Spécification et Modélisation Informatiques

Alexandre Saidi

Ecole Centrale de Lyon
Département Mathématiques-Informatique

7 mai 2009

CNAM NFP108

Spécification et Modélisation Informatiques

(CHAPITRE II)

Partie II : Automates et Machines d'états finis (AEFs/MEFs)

Alexandre Saidi-Glandus

2008-2009

Rappel Plan Général du cours

Deux grands chapitres :

1. Chapitre 1 (vu)
   - Logique (propositionnelle, prédicats)
   - Notions : Théorie des ensembles, Relations, Fonctions,
   - Méthodes de Preuve
   - Induction Mathématique
   - Quelques exemples :
     - ⋆ Notions TDA, graphes, Algorithmes (et leur preuve)
     - ⋆ Notions sur les BDs, Algèbre relationnelle et de Boole, Combinatoires
2. **Chapitre 2**
   - Machines et Automates d'états finis
   - Autre formalismes (UML, RdP)
   - OCL : expression des contraintes
3. Plusieurs exemples traités tout le long.

Automates : motivations et plan

Une introduction

- AEF/MEF : Outil de spécification / modélisation
- Exemples "real world" !
- Avec sortie (MEF), priorité, etc.
- Contrôle et Transformations (minimisation, déterminisme, ...)

- **Outils (libres) :**
  - ➙ LEX (contrôle, génération de code C/C++/...)
  - ➙ FSA (contrôle et validation, minimisation, Non Déterministe $\rightarrow$ Déterministe
  - ➙ D'autres outils ...

  - ➙ MEF (génération de code C/C++/Java/...) d'un automate.

- Classe d'outil et Inférence ?

Exemple : Métro

- Modélisation da la barrière de passage d'un métro.
- Principe de fonctionnement :
  - Par défaut, le passage reste fermé
  - Si ticket introduit, le passage est débloqué
  - L'usager pousse la barrière pour ouvrir et passer
  - Après un passge, la barrière revient à son état bloqué.

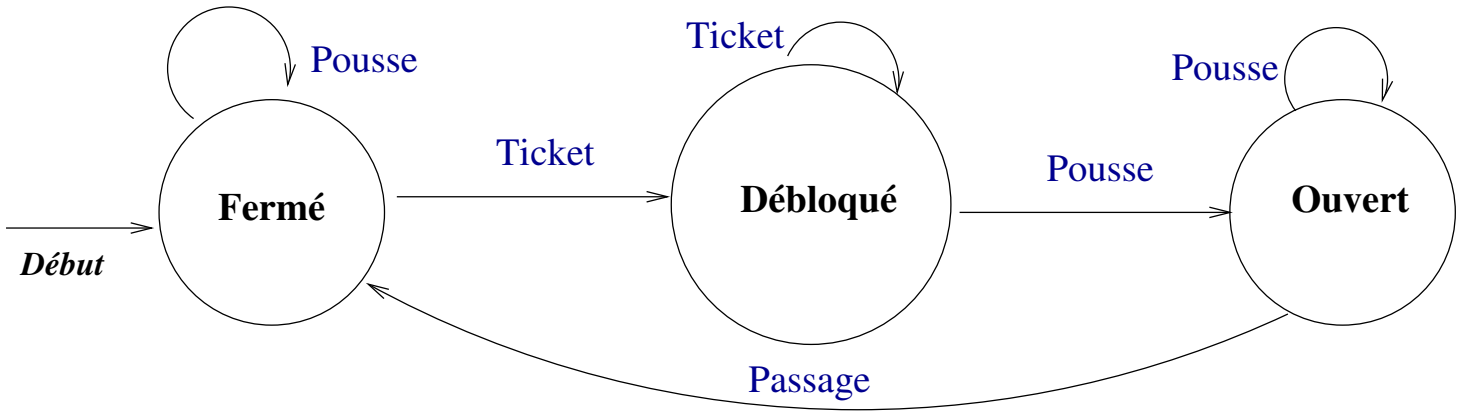


Fig.: Automate d'états fini d'une barrière de Métro

Exemple : Métro (suite)

- Modélisation par une Machine d'états fini :

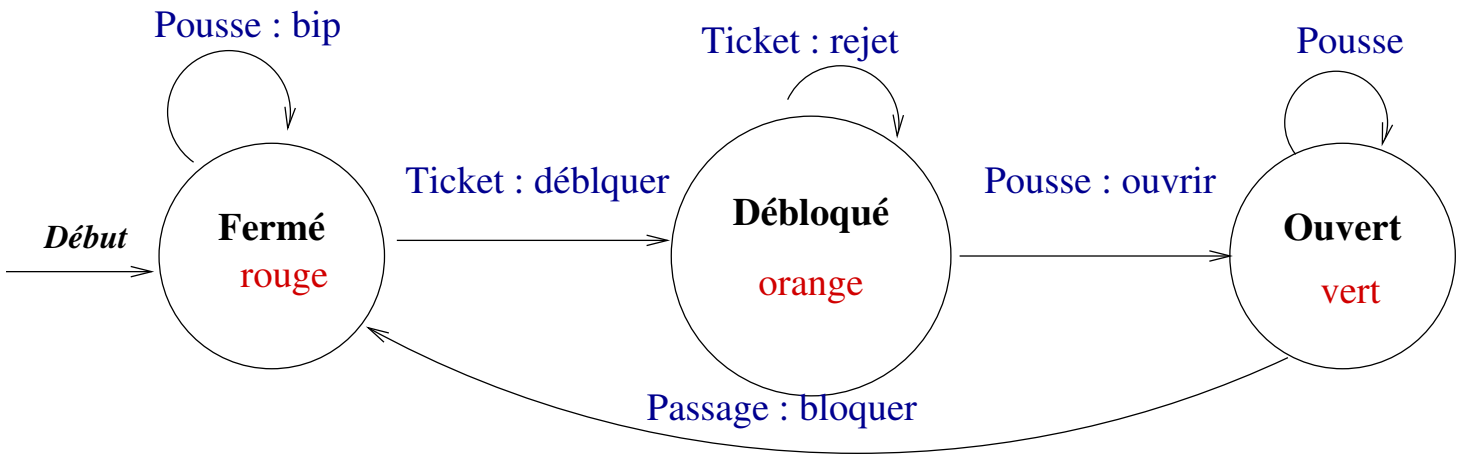


Fig.: Machine d'états fini d'une barrière de Métro

N.B : symbole en entrée représenté par **?** et en sortie par **!** .
Les couleurs à l'intérieur des états : voir Machine de Moore plus loin.

Automates et Machines d'états finis

• Permettent de modéliser toutes sortes de machines (y compris les composantes des ordinateurs).

• Plusieurs sortes d'Automates/Machines de (à) états fini.
• **Leurs points communs** (toutes possèdent) :
  - Un ensemble fini d'états
  - Un état initial donné (mais pas forcément un état final)
  - Un alphabet
  - Une fonction de transition qui affecte un nouvel état à un couple état $\times$ entrée .

✓ En plus : présence d'un symbole en sortie (pouvant être interprété comme une action) ou d'une priorité associée à une transition (dans les MEFs).

• Les AEFs/MEFs sont très utilisées en informatique (TR, Réseaux, etc.).

MEF (suite intrduction)

- **Exemples d'application** :
  - Toutes sortes d'automates (Automatismes),
  - Protocoles réseaux (communication des ordinateurs)
  - Vérification d'orthographe (et de syntaxe)
  - Indexation et recheche dans les BD textuelles larges
  - Reconnaissance de la parole (simple)
  - Transduction et Transformation de texte en / vers HTML/XML
  - ...
- **Quelques exemples étudiés :**
  - Distributeur de boissons
  - Machine de temporisation
  - Machine d'addition
  - Recherche d'un motif dans une chaîne/texte
  - ...

Exemple : robot R1

Exemple : robot basique

• Un robot basique muni de *Bumpers* (détecteurs de contact) frontal, gauche et droit.
• L'activité du robot (dans une pièce sans obstacle) :
✓Avancer tout droit en permanence (action par défaut)
✓Si un Bumper touché (par les murs de la pièce) :
1 - Si bumper gauche : tourner à droite pendant un délai Δ puis avancer tout droit ;
2 - Si bumper frontal : reculer pendant un délai Δ puis tourner à droite (comme en point 1)
3 - Si bumper droit : tourner à gauche pendant un délai Δ puis avancer tout droit ;

Exemple : robot R1 (suite)

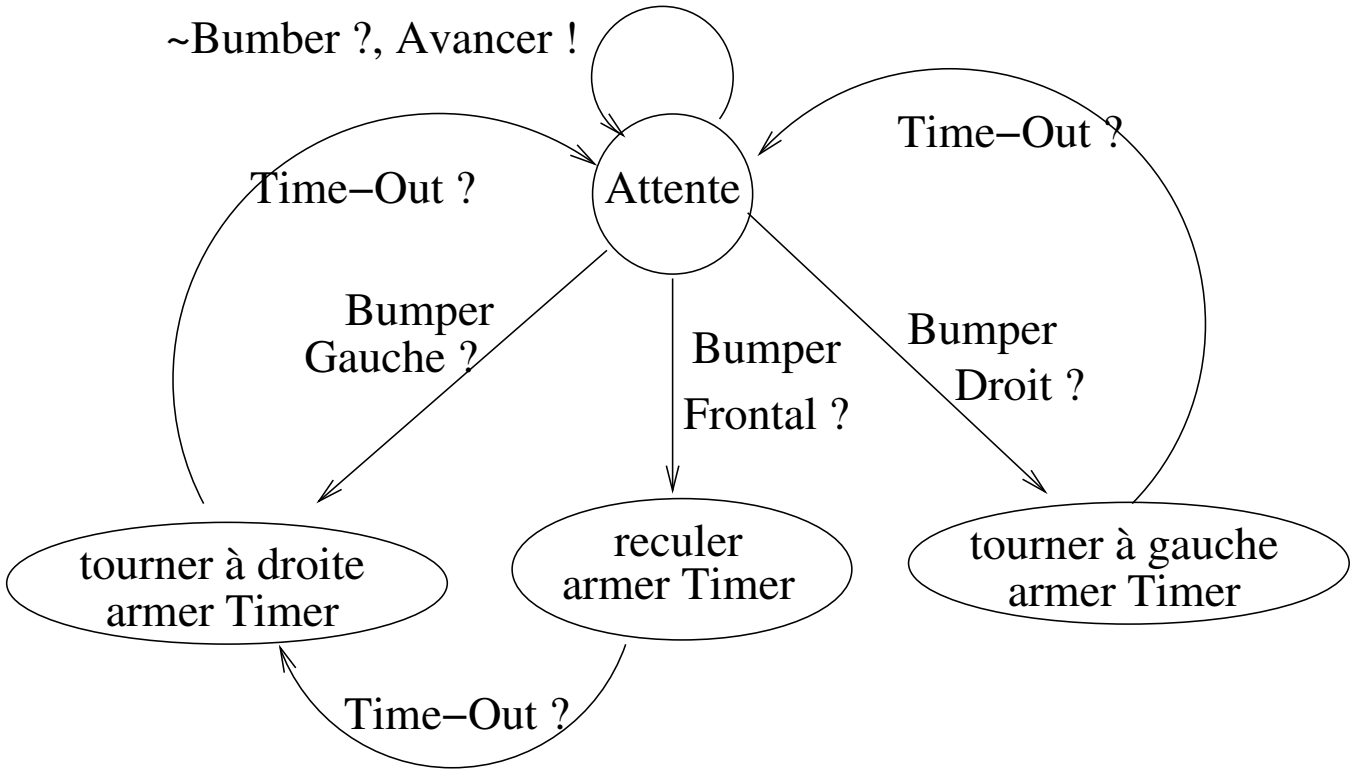


Fig.: Automate d'états fini du Robot R1

Exemple : robot R1 (suite)

N.B : symbole en entrée représenté par **?** et en sortie par **!** .
Sur l'état "Attente", le symbole
$\sim Bumper = Alphabet - \{Bumper\ G/d/F\}$

• La modélisation du Robot R1 précédent par une **Machine** (d'états fini) où l'**action** Armer Timer est associée aux transitions :

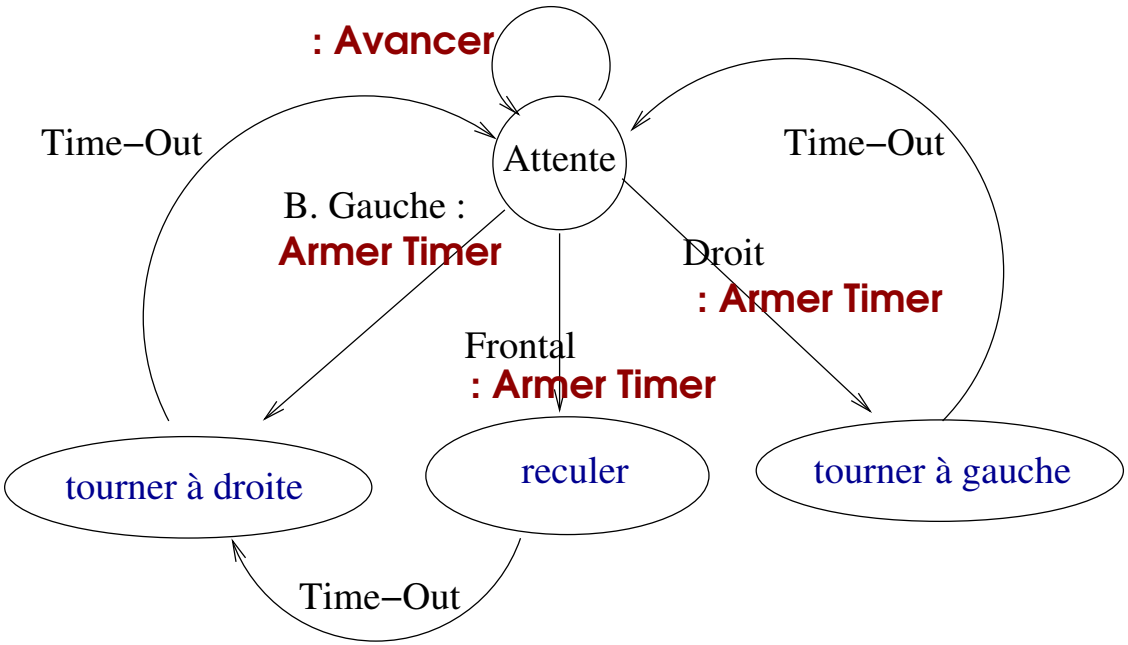


Fig.: Machine d'états fini du Robot R1

Exemple : robot R1 (suite)

N.B : ici, les symboles en **noir** sont des entrées (?) et les symboles en rouge des sorties (!). La notation **: action** correspond à une sortie exécutable (commande).

Exemple 3 : distributeur de café simple

Exemple 3 : distributeur de café simple

- La machine accepte des pièces 10 et 20 centimes
- Le prix du café = 20 c.
- On a le choix entre café court et café long.
- Si plus de 20 centimes ont été introduits, la machine rend le surplus.
- Pendant l'introduction des pièces, on peut annuler la commande.
- Si 20 centimes sont introduits, on peut choisir entre café court ou long.
- Toute pièce différente de 10 ou 20c sera rejetée
- On attend ensuite d'être servi.

Exemple 3 : distributeur de café simple (suite)

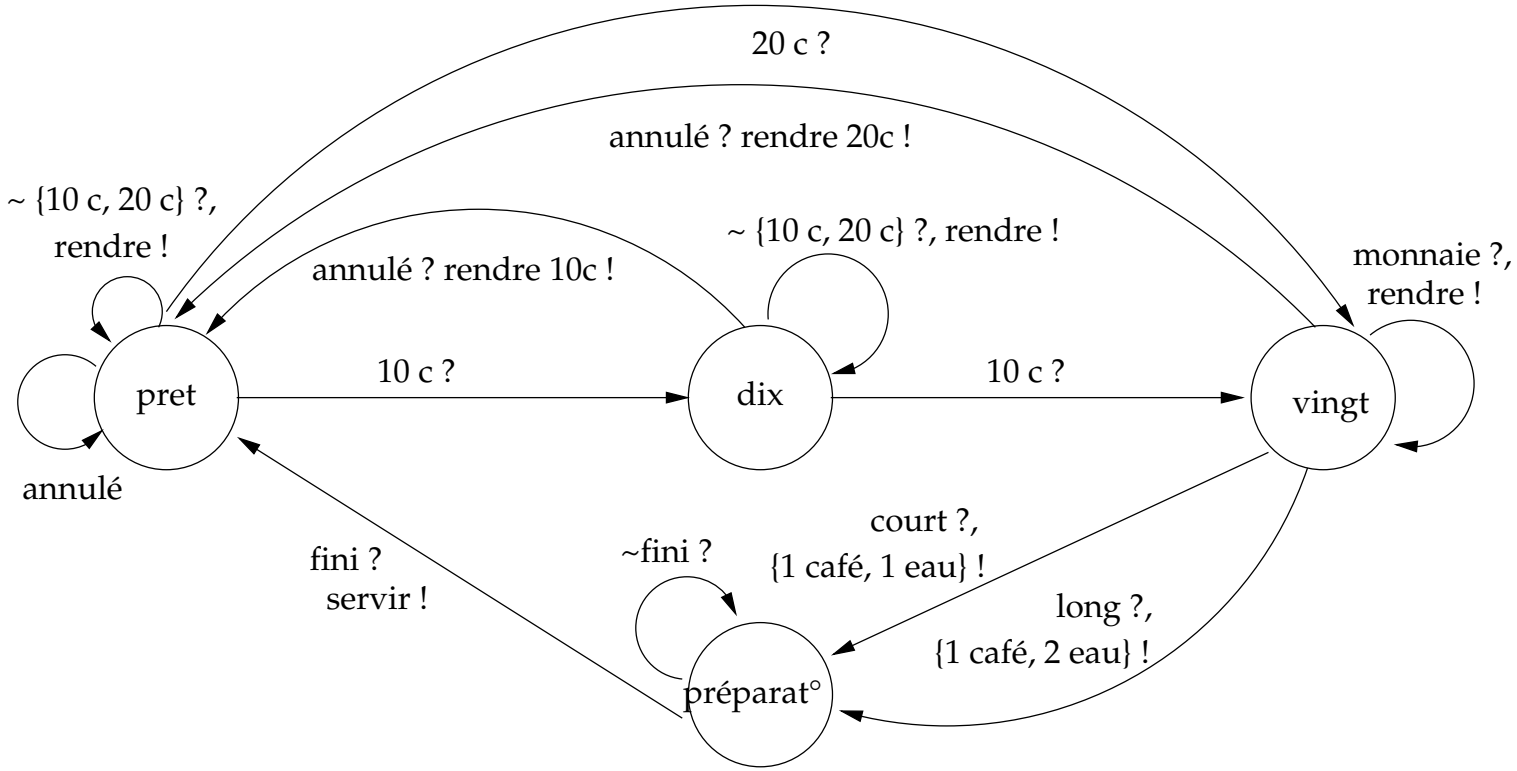


Fig.: Machine d'états fini à café

- Notion de **ruban** (entrée, sortie, commande) : voir l'exemple *passage à niveau*.

Exemple 3 : un distributeur de boissons

Exemple 3 : distributeur de jus de fruits

- La machine accepte des pièces de 5, 10 et 20 centimes
- Si plus de 30 centimes ont été introduits, la machine rend le surplus.
- Si 30 centimes sont introduits, le consommateur peut choisir
  jus d'orange (bouton Orange) ou
  jus de pommes (le bouton Rouge).

- On définit la machine par ses **états**, ses **changements d'états** ainsi que ses **sorties**.

➙ La machine peut être dans un des 7 états S_i , i=0..6.

➙ Elle commence à l'état $S_0 \Rightarrow$ 0 centime reçu.

➙ L'état S_i veut dire : $5i$ centimes introduits ($S_5 \Rightarrow$ 25 centimes).

- A l'état S_0, on **accepte** les pièces de 5, 10 et 20 centimes, ou
  les boutons R = rouge (pommes) ou O = orange.
- Les **sorties** possibles : rien (n), 5 centimes, 10, 15 et 20, un jus d'orange (JO) ou de pommes (JP).

Distributeur : Table d'états

Tableau résumant les états, les transitions et les sorties :

état	**état suivant**					**Sortie**				
	Entrées					**Entrées**				
	5	10	20	O	R	5	10	20	O	R
S_0	S_1	S_2	S_4	S_0	S_0	n	n	n	n	n
S_1	S_2	S_3	S_5	S_1	S_1	n	n	n	n	n
S_2	S_3	S_4	S_6	S_2	S_2	n	n	n	n	n
S_3	S_4	S_5	S_6	S_3	S_3	n	n	5	n	n
S_4	S_5	S_6	S_6	S_4	S_4	n	n	10	n	n
S_5	S_6	S_6	S_6	S_5	S_5	n	5	15	n	n
S_6	S_6	S_6	S_6	S_0	S_0	5	10	20	**JO**	**JP**

n : rien, **JO** : jus d'Ornage, **JP** : jus de pommes
Rappel : les pièces acceptées = 5, 10 et 20. Le prix = 30.
Exercice : refaire (compléter) la table pour accepter 50 centimes aussi.

Distributeur (l'AEF)

L'automate d'états fini associé

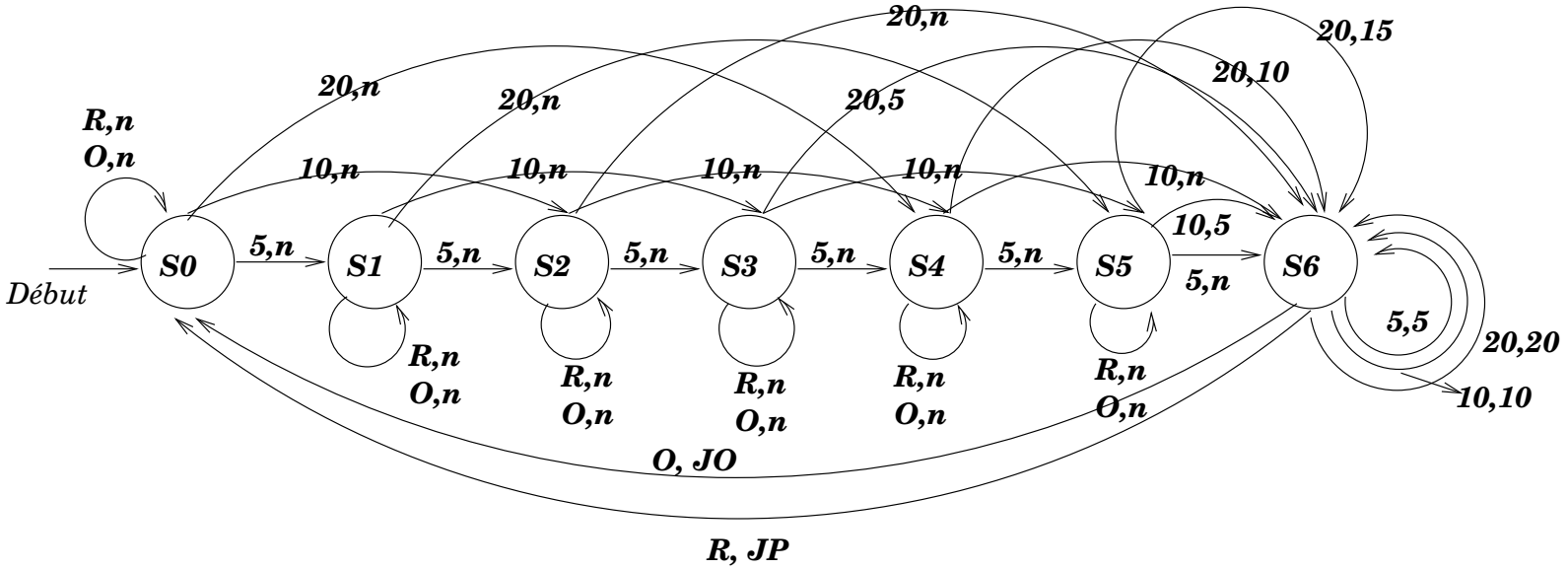


Fig.: Automate du distributeur

• Lorsqu'il n'y a pas d'ambiguïté, la notation **x,y** $\in I \times O$ représente un couple entrée, sortie. Les autres notations courantes sont **x ?** **,y !** et **x : y**.

Le fonctionnement du distributeur

Le fonctionnement du distributeur :

• Scénario 1 : supposons qu'un client introduise une pièce de 10 et une de 20.

➥ Il ne reçoit rien (lettre n) en retour ; puis appuie sur Ornage.

La machine commence à S_0 :

➥ $S_0 \times 10 = S_2$ (sans sortie)

➥ $S_2 \times 20 = S_6$ et sortie=n.

➥ La prochaine entrée = O : $S_6 \times O = S_0$ et la sortie = JO.

➥ La machine est revenue à l'état S_0 : prête !

• Scénario 2 : le client introduit deux pièces de 10 et une de 20.

➥ $S_0 \times 10 = S_2$ et sortie=n.

➥ $S_2 \times 10 = S_4$ et sortie=n.

➥ $S_4 \times 20 = S_6$ et sortie=10.

➥ La prochaine entrée = R : $S_6 \times R = S_0$ et la sortie = JP.

➥ La machine est revenue à l'état S_0 : prête !

• Voir la table d'états ou l'automate pour le fonctionnement.

MEF : définitions

Définition (MEF)

Une Machine d'états fini est représentée par le sextuplet $M = (S, I, O, f, g, s_0)$ *où :*

- S : *un ensemble d'états*
- I : *l'alphabet en entrée (input)*
- O : *l'alphabet de sortie (output)*
- f : *une fonction de transition de la forme* $f : I \times S \rightarrow S$
- g : *une fonction de sortie de la forme* $g : I \times S \rightarrow O$
- s_0 : *un état initial*

• Pour une MEF M, on peut définir une **table d'états** représentant la fonction de transition f et la fonction de sortie g pour toutes paires $(état \times entrée) \in (S \times I)$.
• Parallèlement, un graphe orienté et valué appelé **diagramme d'états** permet de représenter la MEF (cf. exemple suivant).

Définition : exemple

Exemple : (table d'états et diagramme d'états)

➔ Pour cet exemple, on a :

$$S = (s_0, s_1, s_2, s_3),\ I = \{0, 1\},\ O = \{0, 1\}$$

Etats	f		g	
	Entrées		Entrées	
	0	1	0	1
S_0	S_1	S_0	1	0
S_1	S_3	S_0	1	1
S_2	S_1	S_2	0	1
S_3	S_2	S_1	0	0

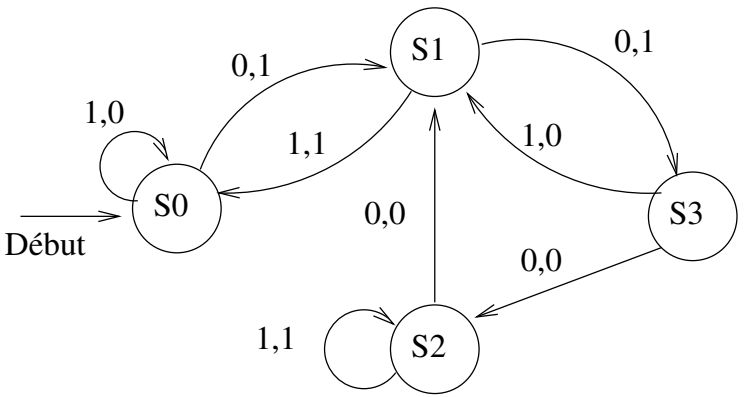


Fig.: Le diagramme des états

- L'entrée (et la sortie) d'une MEF est représentée par une chaîne de caractères.
- Une MEF (avec sorties) est également appelée un **transducteur**.

MEF (suite)

• La fonction de transition f permet à la machine de changer d'état :
$f : I \times S \rightarrow S$.

➥ Par exemple, pour la MEF précédente :

➛ $f : s_1 \times 0 \rightarrow s_3$.

• De même, la fonction de sortie g permet à la machine d'émettre une sortie lors d'une transition : $g : I \times S \rightarrow O$

➥ Par exemple, pour la même MEF et la même transition :

➛ $g : s_1 \times 0 \rightarrow 1$

• Pour une MEF donnée, si $x = x_1 x_2, ..., x_k$ et $y = y_1 y_2, ..., y_k$ tel que y_i est la sortie associée à l'entrée x_i, on dira alors que la sortie y est produite pour l'entrée x et on peut écrire :

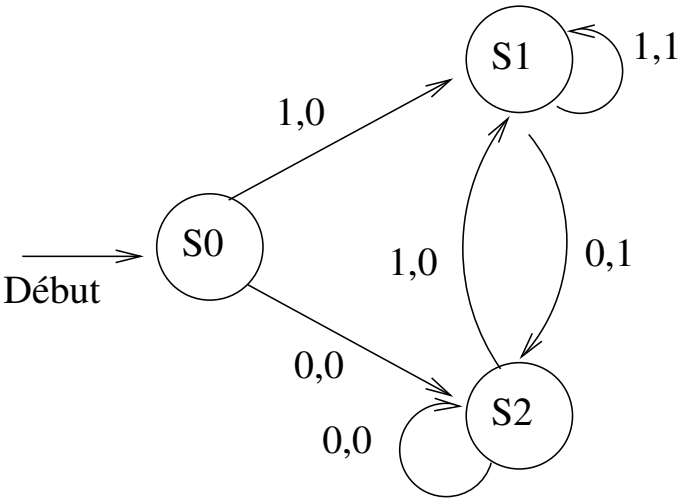
➛ $g(x) = y$. C-à-d : **la traduction de** x **est** y.

• Cette vision du rôle des MEFs permet de multiples applications.

Exemple MEF : unite delay machine

Une machine de retardement d'une unité (*temporisateur*) :
Proposer une MEF basique qui permet de retarder les entrées :
➔ Si l'entrée est la chaîne de bits $x_1 x_2 ... x_k$, alors la sortie $= O\ x_1 x_2 ... x_{k-1}$

La machine mémorise le bit d'entrée et le transmet (avec un décalage) à la prochaine sortie :
➙ si l'entrée $=$ bit 0, la machine entre dans l'état s_2 et accepte tous les bits de cette valeur.
➙ Un bit d'entrée 1 met la machine dans l'état s_1...



➙ La chaîne d'entrée est décalée d'un bit à droite.

