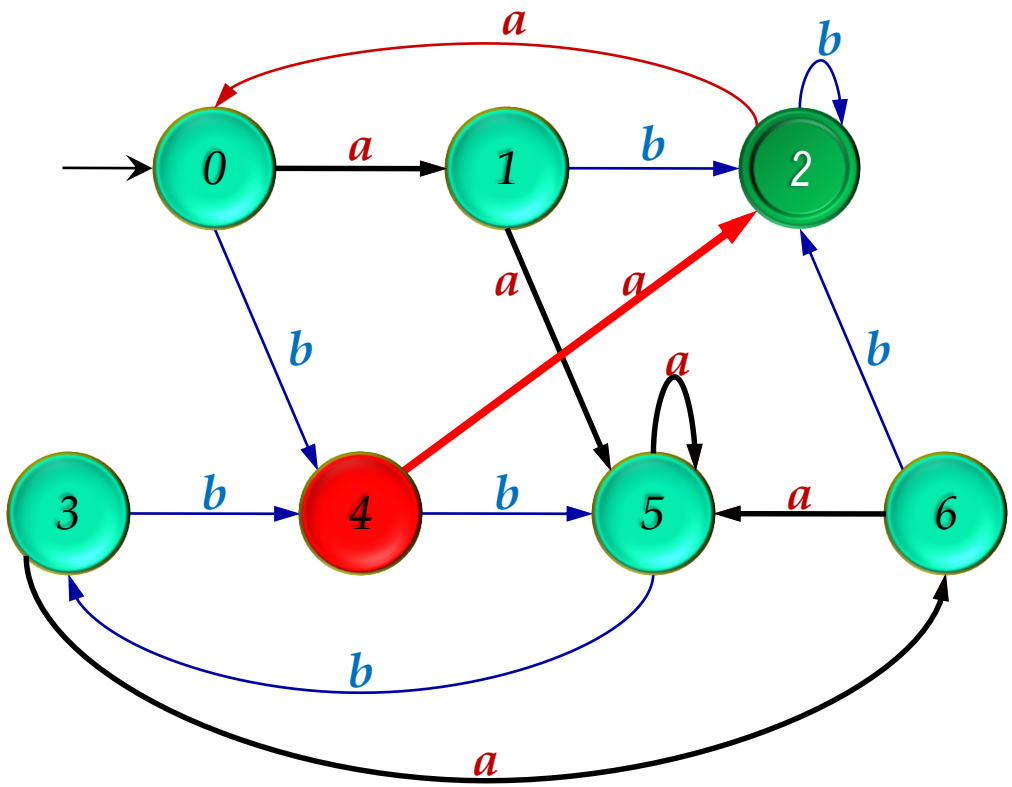


esprit

ALGORITHME DE MINIMISATION D'UN AFD

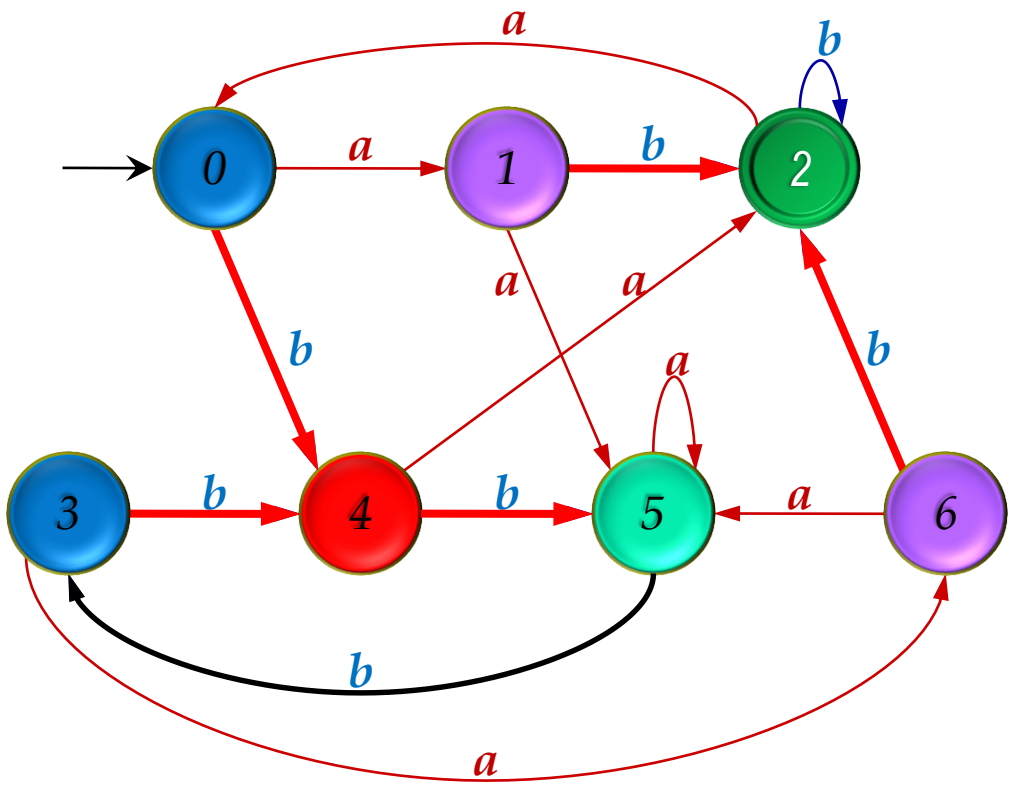


ALGORITHME DE MINIMISATION D'UN AFD



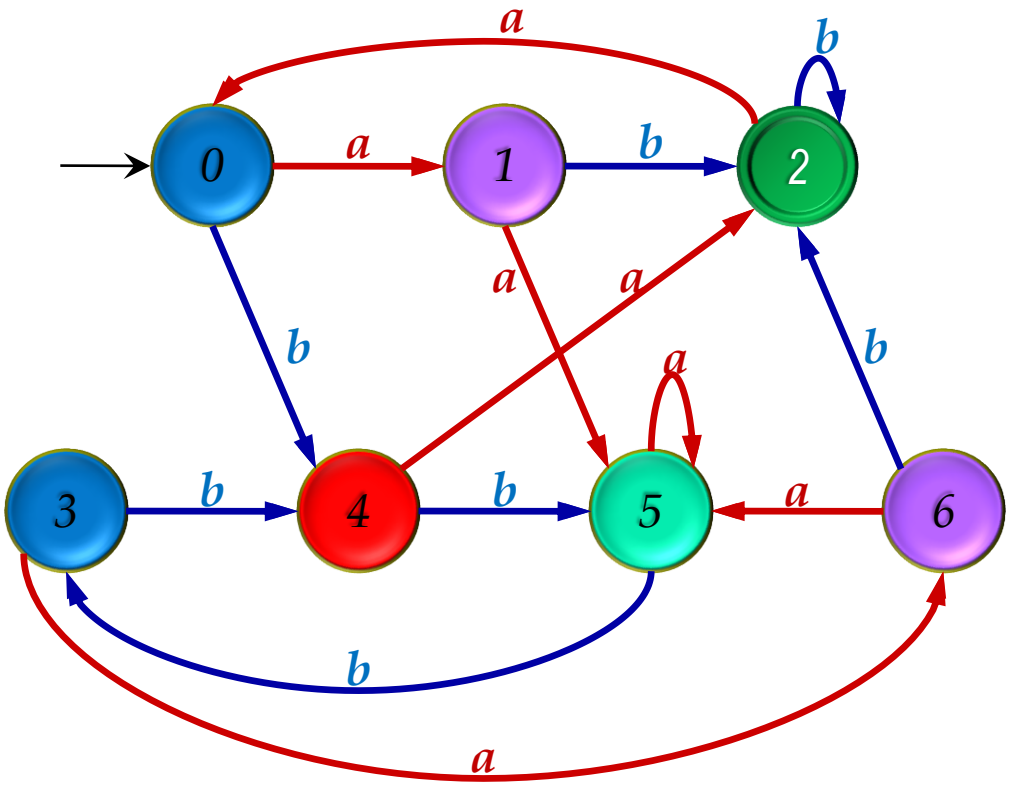
esprit

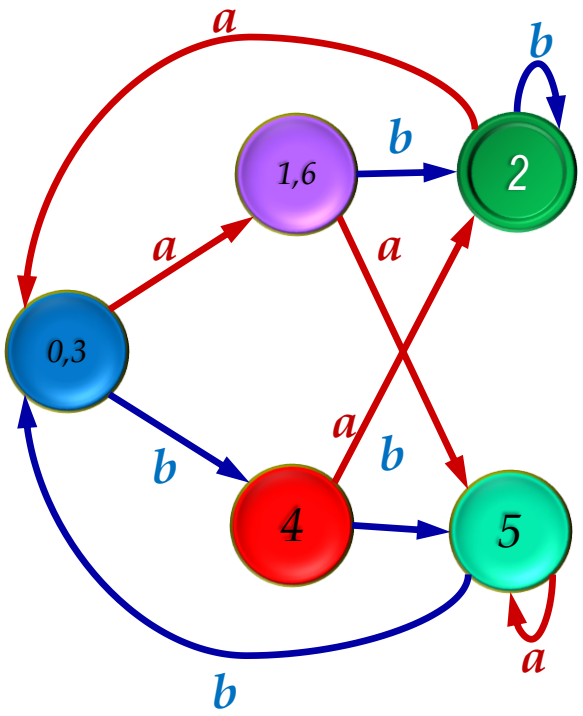
ALGORITHME DE MINIMISATION D'UN AFD



esprit

