IUT de Bordeaux
Département Informatique

M2201 – Graphes et Langages

Éléments de Théorie des Langages

- **Introduction générale : alphabets, mots et langages**
- **Langages rationnels**

Alphabet, mot sur un alphabet (1)

Un **alphabet** est un ensemble (fini) **A** de symboles appelés **lettres**.

Un **mot** m sur un alphabet **A** est une séquence (finie) de lettres prises dans **A** : $\mathbf{m = a_1 \ldots a_k}$. Ce mot est de **longueur** (nombre de lettres) k : $\mathbf{|m| = k}$.

Il existe un unique mot de longueur nulle, le **mot vide**, noté ε.

On note $\mathbf{A}^*$ l'ensemble de tous les mots construits sur l'alphabet **A**. On note $\mathbf{A}^n$ l'ensemble des mots de $\mathbf{A}^*$ de longueur n.

En particulier, $\mathbf{A}^0 = \{ \varepsilon \}$ et $\mathbf{A}^1 = \mathbf{A}$.

Alphabet, mot sur un alphabet (2)

Soit a une lettre de **A** et m un mot de **A***. Le **nombre d'occurrences** de la lettre a dans m, noté $|m|_a$, est le nombre de fois où la lettre a apparaît dans m.

Notons que $|\varepsilon| = 0$ et, pour toute lettre a, $|\varepsilon|_a = 0$.

Exemples.

$\mathbf{A_1} = \{ 0, 1 \}$

$m_1 = 00101011, \quad m_2 = 1101$

$|m_1| = 8, \quad |m_2|_1 = 3$

Alphabet, mot sur un alphabet (3)

$\mathbf{A_2}$ = { a, b, c }

m_3 = baba, m_4 = bac

A_3 = { 0, …, 9, +, -, *, :, (,) }

m_5 = (12 + 4) * (71 – 14:5)

A_4 = { si, alors, sinon, >, a, b, ←, +, 0, 1 , … }

m_6 = si a > b + 1 alors a ← 0 sinon b ← 10

Concaténation de mots

Soient u et v deux mots de **A***. La **concaténation** de u et v est le mot, noté u.v ou plus simplement uv, obtenu en « collant » le mot v à la suite du mot u. Ainsi, $|uv| = |u| + |v|$ et, pour toute lettre a, $|uv|_a = |u|_a + |v|_a$.

On notera u^n le mot **u.u. … .u** (n fois), avec $u^0 = \varepsilon$.

Exemple. u = aba, v = ca, uv = abaca, u^2 = abaaba

$|u| = 3$, $|v| = 2$, $|uv| = 5$, $|u^2| = 6$

Pour tous mots u, v et w, nous avons :

- **εu = uε = u** (ε est élément neutre)
- **u.vw = uv.w = uvw** (associativité)
- mais, en général, **uv ≠ vu** (non commutativité)

Préfixes, suffixes et facteurs

Soient u et v deux mots de **A***.

Le mot u est un **préfixe** du mot v s'il existe un mot w de **A*** tel que v = uw.

De façon similaire, le mot w est un **suffixe** du mot v s'il existe un mot u de **A*** tel que v = uw.

Le mot u est un **facteur** du mot v s'il existe deux mots w_1 et w_2 de **A*** tels que $v = w_1uw_2$.

Exemple. u = aba, v = abac, w = abacabac, t = aca

- u est un *préfixe* de w, car w = u.cabac
- v est un *suffixe* de w, car w = abac.v
- t est un *facteur* de w, cat w = ab.t.bac

Quelques propriétés…

Propriété. Si u, v et w sont trois mots de A*, alors uw = vw $\Leftrightarrow$ wu = wv $\Leftrightarrow$ u = v

Lemme de Levi. Si u et v sont tous deux préfixes de w, alors u est préfixe de v, ou v est préfixe de u.

Théorème (de commutation). Si u et v commutent (c'est-à-dire sont tels que uv = vu), alors u et v sont deux *puissances* d'un même facteur :

$\Rightarrow$ il existe un mot f de **A*** et deux entiers p et q tels que $u = f^p$ et $v = f^q$.

Preuve du théorème de commutation…

- Si $u = f^p$ et $v = f^q$, alors $uv = vu = f^{p+q}$.
- Réciproque : par récurrence sur $N = |u| + |v|$:
  - si $N = 0$, alors $u = v = \varepsilon$ et donc $uv = vu$.
  - si $u = \varepsilon$, alors $u = v^0$ et $v = v^1$ (idem si $v = \varepsilon$).
  - si $|u| = |v|$, alors $uv = vu$ entraîne $u = v$
    d'où $f = u$, $p = q = 1$.
  - sinon, comme u et v sont préfixes de $uv = vu$, l'un est préfixe de l'autre (Lemme de Levi).

    Supposons que u est préfixe de v : $v = uw$.

    On a donc $uv = vu \Rightarrow u(uw) = (uw)u$, c'est-à-dire
    $uuw = uwu$, et donc $uw = wu$ (propriété).

    L'hypothèse de récurrence permet de conclure.

Langages (1)

Un **langage** L sur un alphabet **A** est un sous-ensemble, fini ou infini, de **A***.

Par exemple, sur l'alphabet **A** = {a, b}, on peut définir les langages suivants :

- $L_1 = \mathbf{A}^2 = \{ aa, ab, ba, bb \}$,
- L_2 = { mots d'au plus quatre lettres ayant autant de a que de b }

  = {ε, ab, ba, aabb, abab, abba, baab, baba, bbaa},
- L_3 = { mots ayant deux fois plus de a que de b } (ce langage est infini).

Langages (2)

Autres exemples :

- A = { a, …, z }

  L = { mots de la langue française }

- A = { mots de la langue française }

  L = { phrases correctes }

- A = { … }

  L = { programmes C++ syntaxiquement corrects }

- etc.

Opérations sur les langages (1)

Soit **A** un alphabet. On définit les opérations suivantes sur les langages définis sur **A*** :

Union.

$$L_1 \cup L_2 = \{ m \in \mathbf{A}^* / (m \in L_1) \text{ ou } (m \in L_2) \}$$

Intersection.

$$L_1 \cap L_2 = \{ m \in \mathbf{A}^* / (m \in L_1) \text{ et } (m \in L_2) \}$$

Produit (de concaténation).

$$L_1.L_2 = L_1L_2 = \{m \in \mathbf{A}^* / m = m_1m_2,\ m_1 \in L_1 \text{ et } m_2 \in L_2\}$$

Opérations sur les langages (2)

Puissance.

$$L^0 = \{ \varepsilon \}$$

$$L^n = \{ m \in \mathbf{A}^* / m = m_1 m_2 ... m_n,$$

$$\text{avec } m_i \in L \text{ pour tout } i,\ 1 \leq i \leq n \}, \text{ pour } n \geq 1$$

Étoile et "plus".

$$L^* = L^0 \cup L^1 \cup ... \cup L^n \cup ...$$

$$L^+ = L^1 \cup L^2 \cup ... \cup L^n \cup ...$$

$$(\text{i.e. } L^+ = L.L^*)$$

Langages rationnels

Langages rationnels (1)

Un langage sur un alphabet **A** est **rationnel** (on dit également **régulier**), s'il peut être construit à l'aide des opérations union, étoile et produit à partir des langages élémentaires composés du mot vide ou d'un mot à une seule lettre.

De façon inductive, un L.R. se définit donc ainsi :

- $\{ \varepsilon \}$ est un L.R.,
- si $a \in \mathbf{A}$, $\{ a \}$ est un L.R.,
- si L est un L.R. alors L^* est un L.R. (L^+ et L^n aussi),
- si L_1 et L_2 sont des L.R., alors $L_1 \cup L_2$ et $L_1.L_2$ sont des L.R.

Langages rationnels (2)

Soit **A** = { a, b, c }. Les langages suivants sont rationnels :

- $L_1 = (\{ \varepsilon \} \cup \{ a \}).(\{ b \} \cup \{ c \})^*.\{ a \}^3$

  $L_1 = \{ aaa, bcbbcaaa, ..., aaaa, abaaa, ... \}$

- $L_2 = \{ c \}^2.(\{ a \} \cup \{ b \}^* \cup (\{ a \} \cup \{ b \}.\{ c \})^2)^3$

  $L_2 = \{ ccaaa,cc,cca,ccaaaaaa,ccbcabcabcbc, ... \}$

- $L_3 = (\{ a \} \cup \{ b \})^*.\{ c \}$

  $L_3 = \{ c, ac, bc, aac, abc, bac, bbc, aaac, ... \}$

Expressions rationnelles

On utilise habituellement quelques conventions d'écriture qui permettent d'alléger les expressions précédentes :

- un langage singleton **{ a }** est noté simplement **a**,
- l'union est notée simplement **+**.

