Alphabet, mots et langage
Automates finis déterministes
Réduction des automates
Automates finis non déterministes
Transitions instantanées
Langages reconnaissables

# Automates

## Cours d'Informatique

Denis MONASSE

[1]Classes Préparatoires aux Grandes Ecoles MP*
Lycée Louis le Grand, Paris

Octobre 2007

# Outline

1. Alphabet, mots et langage
2. Automates finis déterministes
   - Automates finis
   - Processus de reconnaissance par un automate
   - Une implémentation simple des AFD
   - Reconnaissance par un AFD
   - Diagramme d'un automate
3. Réduction des automates
   - Accessibilité
   - Graphes
   - Type des graphes
   - Exploration des graphes
   - Exploration en profondeur

## Définition

*On appelle alphabet (ou ensemble des caractères) tout ensemble fini (non vide) A.*

## Remarque

Dans nos applications, l'ensemble *A* sera presque toujours l'ensemble des caractères accessibles sur notre ordinateur (lettres minuscules, lettres majuscules, chiffres, symboles et caractères spéciaux comme l'espace ou le retour à la ligne), mais rien n'empêche d'imaginer d'autres ensembles.

## Définition

*On appelle mot sur A toute suite finie (éventuellement vide) de caractères de l'alphabet A. La longueur d'un mot est le nombre de ses caractères. L'ensemble des mots sur A sera noté $A^*$.*

## Définition

*On appelle langage sur A toute partie L de $A^*$.*

## Remarque

Le but de notre machine est d'analyser un mot de $A^*$ pour savoir si celui-ci appartient ou non à $L$.

## Exemple

Soit $A$ l'ensemble des caractères accessibles sur l'ordinateur. On peut par exemple envisager les langages suivants :

- les représentations décimales de nombres entiers : suites de caractères commençant éventuellement par un symbole $+$ ou $-$ suivi uniquement de chiffres de 0 à 9
- les représentations de nombres décimaux : suites de caractères commençant éventuellement par un symbole $+$ ou $-$ suivi de chiffres de 0 à 9, puis d'un point décimal (pour les anglo-saxons), puis de chiffres de 0 à 9
- les mots composés uniquement de caractères alphabétiques commençant par un *i*, terminés par un *e* et comportant les caractères consécutifs *m,a*
- les mots composés uniquement de caractères alphabétiques et ne contenant pas de *e* (voir G. Perec)

# Résumé

2 Automates finis déterministes
- Automates finis
- Processus de reconnaissance par un automate
- Une implémentation simple des AFD
- Reconnaissance par un AFD
- Diagramme d'un automate

$Q$ l'ensemble des états possibles de la mémoire de notre ordinateur ; cette mémoire étant finie, l'ensemble des états possibles est également fini.

Application $\delta : Q \times A \rightarrow Q$ appelée fonction de transition qui à un état $p$ et un caractère $c$ de l'alphabet $A$ associe $q = \delta(p, c)$ qui est le nouvel état après la lecture du caractère $c$.

Avant la lecture du premier caractère du mot à reconnaître, notre machine sera dans un état connu $p_0 \in Q$ appelé état de départ.

Après la lecture du dernier caractère, la mémoire de l'ordinateur sera dans un état $q$ qui doit déterminer si le mot appartient ou non au langage : certains états seront privilégiés, les états finaux ou acceptants.

Modélisation suivante du travail de reconnaissance (linéaire, c'est à dire sans retour en arrière) d'un mot par un ordinateur :

## Définition

*On appelle automate fini (ou plus simplement automate) la donnée*

- *d'un alphabet $A$*
- *d'un ensemble fini (et non vide) $Q$ appelé ensemble des états de l'automate*
- *d'un élément $p_0$ de $Q$ appelé l'état initial ou état de départ*
- *d'une partie $F$ de $Q$ appelée l'ensemble des états finaux ou états acceptants*
- *d'une application $\delta : Q \times A \to Q$ appelée fonction de transition de l'automate*

## Remarque

Il peut arriver que lorsque la machine est dans un état $p$, on ne veuille pas accepter un caractère donné $c$ ; cela revient à admettre que la fonction de transition n'est pas définie sur $Q \times A$ tout entier, mais simplement sur une partie $\Delta$ de $Q \times A$.
On construit alors facilement un automate équivalent (c'est à dire qui reconnaîtra exactement le même langage) de la manière suivante :

