IUT de Bordeaux
Département Informatique

M2201 – Graphes et Langages

Éléments de Théorie des Langages

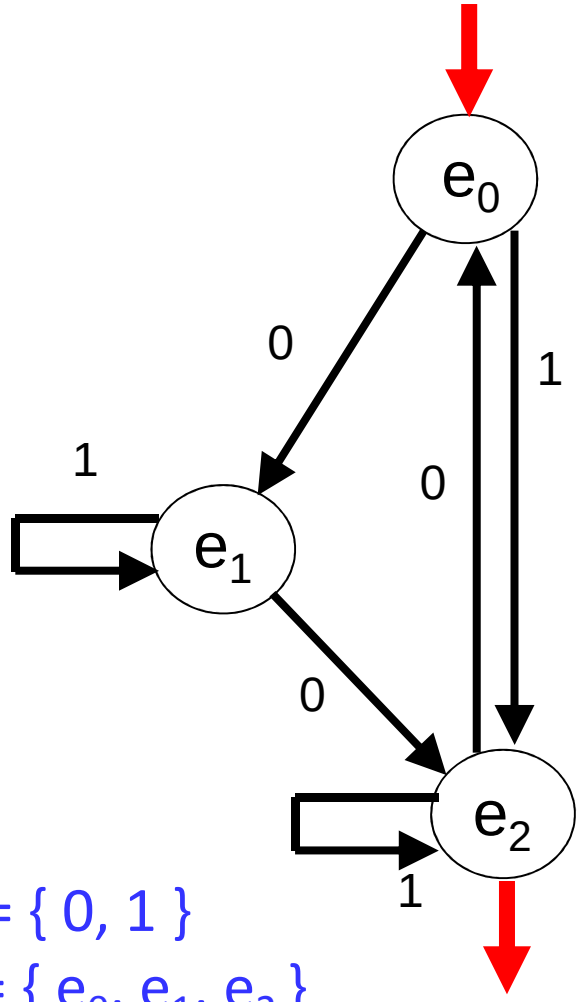
- **Automates**
- **Simulation du fonctionnement d'un automate fini déterministe**
- **Langages reconnaissables**

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**.



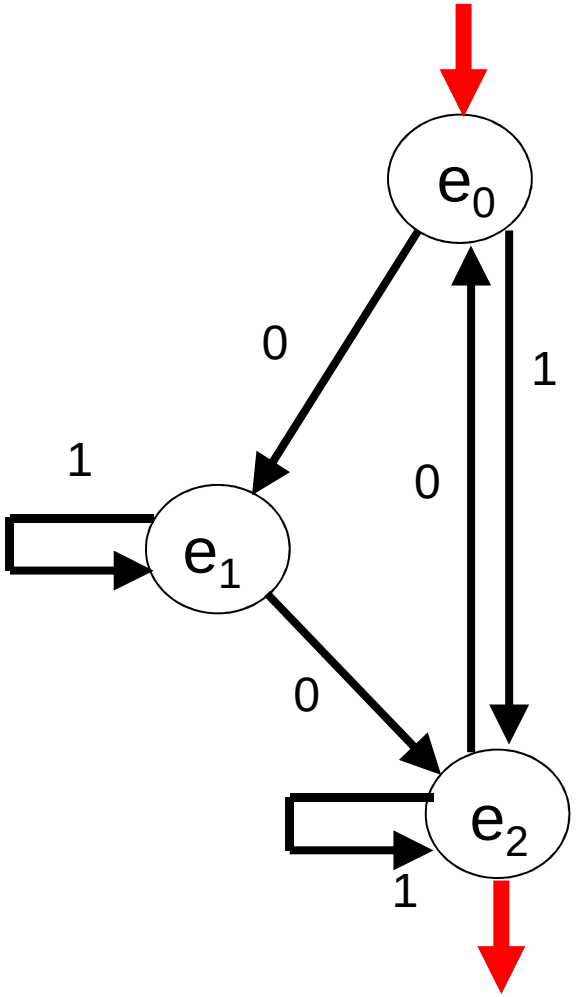
$\Sigma = \{ 0, 1 \}$
$E = \{ e_0, e_1, e_2 \}$
$T = \{ e_2 \}$
$\delta = \{ (e_0,0,e_1), \ldots \}$