Une application de la machine

Cette MEF opère une **division entière** par 2 (un décalage (*shift*) à droite).
➙ L'entrée et la sortie sont sur un nombre fixe de bits.
➙ Par exemple, sur 2 bits (une case fictive à droite peut récupérer le bit poussé à droite ; il représente le **reste** de la division par 2) :

$00 \Rightarrow 00$ (0 / 2 = 0) et un '0' en 3e bit (fictif) droite (00 0)
$01 \Rightarrow 00$ (1 / 2 = 0) et un '1' en 3e bit droite (00 1)
$10 \Rightarrow 01$ (2 / 2 = 1) et un '0' en 3e bit droite (10 0)
$11 \Rightarrow 01$ (3 / 2 = 1) et un '1' en 3e bit droite (01 1)

• On peut envisager le diagramme d'états d'une machine qui opère une **multiplication** par 2.
➙ Elle opère un décalage à gauche : on introduit un '0' à droite. Un bit fictif peut récupérer le bit poussé à gauche.
➙ Sur 2 bits, on doit obtenir :

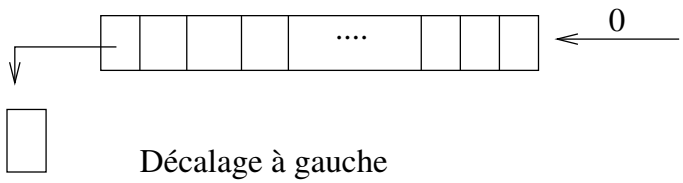
$00 \Rightarrow 00$ (0 * 2 = 0) et un '0' en 3e bit (fictif) gauche (0 00)
$01 \Rightarrow 10$ (1 * 2 = 2) et un '0' en 3e bit gauche (0 10)
$10 \Rightarrow 00$ (2 * 2 = 0) et un '1' en 3e bit gauche (1 00)
$11 \Rightarrow 10$ (3 * 2 = 1) et un '1' en 3e bit gauche (1 10)

Une application de la machine (suite)

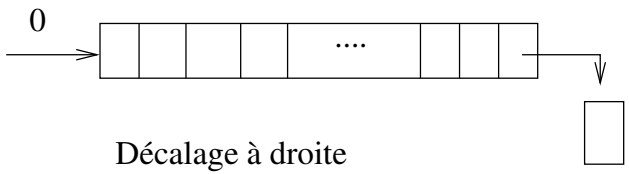
Pour résumer :

➙ Décalage à droite : on peut procéder à une division entière par 2 d'un nombre sur n bits, récupérer le quotient dans le résultat (les mêmes n bits) et le reste de la division dans le $(n+1)$ième bit (fictif dans l'exemple) à droite.

➙ Décalage à gauche : on peut procéder à une multiplication par 2 d'un nombre sur n bits, récupérer le résultat dans le $(n+1)$ bits : le mêmes $n$bits plus un de plus ajouté à droite, bit (fictif dans l'exemple) à droite.



Décalage à gauche



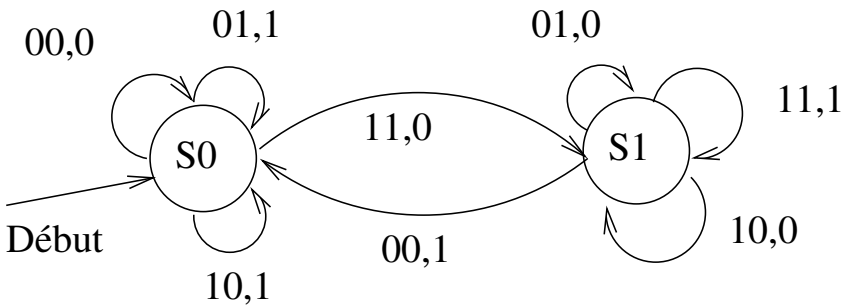
Décalage à droite

- L'Unité Arithmétique et Logique d'un processeur (UAL) sait procèder ainsi.
  ➥ Instruction assembleur (et C/C++) prévue.

Exemple MEF : additionneur

Exemple : une MEF qui aditionne deux nombres binaires (ces nombres sont complétés par un 0 à gauche). On additionne de droite à gauche.

$$\begin{array}{r} 0101 \\ +\ 0111 \\ \hline 1100 \end{array}$$



Pour l'addition de $(x_n ... x_1\ x_0)$ et $(y, ... y_1\ y_0)$:

- $x_0 + y_0 \rightarrow z_0$ et la retenue c_0.
- Ensuite, x_1 et y_1 et c_0 sont additionnés $\rightarrow z_1$ et c_1.
- Etc...
- A l'addition de $x_n + y_n$ (= 0 ici) avec c_{n-1}, on obtient z_n et la retenue c_n (= bit d'addition z_{n-1} lorsque $y_n = x_n$ =0).

MEF (additionneur) : suite

• **La construction de la MEF** :

N.B. : le problème est d'additionner 3 bits, $x_i, y_i,\ i > 0,\ avec\ c_i$

➙ Pour simplifier, on suppose que les bits y_n et x_n sont zéros (sinon, on doit prévoir un $(n+1)$ème bit pour c_n) ;

➥ Au besoin, on ajoute un zéro à gauche de x et de y et le $(n+1)$ème bit du résultat (z_{n+1}) de l'addition sera occupé par c_n.

$$\begin{array}{r} 101 \\ +\ 111 \\ \hline ... \end{array} \qquad \begin{array}{r} 0101 \\ +\ 0111 \\ \hline 1100 \end{array}$$

MEF (additionneur) : suite

• **La construction de la MEF (suite)** :

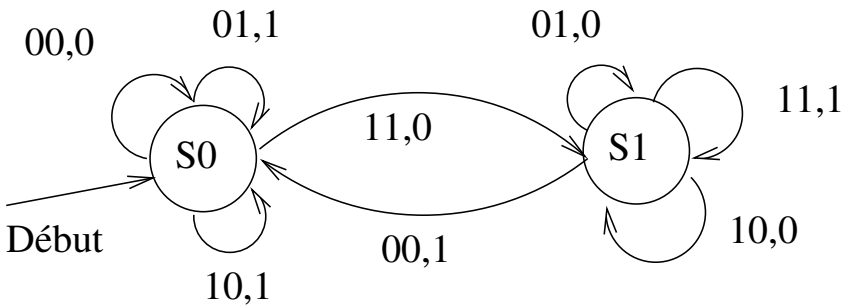
➙ L'état s_0 représente la retenue zéro et s_1 représente la retenue 1.

➙ Au départ, s_0 reflète le fait que la retenue précédente (retenue initiale) $= 0$.

➙ Chaque transition est représentée sous la forme $x_i\ y_i,\ z_i$ et l'état suivant dépend de la retenue de $x_i + y_i + c_{i-1}$.

• Principe : émettre z_i et aller dans l'état qui correspond à la retenue : s_0 si la retenue est 0, s_1 sinon.

$$\begin{array}{r} 0101 \\ +\ 0111 \\ \hline 1100 \end{array}$$



➔ Remarque sur s_1 et ses entrées : y sont représentés $x_i + y_i + c_{i-1}$ (s_1 tient lieu de la retenue=1) avec output$=z_i$ et état d'arrivée$=c_i$.

MEF d'addition (suite)

- Le tableau suivant permet de clarifier ce fonctionnement :
  Considérons la transition $S_{c_{i-1}} \times (x_i \ y_i), z_i \rightarrow S_{c_i}$
  avec $S_{c_{i-1}} = c_{i-1}$ et $S_{c_i} = c_i$.

c_{i-1} = état précédent	x_i	y_i	z_i = output	c_i = état suivant	**Transition**
0	0	0	0	0	$s_0 \times 00, 0 \rightarrow s_0$
0	0	1	1	0	$s_0 \times 01, 1 \rightarrow s_0$
0	1	0	1	0	$s_0 \times 10, 1 \rightarrow s_0$
0	1	1	0	1	$s_0 \times 11, 0 \rightarrow s_1$
1	0	0	1	0	$s_1 \times 00, 1 \rightarrow s_0$
1	0	1	0	1	$s_1 \times 01, 0 \rightarrow s_1$
1	1	0	0	1	$s_1 \times 10, 0 \rightarrow s_1$
1	1	1	1	1	$s_1 \times 11, 1 \rightarrow s_1$

➙ (ligne 4) : $1 + 1 + 0 = 0$ et état suivant=retenue=1 (aller à s_1)
(ligne 5) : $1 + 0 + 0 = 1$ et état suivant=retenue=0 (aller à s_0)

MEF : addition binaire revisitée

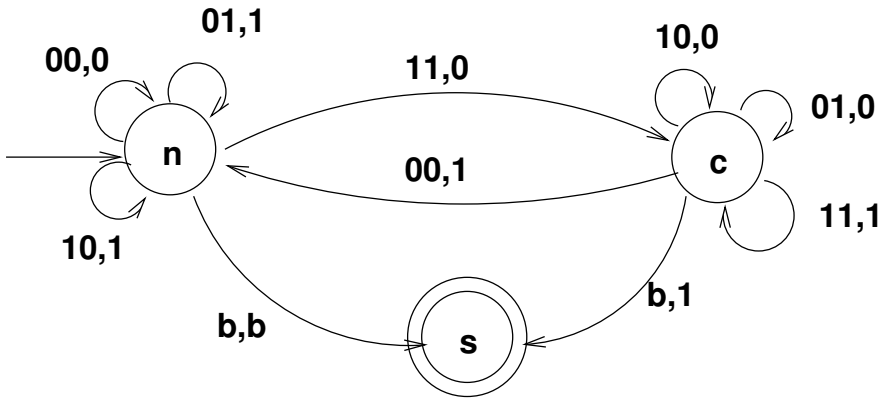
Addition binaire revisitée : exemple

Rappel : pour additionner deux nombres binaires quelconques, on peut ajouter (systématiquement) un 0 au début (à gauche) des nombres x et y pour avoir un résultat (z) de la même taille que x et y.

➙ Dans la variante ci-dessous, ces zéros supplémentaires sont caractérisés par un espace (noté par b) à gauche de x et y.

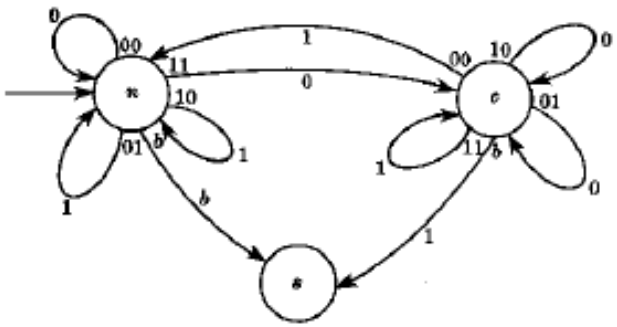
➔ Exemple : si l'entrée est $x = \mathbf{b}1101011$ et $y = \mathbf{b}0111011$, la sortie devra être $z = \mathbf{1}0100110$.

➙ Dans la MEF ci-dessous, les entrées sont les paires de bits dans x et y (**même taille avec b à gauche**), de droite à gauche. L'état S est un état final.



MEF : addition binaire revisitée (suite)

Les entrées :
$I = \{00, 01, 10, 11, b\}$,
Les sorties : $O = \{0, 1, b\}$.
Les états : $S = \{retenue(c), nonretenue(n), stop(s)\}$.
L'état initial = n (pas de retenue).



Théorème

Il n'y a pas de MEF générique capable de faire une multiplication binaire (de taille) quelconque.

Mais, si on limite la taille de ces nombres, alors une telle MEF existe.

➙ Les ordinateurs sont des exemples importants de MEF ; ils limitent la taille des nombres manipulés (➙ multiplication possible).

Exercice MEF : détecteur d'erreur

• **Détecteur d'erreur** : dans certains schémas de codage binaire, la présence de trois '1' consécutifs dans un message permet au récepteur de conclure sur une erreur de transmission.

➙ Construire une MEF qui permet de **détecter ces erreurs**. Le machine émet un '0' pour chaque entrée mais un '1' trois '1' consécutifs présents.

MEF Melay et Moore

- La machine de l'exemple précédent est un **accepteur de langages**.
  ➙ transmet un '1' si une certaine chaîne d'entrée ("111") est lue.
- La reconnaissance de langages est une des applications importantes des MEFs.

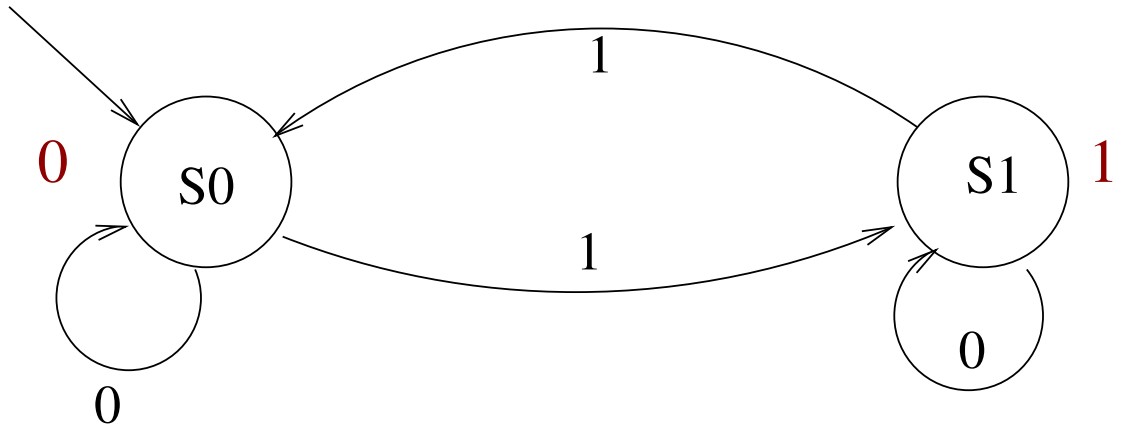
- Les MEFs vues en exemple : machines de **Melay** (G.H. Melay, 1955).
  ➥ La MEF détecteur d'erreur de codage en est un exemple, utilisable pour la reconnaissance d'un langage.

- Si la sortie d'une MEF est uniquement déterminée par un certain état, il s'agira d'une machine de Moore (J.F. Moore, 1956).

- **Une conséquence de la machine de Moore :**
  ➙ Une MEF sans sortie = AEF = un **Automate d'états fini**.
  ➥ Un ensemble d'états particuliers finaux permet de décider, en partant de S_0, de l'acceptation (reconnaissance) d'un langage.

MEF Moore : Exemple

• Exemple : donner une MEF qui lit un flot d'entrée et produit un '1' si le nombre des '1' lus dans une (sous) séquence de '1' est impaire.
• La machine d'états fini (Moore) :



✓ Chaque état "affiche" une valeur.

MEF : Exercices

• Créer une MEF qui change, sur le flot d'entrée, un bit sur deux (en commençant par le 2e bit) et laisse l'autre bit inchangé.

MEF : Exercices (suite)

- Dessiner les diagrammes d'états pour les tables d'états suivantes :

Etats	**f**		**g**	
	Entrées		**Entrées**	
	0	1	0	1
S_0	S_1	S_0	0	1
S_1	S_0	S_2	0	1
S_2	S_1	S_1	0	0

MEF - Exercice : Suite

Dessiner le diagramme d'états pour la table d'états suivante :

Etats	**f**		**g**	
	Entrées		**Entrées**	
	0	1	0	1
S_0	S_2	S_4	1	1
S_1	S_0	S_3	0	1
S_2	S_0	S_2	0	0
S_3	S_1	S_1	1	1
S_4	S_1	S_0	1	0

MEF : Exercice

• Créer la table puis le diagramme d'états pour un distributeur de café basique qui ne rend pas la monnaie.

➙ La machine accepte des pièces de 5,10 et 20 centimes.

➙ Le prix du café est 30.

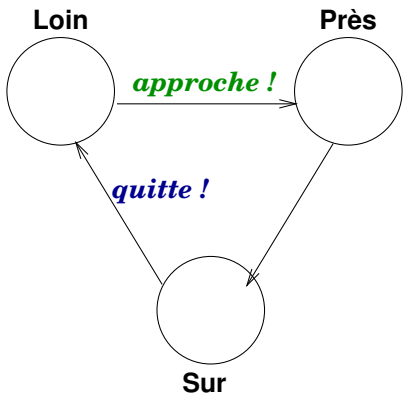
➙ Après l'introduction des 30 centimes (ou plus : tant pis !), la machine délivre un café (une seule sorte de café).

➔ Une fois le service enclenché, toute pièce sera pour le café suivant.

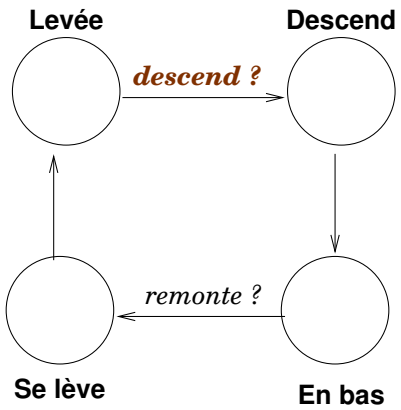
suite café

• Dans la MEF précédente, on sohaite que le surplus soit mis au crédit des suivants. Modifier la solution en conséquence.

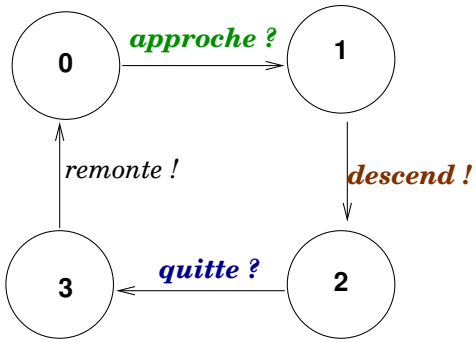
Exemple : controleur de passage à niveau



Le train



La barrière



Le Controleur

• Ces MEFs décrivent le comprtement de : Train, Barrière, Controleur.

- Les capteurs du train **émettent** (en sortie) les signaux "approche" et "quitte" (avec !)

- Ceux de la barrière **reçoivent** des commandes "descend" et "remonte" (en entrée avec ?) émises par le controleur.

- Ceux du controleur reçoivent "approche" et "quitte" (du train) et émettent "descend" et "remonte".

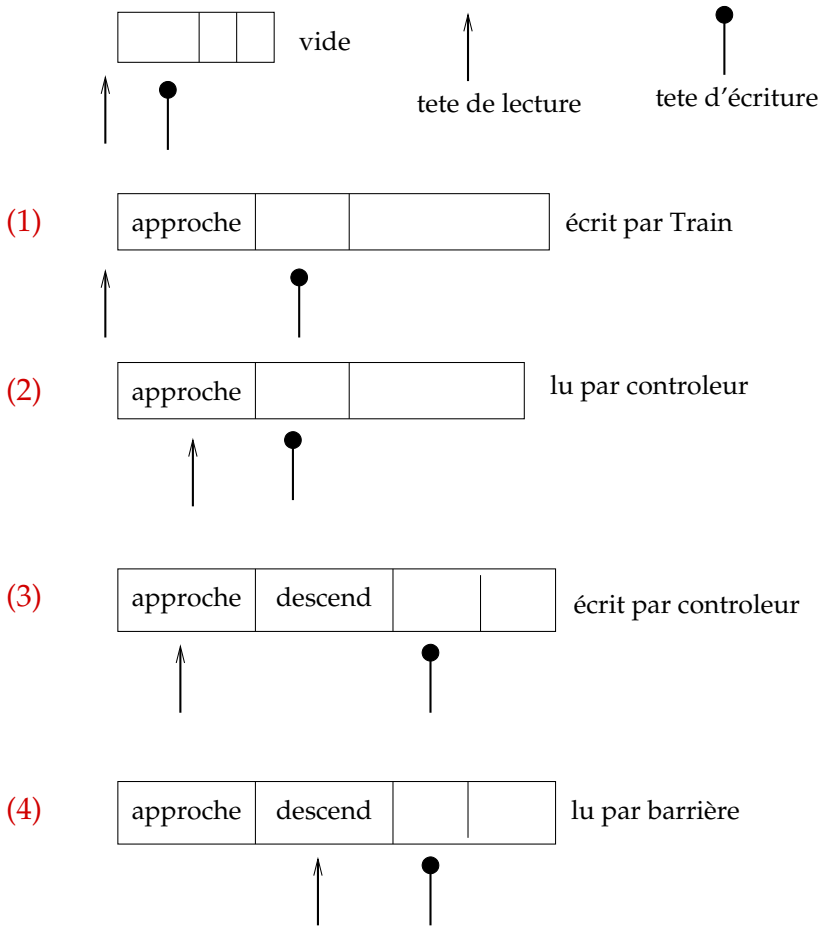
• La communication a lieu par des moyens divers : un réseau pour un poste de passage à niveau (lecture/écriture sur un ruban pour la MEF).

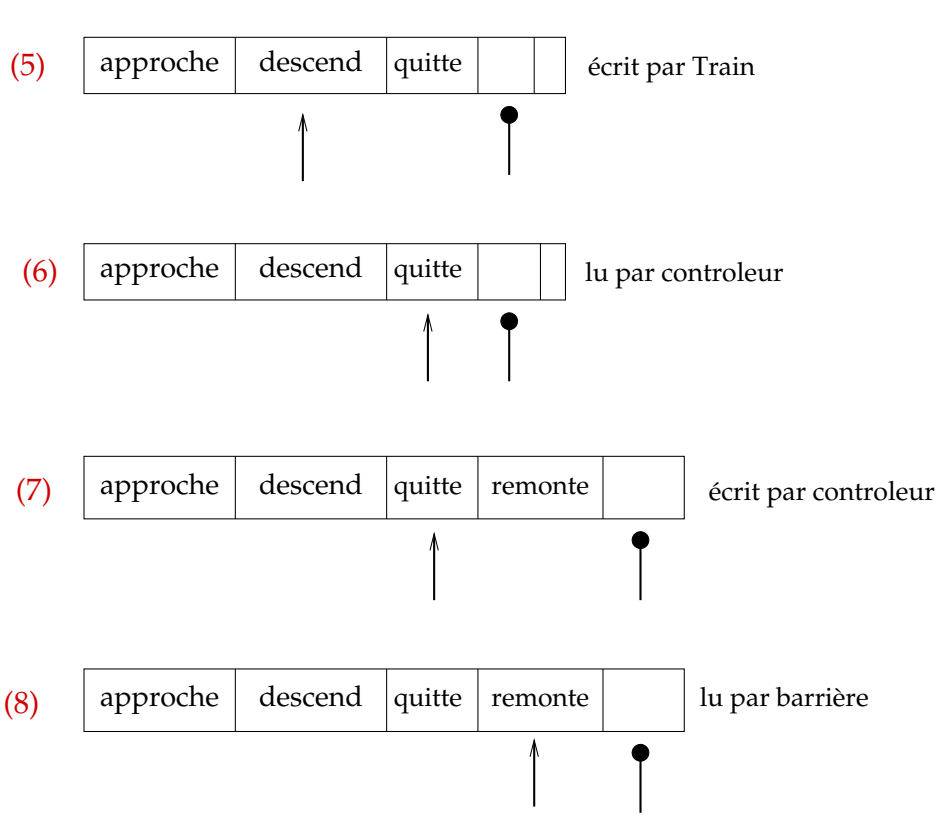
Exemple : controleur de passage à niveau (suite)

Controleur	Symbole	Train	Barrière
➙ Le train s'approche du passage et des capteurs préviennent de cette approche			
$S_0 \rightarrow S_1$	approche	$Loin \rightarrow Pres$	Levée
➙ Le controleur reçoit "approche" et émet "descend"			
$S_1 \rightarrow S_2$	descend	Près	$Levee \rightarrow Descend$
➙ La barrière a reçu "descend"			
S_2		Près	$Descend \rightarrow En\ bas$
➙ La train arrive sur le passage et des capteurs préviennent de cette approche			
S_2		$Pres \rightarrow Sur$	En bas
➙ ... Le train passe ...			
$S_2 \rightarrow S_3$	quitte	$Sur \rightarrow Loin$	En bas
➙ ... Le train s'éloigne ...			
$S_3 \rightarrow S_4$	remonte	Loin	$En\ bas \rightarrow Se\ leve$
S_0		Loin	Levée

Exemple : controleur de passage à niveau (suite)

L'état du Ruban synchronisé :





Exemple : controleur de passage à niveau (suite)

• La MEF précédente est une version *parallèle asynchrone* : les 3 MEFs tournent de manière indépendante et en parallèle mais se partagent un Ruban unique.

➙ La MEF du train représente ses états : {loin, près, sur}.

➙ La MEF de la barrière représente les états de la barrière : {levée, descend, en bas, se lève}.

• Cette exemple utilise des ε-transitions (voir plus loin)

MEF et les calculs

Extensions des MEFs

• Des extensions permettent d'effectuer des calculs dans les MEFs.

• Des outils (cf. LEX) permettent d'ajouter des opérations traduites dans un langage hôte (C/C++,JAVA,PASCAL/ADA, ...)

Exemple : la machine à café précédente est munie d'une sous-machine spécialisée (M1) qui rend la monnaie de n centimes restants, n multiple de 5.

➤ M1 peut rendre des pièces de 5, 10 et 20.

Soit $n \geq 0$ et multiple de 5.

$$
\begin{array}{lll}
S_0 \times \{n \geq 20\}, 20 & \rightarrow & \{n = n - 20\},\ S_0. \\
S_0 \times \{n \in [10, 20[\}, 10 & \rightarrow & \{n = n - 10\},\ S_0. \\
S_0 \times \{n \in [5, 10[\}, 5 & \rightarrow & \{n = n - 5\},\ S_0. \\
S_0 \times \{n = 0\}, rien & \rightarrow & S_1. \quad \text{état final.}
\end{array}
$$

• Le peseudo code correspondant sera directement obtenu. ../..

Par exemple :

MEF et les calculs (suite)

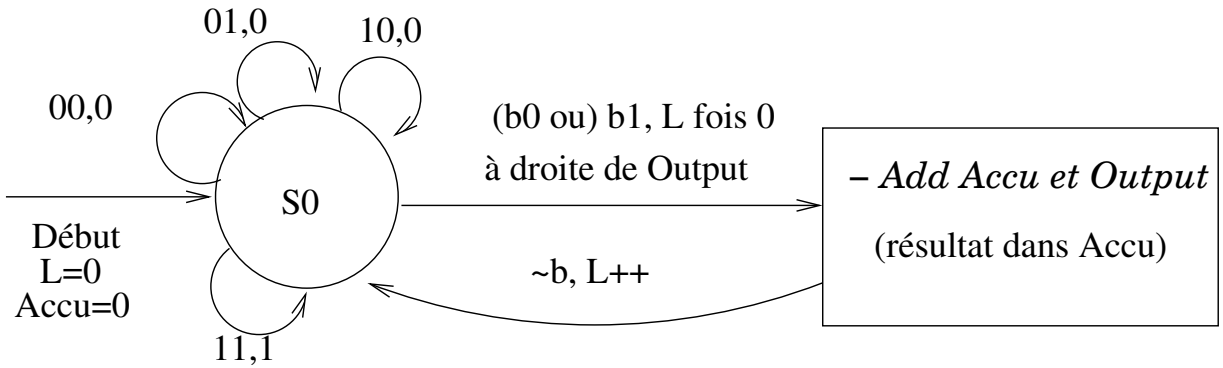
Fonction machine_à_café(n : entier) :
  Début
    étiquette S0 :
      Si n >= 20
      Alors n = n-20 ; émettre 20 ; aller à S0 ;
      Si n >= 10 et n < 20
      Alors n = n-10 ; émettre 10 ; aller à S0 ;
      Si n >= 5 et n < 10
      Alors n = n-5 ; émettre 5 ; aller à S0 ;
    étiquette S1 : fin
  Fin machine

➥ Les calculs (*n=n-20, n=n-10 et n=n-5*) peuvent modifier les entrées :
- on lit (et consomme) la valeur de n qui est retiré du ruban d'entrée et la nouvelle valeur de n est injectée dans ce même ruban.
- à la fin des lectures, $n = 0$ sera consommé (cf. MMG).

Extension des MEFs : multiplicateur binaire

- Une idée du **multiplicateur binaire** (taille limitée)
  ➙ Soit l'exemple de multiplication :

$$\begin{array}{r} 101 \\ \times\ b\ 11 \\ \hline 101 \\ +\ 101\mathbf{0} \\ \hline 1111 \end{array}$$



Principe : les additions sont faites au fur et à mesure entre le flot **Output** et un **accumulateur**, à l'aide de l'additionneur précédent.

➙ Le résultat final sera dans l'accumulateur.

➙ La présence d'un espace (b) dans le 2e multipliant signale la fin des multiplications.

➙ La variable L (init 0) représente le nombre de 0 à ajouter à droite du flot Output (pour les additions).

➙ $\sim b$ veut dire : tout input sauf un espace.

MEF (Suite multiplicateur)

Fonctionnement : pour simplifier, on considère (2) multiplié par (1).

- ➙ Les couples de bits présentés de droite à gauche.
- ➙ La multiplication du premier '1' de (2) par (1) produit (3) : $Output = 101$;

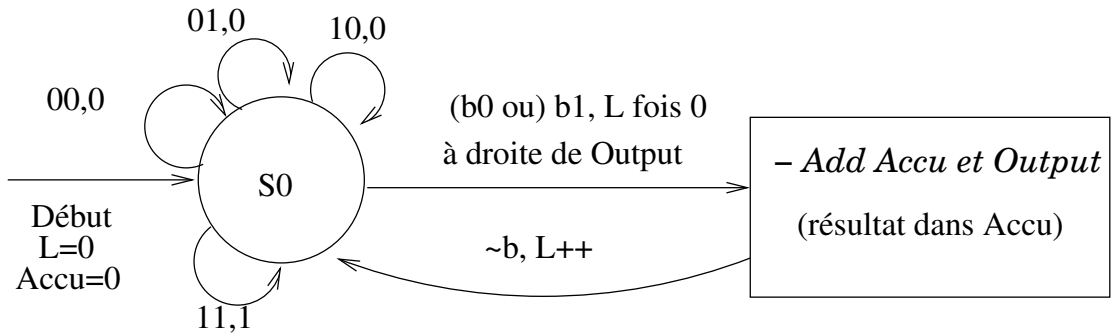
$L = 0 \Rightarrow$ on ajoute aucun 0 à droite de (3).

- ➙ L'addition de Accu (init 0) avec (3) donne 101.
- ➙ input $=\sim b (\neq b)$, on reprend la multiplication avec L=1.
- ➙ La multiplication du second '1' de (2) par (1) produit $Output = 101$;

$L = 1 \Rightarrow$ on ajoute un 0 à droite de (4).

- ➙ L'addition de l'Accu (=101) avec (4) donne (5).
- ➙ Ayant sur (2) un input $=b$, la résultat final est lu dans Accu.

$$
\begin{array}{rl}
101 & (1) \\
\times\ b\ 11 & (2) \\
\hline
101 & (3) \\
+\ 101\mathbf{0} & (4) \\
\hline
1111 & (5)
\end{array}
$$



MEF : distributeur boissons bis

• Créer la table puis le diagramme d'états pour le distributeur de boissons suivant (en vous inspirant de l'exemple du distributeur) :

➙ La machine accepte des pièces de 5,10,20 et 50 centimes.

➙ Le prix des boissons de cette machine est fixé à 35.

➙ La machine rend le surplus.

➙ Après l'introduction des 35 centimes, le choix est entre $X \in \{L, C, T\}$ avec L = limonade, C = cidre et T = jus de tomates.