## Remarque

on choisit un élément $r$ n'appartenant pas à $Q$ (un *rebut*) et l'on pose $Q' = Q \cup \{r\}$ ; on prolonge $\delta : \Delta \to Q$ en $\delta' : Q' \times A \to Q'$ de la manière suivante :

- $\delta'(p, a) = \delta(p, a)$ si $(p, a) \in \Delta$
- $\delta'(p, a) = r$ si $(p, a) \notin \Delta$ (on envoie au rebut les valeurs non définies)
- $\delta'(r, a) = r$ pour tout $a \in A$ (quand on est au rebut, on n'en bouge plus)

On conserve l'état initial $p_0$ et les états finaux $F$ (si bien que le rebut n'est pas un état final). On obtient ainsi un nouvel automate dont la fonction de transition est définie partout et qui effectuera le même *travail* que l'automate initial.

Tenant compte de la remarque précédente, nous poserons la définition :

## Définition

*Soit $\mathcal{A}$ un automate ; on appelle rebut tout état non final $r$ tel que $\delta(r, a) = r$ pour tout $a \in A$*

# Résumé

2 Automates finis déterministes
- Automates finis
- Processus de reconnaissance par un automate
- Une implémentation simple des AFD
- Reconnaissance par un AFD
- Diagramme d'un automate

Automate $\mathcal{A} = (A, Q, p_0, F, \delta)$ ; initialement, notre automate est dans l'état $p_0$.

Avant la lecture du $n$-ième caractère, c'est à dire lorsqu'il a lu les $n-1$ premiers caractères $c_1, \ldots, c_{n-1}$ ou encore lorsqu'il a lu le mot $c_1 \ldots c_{n-1}$, l'automate est dans l'état $p$.

Après la lecture du caractère $c_n$, c'est à dire lorsqu'il a lu le mot $c_1 \ldots c_n$, il passe dans l'état $q = \delta(p, c_n)$.

A tout mot $w = c_1 \ldots c_n$ est associé un état $q$ qui est l'état de l'automate partant de l'état initial $p_0$ une fois qu'il a lu les caractères $c_1, \ldots, c_n$. Nous noterons cet état $\overline{\delta}(p_0, w)$.

Le processus de reconnaissance d'un mot $w$ est simplement le test pour savoir si $\overline{\delta}(p_0, w)$ appartient ou non à l'ensemble $F$ des états finaux.

Ce que nous venons de faire en partant de l'état initial $p_0$ peut naturellement être étendu en partant d'un autre état $p$ : si on suppose qu'à un instant donné l'automate est dans un état $p$ et que l'on procède ensuite à la lecture d'un mot $w = c_1 \dots c_n$, l'automate passera alors dans un état $q = \overline{\delta}(p, w)$. Ceci revient à définir par récurrence une fonction $\overline{\delta} : Q \times A^* \to Q$ de la manière suivante

- $\forall p \in Q,\ \overline{\delta}(p, \varepsilon) = p$ (la lecture du mot vide de modifie pas l'état de l'automate)
- si $n \geq 1$, $\overline{\delta}(p, c_1 \dots c_n) = \delta(\overline{\delta}(p, c_1 \dots c_{n-1}), c_n)$ (en partant de l'état $p$, pour lire le mot $c_1 \dots c_n$ on lit d'abord le mot $c_1 \dots c_{n-1}$ pour se retrouver dans l'état $\overline{\delta}(p, c_1 \dots c_{n-1})$, puis le caractère $c_n$ fait passer dans l'état $\delta(\overline{\delta}(p, c_1 \dots c_{n-1}), c_n)$)

## Définition

*On appelle langage reconnu par l'automate $\mathcal{A}$ l'ensemble des mots $w \in A^*$ tels que $\overline{\delta}(p_0, w)$ est un état final.*

# Résumé

2 Automates finis déterministes
- Automates finis
- Processus de reconnaissance par un automate
- Une implémentation simple des AFD
- Reconnaissance par un AFD
- Diagramme d'un automate

Les noms précis des états n'ont évidemment aucune importance, et tout ensemble $Q'$ qui est en bijection avec $Q$ peut servir à construire un automate $\mathcal{A}'$ équivalent à l'automate $\mathcal{A}$ ;

Nous pouvons, sans nuire à la généralité, supposer que $Q$ est une partie de l'ensemble $\mathbb{N}$ des entiers naturels, et donc en Caml, supposer que les états sont de type `int`, numérotés de 0 à $n-1$.