Ainsi, les langages précédents peuvent s'écrire, plus simplement, sous forme de ce que l'on appelle des **expressions rationnelles** :

- $L_1 = (\varepsilon + a)(b + c)^* a^3$
- $L_2 = c^2(a + b^* + (a + bc)^2)^3$
- $L_3 = (a + b)^* c$

IUT de Bordeaux
Département Informatique

M2201 – Graphes et Langages

Éléments de Théorie des Langages

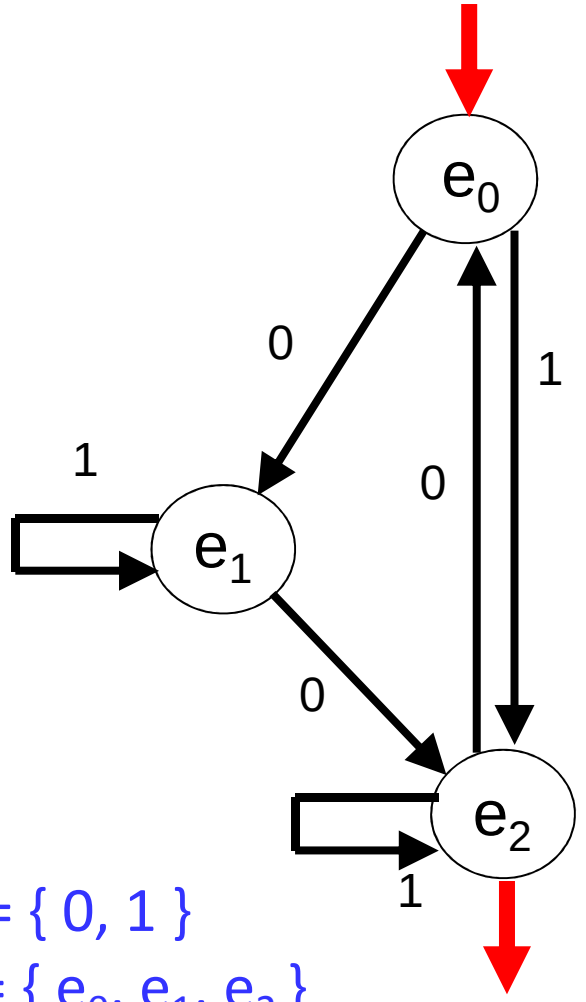
- **Automates**
- **Simulation du fonctionnement d'un automate fini déterministe**
- **Langages reconnaissables**

Automates

Automates finis déterministes

Un **automate** fini, **déterministe** (ou **AFD**) est un 5-uplet
$< \Sigma, E, e_0, T, \delta >$ où :

- Σ est **l'alphabet d'entrée**,
- E est un ensemble fini d'**états**,
- $e_0 \in$ E est l'**état initial,**
- $T \subseteq E$ est l'ensemble des **états terminaux**,
- $\delta : E \times \Sigma \rightarrow E$ est la **fonction de transition**.



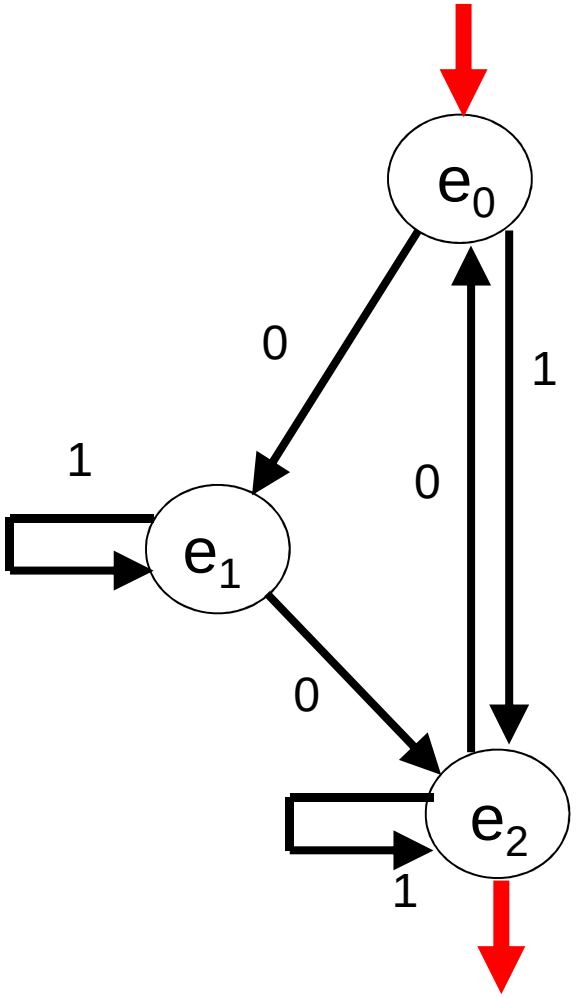
$\Sigma = \{ 0, 1 \}$
$E = \{ e_0, e_1, e_2 \}$
$T = \{ e_2 \}$
$\delta = \{ (e_0,0,e_1), \ldots \}$

Langage reconnu par un AFD (1)

On peut étendre la fonction δ aux mots en posant :

$\delta(e,\varepsilon) = e$

$\delta(e,m) = \delta(\delta(e,a),m')$, si $m = am'$



Soit m = 1101001

$\delta(e_0,m) = e_1$

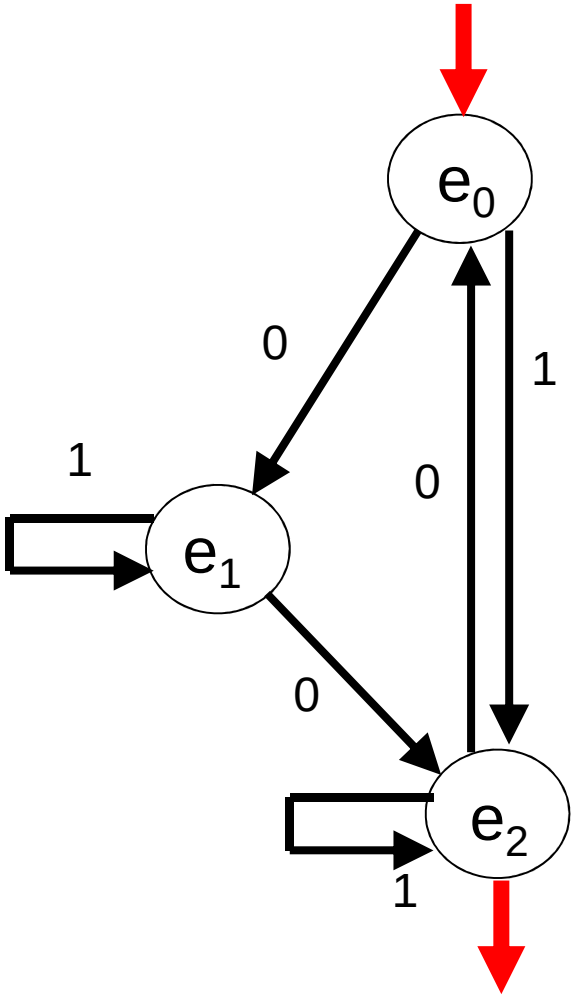
	1	1	0	1	0	0	1
e_0	e_2	e_2	e_0	e_2	e_0	e_1	e_1

Langage reconnu par un AFD (2)

On peut étendre la fonction δ aux mots en posant :

$\delta(e,\varepsilon) = e$

$\delta(e,m) = \delta(\delta(e,a),m')$, si $m = am'$



Le **langage reconnu** par un AFD **M** est alors :

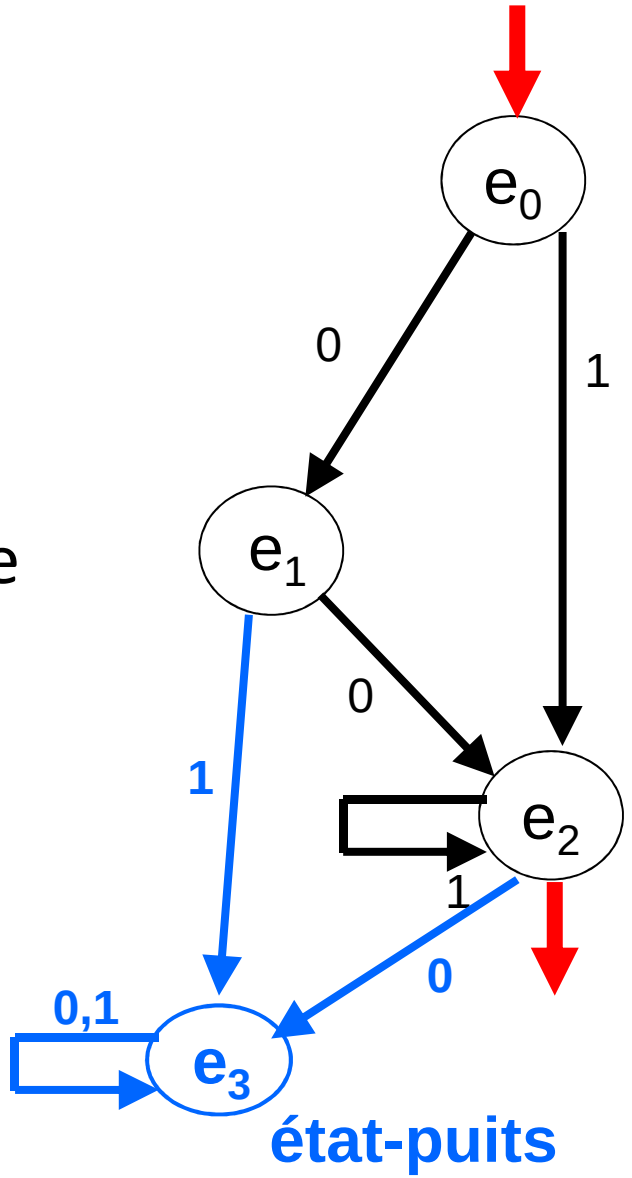
$$L(\mathbf{M}) = \{ m \in \Sigma^* / \delta(e_0,m) \in T \},$$

soit l'ensemble des mots de Σ^* qui conduisent de l'état initial à l'un des états terminaux.

État-puits

Pour simplifier, on s'autorise parfois à ne faire figurer que les transitions *utiles*.

Pour certains états, il existe alors des lettres pour lesquelles nous n'avons aucune transition : dans ce cas, par convention, les transitions non indiquées mènent toutes vers un état, non terminal, d'où l'on ne sort jamais : l'**état-puits**.