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

$\delta(e_0,m) = e_1$

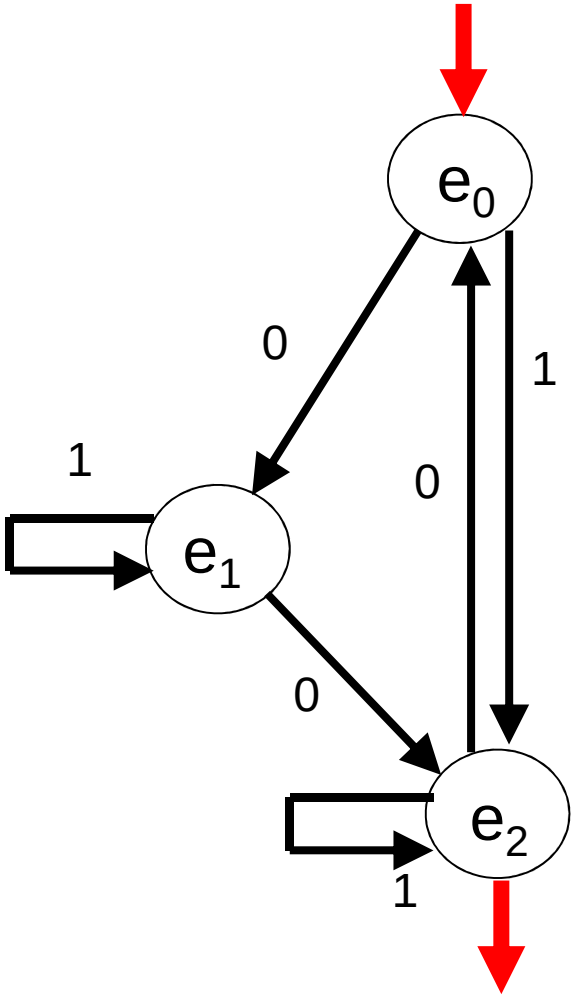
	1	1	0	1	0	0	1
e_0	e_2	e_2	e_0	e_2	e_0	e_1	e_1

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 :

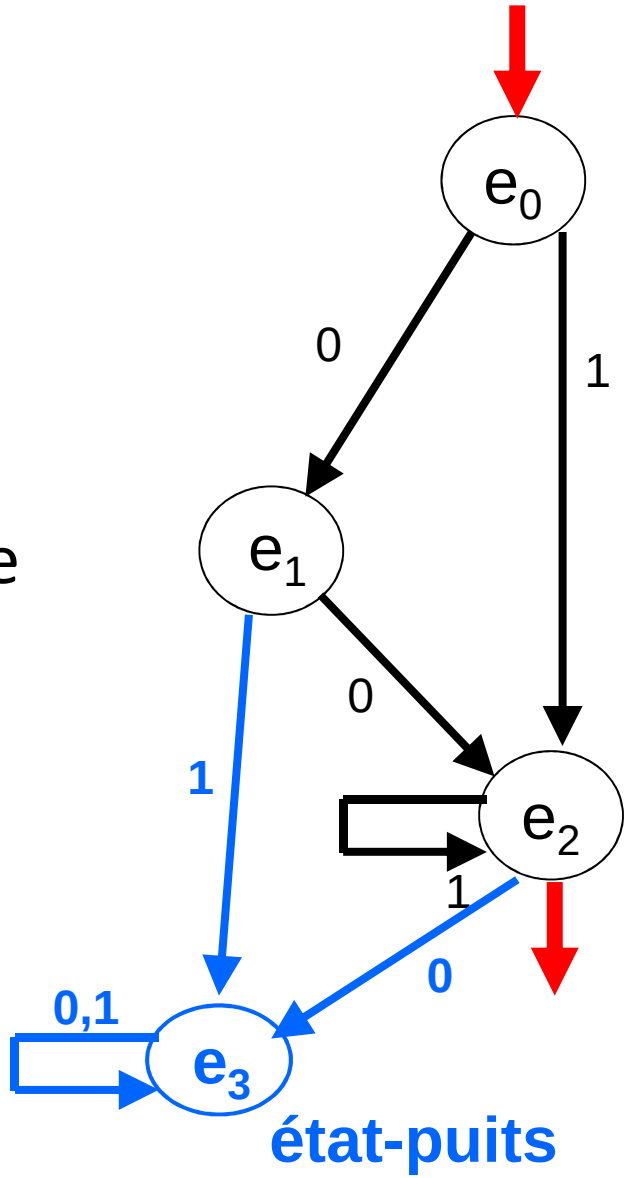
$$L(\mathbf{M}) = \{ m \in \Sigma^* / \delta(e_0,m) \in T \},$$

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**.