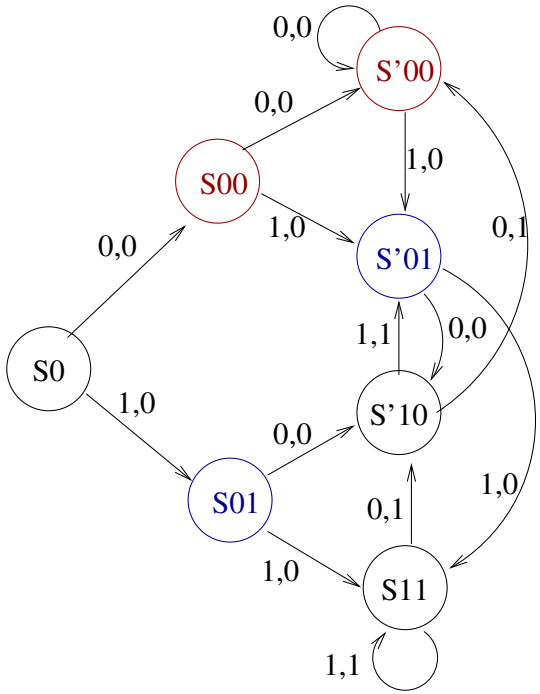
MEF : Exercice

- Construire une MEF de temporisation avec un retardement de 2 bits.
  ➙ La machine placera '00' comme les deux premiers bits de la sortie.

Suite : modus operandi

Comment faire ? : dans un premier temps, on produit une MEF sans chercher à minimiser les états.
N.B. : pour mieux se repérer, appeler $S00$ "je dois 00" (les 2 prochaines sorties devront être 00), ... , appeler $S11$ "je dois 11" , etc.

- On pourra ensuite minimiser le nombre d'états en regroupant les états similaires.
  Par exemple, $S00$ et $S'00$ sont exactement identiques (mêmes entrées abutissent aux mêmes états), idem pour $S01$ et $S'01$.
- On constatera ensuite que $S1$ de la machine précédente $= S00$, $S2 = S01$, $S3 = S11$ et $S4 = S'10$.
- Voir plus loin l'algorithme de minimisation.



MEF : login

• Créer une MEF pour une procédure de *login* sur un ordinateur :

➙ La machine demande un identifiant (une seule entrée) puis un mot de passe (une seule entrée).

➙ La vérification se fait seulement après le mot de passe. En cas d'erreur (ID ou PW), on recommence toute la procédure.

➙ Après le login, un prompte (une invite) est affiché et on lit des commandes quelconques (lignes de commandes).

MEF : Exercices

• Créer une AEF pour un coffre fort qui contient les nombres de 1 à 40.
➙ Le coffre s'ouvre si la combinaison correcte suivante est donnée :
10 droite, 8 seconde (2e) gauche, 37 droite.
➙ Chaque entrée est un triplet : (un nombre, une direction, le nombre de fois où le combiné est rourné dans cette direction).

• Même question avec une entrée = (un nombre, une direction).
➔ On suppose que chaque entrée concerne un tour dans la direction indiquée.
• Que modifier si une erreur dans l'une des 3 étapes devait tout recommencer ?

MEF : Exercices

• Créer une MEF pour une barrière d'autoroute controlée par un panier dans lequel on dépose les pièces de 5, 10 et 20 centimes.

➙ Le prix du passage = 25 centimes.

➙ La machine ne rend pas la monnaie et aucun crédit n'est donné au client suivant lorsque plus de 25 centimes sont introduits.

MEF : Exercices

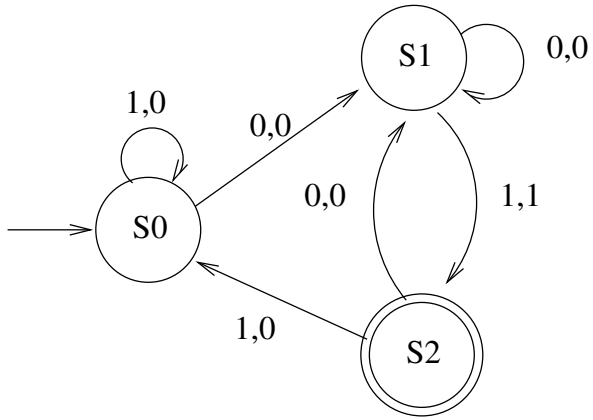
• Créer une MEF (de Moore) qui produit une sortie =1 si le nombre de symboles lus en entrée est divisible par 3 ; 0 sinon.
• N.B. : les sous schéma pour 2, 3, 4 ....

MEF : 0-1 à la fin

• Créer une MEF qui détermine si la chaîne lue en entrée (composée de 0 et de 1) possède un '1' en dernière position et un '0' en avant dernière (pénultiemme).

Suite 0-X-1

• La MEF précédente peut ensuite être transformée en une version déterministe (voir plus loin pour ces conversions).



Remarques : cette machine (déterministe) est capable de reconnaître **une** séquence de n'importe quelle taille terminant par 01 (...01 y compris contenant 01 à l'intérieur : ...01...01, etc.).
➔ Pour obtenir la MEF pour un seul motif terminant par 01, supprimer les transitions qui partent de S_2.
N.B. : l'expression régulière pour un seul motif est $(?)^*01$.

MEF : Exercices

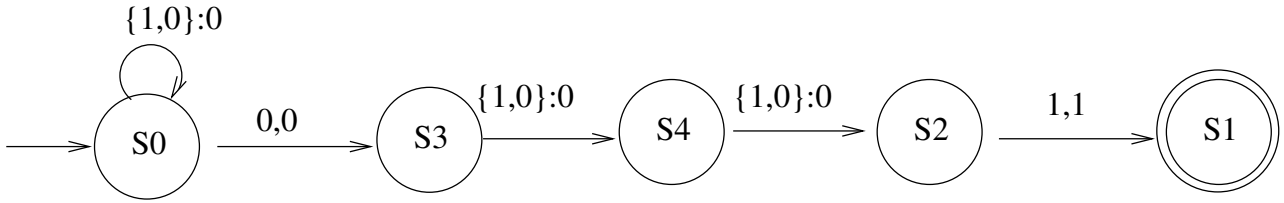
• Créer une MEF qui détermine si la chaîne lue en entrée (composée de 0 et de 1) possède un '1' en dernière position et un '0' en avant avant dernière (antepenultiemme).

MEF : Exercices

• Créer une MEF qui détermine si la chaîne lue en entrée (composée de 0 et de 1) possède un '1' en dernière position et un '0' en 3e position avant la fin (ante-antepenultiemme ?)

MEF (suite exercice)

Pour la même question, la machine non déterministe est :



On peut remarquer que cette machine est bien plus simple mais non déterministe (cf. S_0).
➔ La machine donnée en page précédente est sa forme déterministe.
➔ Voir la section de transformation des AEFs.

Rappel : une machine non déterministe est une machine dont au moins un des états aboutit à plusieurs autres états en présence d'une même entrée.
➔ Dans l'exemple : $S_0 \times 0 = S_0$ et $S_0 \times 0 = S_3$.

Machine de Moore

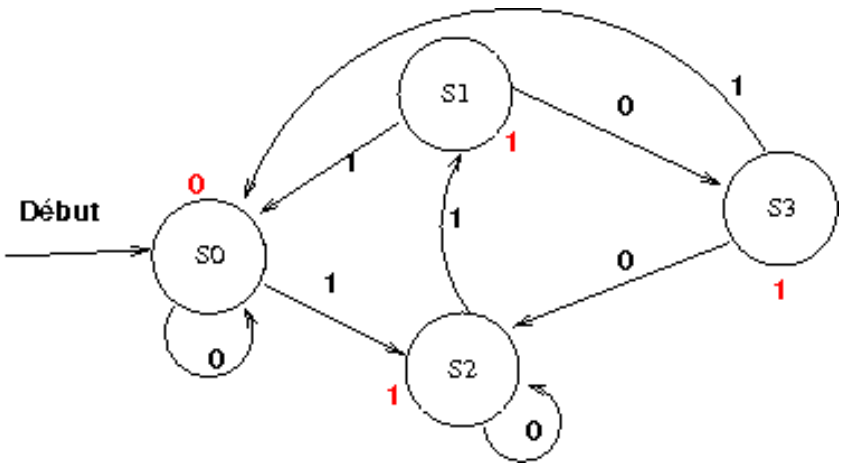
• **Une machine de Moore** est une MEF où une sortie est associée à n'importe quel couple (*état* × *entrée*).

➙ La différence par rapport à une MEF : la sortie ne dépend pas d'une entrée particulière : une sortie est alors associée à chaque état.

➙ La fonction g devient $g : S \rightarrow O$ et la séquence des états activés donne la chaîne de sortie.

• Un exemple où $g(s_0) = 0, g(s_1) = g(s_2) = g(s_3) = 1$:

Etats	**f**		**g**
	Entrées		
	0	1	
S_0	S_0	S_2	0
S_1	S_3	S_0	1
S_2	S_2	S_1	1
S_3	S_2	S_0	1



Exercice Moore

• Construire une MEF (de Moore) qui détermine si une chaîne d'entrée contient un nombre pair ou impair de '1'.

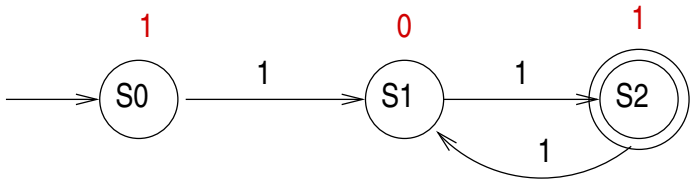
➙ La machine transmet un '1' si ce nombre est pair ; 0 sinon.

• Remarque : la machine ne s'occupe pas du nombre de '0' en entrée : la présence d'un '0' ne remet pas en cause le nombre de '1's .

Exercice : suite

Pour la même question, on peut proposer les étapes suivantes :

- On s'occupe d'abord des '1's :



- On procède ensuite à l'ajout des transitions avec '0'.
  Les schémas ci-dessous représentent trois solutions équivalentes :

Mot FINI

• Donner le diagramme d'une MEF qui détermine si le mot **FINI** a été lu sur un flot d'entrée. Emettre 1 si oui, 0 sinon.
➔ On suppose que l'alphabet A est constitué des lettres habituelles.

Mot "FINI" (suite)

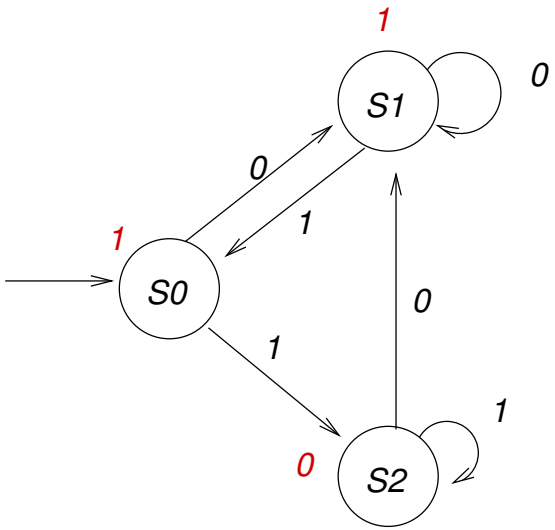
- Donner la machine de *Moore* de la machine de *Melay* précédente.
  ➥ un succès émettra '1', 0 sinon.

Exercices (suite)

• Construire la table d'états de la machine de Moore suivante.

Rappel : une machine de Moore associe une sortie à chaque état :

➙ Si l'entrée est la chaîne $a_1, a_2, ..., a_k$, la sortie sera la chaîne $g(s_0)g(s_1), ..., g(s_k)$ avec $s_i = f(s_{i-1}, a_i), i = 1, 2, ..., k$



• Quelles sont les sorties de cette machine pour les entrées 0101 et 111111 ?

Exemple métro revisité

Extension des MEFs

N.B. : la machine de Moore est une restriction des MEFs ; elle associe une sortie à un état, quelque soit l'entrée.

Pour les besoins des applications réelles, les MEFs ont été étendues.

➙ **Exemple barrière (métro)** :

Comme toute machine de Moore, on a une sortie associée à chaque état.

Les transitions peuvent (éventuellement) émettre une action dite *sémantique* de la forme $I : Actions$.

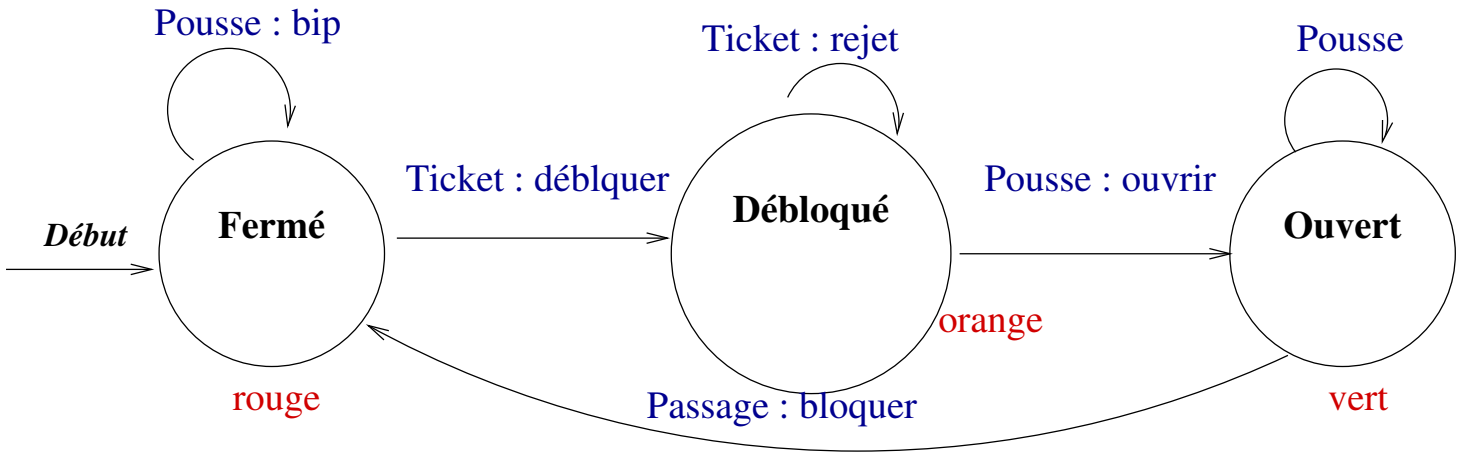


Fig.: AEF basique d'une entrée métro

suite

Remarques :

Une telle machine étendue est définie par le septuplet suivant :

$M = (Etats, Alphabet, Fonction\ de\ transition, Fonction\ de\ sortie,$
$Fonction\ d'action, Etat\ initial, Etat\ final)$ Avec ici :

- Etats $= \{Fermé, Ouvert, Débloqué\}$
- Alphabet $= \{Pousse, Ticket, Passage\}$
- Fonction de sortie (associée aux états) $=$
  $\{Fermé : rouge, Débloqué : vert, Ouvert : vert\}$
- Etat initial $= Fermé$
- Etat final $= \{\}$ état optionnel
- Fonction de Transition (et d'action) $=$
  $\{Fermé \times Ticket \rightarrow Débloqué : débloquer,$
  $Débloqué \times Pousse \rightarrow Ouvert : ouvrir,$
  $Ouvert \times Passage \rightarrow Fermé : bloquer,$
  $Débloqué \times Ticket \rightarrow Débloqué : rejet\}$

Exercice

Exercice :

• Compléter ensuite l'exemple métro en ajoutant un état **violation** enclenché lorsque *pousse* est appliqué à l'état *fermé*.

➔ On sortira de cette état de violation suite à un *reset*.

➔ Prévoir également le traitement des tickets invalides.

MEFs (suite)

N.B. : Dans la version étendue des MEFs, on peut avoir des **préconditions** à l'application d'une transition.

➙ La présence d'un certain symbole en entrée peut alors être considérée comme une précondition, mais l'on peut en avoir d'autres (par exemple, des évènements, le résultat d'un test autre que sur une entrée, ...).

➙ Cependant, cette vision nous éloigne de **l'approche orientée syntaxe**.

➙ De plus, on peut en général éviter ces préconditions en les intégrant dans l'alphabet.

Vers les Automates d'états fini (AEFs) : introduction informelle

- Les MEF sans sortie = AEF.
- Rappel : une des applications des MEFs est la reconnaissance des langages.
  ➔ **Traitement orienté syntaxe**.

- Un AEF est une MEF dont la sortie portera la valeur VRAIE si l'entrée correspond à une reconnaissance avec succès ; FAUSSE sinon.
  ➙ Le succès est matérialisé par le passage par un des états finaux.

➔ Un AEF (de base) n'a donc pas de fonction de sortie.
  ➙ On aura un ensemble d'états finaux dont un (ou plusieurs) correspond à un succès.
  ➙ Dans un AEF, on a un échec de reconnaissance si l'on ne passe pas par un état final.
  ➙ Dans une forme étendue, un AEF peut avoir un (ou plusieurs) états explicitement associés à un échec de reconnaissance.
- Avant de continuer, un exemple puis quelques définitions sont nécessaires.

AEF (définitions)

Définition

✓ *Pour un vocabulaire Σ contenant des caractères.*

➙ *Σ^* désigne toute chaîne (de caractères) de taille ≥ 0.*

➙ *Σ^+ désigne toute chaîne (de caractères) de taille ≥ 1.*

➔ *On a : $\Sigma^+ = \Sigma\ \Sigma^*$*

✓ *Soient A et B deux sous ensembles de Σ^*, où Σ est un vocabulaire.*

➙ *La concaténantion de A et de B, notée AB (ou $A.B$) désigne l'ensemble des chaînes xy avec $x \in A, y \in B$.*

- Exemple-1 : si $A = \{a, b, c\}$ et $B = \{0, 1\}$, alors
  - ➙ $AB = \{a0, a1, b0, b1, c0, c1\}$
  - ➙ $BA = \{0a, 0b, 0c, 1a, 1b, 1c\}$
- Exemple-2 : si $P = \{Pierre, Paul\}$ et $J = \{Jean, Jacques\}$, alors
  - ➙ $PJ = \{PierreJean, PierreJacques, PaulJean, PaulJacques\}$
  - ➔ Aussi, $JJ = \{JeanJean, JeanJacques, JacquesJean, JacquesJacques\}$

... Suite (AEF)

Définition

Pour un alphabet Σ, on définit $\Sigma^n, n = 0, 1, 2, ...$ par

- $\Sigma^0 = \{\lambda\} = \emptyset$ *le langage vide*
- $\Sigma^{n+1} = \Sigma^n.\Sigma$

- Exemple : avec $A = \{1, 00\}$, on a
  - $A^0 = \{\lambda\} = \emptyset$
  - $A^1 = A^0 A = \{\lambda\}A = A = \{1, 00\}$
  - $A^2 = A^1 A = \{1, 00\}A = \{11, 100, 001, 0000\}$
  - $A^3 = A^2 A = \{11, 100, 001, 0000\}A$
    - $= \{111, 1100, 1001, 10000, 0011, 00100, 00001, 000000\}$

Définitions ... (suite)

Définition

Soit A un sous ensemble de Σ^*
La fermeture de Kleen de A*, notée* A^* *est l'ensembe contenant la concaténation de n'importe quelle chaîne arbitraire de* A*.* $A^* = \bigcup_{k=0}^{\infty} A^k$

➙ On constate que A^* est une généralisation de A^n : A^* représente une répétition non bornée de A alors que l'on peut borner A^n en fixant une valeur pour n : A^4 veut dire $AAAA$.

- Exemple : avec $A = \{0\}, B = \{0, 1\}, C = \{11\}$, on a
  - ➙ $A^* = \{0^n | n = 0, 1, 2, ...\}$
  - ➙ $B^* =$ toute séquence composée de 0 et/ou de 1.
  - ➙ $C^* =$ toute séquence contenant un nombre pair de 1 $= \{1^{2n} | n = 0, 1, ...\}$

Automates d'états finis

Les Automates d'états finis (AEFs)

• Les automates d'états finis (AEF) sont équivalents aux langages (et expressions) réguliers (voir plus loin pour les définitions).
• Un AEF se définit par le quintuplet $A = (\Sigma, S, s_0, F, \delta)$ où :

Σ	vocabulaire
S	ensemble d'états
s_0	état initial, $s_0 \in S$
F	états finaux, $F \subseteq S$
δ	fonction de transition, $\delta : \Sigma \times S \rightarrow S$

➙ On constate l'absence de sortie.

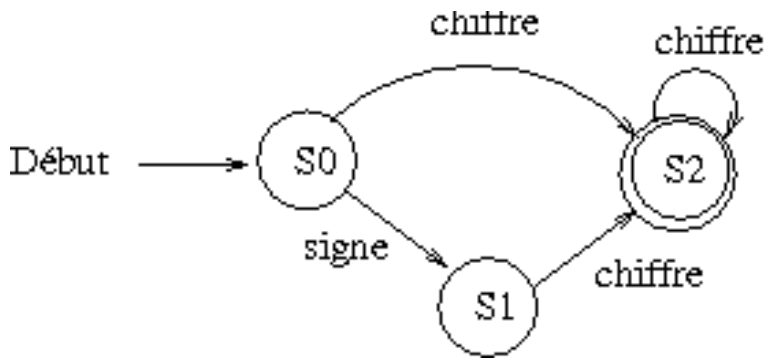
AEF : exemple

Un exemple : structure d'un entier signé :

➙ Le signe $(+)$ peut être omis.

- $S_0 =$ l'état initial,
- $S_2 =$ l'état final (signalé avec un double cercle)
- Le vocabulaire $\Sigma = \{+, -, 0, ..., 9\}$.
- *chiffre* et *signe* dans Σ : signification habituelle.

$$\delta(S_0, signe) = S_1$$
$$\delta(S_0, chiffre) = S_2$$
$$\delta(S_1, chiffre) = S_2$$
$$\delta(S_2, chiffre) = S_2$$



AEFs (suite)

• Dans un AEF, la fonction de transition peut être étendue de la manière suivante :

➙ Soit $\delta : S \times \Sigma \rightarrow S$.

➙ On note $\delta : S \times \Sigma^* \rightarrow S$.

Si $x = x_1 x_2 \dots x_k$ désigne le flot d'entrée ($x \in \Sigma^*$) et
$\delta(S_i, x_i) = S_{i+1}$, $\delta(S_{i+1}, x_{i+1}) = S_{i+2}, \dots, \delta(S_k, x_k) = S_j$

➙ On note $\delta(S_i, x) = \delta(S_k, x_k) = S_j$

➙ L'extention de δ est une **fermeture** partielle de δ.

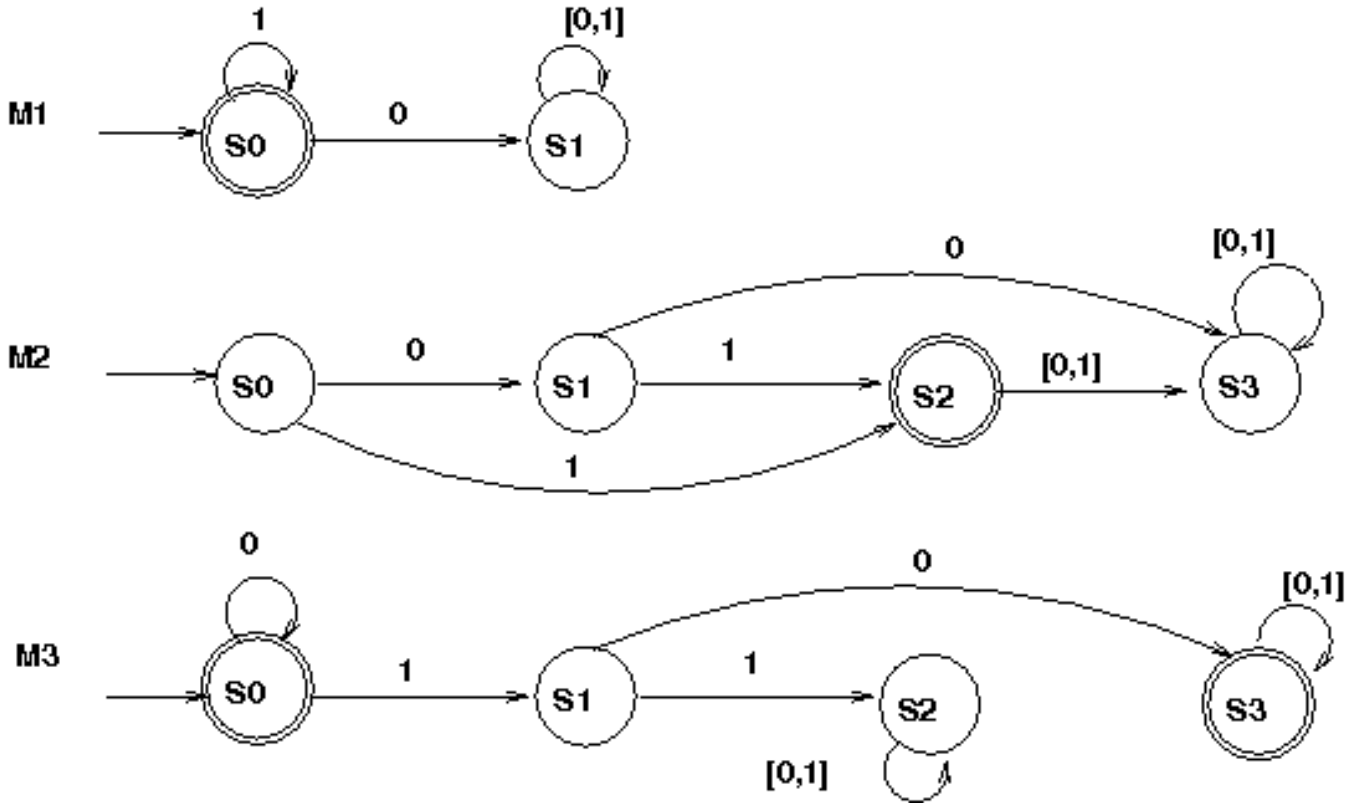
• On dit qu'une chaîne x est **reconnue** ou **acceptée** par un AEF M si l'on a appliqué les transitions de M à partir de S_0 et abouti à un état final : $\delta(S_0, x) \in F$

• Le **langage** accepté par un AEF M, noté $L(M)$ est l'ensemble des chaînes reconnues par M.

➙ Si deux AEFs reconnaissent le même langage, on dira qu'ils sont **équivalents**.

AEF : Execrcice

- Quel est le langage accepté par chacun des automates suivants :



Exercices

Exercices :

- Soit $A = \{0, 11\}$ et $B = \{00, 01\}$.
  ➙ Donner les ensembles AB, BA, A^2 et B^3.

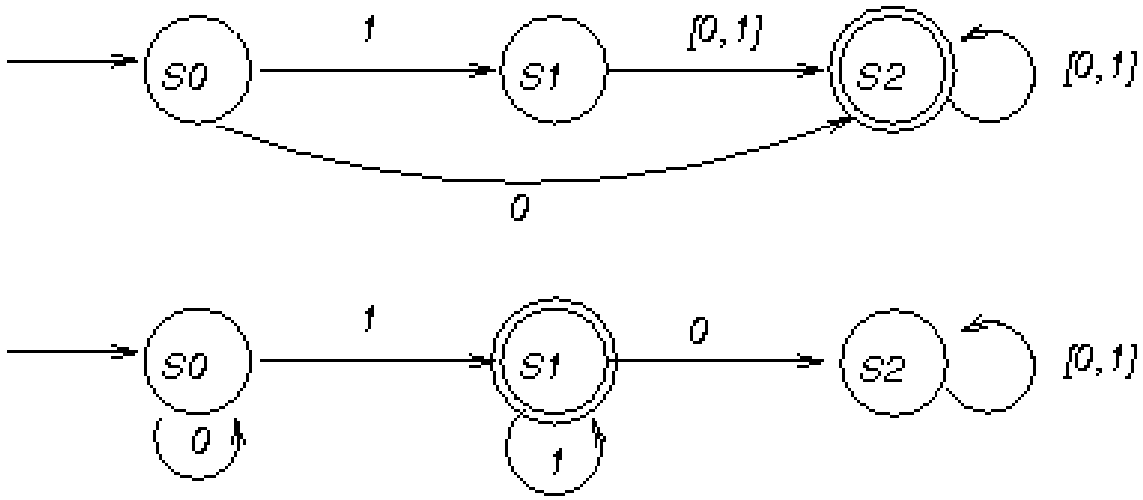
- Quels sont les ensembles A et B possibles tels que $AB = \{10, 111, 1010, 1000, 10111, 101000\}$?

Exercices (suite)

- Déterminer si la chaîne 11101 est dans chacun des ensembles suivants :
  - $\{0,1\}^*$
  - $\{11\}\{1\}^*\{01\}$
  - $\{111\}^*\{0\}^*\{1\}$
  - $\{1\}^*\{0\}^*\{1\}^*$
  - $\{11\}^*\{01\}^*$
  - $\{111,000\}\{00,01\}$

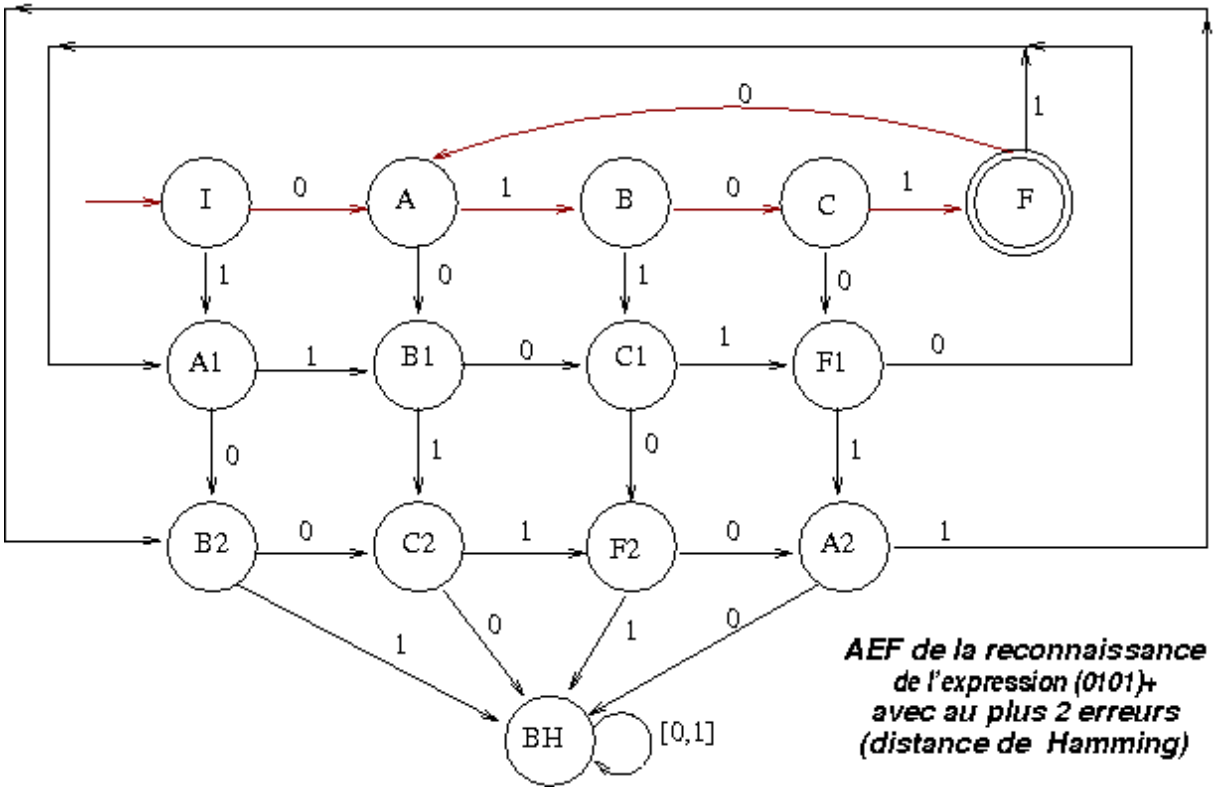
Exercices

- Trouver les langages associés aux diagrammes suivants :



AEF de correction d'erreurs : exemple 1

• AEF de reconnaissance d'une de série 0101 (notée par l'**expressin régulière** $(0101)^+$, voir plus loin) pouvant contenir jusqu'à deux erreurs (au sens distance de **Hamming** : par erreur, un 0 remplacé par un 1 ou inversement).