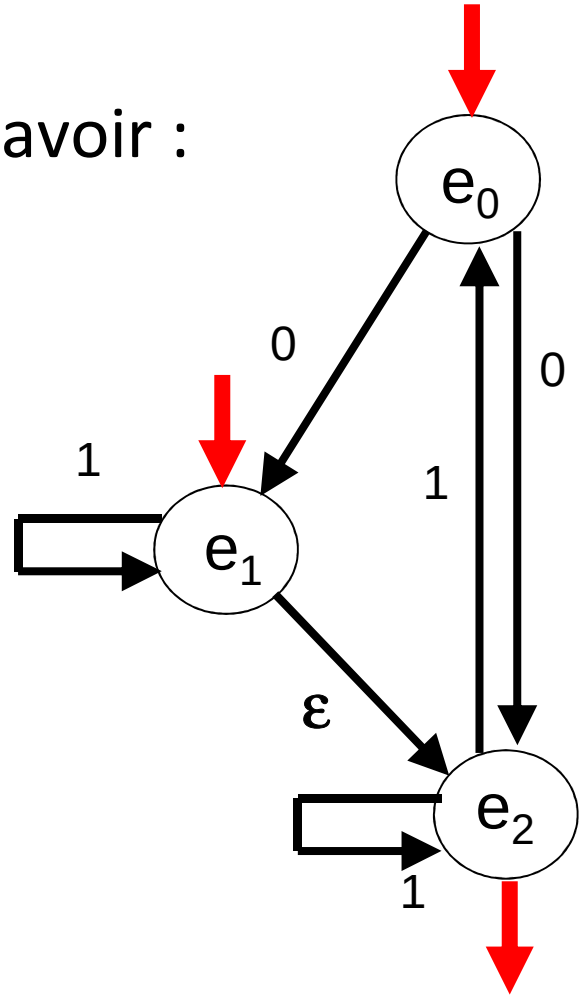


Automates non déterministes (1)

Il est parfois plus simple de produire un automate **non déterministe**.

Pour un tel automate, il est possible d'avoir :

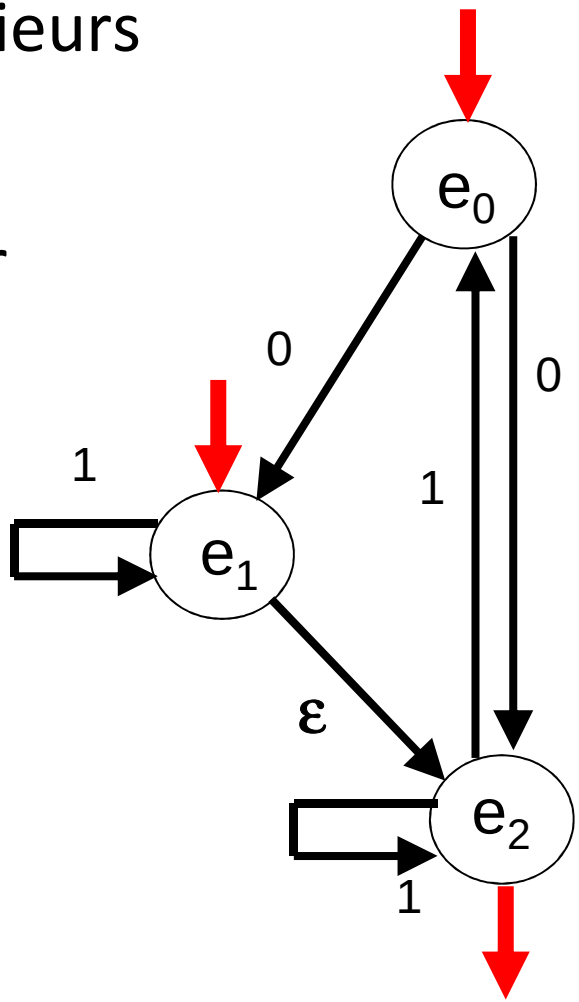
- plusieurs états initiaux,
- plusieurs transitions issus d'un même état et portant la même étiquette,
- des transitions étiquetées par ε (appelées ε-transitions)



Automates non déterministes (2)

Pour un même mot en entrée, un automate fini non déterministe (AF) peut donc avoir plusieurs comportements distincts…

On définit alors le langage reconnu par un AF comme l'ensemble des mots pour lesquels **il existe** (au moins) un comportement conduisant à un état terminal…



*Nous verrons plus tard qu'il est toujours possible de produire un AFD **équivalent** (i.e. **reconnaissant le même langage**).*

Simulation du fonctionnement d'un automate fini déterministe

Représentation d'un AFD (1)

Pour représenter un automate, il est nécessaire de mémoriser :

- l'alphabet Σ,
- l'état initial e_0 (on peut supposer qu'il s'agit de l'état 0 par convention),
- la fonction de transition **delta**, c'est-à-dire les triplets (e_i, a, e_j) pour tout état e_i et toute lettre $a \in \Sigma$,
- l'ensemble **T** des états terminaux.

Représentation d'un AFD (2)

- **l'alphabet Σ**

  On range les lettres dans un tableau Σ, l'indice de la case contenant une lettre devenant le « code » de la lettre, et on mémorise le cardinal de Σ dans une variable **cardΣ** :

Σ

0	1	2	3	4
a	b	c		

cardΣ

3

Représentation d'un AFD (3)

- **la fonction de transition δ**

  On utilise un tableau **delta** à deux dimensions, donnant un état pour chaque couple (état, lettre), et une variable **étatPuits** :

delta

états	0	1	2
0	0	1	2
1	0	1	1
2	4	1	3
3	3	0	4
4	4	4	4

alphabet (code)

étatPuits

4

- **les états terminaux**

  Un tableau **T** de booléens…

Représentation d'un AFD (4)

Σ

0	1	2	3	4
a	b	c		

cardΣ

3

T

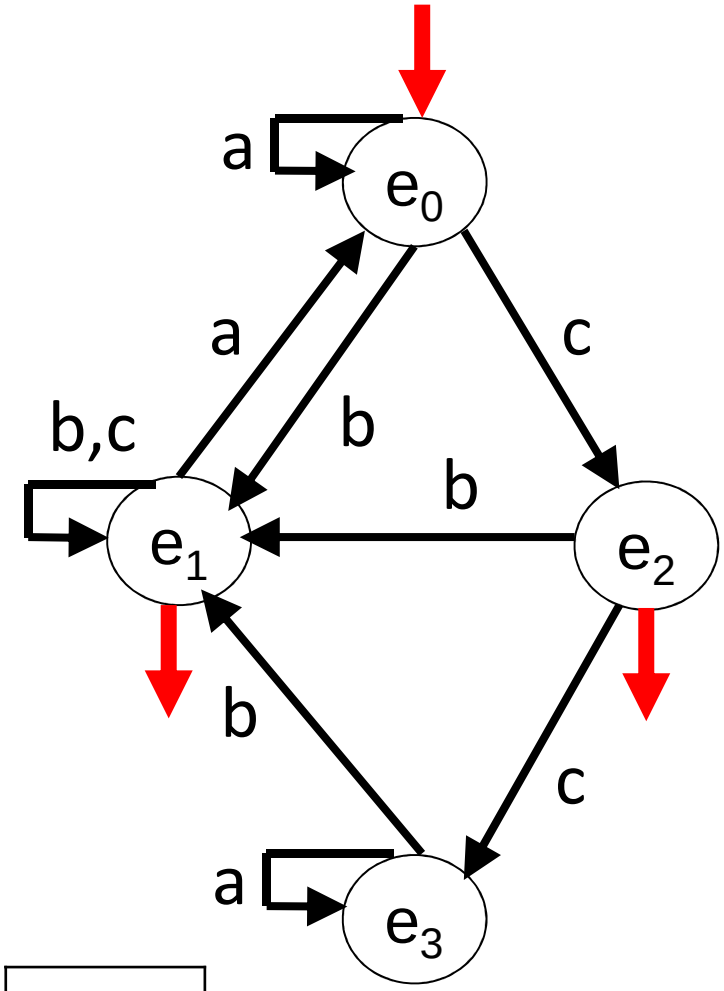
0	F
1	V
2	V
3	F
4	F

delta

	0	1	2
0	0	1	2
1	0	1	1
2	4	1	3
3	3	1	4
4	4	4	4

étatPuits

4



Simulation (algorithme)

Il est ensuite aisé de simuler le fonctionnement de l'AFD :

```
…
lire (car) ; état ← 0
Tant Que { non caractère de fin } Faire
    codecar ← ChercheCode (Σ, cardΣ, car)
    Si codecar = -1
        Alors état ← étatPuits
        Sinon état ← delta [état] [codecar]
    lire (car)
Si T [état]
    Alors écrire ("OUI")
    Sinon écrire (« NON")
```

Langages reconnaissables

Langages reconnaissables

Un langage $L \subseteq \mathbf{A}^*$ est **reconnaissable** si et seulement si il existe un AFD $M = < \Sigma, E, e_0, T, \delta >$ avec $\Sigma = \mathbf{A}$ tel que $L(M) = L$.

Ainsi, par exemple, les langages suivants sont reconnaissables :

- les langages finis,
- $\mathbf{A}^*$ et $\mathbf{A}^+$, pour tout alphabet **A**,
- les mots contenant un certain motif (lettres consécutives),
- les mots ne contenant pas un certain motif,
- …

Théorème de Kleene

En fait, tout langage rationnel (défini par une expression rationnelle) est reconnaissable et… réciproquement !