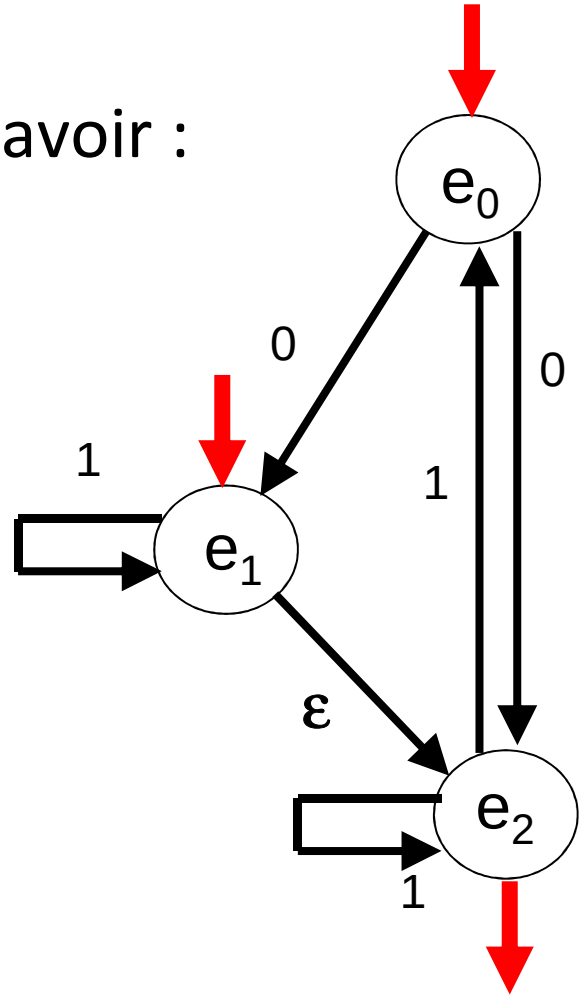


Automates non déterministes (1)

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

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

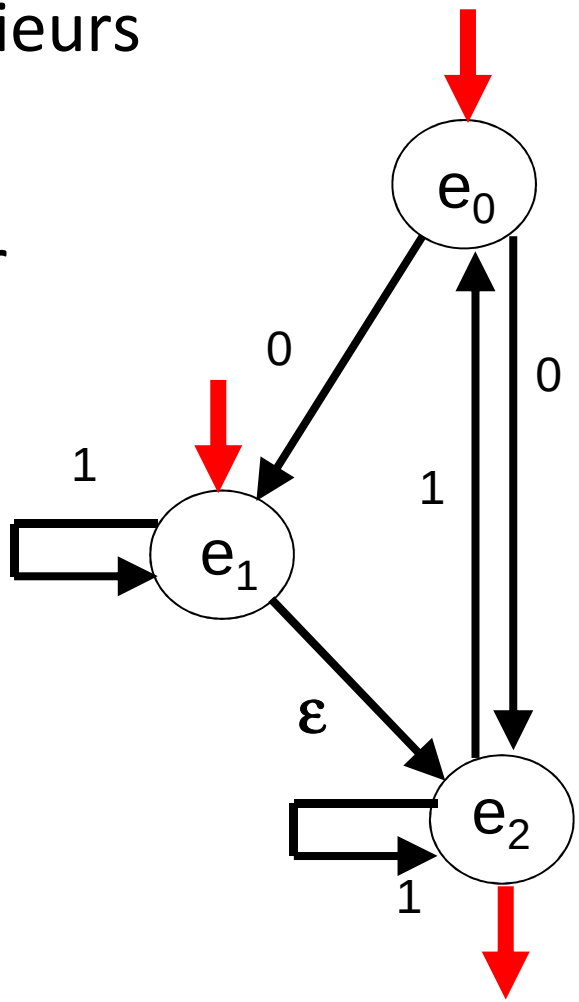
- plusieurs états initiaux,
- plusieurs transitions issus d'un même état et portant la même étiquette,
- des transitions étiquetées par ε (appelées ε-transitions)