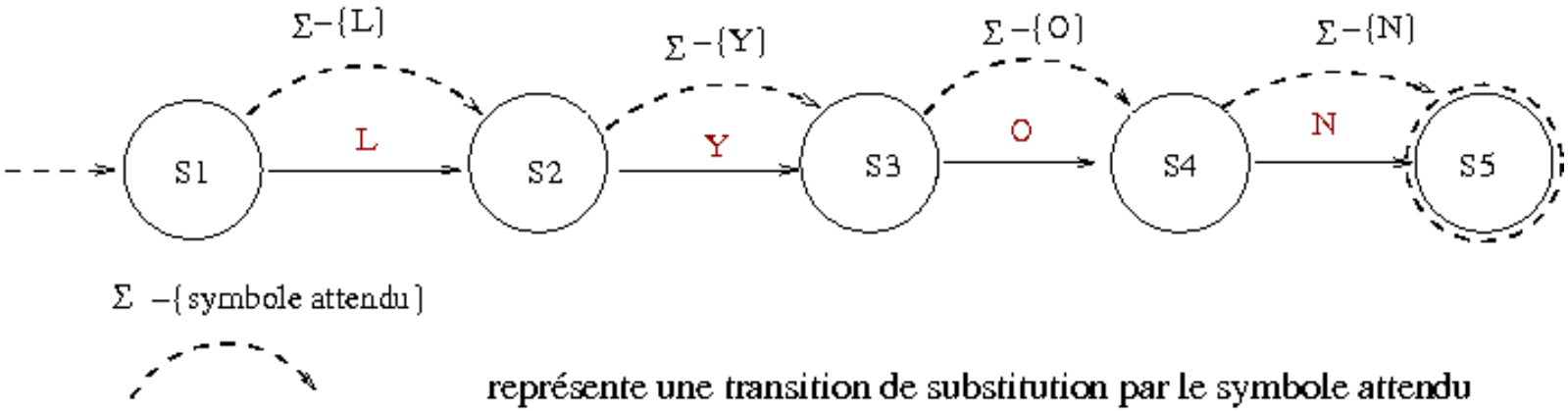
AEF de correction d'erreurs : Exemple 2

• AEF de reconnaissance de la chaîne "LYON" (Σ désigne tout l'alphabet).
• La correction peut nécessiter des transition ($\sim$ opérations) d'Insertion, de Substitution et de Suppression d'un caractère.

➥ Dans le cas présent, seule la substitution (du caractère lu en entrée par le caractère attendu) est utilisée.

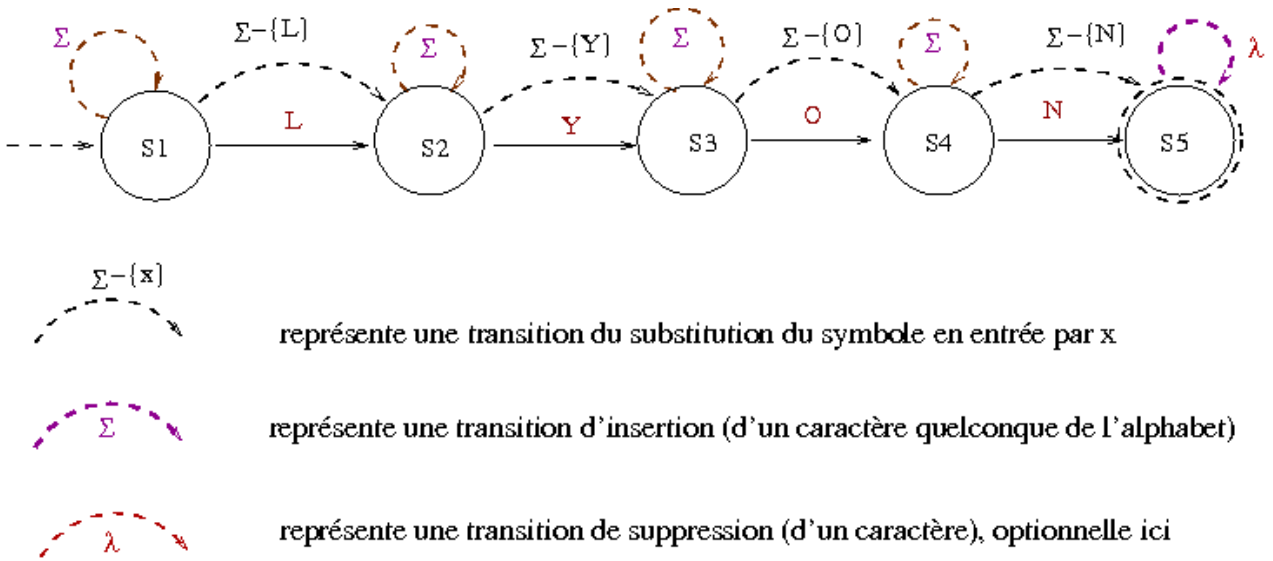
Elle est représentée par $\Sigma - \{x\}$ pour successivement $x \notin \{'L'\}$ puis $x \notin \{'Y'\}$...

➥ On remarque que la transition est confondue avec la substitution.



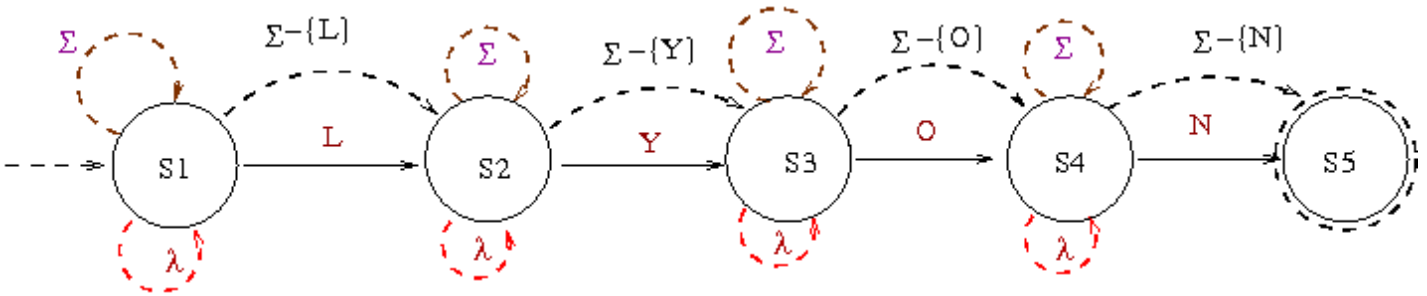
... Suite Exemple 2 : plus complet

- Pour un AEF correcteur, et dans un état S_i donné, on peut :
  ✓trouver en entrée le caractère *attendu* ➟ on avance... (axe principal) ;
  ✓trouver en entrée un caractère *inattendu* ➟ substitution (transition $\Sigma - \{..\}$) ;
  ✓n'avoir *aucun* caractère disponible ➟ on en insère un de Σ (transition Σ) ;
  ✓en avoir lorsqu'il n'en *faut pas* (ou entrée hors Σ) ➟ on en supprime un (transition λ).
  ➥ Dans ce cas présent, on peut utiliser λ sur S_5 (si terminaison exigée).



... Suite

- On complète l'AEF correcteur.



- Un traitement possible de la chaîne "2LïO5N" (Σ = alphabet français) :
  $\delta(S_1,' 2') = S_1$ (transition de suppression λ) en présence de $'2' \notin \Sigma$.
  $\delta(S_1,' L') = S_2$ (transition 'ordinaire')
  $\delta(S_2,' \ddot{i}') = S_3$ (transition de substitution $\Sigma - \{'\ddot{i}'\}$)
  $\delta(S_3,' O') = S_4$ (transition 'ordinaire')
  $\delta(S_4,' 5') = S_4$ (transition de suppression λ) en présence de $'5' \notin \Sigma$.
  $\delta(S_4,' N') = S_5$ (transition 'ordinaire')
  Puis fin de la chaîne.

... Suite : correcteur Stochastique

• Ici, la transition λ change d'état (et supprimer les symboles inattendus) au lieu de boucler comme dans la version précédente.

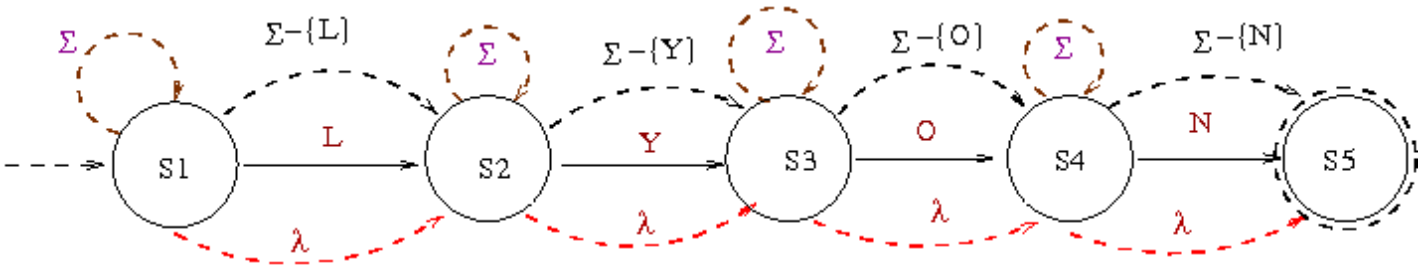


Fig.: L'AEF correcteur stochastique de "LYON"

• Ces automates sont proches de ceux appelés les AEF Stochastiques (pour l'inderminisme de leur comportement dans leur version où l'ordre des transitions deviendrait aléatoire : on substitue/supprime/insère quand on veut !).

➥ Les AEFs correcteurs peuvent compter le nombre de substitutions, insertions et suppressions (distance **Levenshtein** avec *sub*,*del*,*ins*)

AEF non déterminste

- Les AEFs étudiés jusqu'ici sont dits **déterministes**.
  ➙ toutes les transitions $\delta(S_i, x_i)$ sont deux à deux distinctes.

- Une autre classe importante d'AEFs est la classe des AEFs **non déterministes** :
  ➙ On peut avoir plusieurs états d'arrivée différents pour un même couple $\delta(S_i, x_i)$.
  ➙ Par exemple : $\delta(S_2, 0) = S_3$ et $\delta(S_2, 0) = S_4, S_3 \neq S_4$.
  ➙ On ne peut décider quelle transition appliquer à l'entrée 0.

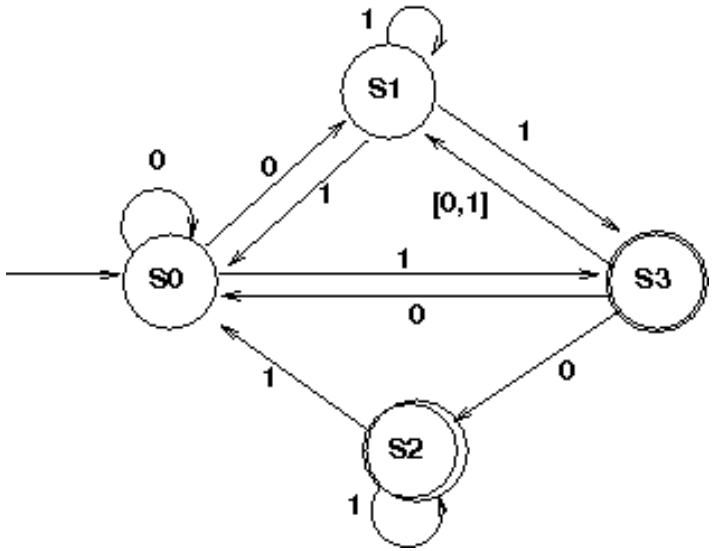
Définition

Un AEF non déterministe M est défini par $M = (S_0, \Sigma, \delta, S, F)$ où la fonction de transition δ attribue un ensemble d'états à une paire (état $\times$ entrée) :
➙ $\delta : S \times \Sigma \rightarrow P(S)$ *avec $P(S)$: parties de S.*

AEF non déterminste (suite)

- Exemple 1 :

Etats	**f**	
	Entrées	
	0	1
S_0	S_0, S_1	S_3
S_1	S_0	S_1, S_3
S_2		S_0, S_2
S_3	S_0, S_1, S_2	S_1

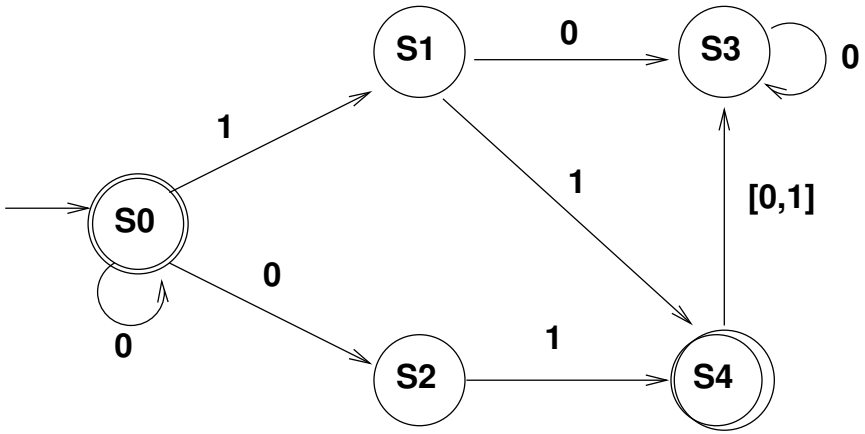


AEF non déterministe : $\delta(S_1, 1) = \{S_1, S_3\}$

AEF non déterminste (suite)

- Exemple 2 :

Etats	f	
	Entrées	
	0	1
S_0	S_0, S_2	S_1
S_1	S_3	S_4
S_2		S_4
S_3	S_3	
S_4	S_3	S_3

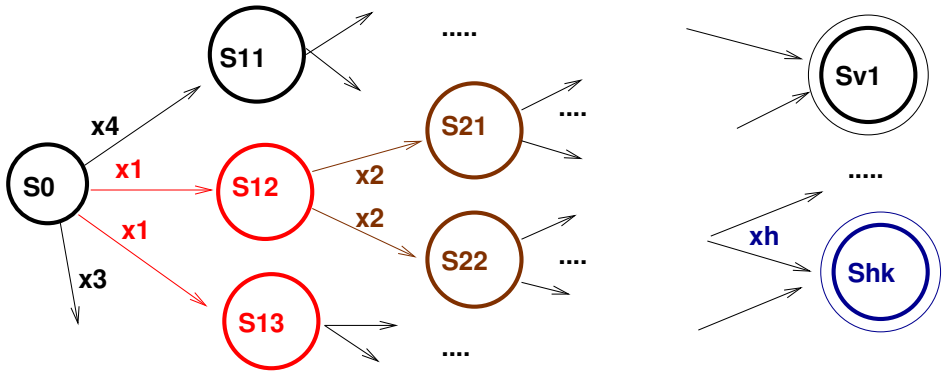


- On constate que ces automates décrivent des transitions qui peuvent sembler inutiles (voir plus loin).

Langages d'un AEF (non déterminsite)

- **Que veut dire** d'accepter un langage par un AEF non déterminsite ?
  - ➙ Soit $x = x_1\ x_2, \ldots x_k$ une chaîne en entrée.
  - ➙ Supposons $\delta(s_0, x_1) = \{s_{11}, \ldots s_{1l}\}$
  - ➙ Puis pour chacun des états dans $\{s_{11}, \ldots s_{1l}\}$,
    $$\delta(s_{1i}, x_2) = \{s_{21}, \ldots s_{2m}\} \ldots \delta(s_{ji}, x_h) = \{s_{h1}, \ldots s_{hk}\}$$
  - ➙ Soit E l'union de tous ces états.
  - ➙ On dira que x a été **accepté** (reconnu) par cet AEF si l'un des états finaux figure dans E tel qu'en partant de s_0 on aboutisse à cet état final.

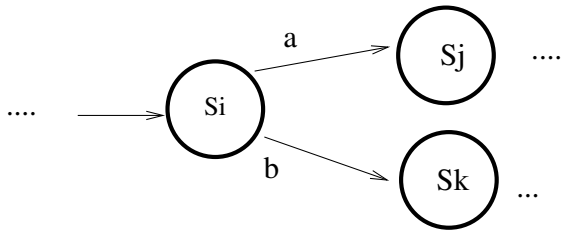
Le **langage** d'un AEF est l'ensemble de toutes les chaînes $x \in \Sigma^*$ acceptés par cet AEF.
N.B. : le langage d'un AEF commence forcément par un état initial (souvent noté S_0)



N.B. : Le langage d'un AEF déterministe se définit de la même manière (plus simple car $\delta(s_0, x_1)$ est un singleton)

Langages d'un AEF (non déterminsite) (suite)

• D'une manière plus générale, on peut définir le langage d'un <u>état</u> dans un AEF (déterministe ou non) :



➥ Soit les transitions d'un AEF :

$$\delta(S_i, a) = S_j, \quad \delta(S_i, b) = S_k, \ ...$$

Le langage de l'état S_i noté $\mathbf{L(S_i)}$ est : $L(S_i) = a.L(S_j) \cup b.L(S_k)$

• Si l'état S_i est (aussi) un état final, on ajoutera (aussi) la chaîne vide ε à $L(S_i)$.
• S'il existe une transition récursive $\delta(S_i, c) = S_i$, $L(S_i)$ contiendra également $c.S_i$.

Langages d'un AEF (non déterminsite) (suite)

• Pour définir l'ensemble des chaînes acceptées par un état S_i tel que $L(S_i) = a.L(S_j) \cup b.L(S_k)$, on définit à leur tour les langages $L(S_j)$ et $L(S_k)$....

• Lorsque ce processus arrive à son terme, $L(S_i)$ ne contiendra uniquement des symboles de Σ (tels que a, b, c, ...)..

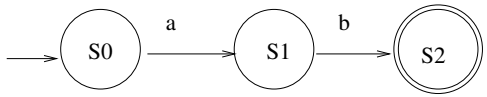
N.B. : un AEF peut contenir des états $S_{impasse}$ dits *inutiles* (état dont les transitions n'aboutiront pas à un état final). Clairement, il ne sera pas possible de définir le langage de tels états car ils n'aboutiront pas à un état final.

• **Le langage d'un AEF** est le langage de son état initial contenant uniquement des chaînes acceptées par S_0.

Calcul du langage d'un AEF

Calcul du langage d'un AEF (déterministe ou non) :

- **Exemple - 1** : Soit l'AEF



AEFD 1

- Le langage d'un AEF est l'ensemble des **mots** (chaînes de symboles $\in \Sigma^*$) acceptés par cet AEF.

  Le langage d'un AEF est calculé à partir des langages de ses états accessibles.

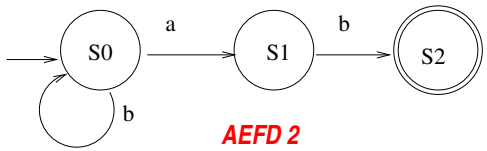
- Pour l'AEFD1, on note :
  - ➥ $L(S_0) = a.L(S_1)$ le langage de l'état $S_0 =$ un a suivi du langage de S_1.
  - ➥ $L(S_1) = b.L(S_2)$ idem
  - ➥ $L(S_2) = \varepsilon$ le langage de l'état $S_2 =$ est vide.

- Donc, $L(S_0) = a.L(S_1) = a.b.L(S_2) = a.b.\varepsilon = ab$

Le langage d'un AEF est le langage se son état initial (S_0).

Calcul du langage d'un AEF (suite)

•**Exemple - 2** : Soit l'AEF



- Pour l'AEFD2, on note :
  ➥ $L(S_0) = b.L(S_0) \cup a.L(S_1)$ introduction de la récursivité
  On constate que l'expression du langage de S_0 suit le même principe de définition, même si un b mème encore à S_0.
- Comment interpréter $L(S_0) = b.L(S_0) \cup ...$?
  Pour généraliser, on peut avoir la forme $L(X) = \alpha.L(X) \cup \beta$
  Un développement possible de l'état $X \rightarrow \alpha.X \cup \beta$ aura la forme :
  $$X \rightarrow \alpha.X \rightarrow \alpha.\alpha X \rightarrow \alpha.\alpha.\alpha X \rightarrow ... \rightarrow \alpha...\alpha.X \rightarrow \alpha...\alpha.\beta$$
  ➥ c-à-d., on aura fini par épuiser les α et on a accepté un β

- Rappel : $X \rightarrow^* \alpha...\alpha.\beta$ est appelé la **fermeture** de X.

Calcul du langage d'un AEF (suite)

• Après avoir accepté un certain nombre (peut être aucun) α, on accepte un β (puis on enchaîne éventuellement sur un autre état). β peut se décliner sous n'importe quelle forme, y compris contenant d'autres $\gamma.X$

La forme $L(X) = \alpha.L(X) \cup \beta$ se traduit par $L(X) = \alpha^*\beta$.

La chaîne acceptée par X dans $L(X) = \alpha.L(X) \cup \beta$ est $\alpha^*\beta$.

• N.B. : on dira que $X = \alpha^*\beta$ est la solution à l'équation linguistique $X = \alpha.X \cup \beta$

• On a donc $L(S_0) = b^*.a.L(S_1)$
• Rappel (de l'AEFD1) :

$$L(S_1) = b.L(S_2)$$
$$L(S_2) = \varepsilon$$

• Donc, $L(S_0) = b^*.a.L(S_1) = b^*.a.b.L(S_2) = b^*.a.b.\varepsilon = b^*.a.b$

Calcul du langage d'un AEF (suite)

• Un exemple analogique pour la règle récursive linguistique précédente :

supposons prendre des billes dans un sac contenant des billes noires ou blanches selon la règle suivante :

on prend une bille ; on arrête si elle est noire, ou (si blanche) on en prend une autre ... ainsi de suite.

➥ A la fin de ces opérations, on aura de 0 à n billes blanches et une noire.

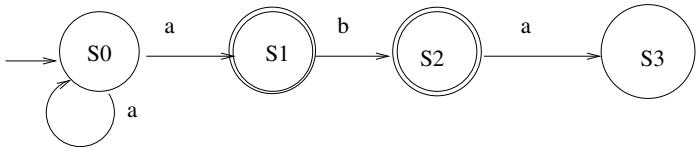
Si on veut conserver l'ordre de tirage des billes, on aura n billes blanches puis une noire.

On peut résumer cet exemple :

prendre une bille = (une bille blanche ET prendre encore une bille)
OU prendre une bille noire (puis rien)
$=$ *prendre* $n \geq 0$ *billes blanches et une noire*
$= blanche^* \ . \ noire.$ (respect de l'ordre du tirage)

Calcul du langage d'un AEF (suite)

- **Exemple - 3** : Soit l'AEFND



AEFND 3

- Par rapport à l'AEFD2, l'AEFND3 (non déterministe) contient un nouvel état S_3. De plus, l'état S_1 est maintenant un état final. On a :

  $L(S_0) = a.L(S_0) \cup a.L(S_1) = a^*.a.L(S_1) = a^+.L(S_1)$ $\quad a^*.a = a^+$

  $L(S_1) = b.L(S_2) \cup \varepsilon$ $\quad S_1$ peut maintenant générer le mot vide.

  $L(S_2) = \varepsilon$ $\quad$ il est inutile de s'intéresser à S_3 car c'est une impasse.

- On peut noter $L(S_2) = \varepsilon \cup a.L(S_3)$ mais on se rappelera que le langage d'un AEF est de la forme : $L(AEF) = \{\omega |\ S_{init} \rightarrow^* \ \omega = S_{init} \rightarrow ... \rightarrow S_{final}\}$

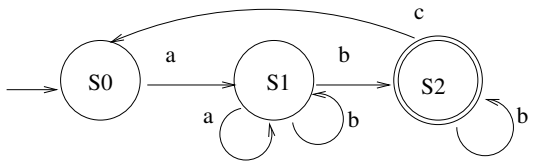
  ➥ S_3 ne mène pas à un état final, il sera (plus tard) éliminé des calculs.

- $L(S_0) = a^+.L(S_1) = a^+.(b.L(S_2) \cup \varepsilon) = a^+.(b.\varepsilon \cup \varepsilon) = a^+.b \cup a^+$

- L'AEFD correspondant :

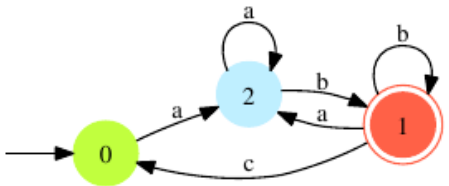
Calcul du langage d'un AEF (suite)

- **Exemple - 4** : Soit l'AEFND



AEFND 4

$$
\begin{aligned}
L(S_2) &= b.L(S_2) \cup \varepsilon \cup c.L(S_0) = b^*.(c.L(S_0) \cup \varepsilon) = b^* c.L(S_0) \cup b^* \\
L(S_1) &= a.L(S_1) \cup b.L(S_1) \cup b.L(S_2) = (a \cup b).L(S_1) \cup b.L(S_2) \\
&= (a \cup b)^*.b.L(S_2) = (a \cup b)^*.b.(b^* c.L(S_0) \cup b^*) \\
&= (a \cup b)^*.b^+ c.L(S_0) \cup (a \cup b)^*.b^+ \\
L(S_0) &= a.L(S_1) = a.(a \cup b)^*.b^+ c.L(S_0) \cup a.(a \cup b)^*.b^+ \\
&= (a.(a \cup b)^*.b^+ c)^*.a.(a \cup b)^*.b^+
\end{aligned}
$$



Calcul du langage d'un AEF (suite)

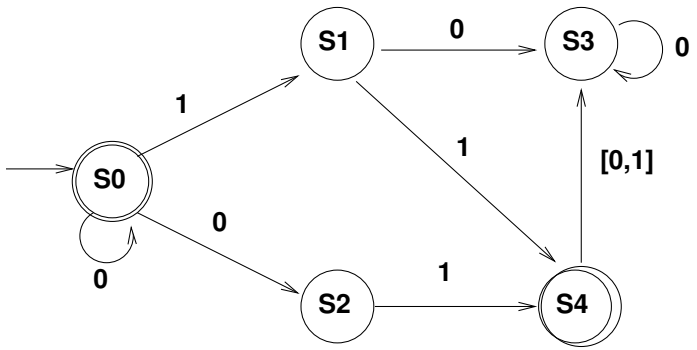
Remarque sur le calcul du langage :
Dans une règle doublement récursive de la forme $L(X) = a.L(X) \cup b.L(X) \cup c$, on peut procéder de 3 manières pour avoir une forme générale $L(X) = \alpha.L(X) \cup \beta$ (qui permettra d'obtenir $L(X) = \alpha^*\beta$) :

1. Mettre en facteur $L(X)$ comme à la page précédente :
   $L(X) = (a \cup b).L(X) \cup c \rightarrow \boxed{L(X) = (a \cup b)^* c}$
2. traiter le premier $L(X)$ à droite :
   $L(X) = a.L(X) \cup [b.L(X) \cup c] = a^*.[b.L(X) \cup c] = a^*.b.L(X) \cup a^*.c$
   $\rightarrow \boxed{L(X) = (a^*.b)^*.a^*.c}$
3. traiter le deuxième $L(X)$ à droite :
   $L(X) = b.L(X) \cup [a.L(X) \cup c] = b^*.[a.L(X) \cup c] = b^*.a.L(X) \cup b^*.c$
   $\rightarrow \boxed{L(X) = (b^*.a)^*.b^*.c}$

Les 3 formes sont strictement équivalentes.
Elles peuvent être utilisées dans d'autres calculs de langages.

Exemples de calculs de langage

- **Exemple** : quel est le langage accepté par l'automate suivant ?



- s_0 est un état final et il y a une transition vers ce même s_0 avec l'entrée 0,
  ➙ Cet AEF accepte 0^* (toute chaîne de 0 ou plus '0').
- De plus, avec l'état final s_4, toute chaîne qui aura s_4 comme état d'arrivée (en partant de s_0) est acceptée.
  ➙ Ces chaînes sont constituées de 0^* suivi de 01 ou 11.

➔ Le langage reconnu par cet AEF est $\{0^*, 0^*01, 0^*11\}$.