C'est le théorème de Kleene :

Théorème de Kleene. Un langage est rationnel si et seulement si il est reconnaissable.

Pour démontrer ce théorème nous allons montrer :

- *comment construire un automate reconnaissant un langage rationnel (à partir de l'expression rationnelle),*
- *comment construire une expression rationnelle définissant le langage reconnu par un automate.*

Kleene : construction de l'AFD

Rappel : de façon inductive, un langage rationnel (LR) se définit donc ainsi :

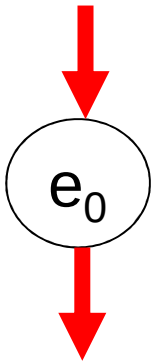
- **{ ε }** est un LR,
- si a ∈ **A**, **{ a }** est un LR,
- si **L** est un L.R. alors **L*** est un LR (**L^+** et **L^n** aussi),
- si **L_1** et **L_2** sont des LR, alors **$L_1 \cup L_2$** et **$L_1.L_2$** sont des LR.

Ainsi, il suffit de savoir construire :

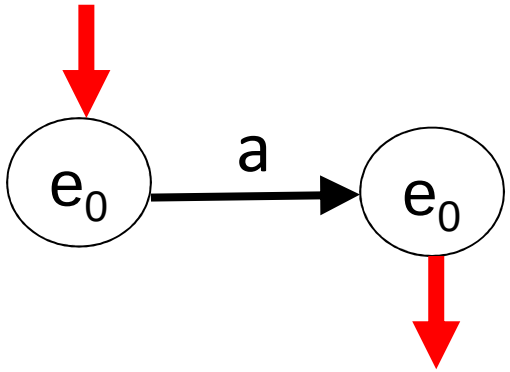
- *des AFD reconnaissant les langages { ε } et { a } pour tout a ∈ **A**,*
- *et, à partir d'AFD reconnaissant les langages L_1 et L_2, des AFD reconnaissant L_1*, $L_1 \cup L_2$ et $L_1.L_2$…*

AFD reconnaissant { ε } et { a }

Aucune difficulté !



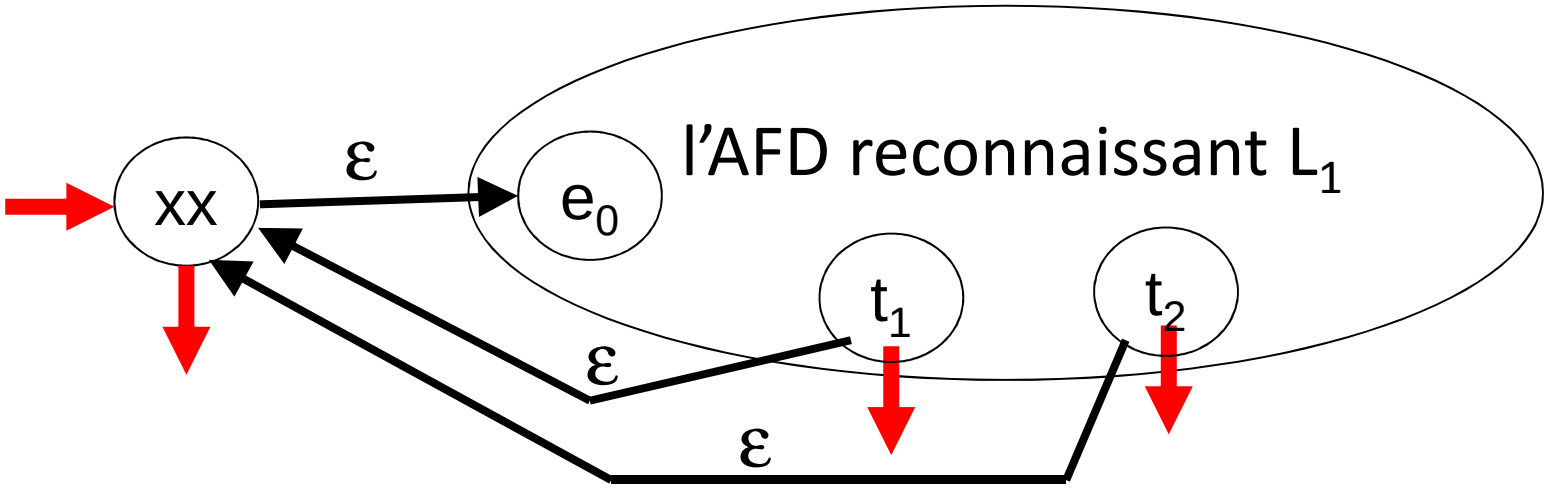
l'AFD $\mathbf{M_\varepsilon}$



l'AFD $\mathbf{M_a}$

AFD reconnaissant L_1*

Pas très compliqué… si l'on produit dans un premier temps un automate non déterministe !

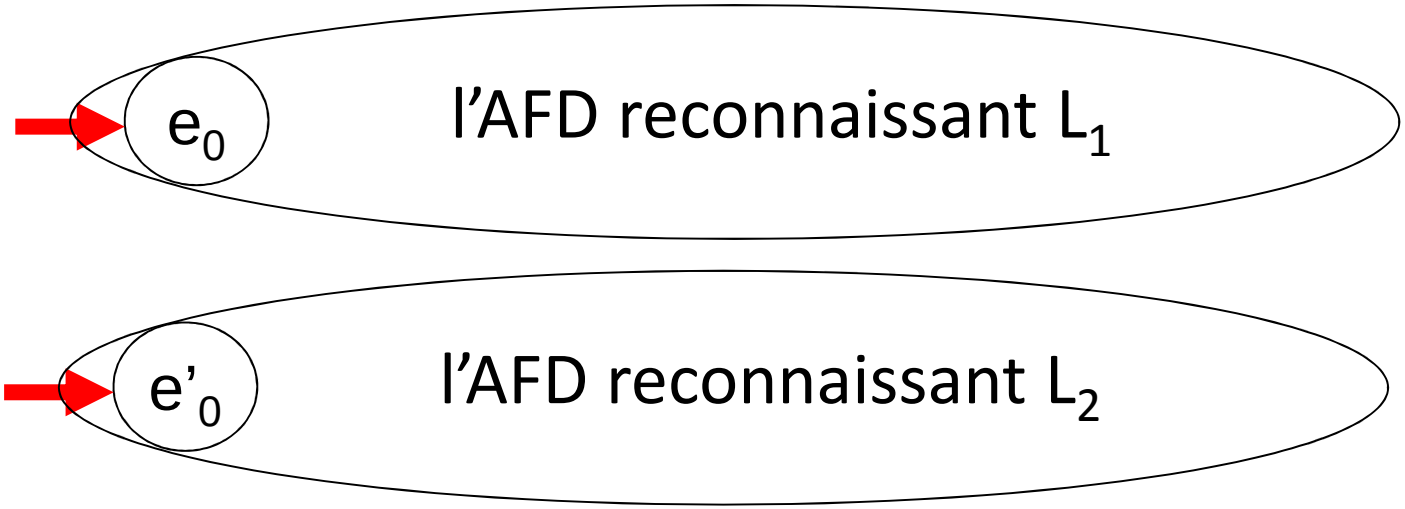


l'AF $\mathbf{M_{L1*}}$

Il suffit ensuite d'utiliser l'algorithme de déterminisation…

AFD reconnaissant $L_1 \cup L_2$

Vraiment pas difficile… si l'on produit dans un premier temps un automate non déterministe !

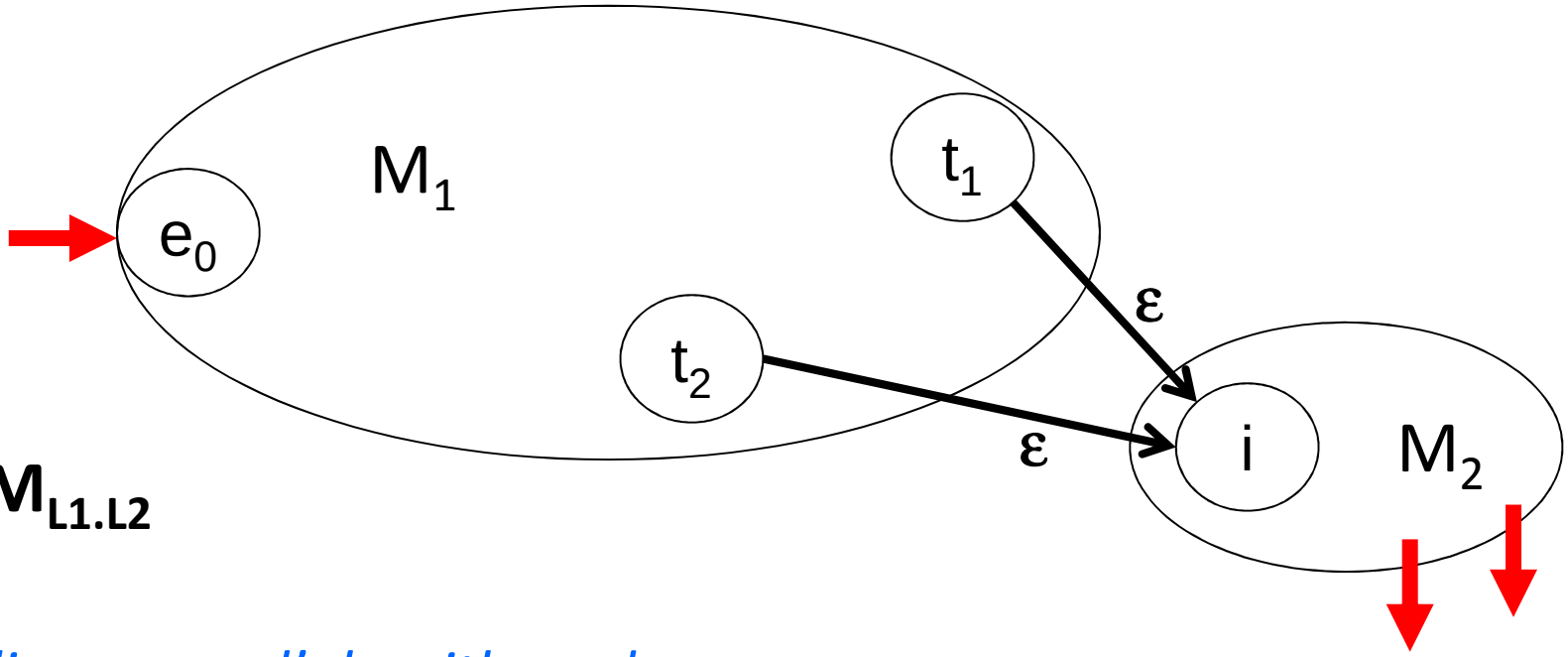


l'AF $\mathbf{M_{L1 \cup L2}}$

Il suffit ensuite d'utiliser l'algorithme de déterminisation…

AFD reconnaissant $L_1.L_2$

Facile : pour chaque état terminal de M_1, on enlève la « flèche de sortie », et on le relie par une ε–transition avec l'état initial de l'automate M_2 (qui n'est plus « initial ») :