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

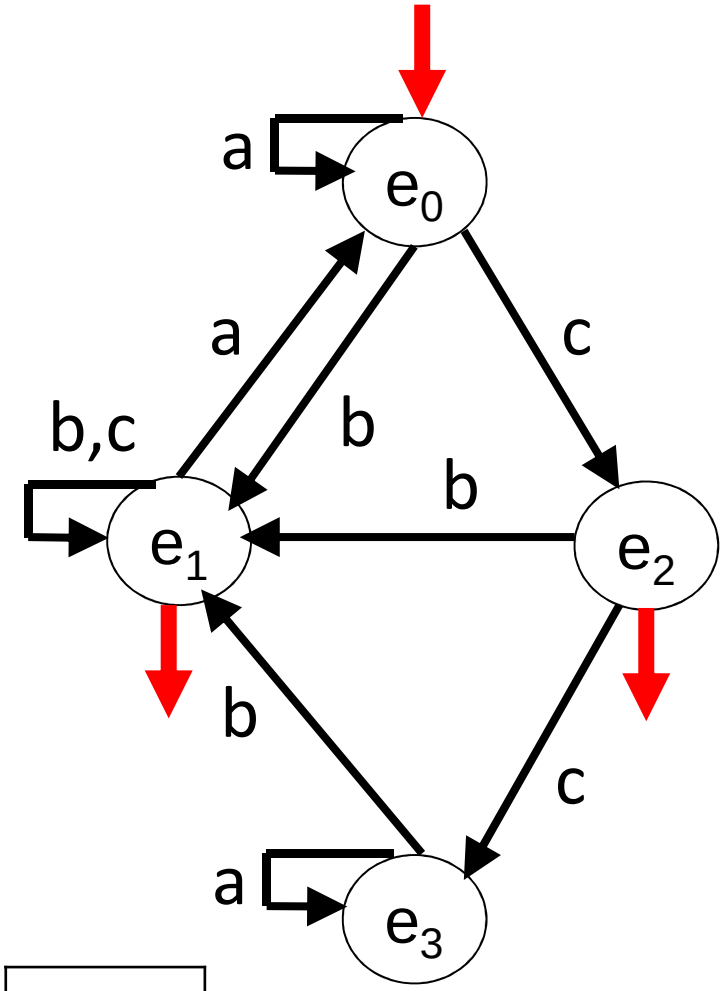
0	F
1	V
2	V
3	F
4	F

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 :

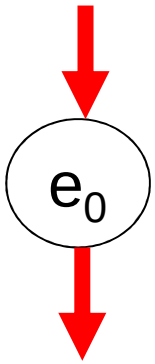
- **{ ε }** est un LR,
- si a ∈ **A**, **{ a }** est un LR,
- si **L** est un L.R. alors **L*** est un LR (**L^+** et **L^n** aussi),
- si **L_1** et **L_2** sont des LR, alors **$L_1 \cup L_2$** et **$L_1.L_2$** sont des LR.

Ainsi, il suffit de savoir construire :

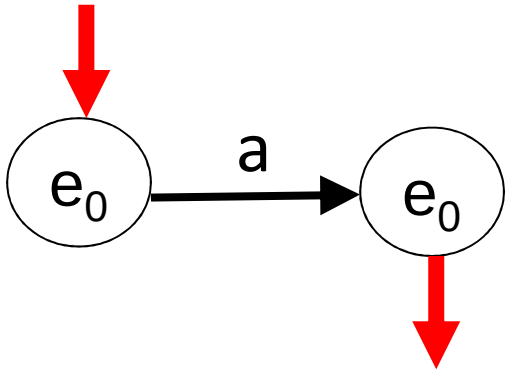
- *des AFD reconnaissant les langages { ε } et { a } pour tout a ∈ **A**,*
- *et, à partir d'AFD reconnaissant les langages L_1 et L_2, des AFD reconnaissant L_1*, $L_1 \cup L_2$ et $L_1.L_2$…*

AFD reconnaissant { ε } et { a }

Aucune difficulté !