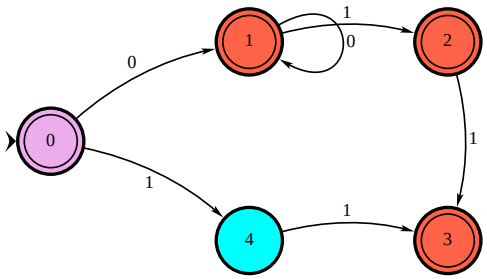
➥**Pourquoi ? méthode plus formelle ?**

Exemples de calculs de langage (suite)

A partir du calcul précédent, proposer un AEFD. :

- Les calculs ont donné L $= \{0^*, 0^*01, 0^*11\}$:
  - $0^* = \varepsilon; OO^*$
  - $0^*01 = O^+1 = (\varepsilon; O)0^*1$
  - $0^*11 = (\varepsilon; O^+)11$

- L'AEFD sera :

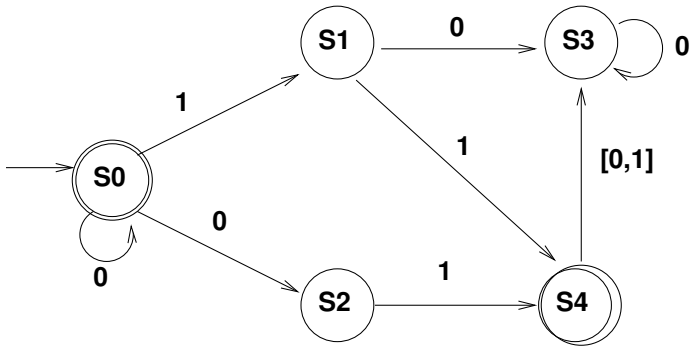


- Dans les cas simples,

le calcul du langage d'un AEFND permet de produire un AEFD

Exemples de calculs de langage (suite)

- Calcul formelle du langage à partir de l'AEF (AEFND ou AEFD) :

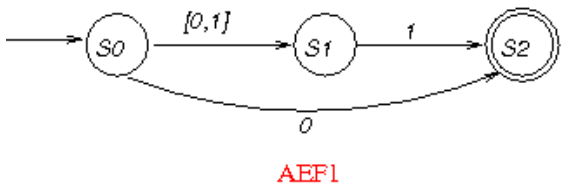


- $L(S_4) = \varepsilon$
- $L(S_1) = 1.L(S_4) = 1$
- $L(S_2) = 1.L(S_4) = 1$
- $L(S_0) = \varepsilon \cup O.L(S_0) \cup 1.L(S_1) \cup 0.L(S_2)$ un état final non terminal a ε de plus.
  ➥ La forme générale $X = \alpha.X \cup \beta$ s'applique ➥ $X = \alpha^*.\beta$:
- $L(S_0) = O.L(S_0) \cup (\varepsilon \cup 11 \cup 01) = 0^*.(\varepsilon \cup 11 \cup 01)$

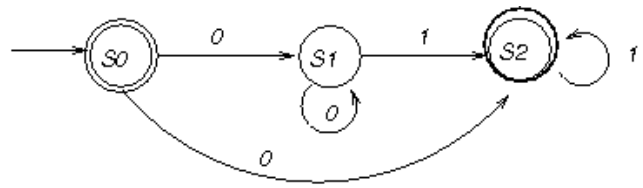
$$\boxed{L(S_0) = 0^* \cup 0^*11 \cup 0^*01}$$

Exercices de calcul de langage

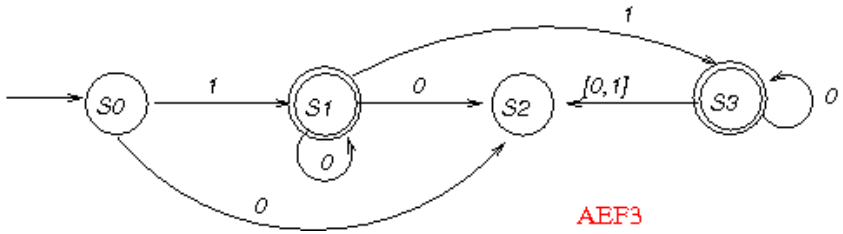
- Trouver les langages associés aux AEFs non déterministes suivants :



AEF1



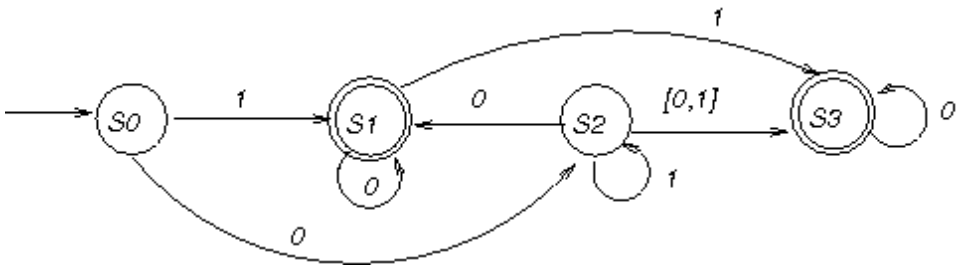
AEF2



AEF3

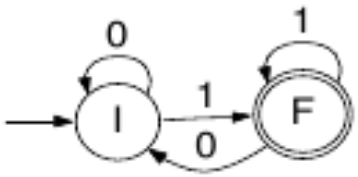
Exercice de calcul de langage (suite)

- Calculer le langage de l'AEF non déterministe suivant et en donner AEFD :



Calcul du langage : Exercice

- Quel est le langage de l'AEF suivant :



AEF (dans tous ses états !) : suite exercices

Trouver un AEF déterministe pour chacun des ensembles suivants
➔ N.B. : prévoir tous les cas de figure pour $\Sigma = \{0, 1\}$.

1. $\{0\}$
2. $\{1, 00\}$
3. $\{1^n | n = 2, 3, 4, ...\}$

• But : se habituer aux automates qui s'occupent de tous les cas de figures !

AEF (suite exercices)

Solution Ex-2 : L$= \{1, 00\}$

AEF (suite exercices)

Solution Ex-3 : L$= \{1^n | n = 2, 3, 4, ...\}$

Prévision de tous les états : exercice

- Donner un AEF de reconnaissance du langage L$= \{1^{2n} | n \geq 0\}$
  Puis compléter la solution en tenant compte de l'alphabet $\Sigma = \{0, 1\}$.

Pouvoir d'expression des AEFs

Lemme

Il n'y a pas d'AEF capable de vérifier s'il y a le même nombre ($N > 0$) de 0 et de 1 sur un flot d'entrée.

Démonstration.

Supposons qu'un AEF **M** capable de reconnaître la chaîne $0^n\ 1^n, n > 0$ possède n états.
Lorsque M lit et reconnaît la chaîne $0^{n+1}\ 1^{n+1}$, on peut dire qu'il passe au moins deux fois par un des états (s) pour reconnaitre les 0s.
➔ Donc, s est un état sur lequel on revient au moins 2 fois (boucle éventle.).
➔ Dans ce cas, M sera toujours dans l'état s lors de la reconnaissance de $0^{n+1+\mathbf{k}}\ 1^{n+1}, k > 0$ (le même que pour $0^{n+1}\ 1^{n+1}$).
➔ M accepterait à la fois $0^{n+1}\ 1^{n+1}$ et $0^{n+1+k}\ 1^{n+1}$, ce qui est une contradiction (puisqu'il reconnaît autant de 0 que de 1). ∎

• N.B. : On peut étendre les AEFs les rendant capables de telles capacités. Voir plus loin l'ajout des actions sémantiques.

AEF non déterminsite (suite)

Théorème

Tout langage reconnu par un AEF non déterministe $M_0 = (S, s_0, F, \delta, \Sigma)$ *peut être reconnu par un AEF déterministe équivalent* M_1 .

Preuve par construction de M_1 (determinsite) à partir de M_0.

- Tout état de M_1 est constitué d'un ensemble d'états S de M_0.
- L'état initial de $M_1 = \{s_0\}$ l'ensemble contenant l'état initial de M_0.
- Le vocabulaire Σ est le même pour les deux.
- Etant donné un état $\{s_{i1}, s_{i2} \dots s_{ik}\}$ de M_1, le symbole d'entrée α conduit à un ensemble d'états d'arrivée tel que :

$$\{s_{i1}, s_{i2} \dots s_{ik}\} \times \alpha \longrightarrow \{\delta(s_{i1}), \delta(s_{i2}) \dots \delta(s_{ik})\}$$

➙ Les états de M_1 sont tous des sous ensembles de S (états de M_0), obtenus de cette manière, en partant de s_0 (il y a 2^n sous ensembles possibles pour M_1, vide compris, S de taille n).

➙ Les états finaux de M_1 = ceux qui contiennent un état final de M_0. ∎

AEF non déterminsite (suite) (suite)

- On pourra prouver (par induction par exemple) :

  ➥ toute chaîne reconnue par M_0 en partant de s_0 et aboutissant à un état final de M_0 sera aussi reconnue par M_1.

  ➥ De même, un échec dans M_0 conduit à un échec dans M_1.

Transformation des AEFND en AEFD

Transformation des AEFs Non Déterministe. en Déterministe

• Constat : on peut plus facilement proposer un AEF non déterministe (AEFND), puis le transformer (par un algorithme) en un AEF déterministe (AEFD).

➙ N.B. : c'est le même principe que pour les algorithmes récursifs vs. itératifs.

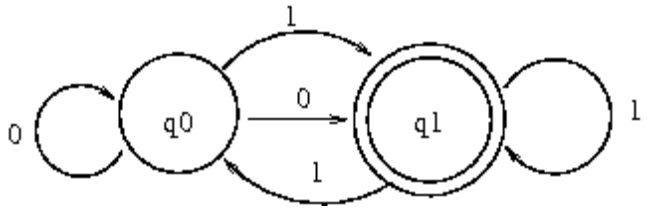
➔ Dans le cas des AEFs, cette transformation est toujours possible car pour un AEFND, il existe un AEFD qui reconnaît le même langage.

• On a vu un exemple de transformation, ci-dessous, une approche plus formelle.

AEFND vers AEFD

AEFND vers AEFD : une approche systématique

- Soit l'AEFND $= (S = \{q_0, q_1\}, \Sigma = \{0, 1\}, \delta, q_0, F = \{q_1\})$ avec
  $\delta(q_0, 0) = \{q_0, q_1\}, \quad \delta(q_0, 1) = q_1, \quad \delta(q_1, 1) = \{q_0, q_1\}$



  ➙ Rappel : on note $\delta(q_1, 0) = \oslash$ pour une transition inexistante.

- L'AEFD correspondant $= (Q, \{0, 1\}, \delta', [q_0], F')$ avec :
  ➙ Q contenant tous les (2^n) sous ensembles de $\{q_0, q_1\}$, c-à-d. :
  ➙ $\{q_0\}, \{q_1\}, \{q_0, q_1\}$ et $\oslash$
  ➙ On considère chacun de ces 4 sous ensembles avec $\Sigma = \{0, 1\}$.

N.B. : on notera les états possibles de l'AEFD entre **[]** pour les distinguer de ceux de l'AEFND.

AEFND vers AEFD (suite)

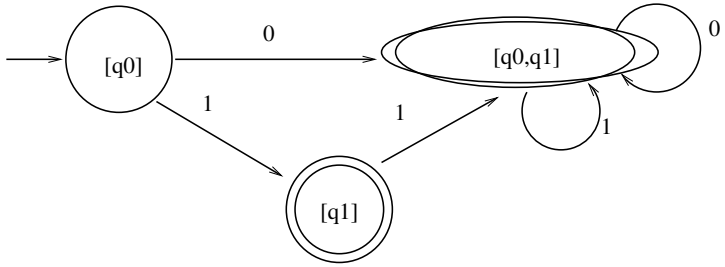
- On a $\delta(q_0, 0) = \{q_0, q_1\}$ $\Rightarrow \delta'([q_0], 0) = [q_0, q_1]$
  - ➙ De même, $\Rightarrow \delta'([q_0], 1) = [q_1]$
  - ➙ $\Rightarrow \delta'([q_1], 0) = \oslash$
  - ➙ $\Rightarrow \delta'([q_1], 1) = [q_0, q_1]$
  - ➙ $\Rightarrow \delta'([q_0, q_1], 0) = [q_0, q_1]$ car

    $\delta'([q_0, q_1], 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \oslash = [q_0, q_1]$
  - ➙ Et $\Rightarrow \delta'([q_0, q_1], 1) = [q_0, q_1]$ car

    $\delta'([q_0, q_1], 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_1\} \cup \{q_0, q_1\} = [q_0, q_1]$

- Naturelllement,

  $\Rightarrow \delta'(\oslash, 0) = \delta'(\oslash, 1) = \oslash$
- L'ensemble F' des états finaux constitué des états de l'AEFD contenant les états finaux de l'AEFND (ici $\{q_1\}$) $= F' = \{[q_1], [q_0, q_1]\}$.



AEFND vers AEFD (suite)

• Le *modus operandi* ci-dessus est systématique.

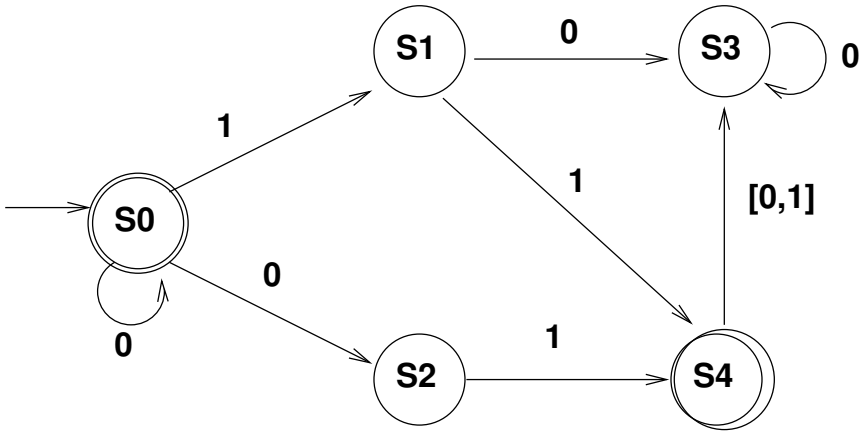
➙ Dans la pratique, on n'envisage pas tous les 2^n sous ensembles de S (états de l'AEFND) qui peuvent représenter un grand nombre sans que tous ne soient accessibles depuis $\{q_0\}$.

➥ Mais plutôt, **on commence par $[q_0]$ et l'on ajoute des états petit à petit.**

AEFND vers AEFD : un autre exemple

- **Exemple 2** : construire la version déterministe de l'AEFND suivant :

Etats	**f**	
	Entrées	
	0	1
S_0	S_0, S_2	S_1
S_1	S_3	S_4
S_2		S_4
S_3	S_3	
S_4	S_3	S_3

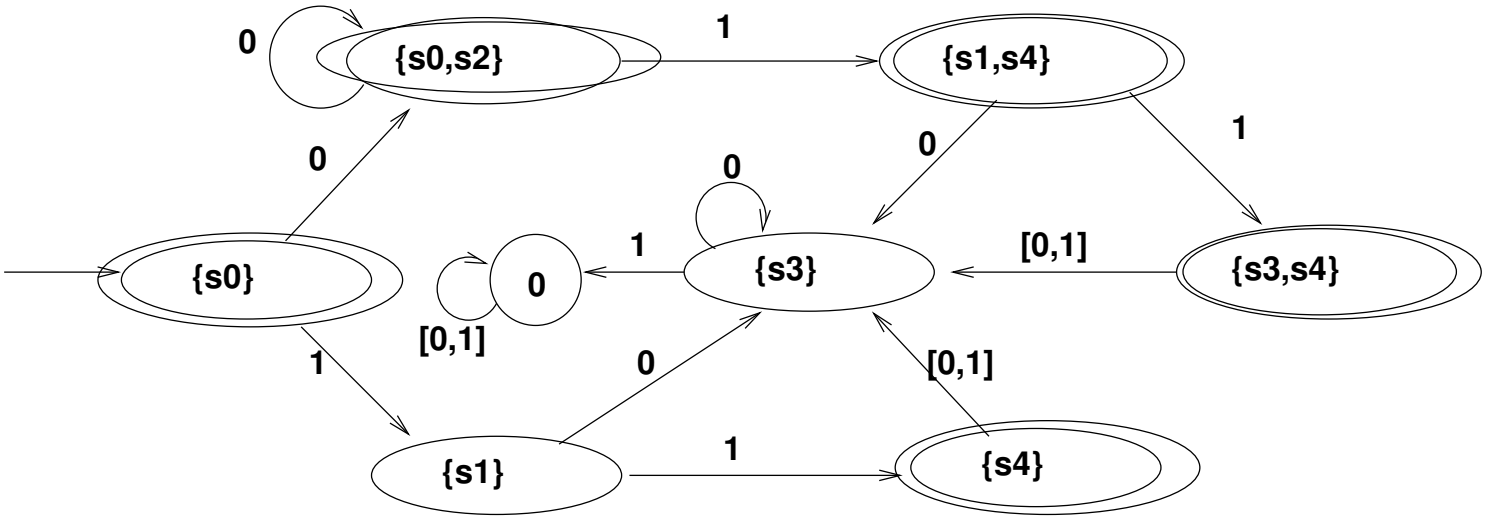


Solution

Rappels :

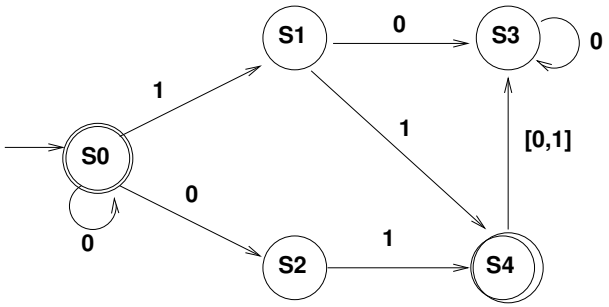
- Les états de M_1 sont des sous ensembles des états de M_0.
- Soit E_1 un état de M_1 de la forme $\{e_1, e_2, .., e_k\}$, e_i état de M_0.

Supposons que dans M_0, chaque état $e_i \in E_1$ participe à une transition $e_i \times \beta = e'_i$. L'état d'arrivée pour la transition $E_1 \times \beta$ de M_1 contiendra tous ces e'_i. (voir exemple).



- Dans la méthode suivante (plus efficace), on ne considère que les états atteignables depuis $[s_0]$ (au lieu des 2^n sous ensembles) ../..

Solution (suite)



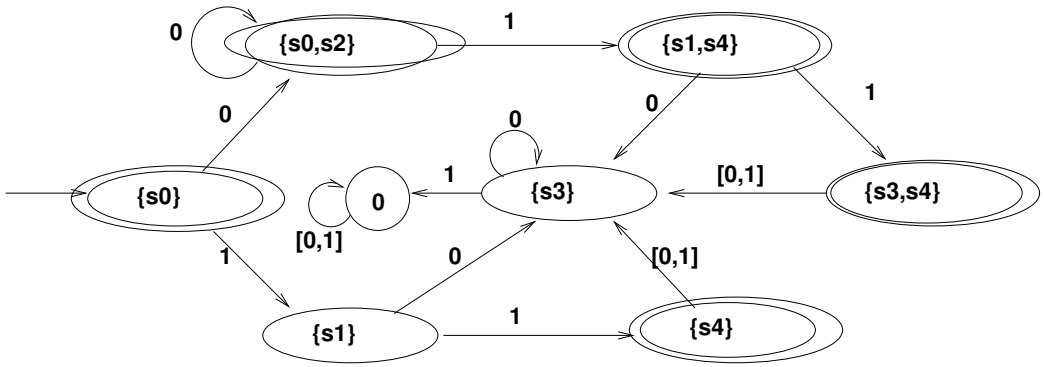
Dans M_0 (AEFND)	$\Rightarrow$ **Dans** M_1 (AEFD)
s_0 état init M_0	$[s_0]$ état init M_1
• Considérons $\{s_0\}$: $s_0 \times 0 = s_0$ $s_0 \times 0 = s_2$	 $[s_0] \times 0 = [s_0, s_2]$
$s_0 \times 1 = s_1$	$[s_0] \times 1 = [s_1]$
• Considérons les états de $\{s_0, s_2\}$: $s_0 \times 0 = \{s_0, s_2\}$ (ci-dessus) $s_2 \times 0$ pas dans M_0 ➡ $\{s_0, s_2\} \cup \oslash = \{s_0, s_2\}$	 $[s_0, s_2] \times 0 = [s_0, s_2]$
$s_0 \times 1 = s_1$ $s_2 \times 1 = s_4$	 $[s_0, s_2] \times 1 = [s_1, s_4]$

Solution (suite)

Dans M_0 (non déter)	$\Rightarrow$ **Dans** M_1 (déter)
• Considérons $\{s_1\}$: $s_1 \times 0 = s_3$	$[s_1] \times 0 = [s_3]$
$s_1 \times 1 = s_4$ ➡ $\{s_1\}$ et $\{s_1, s_4\}$ indépendants : ce sont 2 états différents de M_1	$[s_1] \times 1 = [s_4]$
• Considérons $\{s_1, s_4\}$: $s_1 \times 0 = s_3$ $s_4 \times 0 = s_3$	$[s_1, s_4] \times 0 = [s_3]$
$s_1 \times 1 = s_4$ $s_4 \times 1 = s_3$	$[s_1, s_4] \times 1 = [s_3, s_4]$
• Considérons $\{s_3\}$: $s_3 \times 0 = s_3$	$[s_3] \times 0 = [s_3]$
$s_3 \times 1$ pas dans M_0	$[s_3] \times 1 = [\lambda]$
• Considérons $\{s_4\}$: $s_4 \times 0 = s_3$	$[s_4] \times 0 = [s_3]$
$s_4 \times 1 = s_3$	$[s_4] \times 1 = [s_3]$

Solution (suite)

Dans M_0 (non déter)	$\Rightarrow$ **Dans** M_1 (déter)
• Considérons $\{s_3, s_4\}$: $s_3 \times 0 = s_3$ $s_4 \times 0 = s_3$	$[s_3, s_4] \times 0 = [s_3]$
$s_3\times$ pas dans M_0 Mais $s_4 \times 1 = s_3$	$[s_3, s_4] \times 1 = [s_3]$
• Enfin, considérons λ : ➙ Aucune transition dans M_0 On prévoit une transition générale	$[\lambda] \times \{0, 1\} = [\lambda]$ Pour compléter $[s_3] \times 1$



Solution (suite)

• N.B. : l'ensemble *vide* est un des états de M_1, ce sous ensemble contient tous les états d'arrivée en partant de $\{s_3\}$ avec l'entrée $= 1$.

➟ L'état 0 est pris en compte car l'AEFND d'origine traite les deux symboles d'entrée (0 et 1) pour tous les états de l'automate.

➟ L'absence de transition est notée, hormis par $\{\lambda\}$, également par $\delta(s_i, \alpha) = \oslash$.

• Dans $M1$, l'état initial $= [s_0]$,

➟ L'ensemble des états finaux $=$ tous les états de M_1 où figurent s_0 et s_4 (états finaux de M_0).

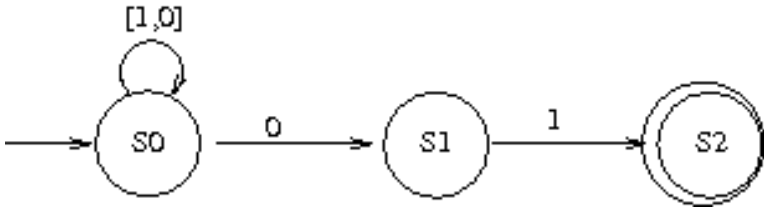
Remarque (générale)

La présence de $\varepsilon - état$ dans un AEF le transforme en non déterministe (voir plus loin).

Un autre exemple de l'approche systématique

Exemple 3 de transformation d'AEFND en AEFD :

• On reprend l'automate de la machine (vue plus haut) qui reconnaît si le flot d'entrée contient un '1' en dernière position et un '0' en avant dernière. Appelons celui-ci **AEFND**.



Pour cet AEFND $= (\{S_0, S_1, S_2\}, \{0, 1\}, \delta, S_0, \{S_2\})$ on a :

$\delta(S_0, 0) = S_0$ (que l'on note aussi $S_0 \times 0 = S_0$)

$\delta(S_0, 0) = S_1$

$\delta(S_0, 1) = S_0$

$\delta(S_1, 1) = S_2$

• On cherche à construire un AEFD (déterministe) :

$$\text{AEFD} = (Q, \{0, 1\}, \delta', [S_0], F).$$

Un autre exemple de l'approche systématique (suite)

➙ Dans l'approche systématique, on pose l'ensemble des parties de $\{S_0, S_1, S_2\}$.
Ce sont les éléments possibles de Q :

➙ $[S_0], [S_1], [S_2], [S_0, S_1], [S_0, S_2], [S_1, S_2], [S_0, S_1, S_2]$

- On considère chaque ensemble avec les entrées $\{0, 1\}$:
  - $[S_0] \times 0 = S_0 \times 0$ (**dans ND**) $= [S_0, S_1]$
  - $[S_0] \times 1 = S_0 \times 1$ (**dans ND**) $= [S_0]$
  - $[S_1] \times 0 = \oslash$ (Aucune transition dans ND)
  - $[S_1] \times 1 = S_1 \times 1 \quad = [S_2]$
  - $[S_2] \times 0 = \oslash$
  - $[S_2] \times 1 = \oslash$
  - $[S_0, S_1] \times 0 = (S_0 \times 0) \cup (S_1 \times 0) = \{S_0, S_1\} \cup \oslash = [S_0, S_1]$
  - $[S_0, S_1] \times 1 = (S_0 \times 1) \cup (S_1 \times 1) = \{S_0\} \cup \{S_2\} = [S_0, S_2]$
  - $[S_0, S_2] \times 0 = (S_0 \times 0) \cup (S_2 \times 0) = \{S_0, S_1\} \cup \oslash = [S_0, S_1]$
  - $[S_0, S_2] \times 1 = (S_0 \times 1) \cup (S_2 \times 1) = \{S_0\} \cup \oslash = [S_0]$

Un autre exemple de l'approche systématique (suite)

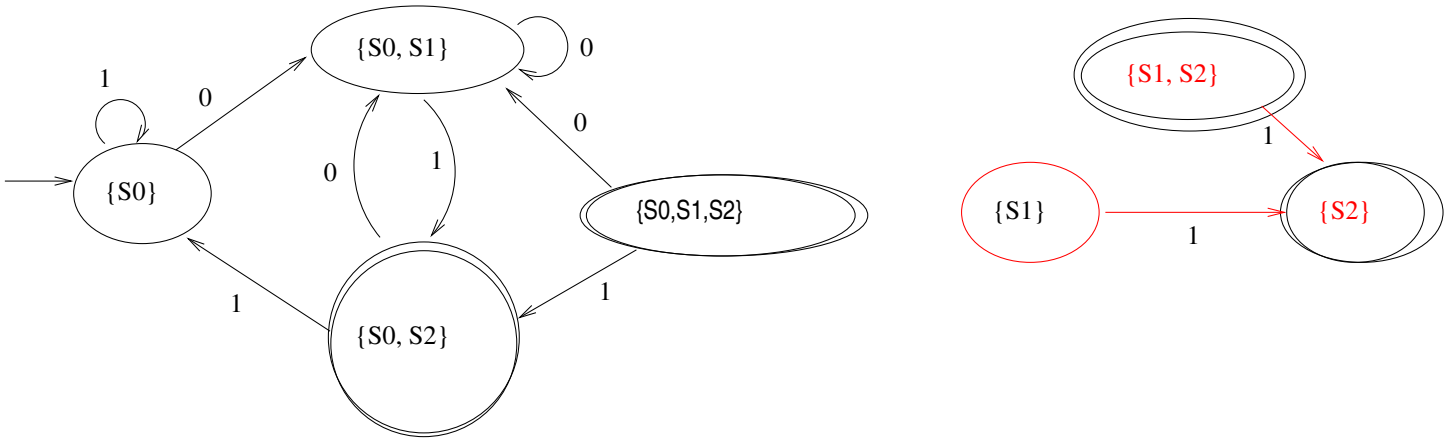
- $[S_1, S_2] \times 0 = \oslash$ (pas de transition dans ND)
- $[S_1, S_2] \times 1 = (S_1 \times 1) \cup (S_2 \times 1) = \{S_2\} \cup \oslash = [S_2]$
- $[S_0, S_1, S_2] \times 0 = (S_0 \times 0) \cup (S_1 \times 0) \cup (S_2 \times 0)$
  $= \{S_0, S_1\} \cup \oslash \cup \oslash = [S_0, S_1]$
- $[S_0, S_1, S_2] \times 1 = (S_0 \times 1) \cup (S_1 \times 1) \cup (S_2 \times 1)$
  $= \{S_0\} \cup \{S_2\} \cup \oslash = [S_0, S_2]$

➙ Les états finaux de l'AEFD = tous les ensembles contenant les états finaux de AEFND (i.e. S_2) $= [S_0, S_2], [S_2], [S_1, S_2], [S_0, S_1, S_2]$

• On crée l'automate *AEFD* en utilisation toutes les transitions qui conduisent à un état(donc $\neq \oslash$) :

➙ Résultat : l'AEFD (brute) suivant (décliné en 2 partie non connexe) :

Un autre exemple de l'approche systématique (suite)

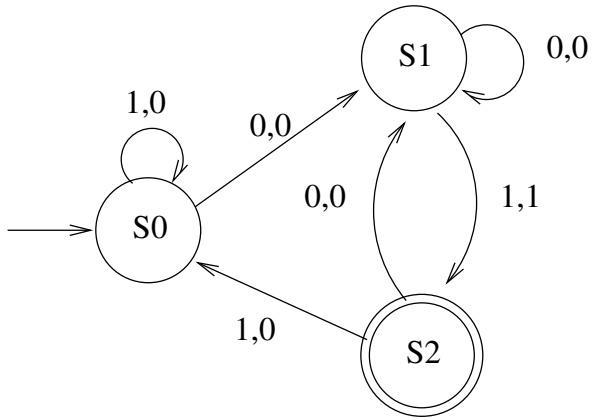


➙ On élimine de cet AEF tous les états que l'on *ne peut pas atteindre* depuis $[S_0]$.

➙ La partie droite disparaît (car non accéssible depuis $[S_0]$).

➙ Ce qui donne l'AEF déterministe suivant (on a renommé les ensembles d'états), abstraction faite des sorties :

Un autre exemple de l'approche systématique (suite)



• Rappel : on peut éviter le développement de tous les sous ensembles des états de l'AEF non déterministe.

➙ Dans les exercices suivants, commencer par l'état initial $[S_0]$ et ne traiter que les ensembles créés au fur à mesure.

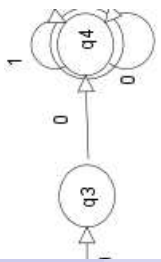
ND vers Det : exercices

• Donner les ensembles d'états d'arrivée dans l'AEFND suivant pour les chaînes d'entrée :

$$0,\ 01\ ,\ 010,\ 0100,\ 01001.$$

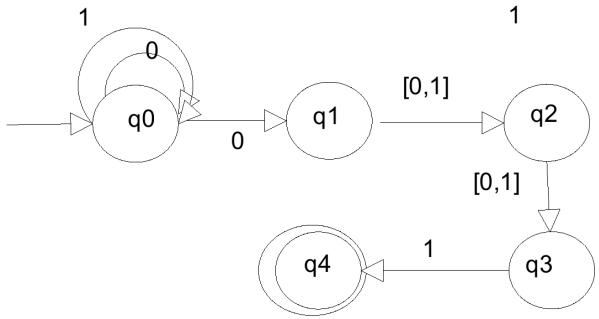
• Transformer ensuite cet AEFND en un AEFD :

Etats	**f**	
	Entrées	
	0	1
q_0	q_0, q_3	q_0, q_1
q_1	$\oslash$	q_2
q_2	q_2	q_2
q_3	q_4	$\oslash$
q_4	q_4	q_4



Exercices (suite)

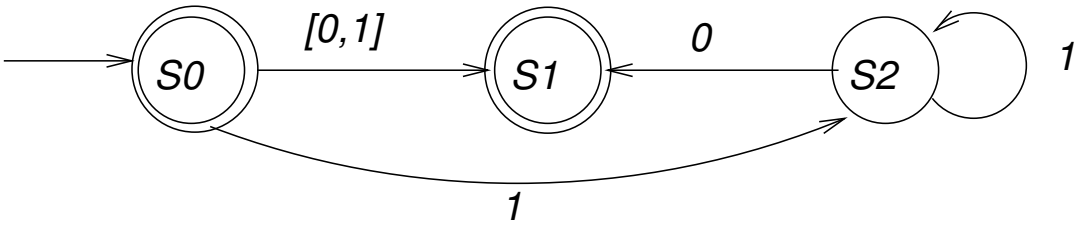
• **Execice** : Soit l'automate de la reconnaissance du flot X^*0XX1 (vu plus haut) avec $X \in \{0, 1\}$.



• Trouver un AEFD qui reconnaît le même langage.