l'AF $\mathbf{M_{L1.L2}}$

On utilise encore l'algorithme de déterminisation…

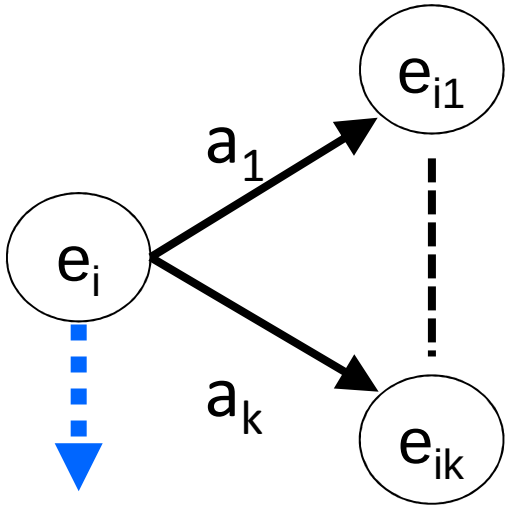
Expression rationnelle pour un AFD (1)

L'idée de base est la suivante :

- À chaque état $\mathbf{e_i}$ de l'AFD, correspond le langage $\mathbf{L_i}$ des mots qui conduisent de l'état $\mathbf{e_i}$ à un état terminal de l'AFD ; on va déterminer une expression rationnelle $\mathbf{E_i}$ décrivant ce langage $\mathbf{L_i}$ pour tout état $\mathbf{e_i}$.
- L'expression rationnelle **E** décrivant le langage reconnu par l'AFD est alors $\mathbf{E_0}$, si l'état initial de l'AFD est $\mathbf{e_0}$…

Expression rationnelle pour un AFD (2)

Pour trouver les expressions rationnelles $\mathbf{E_i}$, on construit un **système d'équations** portant sur les langages $\mathbf{L_i}$, système que l'on va ensuite résoudre.



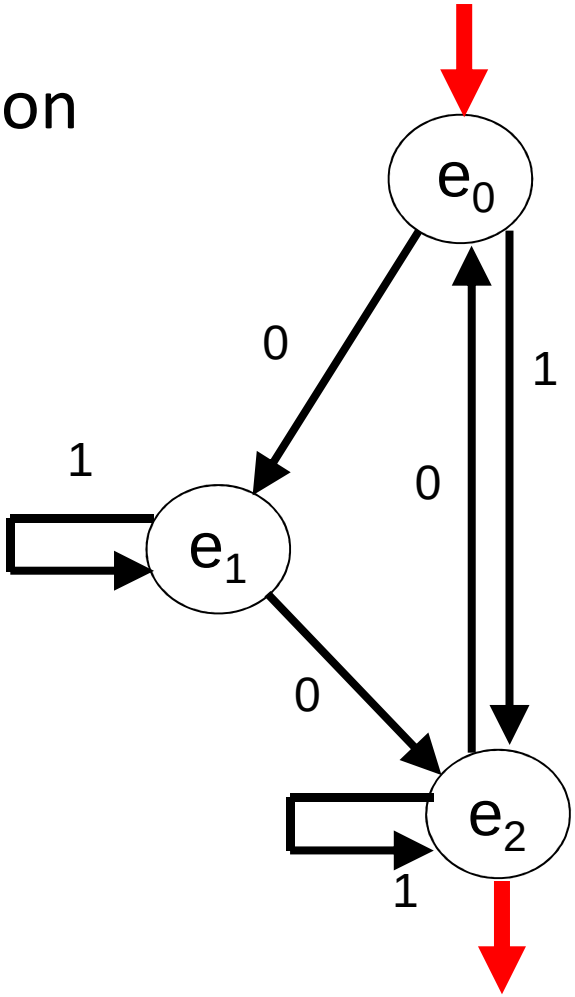
Un mot **u** conduisant de l'état $\mathbf{e_i}$ à un état terminal est nécessairement de la forme $\mathbf{u = a_j u_{ij}}$, où $\mathbf{u_{ij}}$ est un mot conduisant de l'état $\mathbf{e_{ij}}$ à un état terminal [ou $\mathbf{u = \varepsilon}$ si $\mathbf{e_i}$ est terminal].

D'où l'équation :

$$\mathbf{L_i = [\ \varepsilon + \ ]\ a_1.L_{i1} + \ldots + a_k.L_{ik}}$$

Résolution du système d'équations

Pour résoudre le système d'équations, on va procéder de façon classique par substitution / élimination.



- $L_0 = 0.L_1 + 1.L_2$
- $L_1 = 0.L_2 + 1.L_1$
- $L_2 = \varepsilon + 0.L_0 + 1.L_2$

$\Rightarrow$ - $L_2 = \varepsilon + 00.L_1 + (01 + 1).L_2$

et ensuite ?...

Lemme d'Arden

Le lemme d'Arden va nous permettre de compléter la résolution de ce système :

Lemme d'Arden (1960). Si L, A et B sont trois langages satisfaisant l'équation

$$\mathbf{L = A.L + B},$$

alors **L = A*B** est le plus petit langage (au sens de l'inclusion) solution de l'équation. De plus, si A ne contient pas le mot vide, cette solution est unique.

- $L_2 = \varepsilon + 00.L_1 + (01 + 1).L_2$

$\Rightarrow$

- $L_2 = (01 + 1)^*(\varepsilon + 00.L_1)$

Résolution du système d'équations (2)

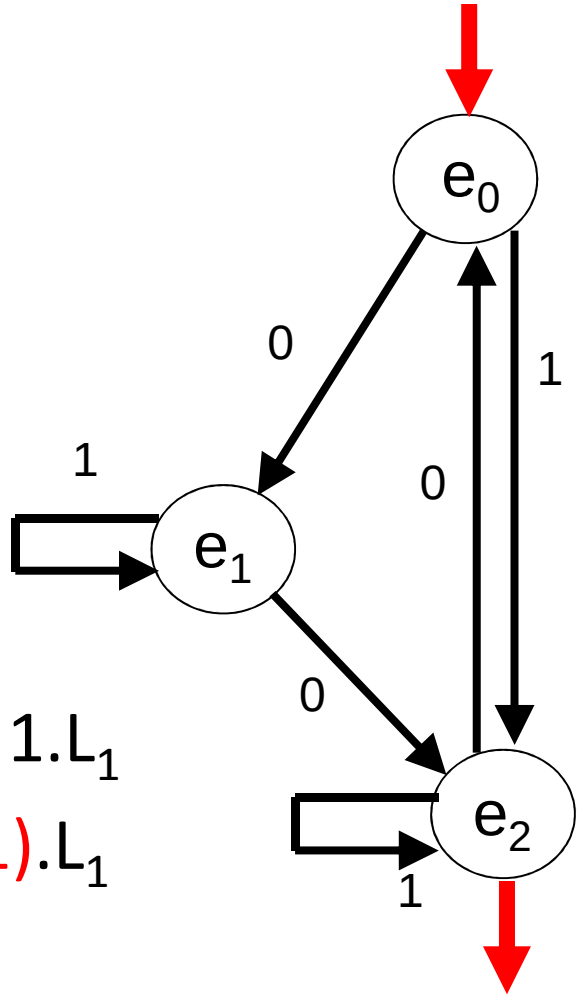
- $L_0 = 0.L_1 + 1.L_2$
- $L_1 = 0.L_2 + 1.L_1$
- $L_2 = \varepsilon + 0.L_0 + 1.L_2$

$\Rightarrow$

- $L_2 = \varepsilon + 00.L_1 + (01+1).L_2$
- $L_2 = (01+1)^*(\varepsilon + 00.L_1)$
  $= (01+1)^* + (01+1)^*00.L_1$

$\Rightarrow$

- $L_1 = 0(01+1)^* + 0(01+1)^*00.L_1 + 1.L_1$
  $= 0(01+1)^* + (0(01+1)^*00 + 1).L_1$
- $L_1 = (0(01+1)^*00 + 1)^*0(01+1)^*$

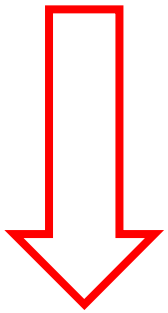


$\Rightarrow$

- $L_2 = (01+1)^* + (01+1)^*00(0(01+1)^*00+1)^*0(01+1)^*$

Résolution du système d'équations (3)