Comme dans beaucoup d'applications, nous supposerons que les éléments de $A$ sont des caractères standards, autrement dit sont de type `char`.

La fonction de transition $\delta$ peut être représentée par les applications partielles $\delta_p : a \mapsto \delta(p, a)$

Chacune de ces application partielle sera représentée par son graphe, c'est à dire comme une liste d'éléments de type `caractere*etat` dans laquelle nous stockerons des objets du type $(a, \delta(p, a))$.

La recherche de la valeur de $\delta(p, a)$ peut alors se faire en Caml à l'aide de la fonction `assoc` ; étant donné une liste de couples de type `('a,'b)`, cette fonction associe à tout élément $a$ de type `'a` l'élément $b$ de type `'b` tel que $(a, b)$ est le premier couple de la liste contenant $a$ comme premier élément ; cette fonction déclenche l'exception `Not_found` si aucun couple de la liste n'a comme première composante $a$.

Chaque état $p$ sera donc représenté par un booléen indiquant s'il est final ou non et par sa fonction $\delta_p$ sous forme de liste d'association, l'automate lui-même étant représenté par un tableau d'états (par convention, l'état initial aura l'indice 0), ce qui conduit au typage

```
type etat = {final: bool; transitions: (char*int) list};;
```

et `type AFD = AFD of etat vect;;`

# Résumé

2 Automates finis déterministes
- Automates finis
- Processus de reconnaissance par un automate
- Une implémentation simple des AFD
- Reconnaissance par un AFD
- Diagramme d'un automate

## Caml

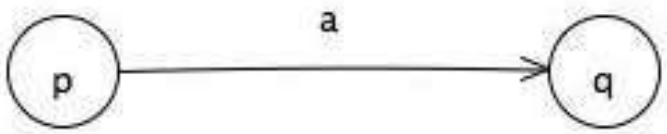
```
type etat = {final: bool; transitions: (char*int) list};;

type AFD = AFD of etat vect;;

let reconnait (AFD v) w =
    let delta q c = assoc c v.(q).transitions
    in let rec delta_barre q = function
           | [] -> q
           | t::r -> delta_barre (delta q t) r
in
try
    let dernier = delta_barre 0 w
    in v.(dernier).final
with Not_found -> false;;
```

# Résumé

2 Automates finis déterministes
- Automates finis
- Processus de reconnaissance par un automate
- Une implémentation simple des AFD
- Reconnaissance par un AFD
- Diagramme d'un automate

Etant donné un automate $\mathcal{A} = (A, Q, p_0, F, \delta)$, il est d'usage d'adopter une représentation dynamique des transitions de l'automate. C'est ainsi que la notation $p \xrightarrow{a} q$ dénotera l'égalité $q = \delta(p, a)$ : la lecture du caractère $a \in A$ fait passer de l'état $p$ à l'état $q$.

On peut même représenter tout l'automate par un diagramme sur lequel figureront tous les états de la machine (il est d'usage de les placer à l'intérieur de cercles) ; ces états seront reliés par des flèches étiquetées par les éléments de l'alphabet comme ci dessus, la relation $q = \delta(p, a)$ étant symbolisée par le sous-diagramme