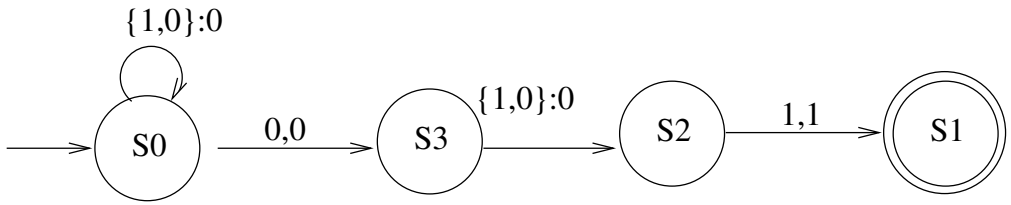
Exercices

• Trouver un AEF déterministe qui reconnaît le même langage que l'AEF non déterministe suivant :



Exercices

• En ignorant les sorties de la machine suivante, trouver un AEF déterministe qui reconnaît le même langage que l'AEF non déterministe suivant (vu plus haut) :



Suite AEFs

- Soit l'expression (régulière) E_1 suivante définissant :
  - ➙ $E_1 = (0+1)^*.(01+10).(0+1)^*$
  - ➙ N.B. : le $(0+1)$ veut dire $(0\ ou\ 1)$, que nous avons également noté par $[0,1]$.

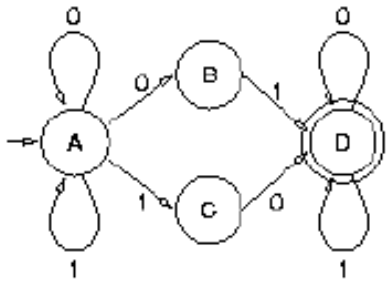


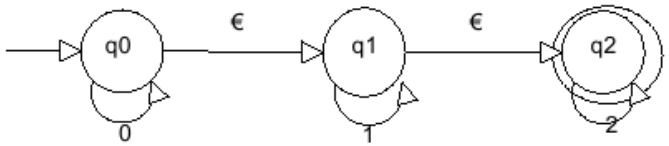
Fig.: AEFND pour l'expression $E_1 = (0+1)^*.(01+10).(0+1)^*$

- **Exercice** : transformer cet AEFND en un AEF déterministe.

ε-transitions

La ε-transition : une introduction informelle

- Un automate comportant des ε-transitions est une sorte d'AEFND.
- Exemple :



- La transformation s'effectue à l'aide de l'opération ε-**fermeture** (notée δ^{ε}).
- Intuitivement, cette opération consiste à mettre en évidence, pour un symbole d'entrée, les états accessibles depuis un état donné en utilisant à la fois des ε-transitions et une transition habituelle (δ).

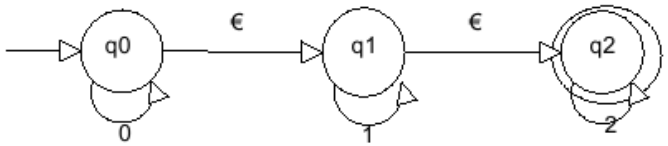
  ➙ Dans l'exemple, depuis l'état q_0 avec l'entrée 0, on atteint les états : $\{q_0, q_1, q_2\}$ car $q_0 \times 0 = q_0$, $q_0 \times \varepsilon = q_1$ et $q_1 \times \varepsilon = q_2$.

- Plus généralement : $\delta^{\varepsilon}(q_i, \alpha) = \bigcup_j q_j$ avec q_j obtenu par
  $q_j = \delta(q_i, \alpha)$ ou $q_j = \delta(q_i, \varepsilon)$ ou
  $q_j = \delta(\delta(q_i, \varepsilon), \varepsilon)$ ou $q_j = \delta(\delta(q_i, \alpha), \varepsilon)$ (transitivement).

ε-transitions (suite)

- L'automate ci-dessus présenté avec sa table de transitions :

q_i	**f**			
	Entrées			
	0	1	2	ε
q_0	$\{q_0\}$	$\oslash$	$\oslash$	$\{q_1\}$
q_1	$\oslash$	$\{q_1\}$	$\oslash$	$\{q_2\}$
q_2	$\oslash$	$\oslash$	$\{q_2\}$	$\oslash$



Calcul informel de la ε-fermeture :

- Pour calculer la ε-fermeture de $(q_0 \times 0)$, on lit dans la table :

$$\delta^{\varepsilon}(q_0, 0) = (q_0 \times 0 = q_0) \cup (q_0 \times \varepsilon = q_1) \cup (q_1 \times \varepsilon = q_2) \cup (q_2 \times \varepsilon = \oslash)$$
$$= \{q_0, q_1, q_2\}.$$

$$\delta^{\varepsilon}(q_0, 1) = (q_0 \times 1 = \emptyset) \cup (q_0 \times \varepsilon = q_1) \cup (q_1 \times 1 = q_1) \cup (q_1 \times \varepsilon = q_2)$$
$$= \{q_1, q_2\}.$$

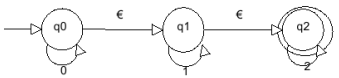
ε-transitions (suite)

- Pour le calcul de $\delta^{\varepsilon}(q, \omega)$, on considère :
  - les $\varepsilon - transitions$ pour aller de q à l'état p capable de reconnaître ω
  - on applique la transition $\delta(p, \omega) = r$ de reconnaissance de ω
  - ainsi que les $\varepsilon - transitions$ sur r et celles qui peuvent le suivre.

- $\delta^{\varepsilon}(S, \omega) =$
  $(S \times \varepsilon = S_1) \cup ... \cup (S_i \times \varepsilon = S_j) \cup (S_j \times \omega = S_k) \cup ... \cup (S_m \times \varepsilon = S_f)$
  N.B. : on arrête de prendre en compte l'état S_m si $(S_m \times \varepsilon = \emptyset)$.

- Dans l'exemple, pour $\delta^{\varepsilon}(q_0, 0)$, on considère tous les cas possibles :
  - ➤ $\delta^{\varepsilon}(q_0, 0) : (q_0 \times 0 = q_0) \cup (q_0 \times \varepsilon = q_1) \cup (q_1 \times \varepsilon = q_2)$
    - ➥ $\delta^{\varepsilon}(q_0, 0) = \{q_0, q_1, q_2\}$

  - ➤ $\delta^{\varepsilon}(q_0, 1) : (q_0 \times \varepsilon = q_1) \cup (q_1 \times 1 = q_1) \cup (q_1 \times \varepsilon = q_2)$
    - ➥ $\delta^{\varepsilon}(q_0, 1) = \{q_1, q_2\}$



ε-transitions (suite)

Algorithme de principe du calcul de Calcul de $\delta^{\varepsilon}(q,\omega), \omega \neq \varepsilon, \omega \in \Sigma$:

Début

S'il n'existe pas une transition $\delta(q_u,\omega) = q_v$ dans l'AEF

Alors Résultat $= \{\}$ (ω ne sera pas reconnu)

Sinon ($\delta(q_u,\omega) = q_v$ existe)

Si on peut atteindre q_u avec une séquence de $\varepsilon - transitions$ de la forme

$\delta(q,\varepsilon) = q_a, \delta(q_a,\varepsilon) = q_b, ..., \delta(q_t,\varepsilon) = q_u$ (puis $\delta(q_u,\omega) = q_v$)

Alors faire l'union des états $q_a ... q_t, q_u, q_v$ (mais pas q) puis

S'il existe des transitions

$\delta(q_v,\varepsilon) = q_w, \delta(q_w,\varepsilon) = q_x, ..., \delta(q_z,\varepsilon) = \emptyset$

Alors y ajouter également $q_w, q_x, ...$ (mais pas q_z)

Finsi

Finsi

Finsi

Résultat = L'ensemble d'états ainsi obtenu pour $\delta^{\varepsilon}(q,\omega)$

Fin

ε-transitions (suite)

• Pour l'automate précédent, après calculs des ε-fermetures, on obtient la table et le diagramme d'états suivants.

➙ On remarque bien l'AEF non déterministe.

q_i	**f**		
	Entrées		
	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	$\oslash$	$\{q_1, q_2\}$	$\{q_2\}$
q_2	$\oslash$	$\oslash$	$\{q_2\}$

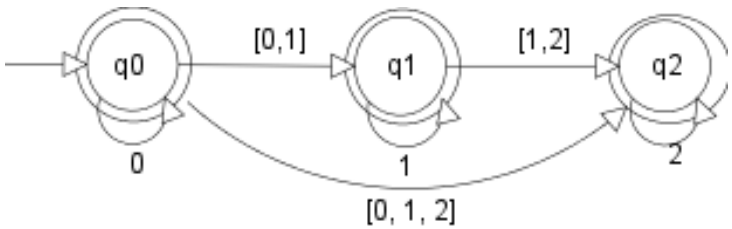
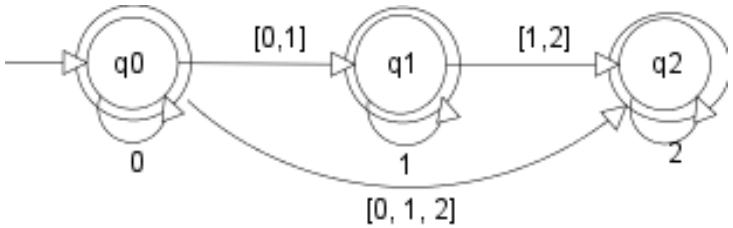


Fig.: l'AEFND correspondant à la table

• N.B. : q_0 et q_1 sont devenus des états finaux car ils sont toujours conjugés avec l'état final q_2 de $\varepsilon - AEFND$ d'origine.

Exercice

• **Exercice** : procéder à la transformation de l'automate non déterministe rappelé ci-dessous en un AEFD.



Elimination des $\varepsilon - transitions$ (approche plus formelle)

Elimination des $\varepsilon - transitions$ par une approche plus formelle :

• L'élimination des $\varepsilon - transitions$ cherche à définir la fonction de transition $\delta^{\varepsilon}(q, \omega)$ qui est une application de $Q \times \Sigma^* \longrightarrow 2^Q$.

• $\delta^{\varepsilon}(q, \omega)$ représentera tous les états p tels que l'on puisse aller de q à p (en incluant éventuellement des arcs portant ε) en présence de ω.

• Dans ce calcul, il est important de trouver les états **atteignables** depuis un état q en suivant seulement les arcs ε.

Ceci est équivalent à la recherche d'un noeud depuis un noeud source dans un graphe orienté :

➙ le graphe en question aura uniquement les arcs portant ε.

On notera par $\varepsilon - fermeture(q)$ l'ensemble des noeuds p tels qu'il y ait un chemin de q à p portant la valeur ε.

➙ Dans l'exemple, $\varepsilon - fermeture(q_0) = \{q_0, q_1, q_2\}$:

- q_0 y figure car il y a toujours un arc fictif de tout noeud (état) à lui-même.
- Y figurement les chemins $q_0 - q_1$ ainsi que $q_0 - q_1 - q_2$.

Elimination des $\varepsilon - transitions$ (approche plus formelle) (suite)

L'algorithme de calcul de $\varepsilon - fermeture(P)$ où P est un ensemble d'états
➟ $\varepsilon - fermeture(P) = \bigcup_{q \in P} \ \varepsilon - fermeture(q)$.

1) $\delta^{\varepsilon}(q, \varepsilon) = \varepsilon - fermeture(q)$
2) Pour $\omega \in \Sigma^*$ et $a \in \Sigma$, $\delta^{\varepsilon}(q, \omega a) = \varepsilon - fermeture(P)$ avec
P=$\{$p | pour un certain r dans $\delta^{\varepsilon}(q, \omega)$, p est dans $\delta(r, a)$ $\}$

On étends δ et δ^{ε} aux ensembles d'états par :
3) $\delta(R, a) = \bigcup_{q \in R} \ \delta(q, a)$ et
4) $\delta^{\varepsilon}(R, \omega) = \bigcup_{q \in R} \ \delta^{\varepsilon}(q, \omega)$ pour les ensembles d'états R.

➟ On notera que dans ces derniers cas, $\delta^{\varepsilon}(q, a)$ n'est pas forcément égale à $\delta(q, a)$ car $\delta^{\varepsilon}(q, a)$ inclue tous les états atteignables depuis q par les chemins portant a (y compris les chemins avec des arcs portant ε), tandis que $\delta(q, a)$ inclue seulement les états atteignables depuis q par des arcs portant a.
➟ De même $\delta^{\varepsilon}(q, \varepsilon)$ n'est pas forcément égale à $\delta(q, \varepsilon)$.

Elimination des $\varepsilon - transitions$ (approche plus formelle) (suite)

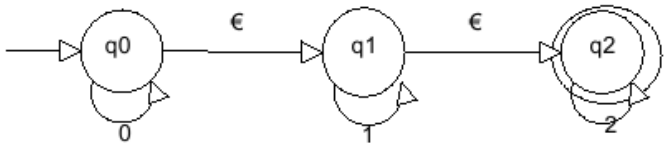
➙ Par conséquent, il faut distinguer δ de δ^{ε} lorsque l'on parle d'un AEFND avec des $\varepsilon - transitions$.

- **Le langage** $L(M)$ accepté par $M = (Q, \Sigma, \delta, q_0, F)$ est :
$$L(M) = \{w \mid \delta^{\varepsilon}(q_0, w) \textit{ contient un état dans } F\}$$

Elimination des $\varepsilon - transitions$ (approche plus formelle) (suite)

Exemple : on reprend l'AEFND précédent :

q_i	**f**			
	Entrées			
	0	1	2	ε
q_0	$\{q_0\}$	$\oslash$	$\oslash$	$\{q_1\}$
q_1	$\oslash$	$\{q_1\}$	$\oslash$	$\{q_2\}$
q_2	$\oslash$	$\oslash$	$\{q_2\}$	$\oslash$



On a :

$$\delta^{\varepsilon}(q_0, \varepsilon) = \varepsilon - fermeture(q_0) = \{q_0, q_1, q_2\}$$
$$\delta^{\varepsilon}(q_1, \varepsilon) = \varepsilon - fermeture(q_1) = \{q_1, q_2\}$$
$$\delta^{\varepsilon}(q_2, \varepsilon) = \varepsilon - fermeture(q_2) = \{q_2\}$$

Elimination des $\varepsilon - transitions$ (approche plus formelle) (suite)

Donc :

$$\begin{aligned}\delta^{\varepsilon}(q_0, 0) &= \varepsilon - fermeture(\delta(\delta^{\varepsilon}(q_0, \varepsilon), 0)) \\ &= \varepsilon - fermeture(\delta(\{q_0, q_1, q_2\}, 0)) \\ &= \varepsilon - fermeture(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)) \\ &= \varepsilon - fermeture(\{q_0\} \cup \oslash \cup \oslash) \\ &= \varepsilon - fermeture(\{q_0\}) = \{q_0, q_1, q_2\}\end{aligned}$$

Et

$$\begin{aligned}\delta^{\varepsilon}(q_0, 01) &= \varepsilon - fermeture(\delta(\delta^{\varepsilon}(q_0, 0), 1)) \\ &= \varepsilon - fermeture(\delta(\{q_0, q_1, q_2\}, 1)) \\ &= \varepsilon - fermeture(\{q_1\}) = \{q_1, q_2\}\end{aligned}$$

Calcul de ε-fermeture revisitée

Calcul de ε-fermeture revisitée via un aure exemple :

• Rappel : dans un AEF avec un ensemble d'états Q et l'état $q \in Q$, $\alpha - successeur(q)$ désigne l'ensemble des états que l'on peut atteindre depuis l'état $q \in Q$ en utilisant le symbole d'entrée α.

➥ On étend facilement cette notion à un sous-ensemble $E \subseteq Q$.

• La définition de l'ε-fermeture d'un état q, notée $\varepsilon - fermeture(q)$, et de son extension sont utiles pour l'élimination des epsilon-transitions.

• Etant donné état $q \in Q$, on définit l'ε-fermeture de q par :

$$\varepsilon - fermeture(q) = \{p | q \xrightarrow{\varepsilon}{}^{*} p\}$$

➤ En d'autres termes, $\varepsilon - fermeture(q)$ est l'ensemble de tous les états dans Q que l'on pourrait atteindre à partir de q en éfectuant 0, 1 ou plusieurs ε-transitions (0 transitions sur q veut dire : $q \in \varepsilon - fermeture(q)$) .

• L'opération $\varepsilon - fermeture(q)$, comme pour $\alpha - successeur(q)$, pourrait être étendue aux ensembles :

$$\varepsilon - fermeture(E) = \bigcup_{q \in E} \ \varepsilon - fermeture(q) \quad avec \ E \subseteq Q$$

Calcul de ε-fermeture revisitée (suite)

Exemple :

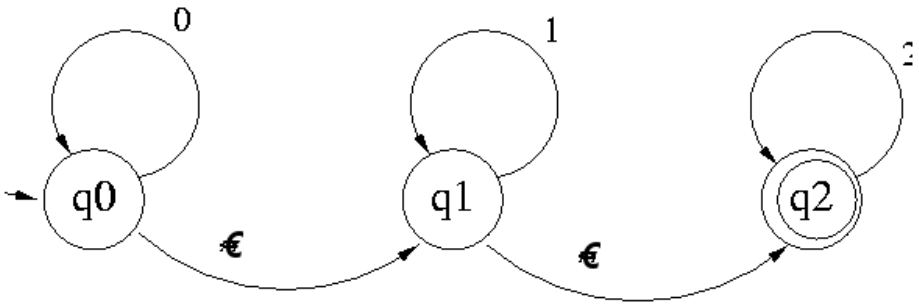
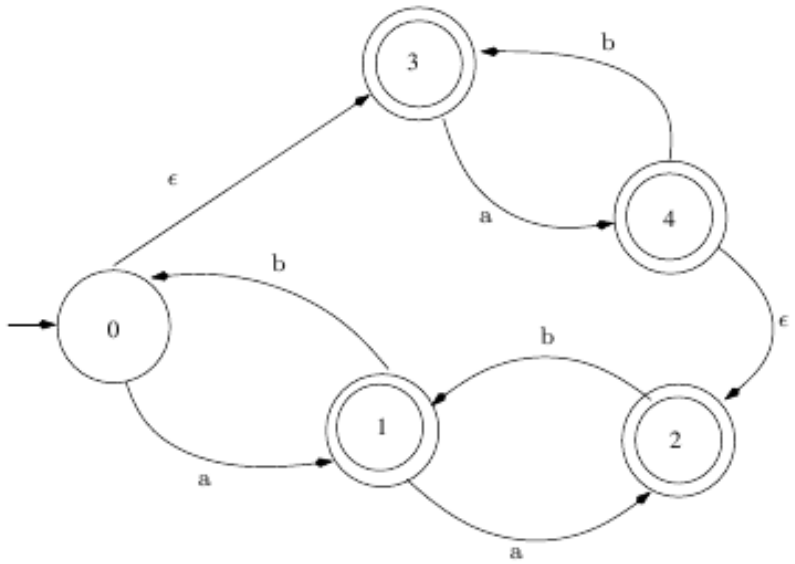


Fig.: AEFND reconnaissant $0^*1^*2^*$ avec des $\varepsilon - transitions$

ε-fermeture(q0) = {q0,q1,q2},
ε-fermeture(q1) = {q1,q2},
ε-fermeture(q2) = {q2},
ε-fermeture({q1,q2}) = ε-fermeture(q1) $\cup$ ε-fermeture(q2)
= {q1,q2} $\cup$ {q2} = {q1,q2}

Calcul de ε-fermeture revisitée (suite)

Exemple-2 et tous les calculs jusqu'à l'AEFD



Etape-1 : transformation en AEFND sans $\varepsilon - transitions$:

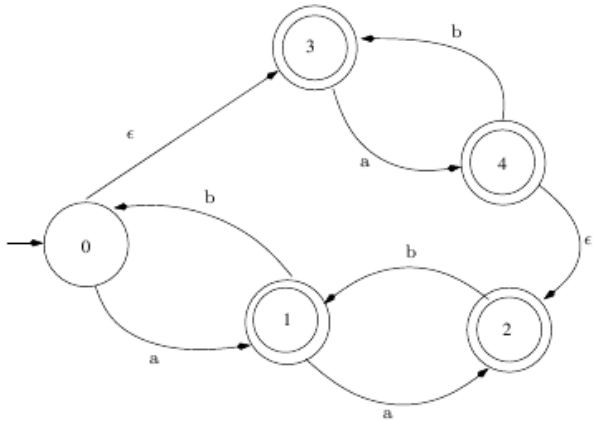
• Cet automate est assez complexe et on va appliquer la méthode formelle vue plus haut.

• On commence par calculer les $\varepsilon - fermetures(q_i) = \delta^{\varepsilon}(q_i, \varepsilon)$ pour tous les états q_i, puis $\delta^{\varepsilon}(q_i, \alpha)$ pour $q_i \in Q, \alpha \in \Sigma$.

Calcul de ε-fermeture revisitée (suite)

Calcul des $\varepsilon - fermetures(q_i)$:

$\varepsilon - fermeture(q_0) = \delta^{\varepsilon}(q_0, \varepsilon) = \{q_0, q_3\}$,
$\varepsilon - fermeture(q_1) = \delta^{\varepsilon}(q_1, \varepsilon) = \{q_1\}$,
$\varepsilon - fermeture(q_2) = \delta^{\varepsilon}(q_2, \varepsilon) = \{q_2\}$,
$\varepsilon - fermeture(q_3) = \delta^{\varepsilon}(q_3, \varepsilon) = \{q_3\}$,
$\varepsilon - fermeture(q_4) = \delta^{\varepsilon}(q_4, \varepsilon) = \{q_2, q_4\}$



• Rappel : voir l'encadré plus haut pour l'assimiliation de ce calcul à un parcours de graphe. Ces chemin sont directement visibles sur l'AEF ci-dessus.

Ici, le chemin ne contenant que des $\varepsilon - transitions$ en partant de q_0 est $q_0 - q_3$. De même pour q_4 qui donne le chemin $q_2 - q_4$.

Pour les autres q_i, le chemin se réduit à l'état (noeud) lui même seul.

• La phase suivante est le calcul des $\delta^{\varepsilon}(q_i, \alpha)$ pour $q_i \in Q, \alpha \in \Sigma$.

Calcul de ε-fermeture revisitée (suite)

$$\begin{aligned}
\delta^{\varepsilon}(q_0, a) &= \varepsilon - fermeture(\delta(\delta^{\varepsilon}(q_0, \varepsilon), a)) \\
&= \varepsilon - fermeture(\delta(\{q_0, q_3\}, a)) \\
&= \varepsilon - fermeture(\delta(\{q_0\}, a) \cup \delta(\{q_3\}, a)) \\
&= \varepsilon - fermeture(\{q_1\} \cup \{q_4\}) \\
&= \varepsilon - fermeture(\{q_1\}) \cup \varepsilon - fermeture(\{q_4\}) \\
&= \{q_1\} \cup \{q_2, q_4\}) = \{q_1, q_2, q_4\}
\end{aligned}$$

Att : ici δ , pas δ^{ε}

$\varepsilon - fermeture(q_4) = \{q_2, q_4\}$

$$\begin{aligned}
\delta^{\varepsilon}(q_0, b) &= \varepsilon - fermeture(\delta(\delta^{\varepsilon}(q_0, \varepsilon), b)) \\
&= \varepsilon - fermeture(\delta(\{q_0, q_3\}, b)) \\
&= \varepsilon - fermeture(\delta(\{q_0\}, b) \cup \delta(\{q_3\}, b)) \\
&= \varepsilon - fermeture(\emptyset) \\
&= \emptyset
\end{aligned}$$

Calcul de ε-fermeture revisitée (suite)

$$\begin{aligned}\delta^{\varepsilon}(q_1, a) &= \varepsilon - fermeture(\delta(\delta^{\varepsilon}(q_1, \varepsilon), a)) \\ &= \varepsilon - fermeture(\delta(\{q_1\}, a)) \\ &= \varepsilon - fermeture(\{q_2\}) \\ &= \varepsilon - fermeture(\{q_2\}) \\ &= \{q_2\}\end{aligned}$$

$$\begin{aligned}\delta^{\varepsilon}(q_1, b) &= \varepsilon - fermeture(\delta(\delta^{\varepsilon}(q_1, \varepsilon), b)) \\ &= \varepsilon - fermeture(\delta(\{q_1\}, b)) \\ &= \varepsilon - fermeture(\{q_0\}) \\ &= \{q_0, q_3\}\end{aligned}$$

$$\delta^{\varepsilon}(q_2, a) = ... = \emptyset$$

$$\delta^{\varepsilon}(q_2, b) = ... = \{q_1\}$$

Calcul de ε-fermeture revisitée (suite)

$$\begin{aligned}\delta^{\varepsilon}(q_3, a) &= \varepsilon - fermeture(\delta(\delta^{\varepsilon}(q_3, \varepsilon), a)) \\ &= \varepsilon - fermeture(\delta(\{q_3\}, a)) \\ &= \varepsilon - fermeture(\{q_4\}) \\ &= \{q_2, q_4\}\end{aligned}$$

$$\delta^{\varepsilon}(q_3, b) = ... = \emptyset$$

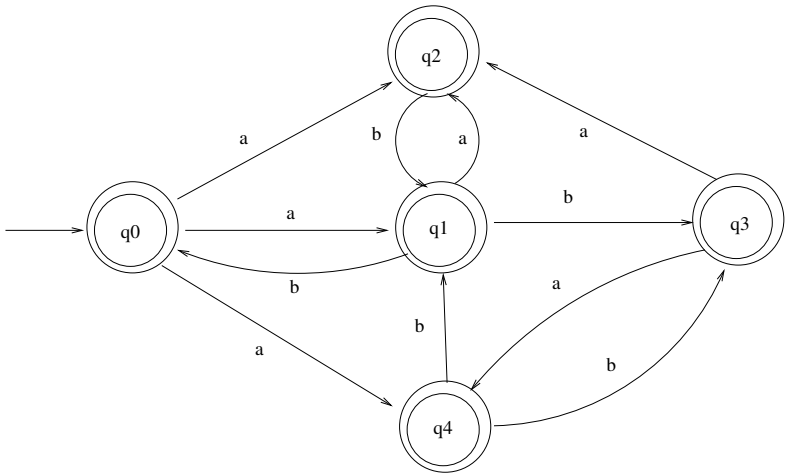
$$\delta^{\varepsilon}(q_4, a) = ... = \emptyset$$

$$\begin{aligned}\delta^{\varepsilon}(q_4, b) &= \varepsilon - fermeture(\delta(\delta^{\varepsilon}(q_4, \varepsilon), b)) \\ &= \varepsilon - fermeture(\delta(\{q_2, q_4\}, b)) \\ &= \varepsilon - fermeture(\{q_3\}) \cup \varepsilon - fermeture(\{q_1\}) \\ &= \{q_1, q_3\}\end{aligned}$$

Calcul de ε-fermeture revisitée (suite)

On peut maintenant procéder à la création de l'AEFND (sans $\varepsilon - transition$)

état	a	b
q_0	$\{q_1, q_2, q_4\}$	$\emptyset$
q_1	$\{q_2\}$	$\{q_0, q_3\}$
q_2	$\emptyset$	$\{q_1\}$
q_3	$\{q_2, q_4\}$	$\emptyset$
q_4	$\emptyset$	$\{q_1, q_3\}$



N.B. : du fait de la $\varepsilon - transition$ sur q_0, q_0 sera aussi un état final. Ceci est visible par $\varepsilon - fermeture(q_0) = \{q_0, q_3\}$ et q_3 est un état final.

- L'étape suivante sera la transformation de cet AEFNF en AEFD.

Calcul de ε-fermeture revisitée (suite)

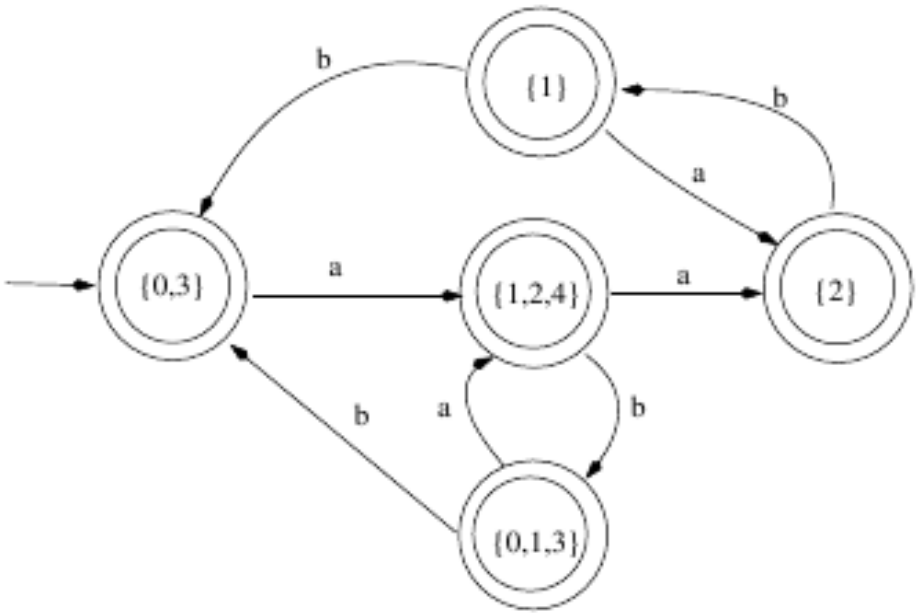
Etape-2 : transformation de l'AEFNF en AEFD.

- Transformations en AEFD en commençant par $[q_0]$

$$
\begin{aligned}
&[q_0] \times a = [q_1, q_2, q_4] \\
&[q_0] \times b = \emptyset \\
&[q_1, q_2, q_4] \times a = (q_1 \times a) \cup (q_2 \times a) \cup (q_4 \times a) = [q_2] \\
&[q_1, q_2, q_4] \times b = ... = [q_0, q_1, q_3] \\
&[q_2] \times a = \emptyset \\
&[q_2] \times b = [q_1] \\
&[q_0, q_1, q_3] \times a = ... = [q_1, q_2, q_4] \\
&[q_0, q_1, q_3] \times b = ... = [q_0, q_3] \\
&[q_1] \times a = [q_2] \\
&[q_1] \times b = [q_0, q_3] \\
&[q_0, q_3] \times a = ... = [q_1, q_2, q_4] \\
&[q_0, q_3] \times b = \emptyset
\end{aligned}
$$

Calcul de ε-fermeture revisitée (suite)

- L'AEFD :



N.B. : l'ajout de l'état final $[q_0]$ n'apporte rien : l'état $[q_0, q_3]$ joue exactement le même rôle.

➥ $[q_0]$ disparaîtra dans un processus de minimisation (voir plus loin).

Minimisation

Minimisation des AEFs :

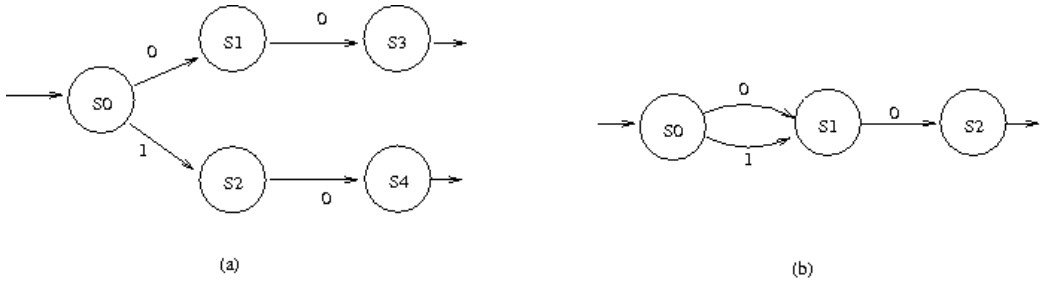
• Principe de base : pour tout langage L, s'il existe plusieurs AEFs reconnaissant L, alors l'un des ces AEFs est déterministe et **minimal** (en nombre d'états).