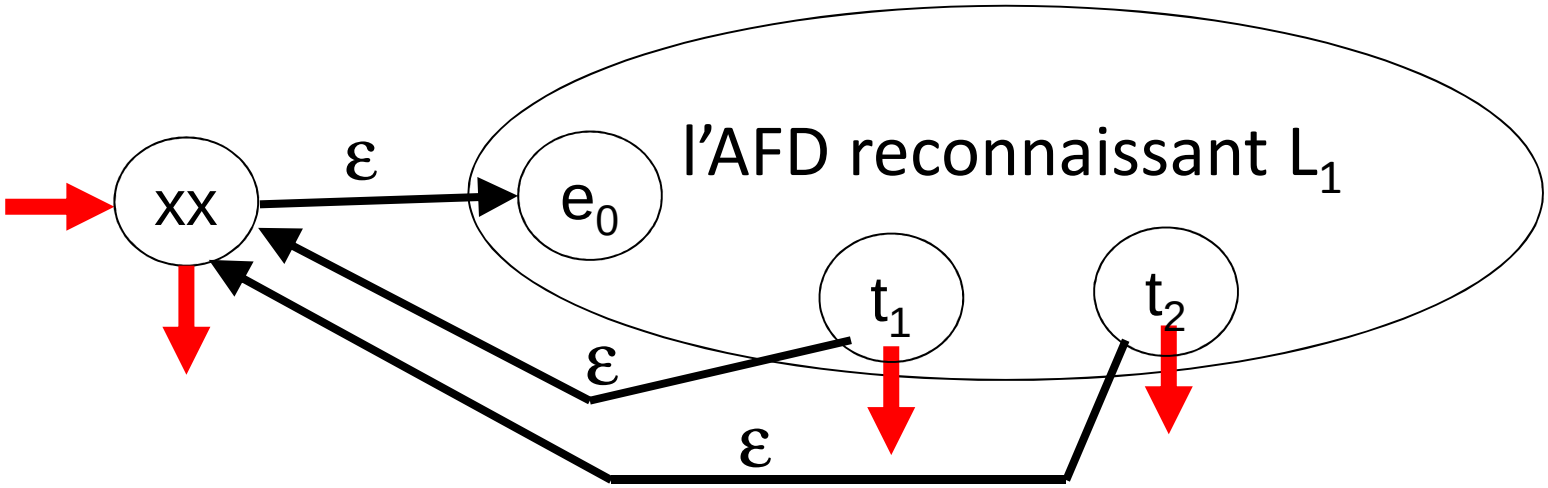
l'AFD $\mathbf{M_\varepsilon}$



l'AFD $\mathbf{M_a}$

AFD reconnaissant L_1*

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

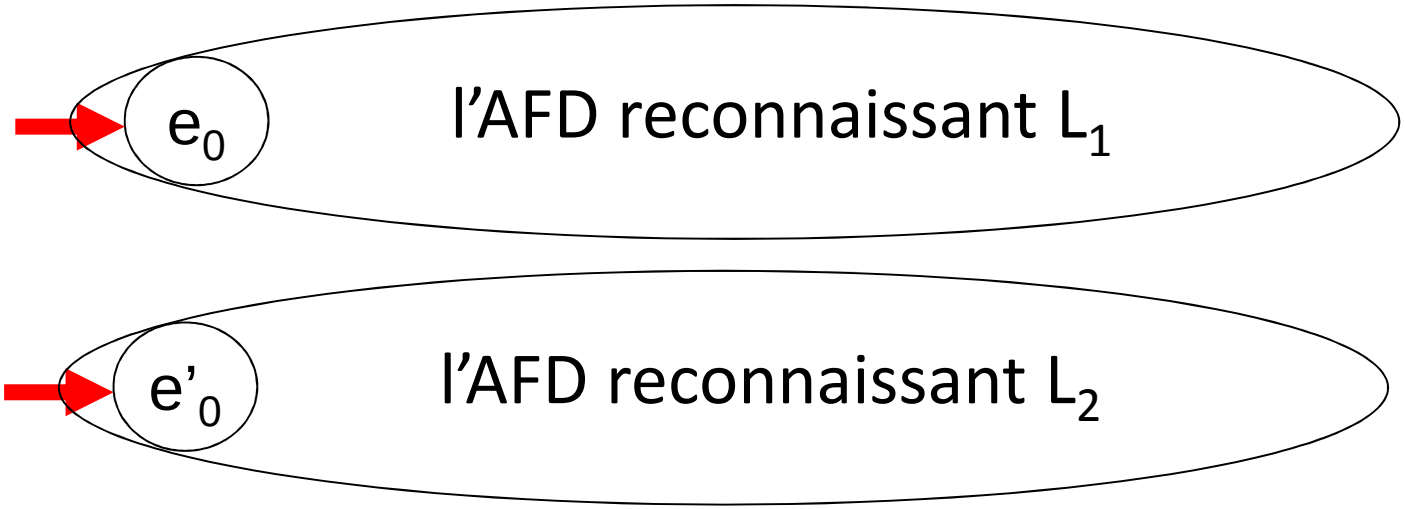


l'AF $\mathbf{M_{L1*}}$

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 !