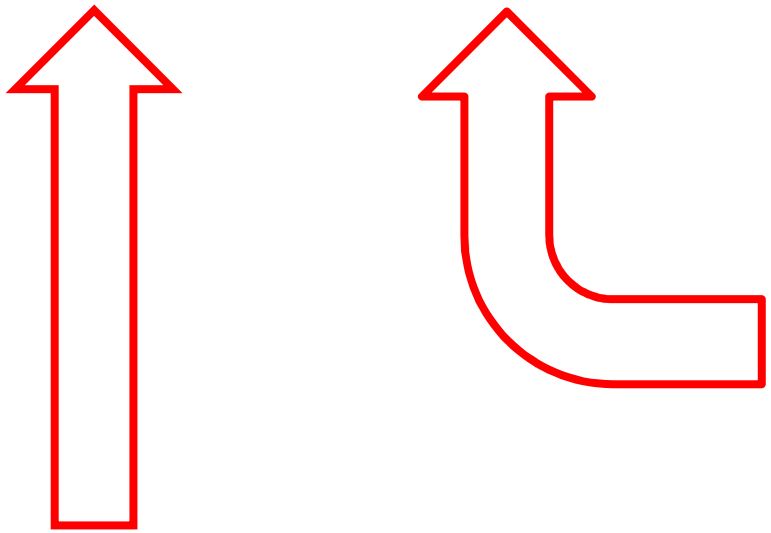
- $L_0 = 0.L_1 + 1.L_2$
- $L_1 = (0(01+1)^*00 + 1)^*0(01+1)^*$
- $L_2 = (01+1)^* + (01+1)^*00(0(01+1)^*00+1)^*0(01+1)^*$

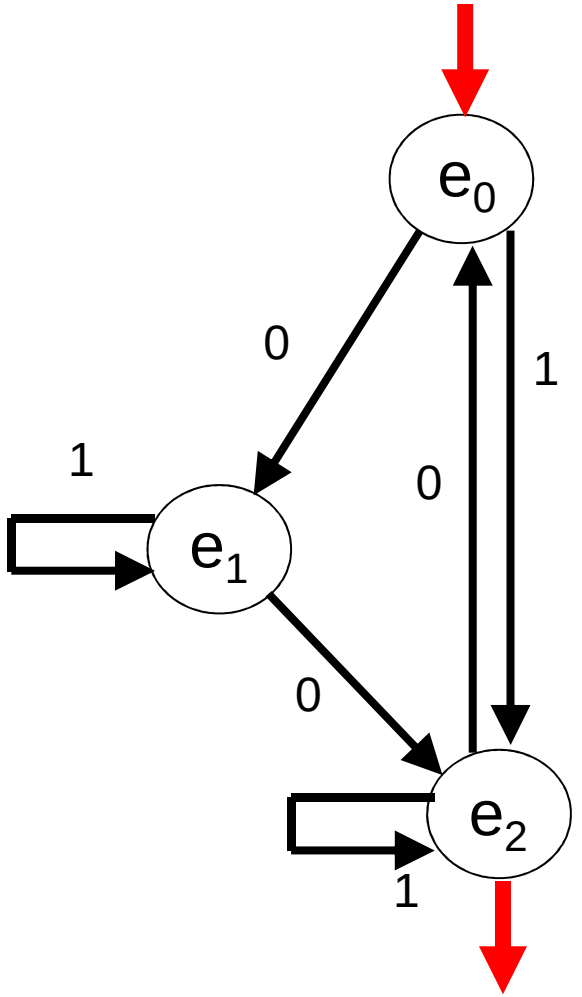


- $L_0 = 0.(0(01+1)^*00 + 1)^*0(01+1)^*$
  $+ 1.(01+1)^* + (01+1)^*00(0(01+1)^*00+1)^*0(01+1)^*$

Résolution du système d'équations (4)

- $L_0 = (1 + 01*0)1*(0(1 + 01*0)1*)*$





- $L_0 = 0.(0(01+1)*00 + 1)*0(01+1)*$
  $+ 1.(01+1)* + (01+1)*00(0(01+1)*00+1)*0(01+1)*$

IUT de Bordeaux
Département Informatique

M2201 – Graphes et Langages

Éléments de Théorie des Langages

- **Déterminisation d'un AF**
- **Minimisation d'un AFD**

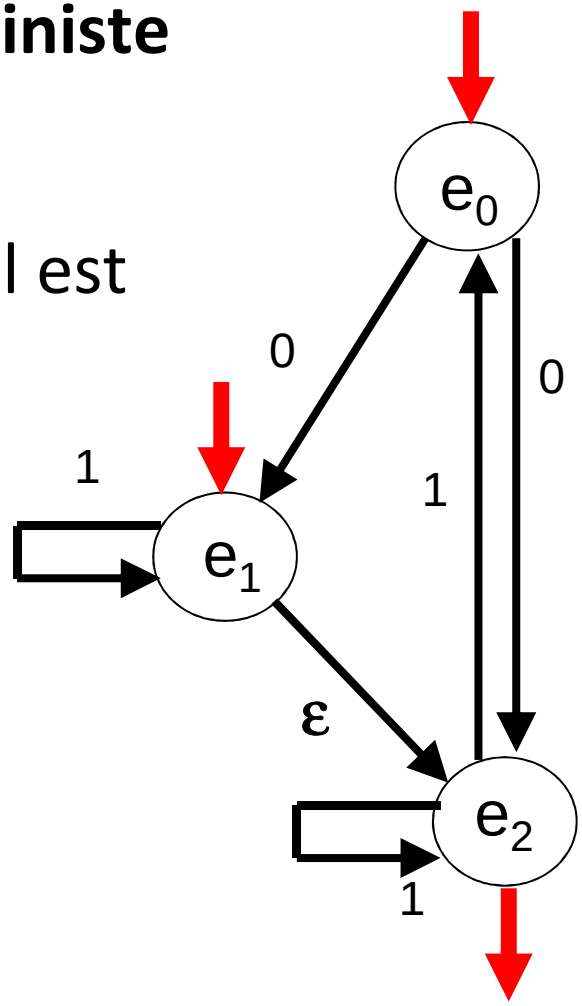
Déterminisation d'un automate fini

Automates non déterministes (rappel)

Nous avons vu qu'il était parfois plus simple de produire un automate fini **non déterministe (AF)**.

Rappelons que pour un tel automate, il est possible d'avoir :

- plusieurs états initiaux,
- plusieurs transitions issus d'un même état et portant la même étiquette,
- des transitions étiquetées par ε (appelées ε-transitions)



Équivalence entre AFs et AFDs

Nous allons démontrer que les AF ne sont pas plus « puissants » que les AFD : les langages reconnus par les AF sont exactement les mêmes que ceux reconnus par les AFD.

Comme tout AFD est un AF, il suffit de montrer la proposition suivante :

Proposition. Pour tout automate fini **M**, il existe un automate fini déterministe **M'** tel que L(**M'**) = L(**M**).

*Pour démontrer ce résultat, nous allons donner un algorithme qui, à partir d'un AF **M**, construit un AFD **M'** équivalent…*

Déterminisation d'un AF : principe

L'idée de base est la suivante :

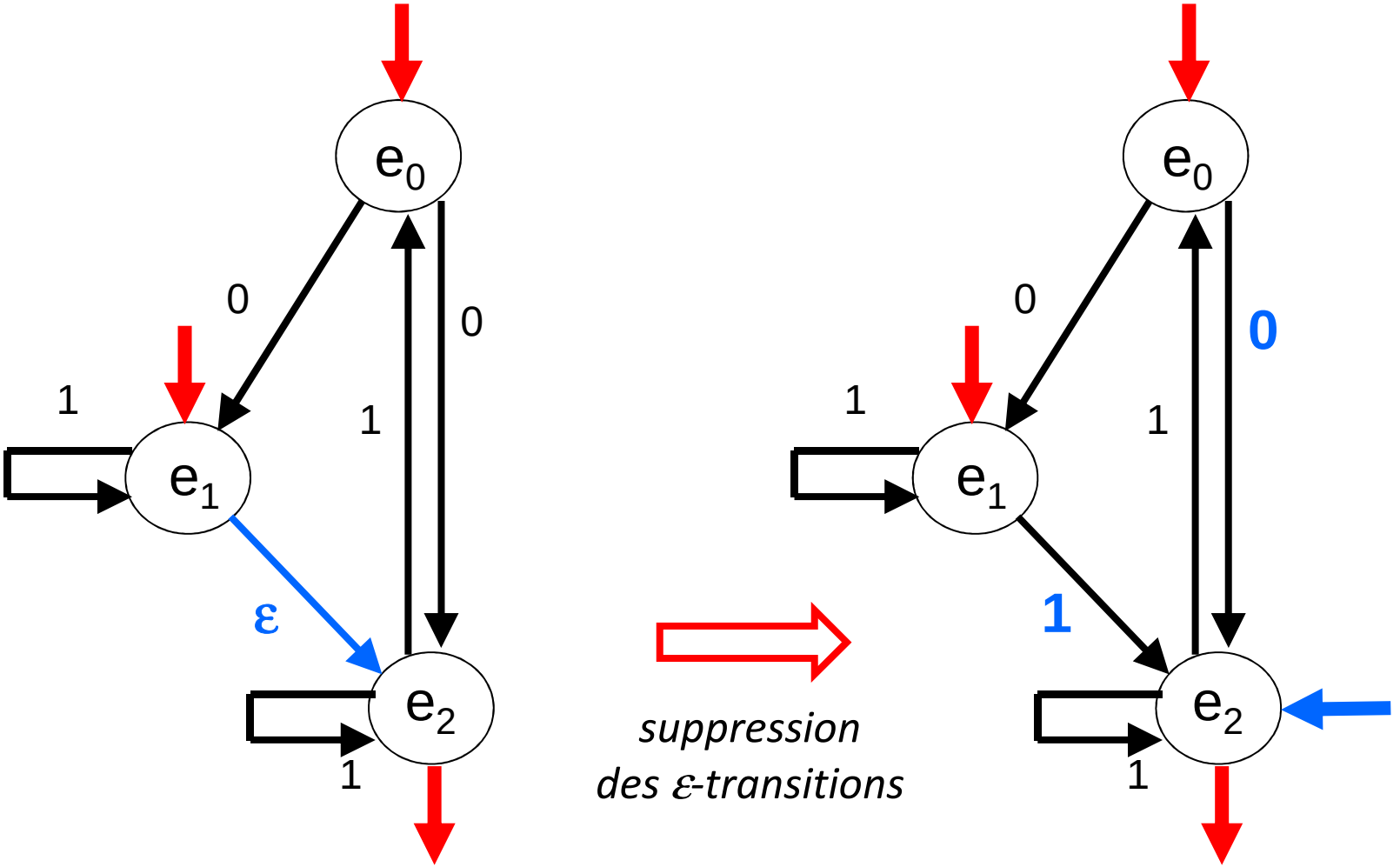
- On supprime les ε-transitions de **M**, une à une, pour obtenir un AF **M*** sans ε-transition :