➙ Notion informelle de classe d'équivalence : on dira que chaque état de cet AEF **minimal** représente (regroupe) plusieurs états pris dans un AEF équivalent (non minimal). Il faudra ensuite préciser la relation d'équivalence entre états (et langages).

➙ Formellement : il existe un représentant unique (à une renumérotation des états près) et **minimal** (en nombre d'états) pour les classes de relations d'équivalence sur les automates finis. ../..

Minimisation (suite)

Une idée intuitive de la minimisation :
Dans la figure, la partie (a) sera réduite en partie (b) :



Automate Canonique : pour un langage L donné et une famille d'AEFs tous reconnaissant L, on peut trouver un automate A_L minimal :

Théorème

L'automate A_L , fondé sur la relation d'équivalence $\equiv_L$, est le plus petit automate déterministe complet reconnaissant L. Cet automate est unique (à une renumérotation des états près) et appelé automate **canonique** *de L.*

Minimisation (suite) (suite)

Démonstration.

soit A un automate fini déterministe reconnaissant L.

➙ $\equiv_A$ définit une relation d'équivalence.

On peut montrer que chaque état de A correspond à une classe d'équivalence (pour $\equiv_A$) incluse dans une classe d'équivalence pour $\equiv_L$.

➙ Le nombre d'états de A est donc nécessairement plus grand que celui de A_L.

Le cas où A et A_L ont le même nombre d'états correspond au cas où les classes d'équivalences sont toutes semblables, permettant de définir une correspondance biunivoque entre les états des deux machines. ∎

L'existence de A_L étant garantie, reste à savoir comment le construire :

➙ la construction directe des classes d'équivalence de $\equiv_L$ n'est en effet pas nécessairement immédiate.

Minimisation (suite) (suite)

Principe de minimisation :
On présente ici les éléments d'un algorithme permettant de construire A_L à partir d'un automate déterministe quelconque reconnaissant L.

➙ Comme préalable, définissons une troisième relation d'indistinguabilité, portant cette fois sur les états :

Définition

Deux états q et p d'un automate fini déterministe A sont **distinguables** *s'il existe un mot w tel que le calcul $\delta(q, w)$ termine dans un état final alors que le calcul $\delta(p, w)$ ne termine pas dans un état final (voire, $\delta(p, w)$ échoue).*

➙ *Si deux états ne sont pas distinguables, ils sont indistinguables.*

L'indistinguabilité est une relation d'équivalence, notée $\equiv_v$ sur les états de Q.
L'ensemble des classes d'équivalence $[q]_v$ est notée Q_v.

Minimisation (suite)

Principe du calcul de l'automate minimal :
Pour un automate fini déterministe $A = (\Sigma, Q, q_0, F, \delta)$, on définit l'automate fini $A_v = (\Sigma, Q_v, [q_0]_v, F_v, \delta_v)$, avec :

$\delta_v([q]_v, a) = [\delta(q, a)]_v$ et $F_v = [q]_v$, si q dans F.

δ_v est correctement défini en ce sens que si p et q sont indistinguables, alors nécessairement $\delta(q, a) \equiv_v \delta(p, a)$.

Le but de l'algorithme de minimisation de $A = (\Sigma, Q, q_0, F, \delta)$ consiste alors à chercher à identifier les classes d'équivalence pour $\equiv_v$ de manière à construire l'automate A_v (alias A_L).

➙ La finitude de Q nous garantit l'existence d'un algorithme pour calculer ces classes d'équivalence.

➙ La procédure itérative suivante permet une implantation naïve de cet algorithme, qui construit la partition correspondant aux classes d'équivalence par raffinement d'une partition initiale $\prod_0$ qui au départ distingue simplement les états finaux des états non-finaux.

Algorithme de Minimisation

L'algorithme (de Moore) se décrit comme suit :

> *- Initialiser avec deux classes d'équivalence :* F *et* $Q \setminus F$ *(➙* $\prod_0$*)*
> *- Itérer jusqu'à stabilisation :*
> *- pour toute paire d'état* (q, p) *dans la même classe de la partition* $\prod_k$*, s'il existe* $a \in \Sigma$ *tel que* $\delta(q, a)$ *et* $\delta(p, a)$ *ne sont pas dans la (même) classe dans* $\prod_k$*, alors ils seront dans deux classes différentes de* $\prod_{k+1}$.

On vérifie que lorsque cette procédure s'arrête (après un nombre fini d'étapes), **deux états sont dans la même classe Si-et-Seulement-Si ils sont indistinguables**.

➙ Cette procédure est connue sous le nom d'*algorithme de Moore*.

Dans sa version naïve, elle est d'une complexité quadratique (à cause de l'étape de comparaison de toutes les paires d'états).

➙ En utilisant des structures auxiliaires, il est toutefois possible de diminuer la complexité à $nlog(n)$, avec n le nombre d'états.

Exemple de Minimisation

Premier exemple : soit à minimiser l'AEF suivant :

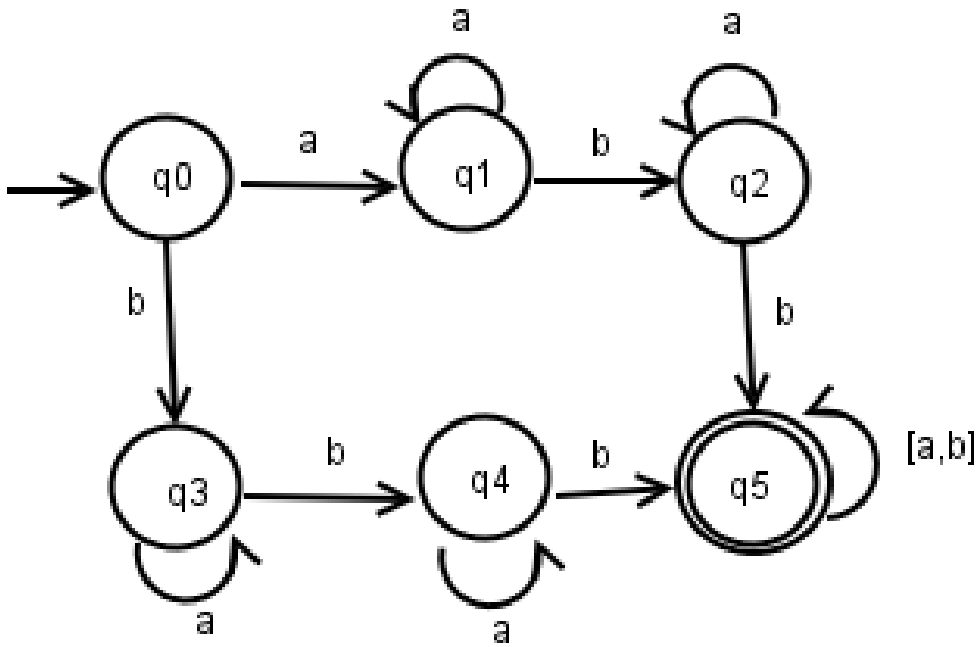


Fig.: AEF non minimal à minimiser

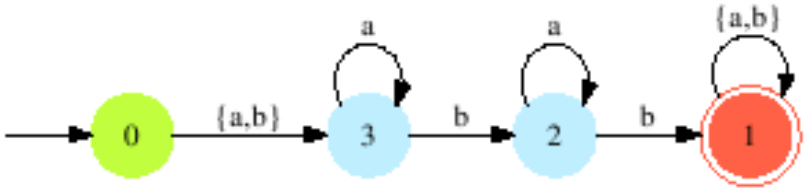
➙ N.B. : le langage $L = (a+b)a^*ba^*b(a+b)^*$

Exemple de Minimisation (suite)

Les itérations successives de l'algorithme de construction des classes d'équivalence pour $\equiv_v$ se déroulent alors comme suit :

$\prod_O = \{\{q_0, q_1, q_2, q_3, q_4\}, \{q_5\}\}$ (q_5 est le seul état final)
$\prod_1 = \{\{q_0, q_1, q_3\}, \{q_2, q_4\}, \{q_5\}\}$ (car q_2 et q_4 , sur le symbole b, atteignent q_5 qui n'est pas dans leur classe).
$\prod_2 = \{\{q_0\}, \{q_1, q_3\}, \{q_2, q_4\}, \{q_5\}\}$ (car q_1 et q_3, sur le symbole b, atteignent respectivement q_2 et q_4 qui sont dans une classe autre d'équivalence)
$\prod_3 = \prod_2$ pas de changement ➙ fin de la procédure.

L'automate minimal résultant de ce calcul est l'AEF minimal suivant (dont le langage $L = (a+b)a^*ba^*b(a+b)^*$) :



Minimisation (Ex2)

Minimisation : exemple 2 :
L'AEF à minimser

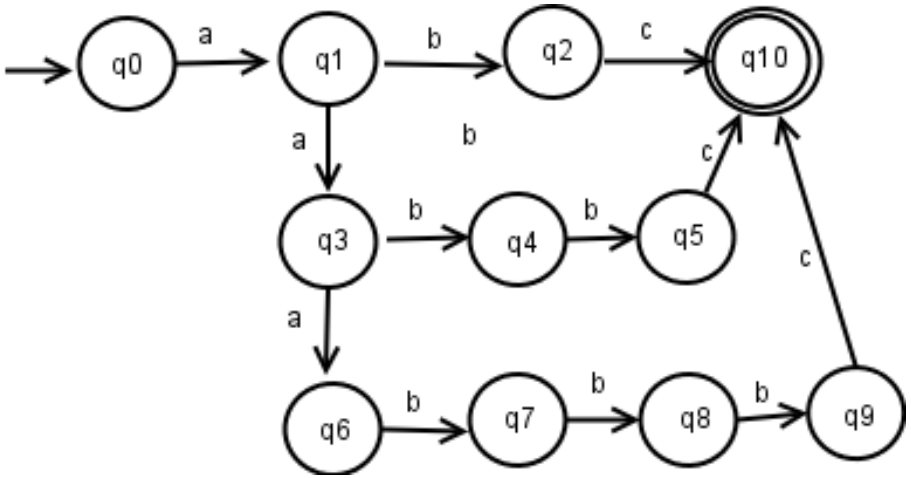


Fig.: AEF à minimiser

On utilisera une structure de donnée tableau pour cet exemple.

Minimisation (Ex2) (suite)

Rappels

➙ L'idée de base de la minimisation est de trouver des états *équivalents* que l'on pourra regrouper pour diminuer le nombre total d'états.

➙ Deux états seront équivalents s'ils nous mènent à un même état.

• L'algorithme pour trouver l'AEF minimum nécessite de partitionner l'ensemble des états en les regroupant en paquets d'états équivalents (classes d'équivalence).

La mise en oeuvre : on procède comme suit :

- On commence par créer deux partitions contenant deux ensembles : l'ensemble contenant les états finaux et l'ensemble contenant tous les autres états.
- Ensuite, pour chacun des classes de la partition, on crée une matrice comme la suivante où

La première colonne contiendra le numéro de l'étape, la deuxième colonne les classes de la partition en cours de raffinement, et la troisième les transitions pour les états de la classes de la partition.

Minimisation (Ex2) (suite)

Etape 1

No	Partition	Transitions

Etat	a	b	c
q10			

No	Partition	Transitions
	O2={q0,q1,q2,q3,q4,q5,q6,q7,q8,q9}	

Etat	a	b	c
q0	O2		
q1	O2	O2	
q2			O1
q3	O2	O2	
q4		O2	
q5			O1
q6		O2	
q7		O2	
q8		O2	
q9			O1

On constate que dans la colonne 3 de ce tableau, pour une transition donnée, on note la **partition** de l'état d'arrivée (au lieu de l'état).

Minimisation (Ex2) (suite)

➙ Par exemple, avec la transition $q_1 \times a = q_3$, puisque q_3 est dans la partition $O2$, on a noté $O2$ dans la table des transitions de $q_1 \times a$ (au lieu de noter q_3).

➙ L'intérêt de cette convention est que l'on peut immédiatement repérer les états équivalents : ce sont les états dont les contenus des cellules (colonnes a,b,c) sont identiques.

➙ Par exemple, q_1 et q_3 ont les mêmes valeurs dans leurs cellules respectives. Idem pour $\{q_2, q_5, q_9\}$

➙ Attention : ces ensembles équivalents sont susceptibles de se modifier pendant les prochaines phases (de recherche d'équivalences).

Etape 2

On reprend les deux partitions O1 et O2 ci-dessus.

➙ Il n'y a pas de raffinement à faire sur O1 (un seul état).

➙ Dans la partition O2, $\{q_0\}$ constitue sa propre partition. Notons sa partition définitive **1_o2_1** : "1" est la niveau précédent de partitionnement, "o2" est la partition parente, et "1" est le numéro pour la distinguer des autres "1_o2".

Minimisation (Ex2) (suite)

<table>
<tr><th>No</th><th>Partition</th><th>Transitions</th></tr>
<tr><td>2</td><td>1_o2_1 ={ q0 }</td><td></td></tr>
<tr><td></td><td>1_o2_2 ={ q1,q3}</td><td>
<table>
<tr><th>état</th><th>a</th><th>b</th><th>c</th></tr>
<tr><td>q1</td><td>1_o2_2</td><td>1_o2_3</td><td></td></tr>
<tr><td>q3</td><td>1_o2_4</td><td>1_o2_4</td><td></td></tr>
</table>
</td></tr>
<tr><td>3</td><td>1_o2_3 ={ q2,q5,q9 }</td><td>pas d'autres partitions</td></tr>
<tr><td></td><td>1_o2_4={q4, q6,q7,q8 }</td><td>
<table>
<tr><th>Etat</th><th>a</th><th>b</th><th>c</th></tr>
<tr><td>q4</td><td></td><td>1_o2_3</td><td></td></tr>
<tr><td>q6</td><td></td><td>1_o2_4</td><td></td></tr>
<tr><td>q7</td><td></td><td>1_o2_4</td><td></td></tr>
<tr><td>q8</td><td></td><td>1_o2_3</td><td></td></tr>
</table>
</td></tr>
<tr><td>4</td><td>2_1_o2_2_1 ={ q1 }</td><td></td></tr>
<tr><td>5</td><td>2_1_o2_2_2 ={ q3 }</td><td></td></tr>
<tr><td>6</td><td>2_1_o2_4_1 ={ q4,q8 }</td><td></td></tr>
<tr><td></td><td>2_1_o2_4_2={q6, q7 }</td><td>
<table>
<tr><th>Etat</th><th>a</th><th>b</th><th>c</th></tr>
<tr><td>q6</td><td></td><td>2_1_o2_4_2</td><td></td></tr>
<tr><td>q7</td><td></td><td>2_1_o2_4_1</td><td></td></tr>
</table>
</td></tr>
<tr><td>7</td><td>3_2_1_o2_4_2_1 ={ q6 }</td><td></td></tr>
<tr><td>8</td><td>3_2_1_o2_4_2_2 ={ q7 }</td><td></td></tr>
</table>

Minimisation (Ex2) (suite)

- Le schéma suivant donne l'arborescence des partitions.

Minimisation (Ex2) (suite)

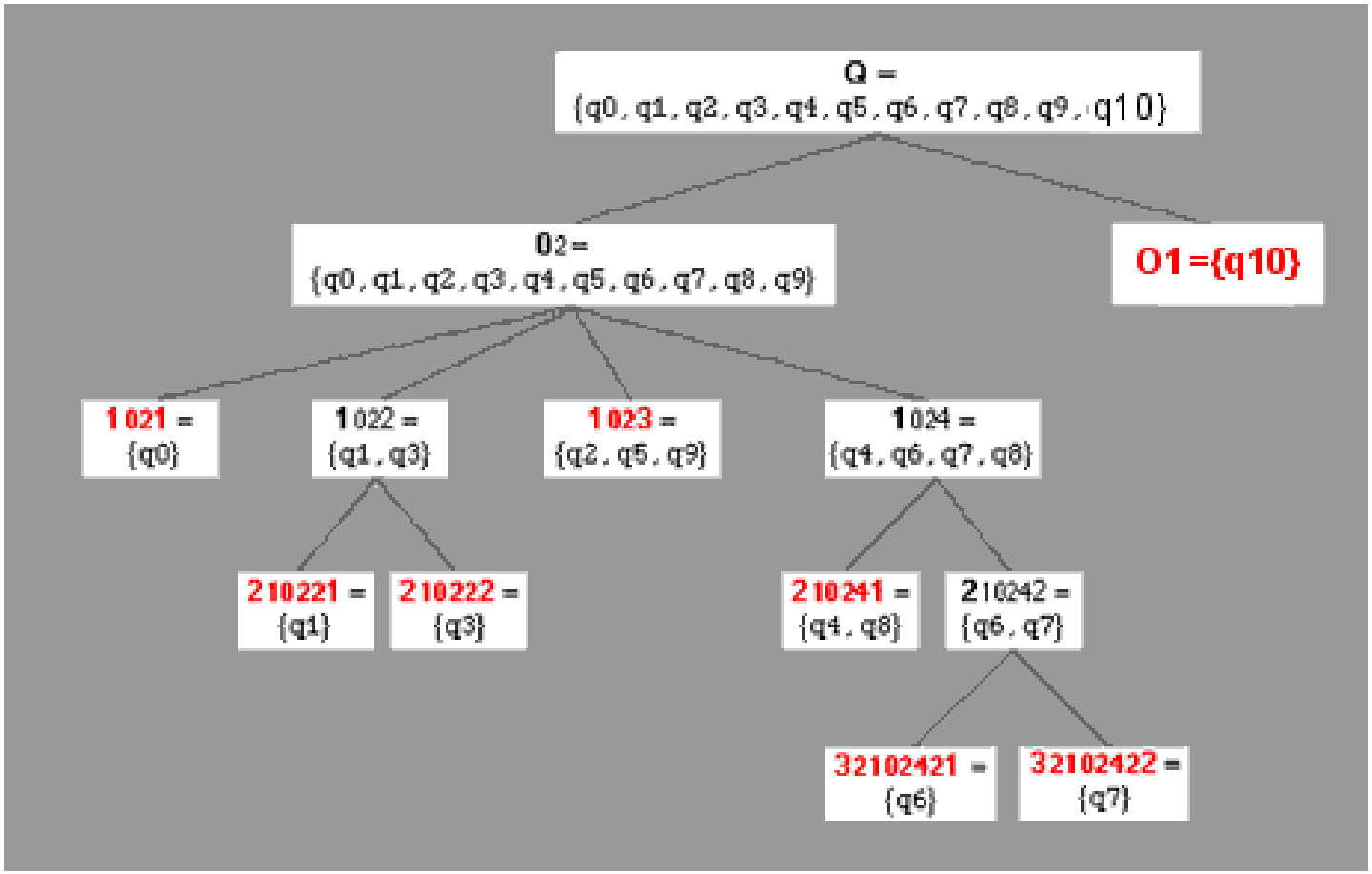


Fig.: Les partitions

Minimisation (Ex2) (suite)

L'AEF minimisé :

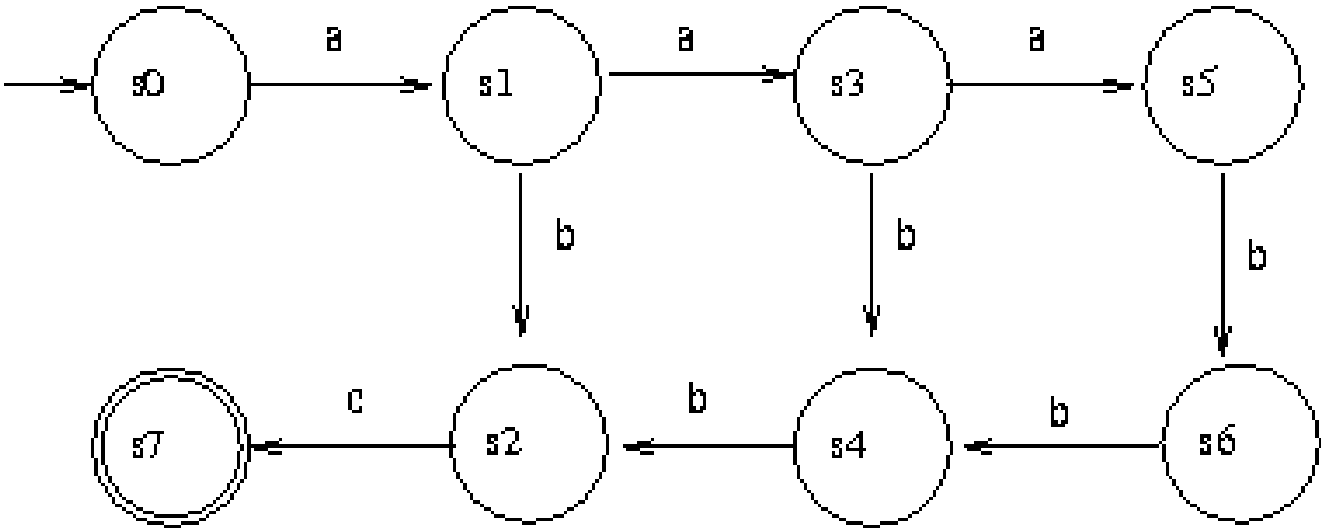
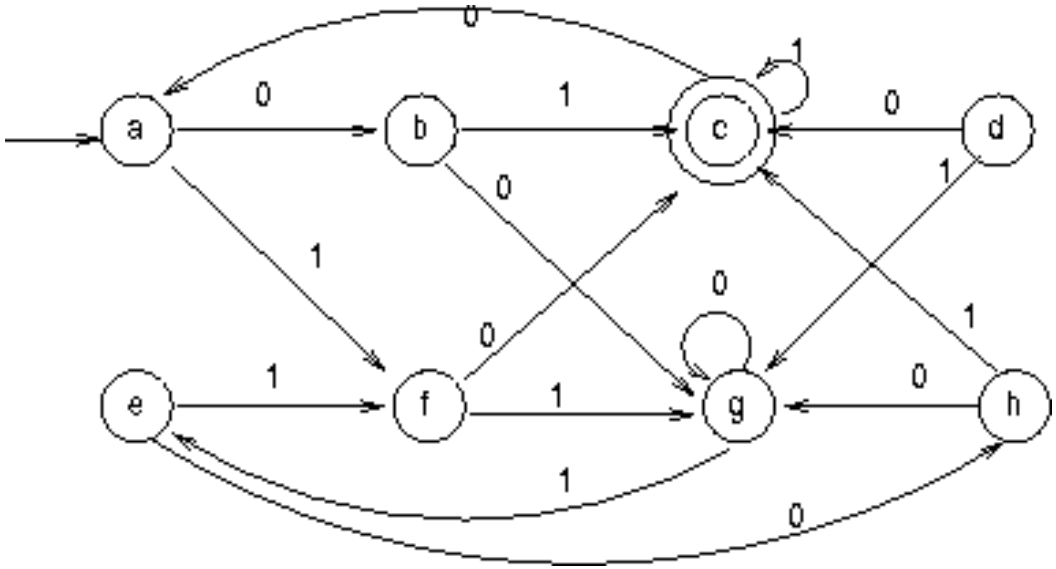


Fig.: AEF minimisé final

Minimisation : algorithme via un autre exemple

Minimisation à l'aide d'une matrice illustrée par l'exemple-3.

➙ Cet autre exemple est plus détaillé :



On détaillera la construction des classes d'équivalence par l'algorithme itératif vu plus haut dans une **version optimisée** (à l'aide d'une table). Un algorithme décrivant cette méthode est donné.

➙ Néanmoins, le traitement des exemples 2 et 3 suit le même principe. ../..

Minimisation : algorithme via un autre exemple (suite)

• Pour cet AEF (appelé M), on utilise la matrice suivante.

➙ Dans cette matrice, on place un X dans la case $(p, q),\ p, q \in Q$ à chaque fois que l'on découvre une paire d'états sur ces entrées qui ne sont pas équivalents.

➥ Au départ, on place un X dans les entrées qui correspondent à un état final (ici c) et un état non final (voir explications plus haut).

➙ Dans notre exemple, on place initialement un X dans les entrées $(a, c), (b, c), (c, d), (c, e), (c, f), (c, g)$ et (c, h).

➙ N.B. : seule la zone sous la diagonle inférieure du tableau est nécessaire (les couples (p, q) d'états ne sont pas ordonnés).

b							
c	X	X					
d			X				
e			X				
f			X				
g			X				
h			X				
	a	**b**	**c**	**d**	**e**	**f**	**g**

Minimisation : algorithme via un autre exemple (suite)

Algorithme de minimisation suivant permet de marquer (par un X) les paires non équivalentes (distinguables).

Soit l'AEF $M = (Q, \Sigma, \delta, q_0, F)$ à minimiser ;
Début
  Pour $p \in F$ et $q \in Q \setminus F$ faire marquer (p, q) fin faire
  Pour toute paire distinguable $(p, q) \in (F \times F)$ ou $(Q - F) \times (Q - F)$ faire
    Si pour un symbole d'entrée α, la case $(\delta(p, \alpha), \delta(q, \alpha))$ est marquée alors
      Marquer (p, q) ;
      Marquer récursivement toutes les cases non marquées dans la liste associée à (p, q) et dans celles associées aux autres paires marquées à cette même étape ;
    Sinon % case $(\delta(p, \alpha), \delta(q, \alpha))$ non marquée
      Pour tout symbole d'entrée α faire
        Placer (p, q) dans la liste associée à $(\delta(p, \alpha), \delta(q, \alpha))$
          sauf si $\delta(p, \alpha) = \delta(q, \alpha)$
      Fin Pour
    Fin si
  Fin pour
Fin

➙ On applique l'algorithme à l'AEF ci-dessus.

Minimisation : algorithme via un autre exemple (suite)

Retour à l'AEF et l'application de l'algorithme :

... Pour chaque état p et q qui ne sont pas distinguables, on considère les paires d'états $r = \delta(p, a)$ et $s = \delta(q, a)$ pour chaque symbole d'entrée a.

➙ Si les états r et s sont *distinguables* pour un certain symbole x, alors p et q sont distinguables pour l'entrée ax.

➙ Donc, si l'entrée (r, s) dans la table possède un X, alors un X est placé dans l'entrée (p, q).

➙ Aussi, si l'entrée (r, s) ne possède pas encore un X, alors la paire (p, q) est placée dans une liste associée avec l'entrée (r, s).

➙ Plus tard, si l'entrée (r, s) reçoit un X, alors chaque paire dans cette liste associée à l'entrée (r, s) reçoit aussi un X.

Minimisation : algorithme via un autre exemple (suite)

Déroulement :
Pour plus de clarté, on numérote les X insérés dans la table :

- (c, f) possède déjà un X
  ➙ $b \times 1 = \boxed{c}$ et $a \times 1 = \boxed{f}$
  ➙ On met X_1 dans la case (a, b) de la table.
- (b, c) possède déjà un X
  ➙ $a \times 0 = \boxed{b}$ et $d \times 0 = \boxed{c}$
  ➙ On met X_2 dans la case (a, d) de la table.
- (c, e) possède déjà un X
  ➙ $b \times 1 = \boxed{c}$ et $h \times 1 = \boxed{c}$
  ➙ et $g \times 1 = \boxed{e}$
  ➙ On met X_3 dans les deux cases (b, g) et (h, g) de la table.

Minimisation : algorithme via un autre exemple (suite)

- (c, g) possède déjà un X
  ➙ $d \times 0 = \boxed{c}$ et $f \times 0 = \boxed{c}$
  ➙ et $b \times 0 = \boxed{g}$ et $h \times 0 = \boxed{g}$
  ➙ On met X_4 dans les 4 cases (d, b), (d, hg), (f, b), (f, hg) de la table.

  ➤ N.B. : le même calcul avec (c, g) et l'entrée 1 donne les mêmes X_4.
- (b, h) ne possède pas de X mais on a
  ➙ $a \times 0 = \boxed{b}$ et $e \times 0 = \boxed{h}$
  ➙ On met (a, e) dans la liste associée à (b, h).
- (c, h) possède un X et il y a des transitions partant de d et f qui arrivent à c avec l'entrée 0 mais il n'y a pas de transition vers h avec cette entrée.
- (a, c) possède un X et $c \times 0 = a$ et $[f, d] \times 0 = c$ mais ceci n'ajoute rien de nouveau à la table.
  De même (b, g) possède un X et $a \times 0 = b$ et $[b, h] \times 0 = g$ mais ceci n'ajoute rien de nouveau à la table ?

Minimisation : algorithme via un autre exemple (suite)

- Ainsi, sur le symbole 1, a et g vont tous les deux à l'état f et donc aucune chaîne commencant avec 1 ne distingue a de e.
  ➙ De même et à cause du symbole 0, la paire (a, g) est placée dans la liste associée à (b, g).
- Lorsque l'entrée (b, g) est considérée, elle reçoit un X pour le compte du symbole 1 et donc la paire (a, g) reçoit un X car elle était dans la liste (b, g) : la chaîne 01 permet de distinguer a de g ...

➔ En itérant jusque la fin, on obtient la table suivante :

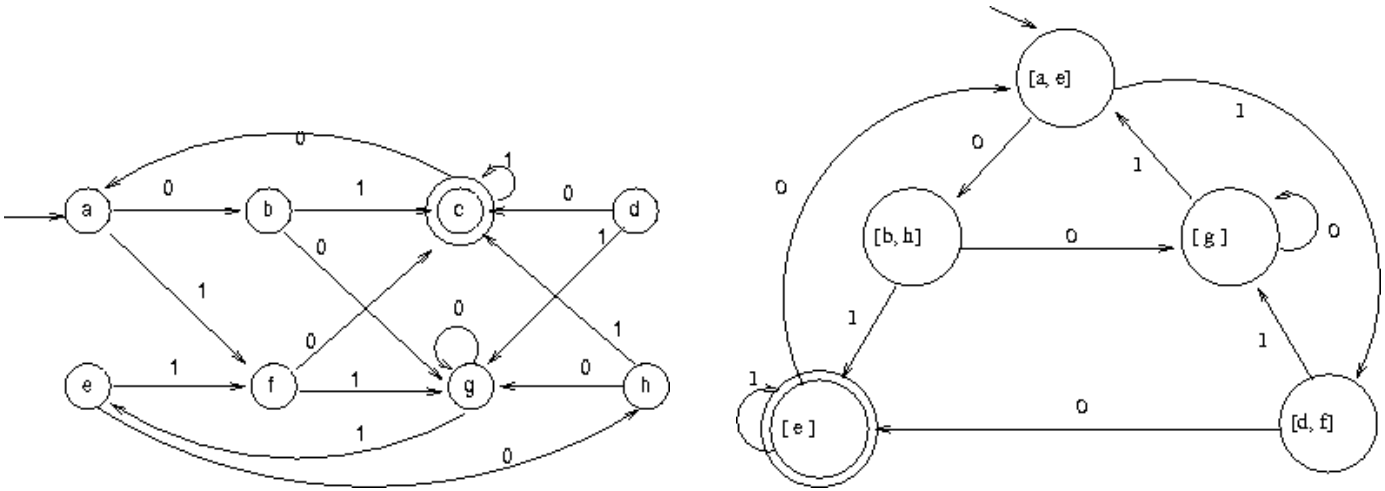
b	X_1						
c	X	X					
d	X_2	X_4	X				
e		X	X	X			
f	X	X_4	X		X		
g	X	X_3	X	X	X	X	
h	X		X	X_4	X	X_4	X_3
	a	**b**	**c**	**d**	**e**	**f**	**g**

Minimisation : algorithme via un autre exemple (suite)

➙ Les cases marquées représentent les paires d'états distinguables.
➙ Les états équivalents seront les cases restées vides :
➙ On a donc $\mathbf{a} \equiv \mathbf{e}$, $\mathbf{b} \equiv \mathbf{h}$ et $\mathbf{d} \equiv \mathbf{f}$.