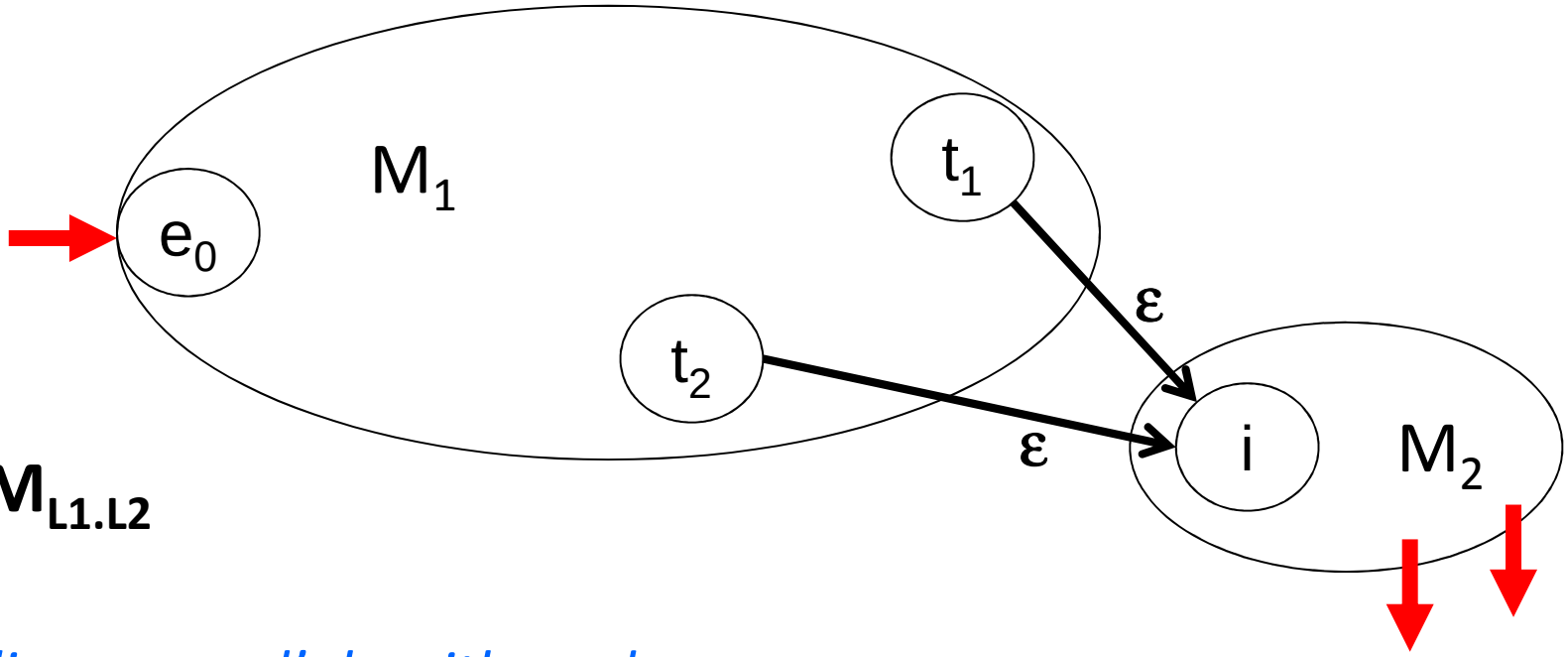


l'AF $\mathbf{M_{L1 \cup L2}}$

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…

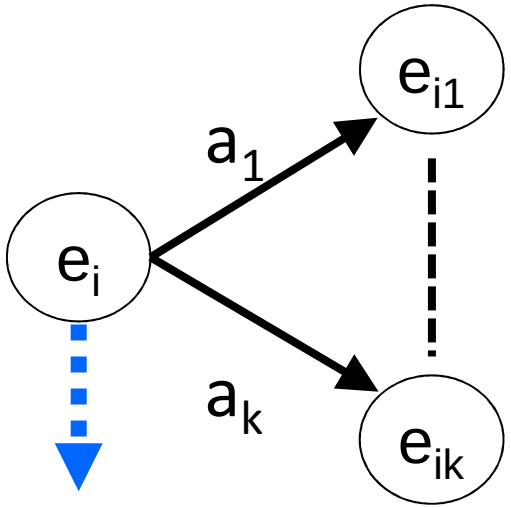
Expression rationnelle pour un AFD (1)

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.