ALGORITHME DE MINIMISATION D'UN AFD

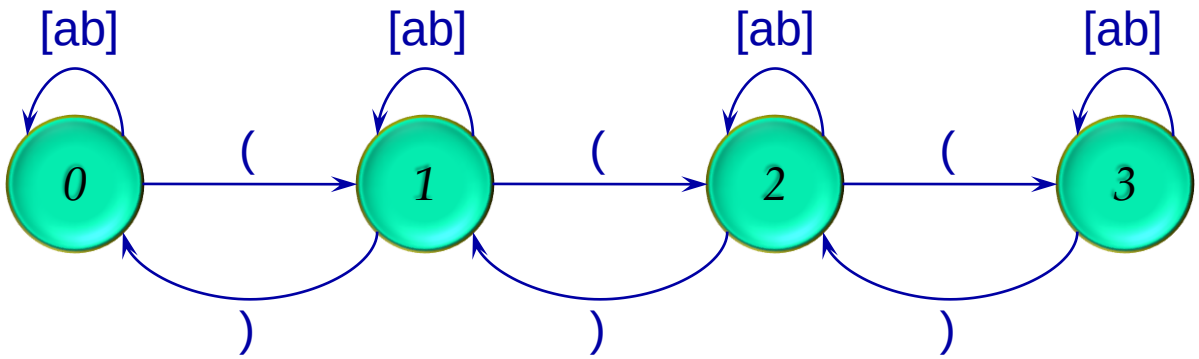




esprit

LIMITE DES AUTOMATES

Les automates ont une limite théorique connue. Par exemple, un automate ne peut pas vérifier qu'un parenthésage est correct jusqu'à une profondeur arbitraire.



abba(aba)ba(aa(babab))aababa((a)(b()aa(aba)a)b)baba

LIMITE DES AUTOMATES

Aucun automate ne peut reconnaître le langage $\{a^n b^n\}$ (Autant de a que de b).

En clair, pour obtenir des mots très long, un automate ne peut que répéter plusieurs fois le même modèle.

Tous les langages ne sont pas reconnaissables par les automates.