• L'algorithme ci-dessus améliore la méthode de marquage mais il reste de complexité $O(n^2)$ (il n'est pas $nlog(n)$).
➙ On peut démontrer qu'il permet de construire un AEF minimum.

• Son application à l'AEF précédent donne l'AEF minimisé suivant :



Extension des AEFs : actions sémantiques

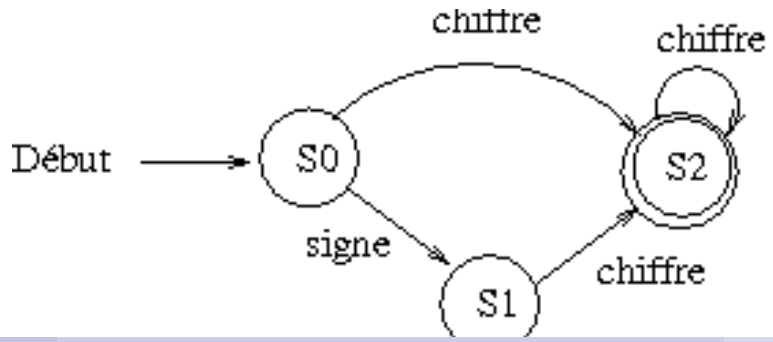
Ajout d'actions sémantiques aux AEFs

Les transitions dans un AEF peuvent être enrichies d'une **action sémantique** exécuté en cas d'application de la transition.

Pour l'exemple des entiers signés (les actions S_{ij} sont notées après les transitions, *atoi(char)* calcule la valeur d'un chiffre) :

init : *entier sg := 1, val := 0 ;*
$\delta(0, signe) = 1$ et S_{01} : si $(signe = \ '-')$ $sg := -1$;
$\delta(0, chiffre) = 2$ et S_{02} : $val := atoi(chiffre)$; **%chiffre reconnu**
$\delta(1, chiffre) = 2$ et S_{12} : $val := atoi(chiffre)$;
$\delta(2, chiffre) = 2$ et S_{22} : $val := val * 10 + atoi(chiffre)$;
output : S_f : $val := sg * val$;

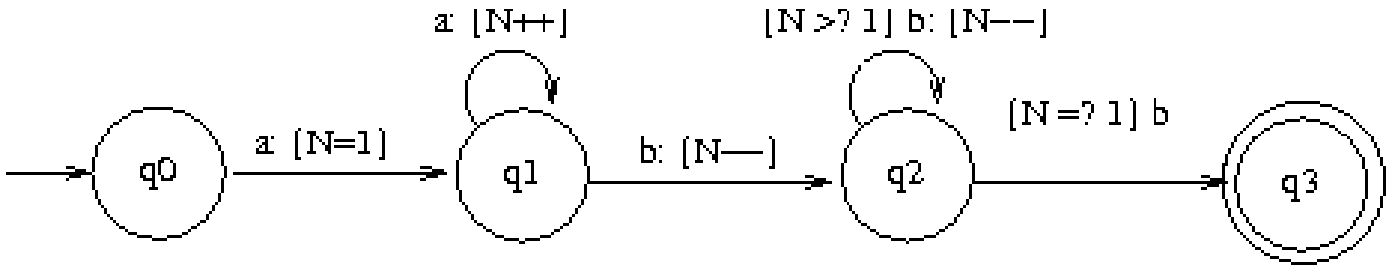
L'action finale S_f sera appliquée à la fin de la reconnaissance=sortie.



Extension des AEFs : exemple 2

Exemple 2 : il est bien connu (vu plus haut) qu'aucun AEF ne peut reconnaître le langage $a^n b^n, n \geq 1$.

➟ On peut cependant proposer un AEF étendu permettant de résoudre ce problème :



La forme générale d'une transition est $\{tests?\}\ input,\ output\ : \{actions\}$ où si $tests?$ est évalué à vrai et le symbole $input \in \Sigma$ est présent sur le flot d'entrée alors la transition aura lieu, le symbole $output$ éventuel (cas de transducteurs) est émis en sortie et $actions$ est exécutée.

➟ On remarque que parmi ces 4 composantes, seul $input \in \Sigma$ est exigé dans un AEF sans $\varepsilon - transition$; cette obligation est levée (signalée par ε) dans un AEFND avec $\varepsilon - transition$.

Extension des AEFs : exemple 2 (suite)

➙ Par exemple, sur l'état $q2$, si $N > 1$ et le symbole b est présent alors la transition a lieu et on décrémente la valeur de N.

• La compilation d'un AEF en un analyseur est assez simple et directe.

➙ Une table $Q \times \Sigma$ permet de décider de la transition à appliquer.

➙ Les AEF permettent de mieux visualiser les expressions (grammaires) régulières.

Parmi les outils, LEX permet la production d'analyseurs lexicaux en divers langages.

Les AEFs acceptés par LEX peuvent comporter des actions sémantiques.

A noter que l'émission des symboles en sortie fait l'objet d'une action sémantique.

Exercices

• Proposer une MEF sur l'alphabet {a,b} telle qu'elle puisse reconnaître et terminer toute séquence $[a, b]^+$ contenant un seul 'b'.

AEF non déterministe (exercice)

Exercice :
Donner un AEF non déterministe reconnaissant le langage $[\mathbf{a}, \mathbf{b}]^*\mathbf{abb}$ puis l'AEF déterministe équivalent (reconnaissant le même langage).

Applications : boot

Application des MEFs :

• **La séquence de boot d'une machine**

L'automate suivant modélise la séquence de démarrage d'une ordinateur.

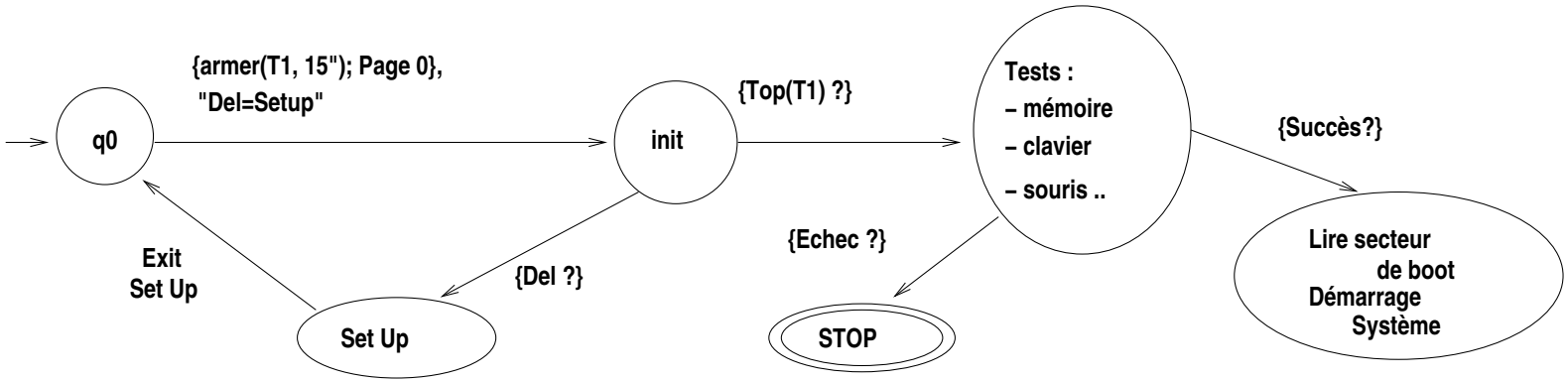


Fig.: Séquence simplifiée de démarrage (boot) d'un ordinateur

➙ Etudier cet MEF et compléter.

Applications

Application des MEFs : A compléter

• **Hyphonation** : coupage des mots en fin de ligne dans un texte (appelé également *Syllabification*).

➙ Règles :

➙ 1) Respecter les limites des mots : ne pas décuper *courts-circuits* → *court-scircuits* ;

➙ 2) Ne pas couper les syllabes : *al-pha-bet* → a-lpha-bet ;

➙ 3) Ne pas couper dès que possible : *al-pha-bet* → alph-a-bet, al-phab-et, alph-ab-et.

➙ Une syllabe = une expression régulière (ER).

AEF et Expressions régulières (ER)

Equivalence AEF et grammaires régulières :
Une transition de la forme $\delta(A, \alpha) = B$ devient $A \rightarrow \alpha\, B$.
Aussi, $A \rightarrow \alpha$ représente $\delta(A, \alpha) = q_f$.
Attention : une règle d'une grammaire régulière de la forme $A \rightarrow \alpha$ ne débouche pas toujours sur un état final (mais l'inverse est vrai).
➙ Par exemple, dans :

$$S \rightarrow Aa$$

$$A \rightarrow Ab$$

$$A \rightarrow b$$

➔ la règle $A \rightarrow b$ ne doit pas aboutir à un état final.
• Exemples d'AEF : métro, chèvre et loup, ...

Langages rationnels

Langages rationnels :
• **expressions rationnelles (regular expressions)** : $\epsilon, +, \ . \ , \ *$
• **automates finis non déterministes :** (Q, I, T, F)
• **automates finis déterministes :** (Q, I, T, F) avec F fonctionnelle et I singleton
• **déterminisation :** construction de l'ensemble puissance
• **minimalisation :** construction de Moore

➙ **propriétés :** lemme de l'étoile ; fermeture par complément, intersection, union ; décidabilité du langage vide, fini ou infini

Formalismes Grammaticaux

Σ : vocabulaire ou alphabet
ensemble non vide de symboles
e.g. {a,b,c}, caractères ASCII, lettres de l'alphabet latin (français), ...

Σ^* : ensemble Σ muni de la concaténations (le point '.')
i.e. ensemble des séquances finies de symboles de Σ
La concaténation :
- $\forall x \in \Sigma, \quad x \in \Sigma^*$
- $\forall x \in \Sigma, \forall y \in \Sigma^*, \quad x.y \in \Sigma^*$
- ε est l'éléments neutre de la concaténation :
  $\forall x \in \Sigma^*, \quad x.\varepsilon = \varepsilon.x = x$
- le '.' est un opérateur produit associatif :
  $a^n = aa...a$ et $a^0 = \varepsilon$
  $a, b \in \Sigma, ab \in \Sigma^*, aabab \in \Sigma^*$

Σ^* : ensemble de chaînes (mots) constructibles sur Σ, y compris la chaine vide ε

$\Sigma^+ \quad = \Sigma^* - \{\varepsilon\}$

Langages et Formalismes : pouvoir d'expression

Pouvoir d'expression : du moins évolué vers le plus évolué :

- Microcode, Langage Machine
- Langage assemblé (assembleur)
- Langage Fortran, C, Pascal, ADA, ...beaucoup de dialectes
  (Langages Objets ?, Fonctionnels)
- *L4G* : SQL, méta langages divers
- Langages (de programmation) Logiques, Avec cntraintes
- Langues Naturelles

Remarque (Exemple)

En français : *trouver les produits dont le prix > 100 Euros*

Prolog : $\mathcal{R}$ *(Nom,Prix, ...), Prix > 100.*

SQL : *select Nom from* $\mathcal{R}$ *where Prix > 100*

C :
```
while( ! feof(f)) {
  read(f,Enreg) ;
    if (Enreg.Prix > 100) output(Enreg.nom) ; }
```

Autres : *Compiler le code* $C \Rightarrow$ *Assembleur* $\Rightarrow$ *binaire ...*

Langages Formels

Un **langage (formel)** est une partie de Σ^*

Exemples :

- T=$\{$a,b$\}$, $L_T = \{a^n b^m | n > m \geq 0\}$ ou $L_T = \{a^n b^n | n \geq 0\}$
- D=$\{$0,1$\}$, $L_D = \{x \in D^* | \quad |x| \ mod \ 2 = 0\}$
- F=$\{$a,...,z, A, ..., Z$\}$, $L_F = \{x \in F^* | \quad |x| = 8\}$
- C=$\{$0, ...,9$\}$,

$$L_C = \{xzy, \ x, y \in C^*, z \in C | xzy = yzx\}$$

➙ N.B. : Pour ce dernier exemple, peut-on déduire $\{|x| = |y| \geq 0\}$?

Opérations sur les langages formels

Opérations sur les langages formels : Soit les langages $L_1, L_2 \subseteq \Sigma^*$

☒ **Opérations ensemblistes :**

1) Union de deux langages (somme)

$$L_1 \cup L_2 = L_1 + L_2 = \{x \in \Sigma^* | x \in L_1 \; ou \; x \in L_2\}$$

2) Intersection de langages

$$L_1 \cap L_2 = \{x \in \Sigma^* | x \in L_1 \; et \; x \in L_2\}$$

3) Complémentation ...

- Exemple : $L = \{a^n b^n | n \geq 0\}$

  $\bar{L} = \{a^n b^m | n \neq m\}$ L-barre = complément de L.

suit opérationse

☒ **Opérations spécifiques :**

1) Produit de deux langages (concaténation) :

$L_1 L_2 = \{xy | x \in L_1, y \in L_2\}$

$\Rightarrow$ opérateur associatif et non commutatif :

$$\Sigma\Sigma^* = \Sigma^* - \{\varepsilon\} = \Sigma^+$$

2) Fermeture d'un langage :

On utilise cette propriété pour prouver que certains langages ne sont pas de telle classe (voir plus loin).

$L^0 = \{\varepsilon\}$ et $L =$ le langage sur Σ^*

$L^n = LL^{n-1}, n \geq 1$

$L^* = \bigcup_{n \geq 0} L^n = \{\varepsilon\} \cup L^1 \cup L^2 \cup L^3 ...$

$L^+ = LL^* = L^* L$

Voir aussi les propriétés des grammaires et langages.

Expressions Régulières (ER)

Expressions régulières sur un alphabet Σ :

$\emptyset$ est une ER décrivant $\emptyset$

ε est une ER décrivant ε

$a \in \Sigma$ est une ER décrivant {a}

a et b deux ER sur Σ $(soient\ L_a, L_b)$:

- $(a + b)$ est une ER décrivant $L_a \cup L_b$
- (ab) est une ER décrivant $L_a L_b$
- (a^*) est une ER décrivant L_a^*

N.B. $(a^+) = a(a^*)$ est un raccourcie décrivant $L_a^+ = L_a L_a^*$

Exemple

Exemple :

$l = \{A..Z, a..z\}, \quad c = \{0..9\}$

➙ N.B. : on peut écrire $c = (0 + 1 + .. + 9)$

c^+ est une ER décrivant les constantes entières

$l(l + c)^*$ est une ER décrivant les identificateurs

Priorités des opérateurs :

* $<$ produit (concaténation) $<$ somme (union)

Simplifications et calculs algébriques

Simplifications et calculs algébriques :

$r+s = s+r$	$r+(s+t) = (r+s)+t$
$(rs)t=r(st)$	$r(s+t)=rs+rt$
$(s+t)r = sr+tr$	$r^*=(r+\varepsilon)^*$
$r^{*^*} = r^*$	$r^+ = r\, r^*$ ...

- ➙ Le formalisme ER est apart (des grammaires algébriques).
- ➙ A ne pas confodre avec les grammaires régulières.

- ➙ Il permet de décrire les langages régulièrs et finis.

L'outil Lex accepte les expressions régulières (étendues)

Lex

Lex utilise une forme étendue des expressions régulières.
Un expression de base est de l'une des formes suivantes :

Formes de base :

x	le caractère 'x'
.	un caractère quelconque (mais pas eof)
$<< EOF >>$	la fin d'input
[xyz]	un de ces 3 caractères
[^ aeiou]	aucun voyelle
[a-z]	un caractère entre 'a' ... 'z'
[^ a-z]	aucun minuscule
[a-zA-Z]	une des lettres
[0-9]	un chiffre
"une chaine"	la chaîne de caractères

Lex (suite)

Lex : Les Formes Rationnelles :

r*	0 ou plus expression régulière r
r+	1 ou plus expression régulière r
r ?	expression régulière r optionnelle (0 ou 1 fois)
r{2,5}	de 2 à 5 occurrences de l'expression r
r{2,}	de 2 à plus occurrences de l'expression r
r{2}	exactement 2 occurrences de l'expression r
{nom}	une expansion appelée 'nom' (voir ci-dessous)
\x	si x est l'un des caractères $\{a,b,f,n,r,t,v\}$ alors ce sera l'interprétation ANSI de ce caractère (e.g. \n pour le retour chariot) sinon ce sera le caractère x (e.g $\backslash*$ pour l'opérateur *, $\backslash\backslash$ pour $\backslash$, etc.).
(r)	même chose que r. On utilise les () pour la priorité.
rs	l'expression régulière r suivie de l'expression s.
r | s	l'expression régulière r ou de l'expression s.

Lex (suite) (suite)

r / s l'expression régulière r seulement si elle est suivie de s. Seule r est consommée en entrée.

^ r l'expression régulière r si elle est en début d'une ligne.

r$ l'expression régulière r si elle est à la fin d'une ligne.

Remarques :

• On peut définir des noms, par exemple :

$$DIGITS[0-9]$$

et des les utiliser entre { } :

$$ID[a-zA-Z_][a-zA-Z0-9_]*$$

$$DIGITS[0-9]+$$

$$DIGITS\{printf(\text{"}unnombre :$$

$$val = atoi(yytext)); \}$$

Lex (suite) (suite)

$$ID\{printf(\text{"}unident : \text{"}); ECHO;$$

$$add_t o_t ab_s ym(tab, yytext); \}$$

• Les caractères spéciaux doivent être précédés de \, par exemple \" pour les guillemets.
• La précédence : '*' et '+' précèdent '?'.
• Les caractères spéciaux : { *a,b,f,n,r,t,v* }, \0 (nul ASCII), \0123 (code octal 123), \x2a (code hexa 2a),
• Il existe d'autres possibilités. Consulter les documents de Lex. *man lex* donnera également des explications utiles.
• Voir également les documents pour les actions que l'on peut mettre en partie droite d'une reconnaissance lexicale.

Les Grammaires (II)

- Une grammaire est un quadruplet G $= (\Sigma, V_N, \mathcal{S}, \mathcal{P})$ où :
  - Σ est un alphabet fini de "terminaux",
  - V_N est un ensemble fini de symboles "non terminaux", avec $\Sigma \cap V_N = \emptyset,\ V = \Sigma \cup V_N$;
  - $\mathcal{S}$ est un non terminal distingué nommé le "symbole de départ" (start symbol)
  - $\mathcal{P}$ est une relation finie sur $V_N^+ \times \mathrm{V}^*$ qui définit les "règles" (de "production") de la grammaire.

Une règle $(\alpha, \beta) \in P$ s'écrit aussi $\alpha \rightarrow \beta$.

Exemple (Grammaire G) :

$$
\begin{array}{ll}
< Phrase > & \rightarrow < Groupe\_nominal >< Groupe\_verbal > . \\
< Groupe\_nominal > & \rightarrow < Determinant >< Nom. > \\
< Groupe\_verbal > & \rightarrow < Verbe\_nt > . \\
< Groupe\_verbal > & \rightarrow < Verbe\_t >< Groupe\_nominal > . \\
< Verbe > & \rightarrow\ 'mange' |\ 'danse' | ... \\
< Nom > & \rightarrow\ 'chien' |\ 'soupe' | ...
\end{array}
$$

Définitions

Définitions :
On peut appliquer la règle $\alpha \rightarrow \beta$ à n'importe quelle séquence $x\alpha y$ pour obtenir $x\beta y$.
On écrit alors $x\alpha y \Rightarrow x\beta y$.
On dira que l'on a *réécrit* $x\alpha y$ en $x\beta y$ (à l'aide de la règle $\alpha \rightarrow \beta$).
On écrira $\omega \Rightarrow^* z$ s'il existe $\omega 1 \Rightarrow \omega 2... \Rightarrow \omega n$, $n \geq 1$,avec
$\omega = \omega 1 \Rightarrow \omega 2... \Rightarrow \omega n = z$.
L'opérateur $\Rightarrow^*$ (appelée une *dérivation*) est la fermeture réflexo-transitive de l'opérateur $\Rightarrow$.
Le *langage engendré* par une grammaire G est

$$L(G) = \{\omega | S \Rightarrow^* \omega\}$$

BNF

Backus Naur Form (BNF)

La **forme normale de Backus** est une extension du formalisme des grammaires hors contextes qui n'ajoute pas de pouvoir expressif (on ne définit pas une classe plus large de langages), mais qui permet des définitions plus concises.
A la structure des règles on ajoute les notations suivantes :

• **Opérateur étoile de Kleene** :

On peut écrire $A \rightarrow \alpha\beta^*\gamma$ à la place de :

$$A \rightarrow \alpha B\gamma$$
$$B \rightarrow \beta B$$
$$B \rightarrow \epsilon.$$

• **Optionel :**

on peut écrire $A \rightarrow \alpha[\beta]\gamma$ à la place de :

$$A \rightarrow \alpha B\gamma$$
$$B \rightarrow \beta$$
$$B \rightarrow \epsilon.$$

BNF (suite)

• **Choix :**
on peut écrire $A \rightarrow \alpha_1|\alpha_2...|\alpha_n$ à la place de :

$A \rightarrow \alpha_1$
$A \rightarrow \alpha_1$
...
$A \rightarrow \alpha_n$

Nota Bene : en s'inspirant des expressions régulières (cf. Lex), parfois :

β^* est noté $\{\beta\}$ ou encore $[\beta]^*$
$\beta(\beta)^*$ est noté $(\beta)^+$ ou encore $[\beta]^+$.

• Attention à ne pas confondre les symboles BNF avec les terminaux de Σ.
➙ Pour éviter ce cas, les éléments $s \in \Sigma$ sont notés par "s" (sauf s'il n'y a a pas d'ambiguïté).

Exemples

$X \to AC.$	$X \to BC.$	donne	$X \to (A\ ;\ B)\ C.$
$A \to BC.$	$A \to B.$	donne	$A \to B(C\ ;\ \varepsilon)$
$B \to CB.$	$B \to \varepsilon.$	donne	$B \to \{C\}$ ou $(C)^*$ ou $[C]^*$
$C \to aC.$	$C \to a.$	donne	$B \to (a)^+$ ou $[a]^+$

Exemple (suite) : BNF spécifié en BNF

$$
\begin{array}{lcl}
Grammaire & \longrightarrow & Regles \\
Regles & \longrightarrow & Regle \quad Regles \mid \varepsilon. \\
Regle & \longrightarrow & SymboleDeV_N \text{ ” :: ” } Alternants \text{ ”.”}. \\
Alternants & \longrightarrow & Sequence\ Sommes\ . \\
Sequence & \longrightarrow & Element \quad Produit\ . \\
Sommes & \longrightarrow & \text{”;”}\ Alternants \mid \varepsilon. \\
Produit & \longrightarrow & \text{”,”}\ Sequence \mid \varepsilon. \\
Element & \longrightarrow & \text{”(”}\ Alternants\ \text{”)”}. \\
Element & \longrightarrow & \text{”}\{\text{”}\ Alternants\ \text{”}\}\text{”}. \\
Element & \longrightarrow & \text{”[”}\ Alternants\ \text{”]”}. \\
Element & \longrightarrow & \text{”}\varepsilon\text{”}. \\
Element & \longrightarrow & SymboleDe\Sigma \mid SymboleDeV_N. \\
SymboleDeV_N & \longrightarrow & \text{élément de } V_N. \\
SymboleDe\Sigma & \longrightarrow & \text{élément de } \Sigma.
\end{array}
$$

Classification de Chomsky

Classification de Chomsky

Graduation (et restrictions) sur la forme des productions.

➙ Pouvoir de description, propriétés formelles.

➙ On classifie les grammaires selon la forme de $\mathcal{P}$ (productions) :

type 4 (langages finis) :

Règles de la forme $A \longrightarrow b$, $A \in V_N, b \in \Sigma$.

type 3 (rationnelle, linéaire, régulier, Kleen) :

Règles de la forme $A \longrightarrow bB$ ou $A \longrightarrow b$, $A, B \in V_N, b \in \Sigma$.

➙ N.B. : on distingue linéaire à *gauche* ou à *droite* :

- linéaire à gauche (cf. ci-dessus)
- linéaire à droite : règle de la forme

$$A \longrightarrow Bb \text{ ou } A \longrightarrow b,\ A, B \in V_N, b \in \Sigma.$$

type 2 (hors contexte) :

règle de la forme $A \longrightarrow \beta$, $A \in V_N, \beta \in V^*$

Classification de Chomsky (suite)

type 1 (contextuelle) :

pour toute règle $\alpha \rightarrow \beta$, $|\alpha| \leqslant |\beta|$, $\alpha, \beta \in V^+$

type 0 : (grammaire)

$\mathcal{P}$ quelconque ($\alpha \rightarrow \beta$, $\alpha \in V^+, \beta \in V^*$)

Pour la définition des langages de programmation, on utilise les grammaires de type 2 (hors contexte = CFG).

Exemples

Notation : distinction terminaux et non terminaux (en cas d'ambiguïté dans la notation),

on note $A \rightarrow C\ 'a' B$ (terminaux entre ' ou ")

ou $A \rightarrow < C > a < B >$ (non terminaux entre <>)

ou même $A \rightarrow < C > "a" < B >$

- **type 4 : langages finis**

$$amphi \rightarrow \text{'A1'} \mid \text{'A2'} \mid \text{'A3'} \mid \text{'A201'} \mid \text{'A202'}\ .$$

- **type 3 : langages réguliers**

Exemple : identificateur avec la lettre 'a' et le chiffre '0' :

$$\begin{aligned} &< identificateur > &&\rightarrow\ a < chiffres\_ou\_lettres > \\ &< chiffres\_ou\_lettres > &&\rightarrow\ a < chiffres\_ou\_lettres >. \\ &< chiffres\_ou\_lettres > &&\rightarrow\ '0' < chiffres\_ou\_lettres >. \end{aligned}$$

Exemples (suite)

- **type 2 : langages Hors Contextes**
  - ➙ Exemple-1 : L=$\{a^n b^n | n \geq 1\}$:

$$
\begin{array}{lll}
S & \rightarrow & 'a'\ E\ 'b'. \\
E & \rightarrow & 'a'\ E\ 'b'. \\
E & \rightarrow & \varepsilon.
\end{array}
$$

  - ➙ Exemple-2 : L=$\{a^n b^m | n, m \geq 0\}$

$$
\begin{array}{lll}
S & \rightarrow & A\ B. \\
A & \rightarrow & 'a'\ A. \\
A & \rightarrow & \varepsilon. \\
B & \rightarrow & 'b'\ B. \\
B & \rightarrow & \varepsilon.
\end{array}
$$

Exemples (suite)

• **type 1 : langages Contextuels**

Exemple : L= $\{a^n b^n c^n | n \geq 1\}$ avec $\Sigma = \{a, b, c\}$:

(1) $S \rightarrow a\ S\ B\ C.$
(2) $S \rightarrow a\ B\ C.$
(3) $C\ B \rightarrow B\ C.$
(4) $a\ B \rightarrow a\ b.$
(5) $b\ B \rightarrow b\ b.$
(6) $b\ C \rightarrow b\ c.$
(7) $c\ C \rightarrow c\ c.$

➙ Un exemple de dérivation de $a^2 b^2 c^2 \in L$:

$\underline{S} \Rightarrow_1 a\underline{S}BC \Rightarrow_2 a\ \underline{a\ B}CBC \Rightarrow_4 a\ a\ b\ \underline{CB}C$
$\Rightarrow_3 a\ a\ \underline{b\ B}CC \Rightarrow_5 a\ a\ b\ \underline{b\ C}C$
$\Rightarrow_6 a\ a\ b\ b\ \underline{c\ C} \Rightarrow_7 a\ a\ b\ b\ c\ c$

Exemples (suite)

- **type 0 : langages formels**
  ➙ Aucune restriction sur la taille de la partie gauche (ou droite) de la règle (de part et d'autre du symbole $\rightarrow$).
  ➙ Formalisme non exploitable !

Exemple (semblable au type 1) : L= $\{a^i | i = 2^n, n \geq 1\}$ avec $\Sigma = \{a\}$:

$$
\begin{array}{ll}
S & \rightarrow a\ C\ aB. \\
C\ a & \rightarrow a\ a\ C. \\
C\ B & \rightarrow D\ B. \\
C\ B & \rightarrow E. \\
a\ D & \rightarrow D\ a. \\
A\ D & \rightarrow A\ C. \\
a\ E & \rightarrow E\ a. \\
A\ E & \rightarrow \varepsilon.
\end{array}
$$

Remarque : les grammaire de type 0 sont équivalentes à la machine de Turing.

Quelques outils de spécification et de modélisation Formelles

- Graphique :
  - AEF / MEF (voir outils asf, DCG, ...)
  - Réseau de Pétri
  - OMG, UML (et ses variantes)
  - SADT, Merise
- Algébriques (utilisent des formes diverses des TDAs) :
  - Z et ses variantes : (Z+UML, SAZ, ...)
  - ASM (abstract state machine) $\Longrightarrow$ Technique
  - CafeOBJ (free) : langage (sous lisp), Spécif alg exécutable (voir sur G5)
  - CASL (algébrique), commercialisé, sous Win, preuve, ...
  - DC : Duration calculi (logique d'intervalle et temporelle)
  - RAISE/RSL : langage de spécif
  - VDM (le 1er 1974)
  - TLA/TLA+ : linear Tomporal Logic (TR ?)
  - B / event-B : méthode de modélisation , state basesed
- Logique

Quelques outils de spécification et de modélisation Formelles (suite)

Propositionnelle & Prédicats
Modale, Ternaires, etc.

- NB : AltaRica est aussi un formalisme utilisé dans l'ex ascenseur (lift.pdf)

Quelques Références Bibliographiques :

- *Introduction to Automata Theory, Languages, and Computation*. J.E.Hopcroft & J.D.Ullman.
- *Mathématiques Discrètes.* Notes personnelles.
- *The Complexity of Theorem Proving Procedure.* S.A. Cook.
- *Compilers.* J.E.Hopcroft & J.D.Ullman.
- *Finite State Languages.* N. Chomsky & G.A. Miller
- *Computational Logic.* D. Warren.
- *Notes Personnelles*

...