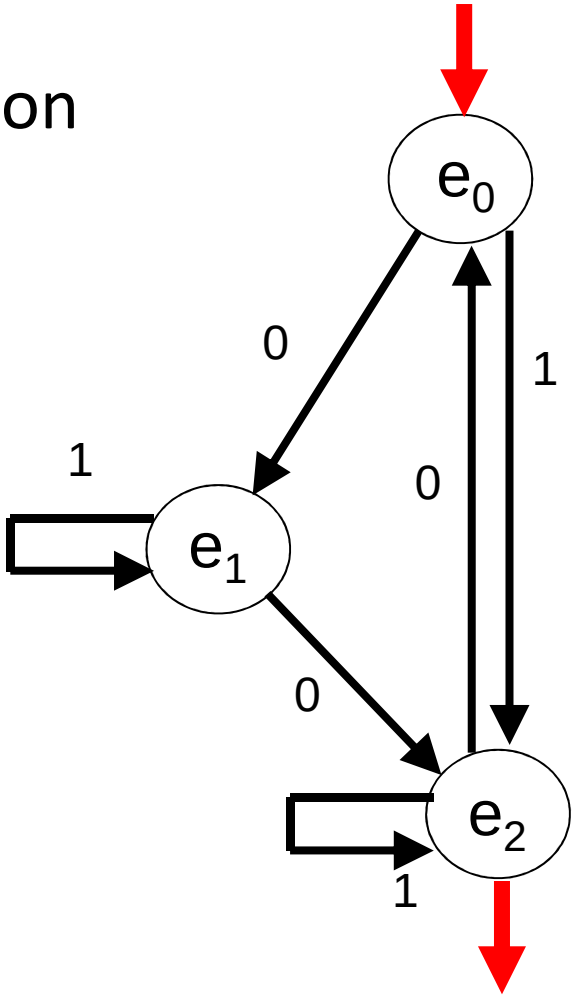
Un mot **u** conduisant de l'état $\mathbf{e_i}$ à un état terminal est nécessairement de la forme $\mathbf{u = a_j u_{ij}}$, où $\mathbf{u_{ij}}$ est un mot conduisant de l'état $\mathbf{e_{ij}}$ à un état terminal [ou $\mathbf{u = \varepsilon}$ si $\mathbf{e_i}$ est terminal].

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)

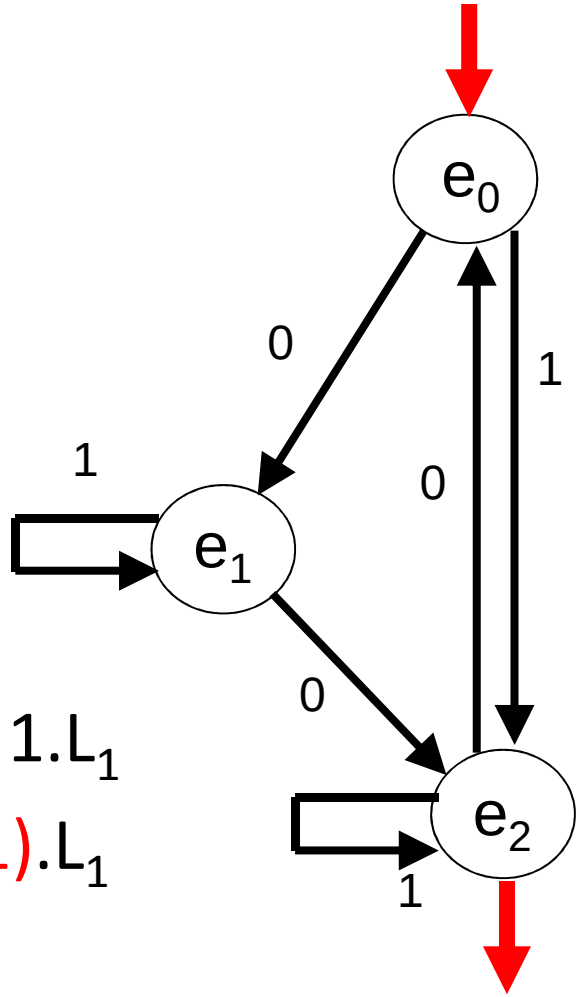
- $L_0 = 0.L_1 + 1.L_2$
- $L_1 = 0.L_2 + 1.L_1$
- $L_2 = \varepsilon + 0.L_0 + 1.L_2$