  si (**e,ε,f**) $\in$ δ et **e** est un état initial alors **f** devient un état initial ;
  si (**e,ε,f**), (**g,a,e**) $\in$ δ alors (**g,a,f**) $\in$ δ^*.

- Les états de **M'** seront des ensembles d'états de **M***.
- L'état initial de **M'** est l'ensemble des états initiaux de **M***.
- Ensuite, pour chaque lettre **a** $\in$ **A** et chaque état **S** de **M***, on pose : $\delta'(S,a) = \{ \delta^*(e_i,a) / e_i \in S \}$.
- Si **S** contient un état terminal de **M***, alors **S** est un état terminal de **M'**.
- Cet algorithme s'arrête car **M*** possède un nombre fini d'états : il n'y a donc qu'un nombre fini d'états possibles pour **M'**…

Déterminisation d'un AF : exemple (1)

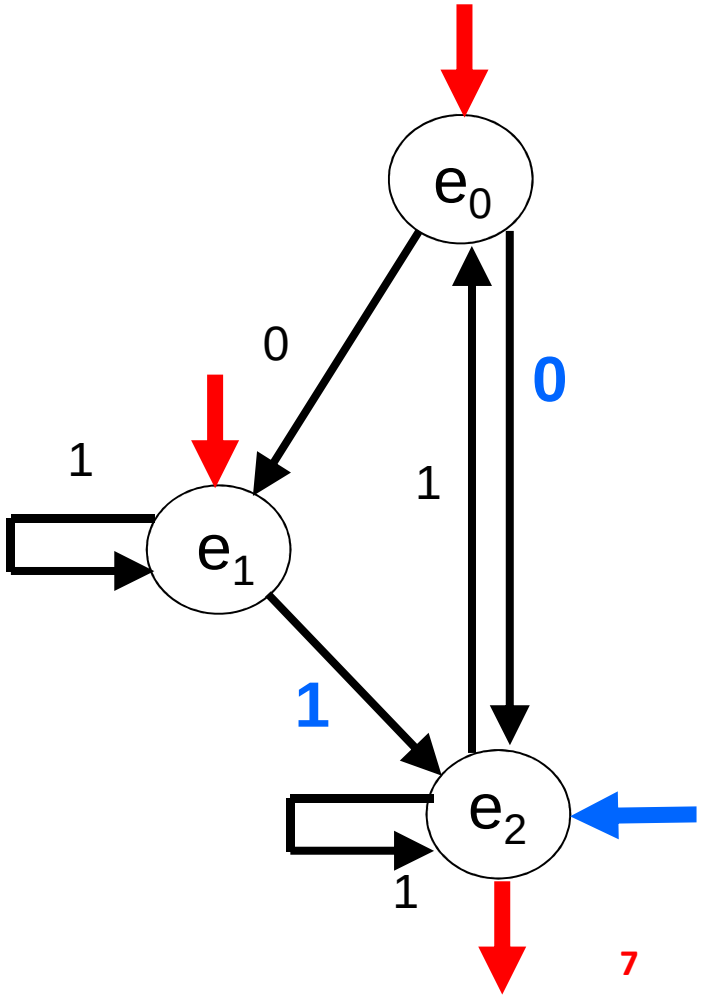
On notera e_3 l'état-puits de notre AF.



Déterminisation d'un AF : exemple (2)

La fonction de transition de l'AFD s'obtient ainsi :

état		0	1	∈ T'
S_0 initial	$e_0e_1e_2$	$e_1e_2e_3$	$e_0e_1e_2e_3$	O
S_1	$e_1e_2e_3$	e_3	$e_0e_1e_2e_3$	O
S_2	$e_0e_1e_2e_3$	$e_1e_2e_3$	$e_0e_1e_2e_3$	
S_3 puits	e_3	e_3	e_3	O

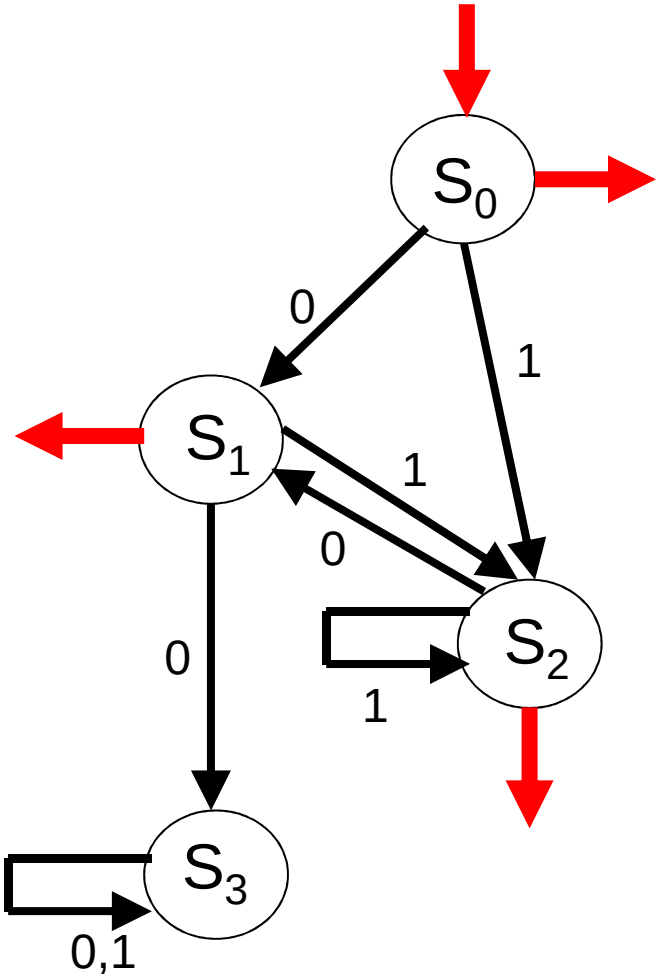


On démarre avec l'état initial $S_0 = \{e_0, e_1, e_2\}$.
Quand un nouvel état « apparaît », on crée une nouvelle ligne…

Déterminisation d'un AF : exemple (3)

On obtient l'AFD suivant :

état		0	1	∈ T'
S_0 initial	$e_0e_1e_2$	$e_1e_2e_3$	$e_0e_1e_2e_3$	O
S_1	$e_1e_2e_3$	e_3	$e_0e_1e_2e_3$	O
S_2	$e_0e_1e_2e_3$	$e_1e_2e_3$	$e_0e_1e_2e_3$	O
S_3 puits	e_3	e_3	e_3	

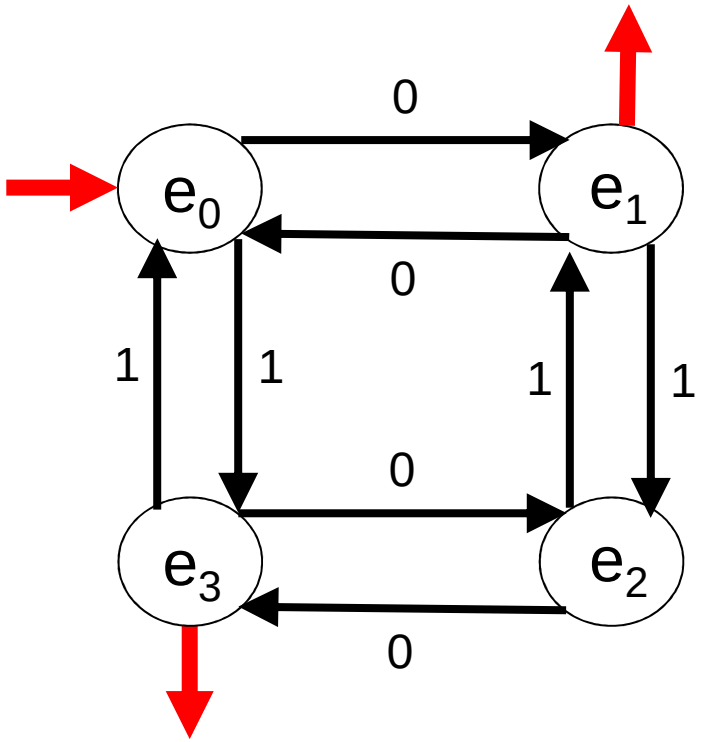


Remarque. *L'automate ainsi construit n'est pas nécessairement minimal…*

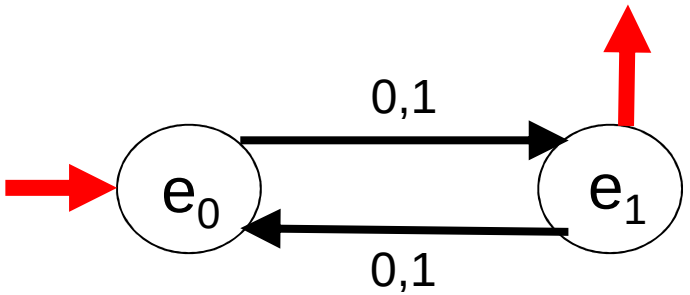
Minimisation d'un automate fini déterministe

AFD minimal

Nous avons vu que plusieurs AFD distincts pouvaient reconnaître le même langage. Un tel AFD est dit **minimal** s'il contient un nombre minimum d'états.



non minimal…



minimal !