$\Rightarrow$

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

$\Rightarrow$

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

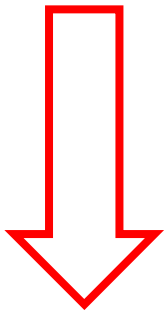


$\Rightarrow$

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

Résolution du système d'équations (3)

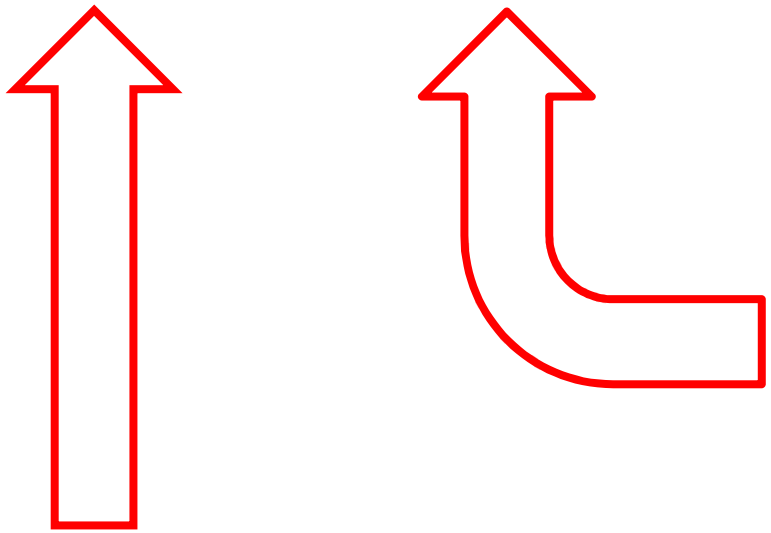
- $L_0 = 0.L_1 + 1.L_2$
- $L_1 = (0(01+1)^*00 + 1)^*0(01+1)^*$
- $L_2 = (01+1)^* + (01+1)^*00(0(01+1)^*00+1)^*0(01+1)^*$

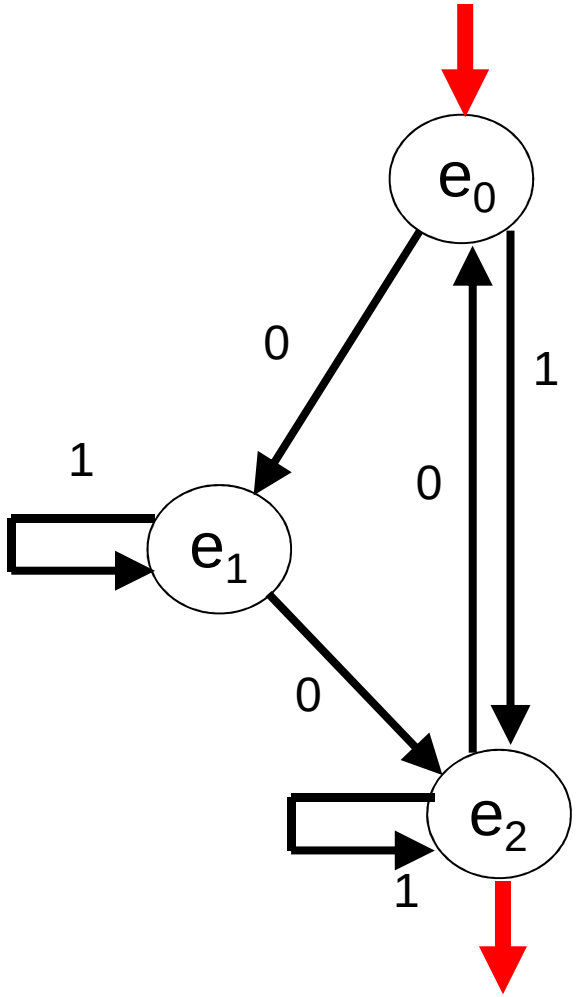


- $L_0 = 0.(0(01+1)^*00 + 1)^*0(01+1)^*$
  $+ 1.(01+1)^* + (01+1)^*00(0(01+1)^*00+1)^*0(01+1)^*$

Résolution du système d'équations (4)

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





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