Minimisation d'un AFD : principe (1)

L'idée de base est la suivante : on va raisonner sur des **classes d'équivalence** d'états.

Deux états **e** et **f** seront considérés comme **équivalents** si, pour toute lettre **a** $\in$ **A**, δ**(e,a)** et δ**(f,a)** sont deux états équivalents… (Intuition : ils ont « même futur ».)

Pour déterminer ces classes d'équivalence, on va procéder par raffinements successifs en « divisant » les ensembles contenant des états non équivalents…

Remarque. *Un pré-traitement de l'AFD permet d'éliminer les états non accessibles à partir de l'état initial…*

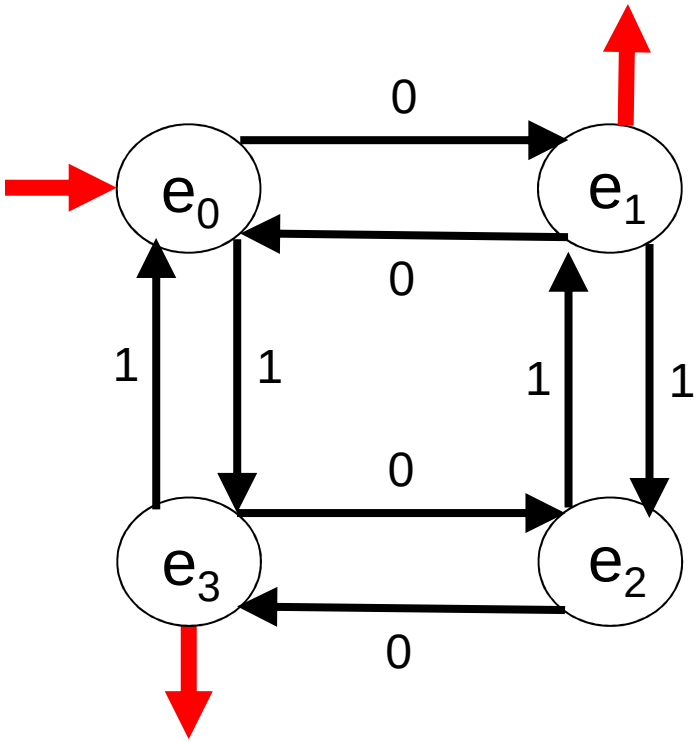
Minimisation d'un AFD : principe (2)

Principe de minimisation : **Algorithme de Moore.**

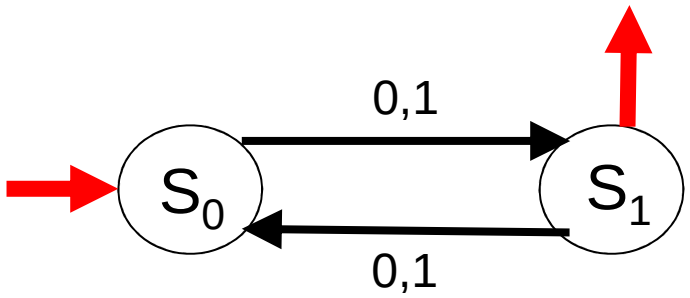
On suppose que tous les états sont accessibles à partir de l'état initial.

- On démarre avec deux ensembles, **T** et **E \ T.**
- Si, dans un ensemble **S**, il existe deux états **e** et **f** et une lettre a tels que **δ(e,a)** et **δ(f,a)** n'appartiennent pas au même ensemble, on divise **S** de façon à mettre ensemble tous les états qui ont même « ensemble successeur » par la lettre **a**.
- L'algorithme se termine lorsque plus aucun ensemble ne doit être divisé : chaque ensemble est alors un état de l'automate minimal. Les sous-ensembles de T sont des états terminaux, et l'ensemble contenant l'état initial d'origine est l'état initial.

Minimisation d'un AFD : exemple (1)

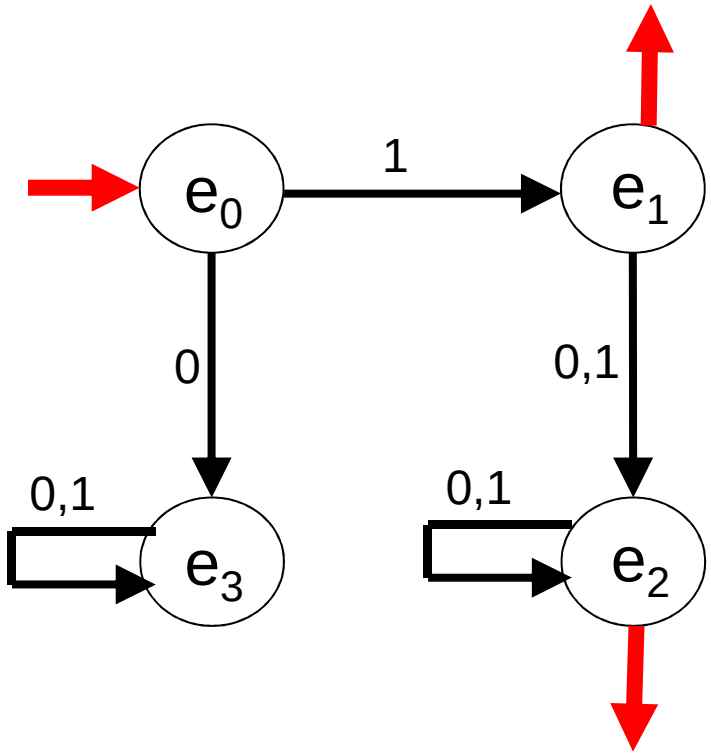


- étape 1 : { e_0, e_2 }, { e_1, e_3 }.
- étape 2 : rien à faire, aucune partie ne satisfait la condition de séparation !
- $S_0 \leftarrow$ { e_0, e_2 }, initial
  $S_1 \leftarrow$ { e_1, e_3 }, terminal.



Trop facile !... ;-)

Minimisation d'un AFD : exemple (2)

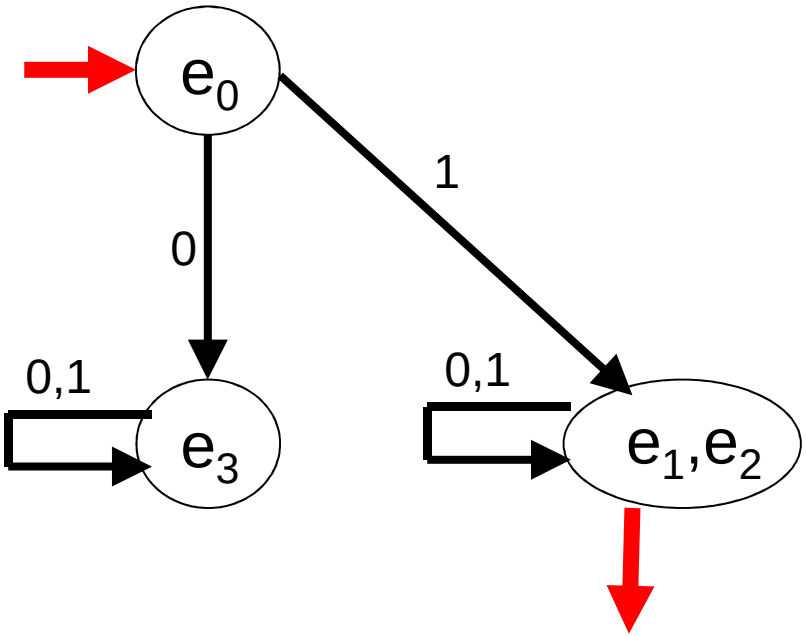


- étape 1 : { e_0, e_3 }, { e_1, e_2 }.
- étape 2 :
  { e_0, e_3 } → séparés par 1
  { e_0 }, { e_3 }, { e_1, e_2 }
- étape 3 :
  { e_1, e_2 } → OK
  { e_0 }, { e_3 }, { e_1, e_2 }

	e_0	**e_1**	**e_2**	**e_3**
0	e_3	e_2	e_2	e_3
1	e_1	e_2	e_2	e_3

- résultat final, 3 états :
  { e_0 }, { e_3 }, { e_1, e_2 }

Minimisation d'un AFD : exemple (3)



	e_0	e_1	e_2	e_3
0	e_3	e_2	e_2	e_3
1	e_1	e_2	e_2	e_3

- résultat final, 3 états :
  $\{ e_0 \}, \{ e_3 \}, \{ e_1, e_2 \}$