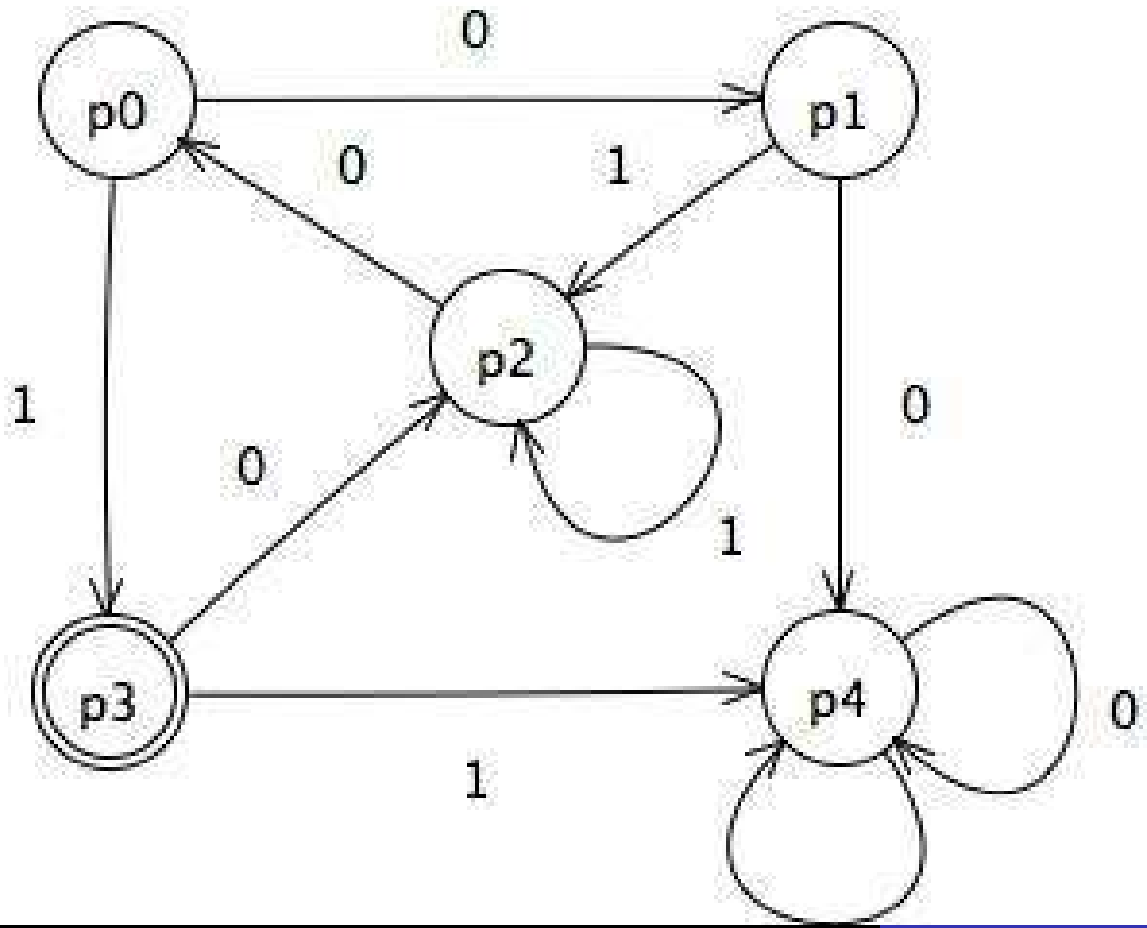

L'état initial est repéré par une flèche entrante provenant de l'extérieur et les états finaux sont distingués par un double cerclage

Considérons par exemple l'automate suivant :

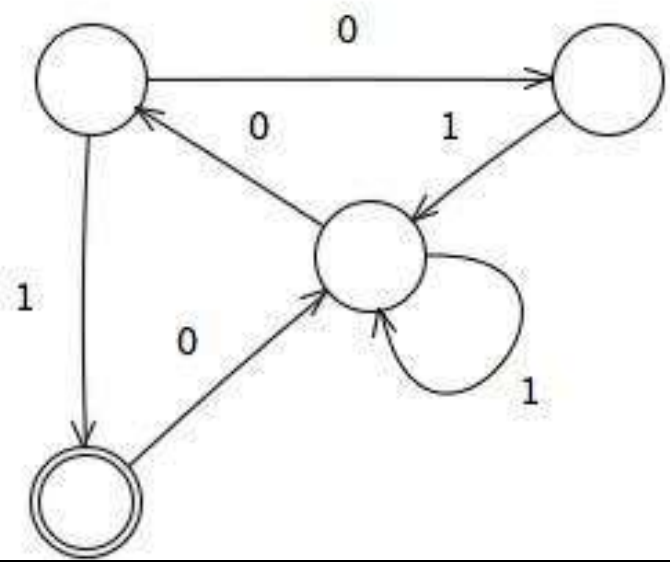
- l'alphabet est $\{0, 1\}$
- $Q = p_0, p_1, p_2, p_3, p_4$
- l'état initial est $p_0$
- le seul état final est $p_3$
- $p_4$ est un rebut
- la fonction de transition est définie par

$$\begin{array}{llll} \delta(p_0, 0) = p_1, & \delta(p_0, 1) = p_3, & \delta(p_1, 0) = p_4, & \delta(p_1, 1) = p_2, \\ \delta(p_2, 0) = p_0, & \delta(p_2, 1) = p_2, & \delta(p_3, 0) = p_2, & \delta(p_3, 1) = p_4, \\ \delta(p_4, 0) = \delta(p_4, 1) = p_4 & & & \end{array}$$

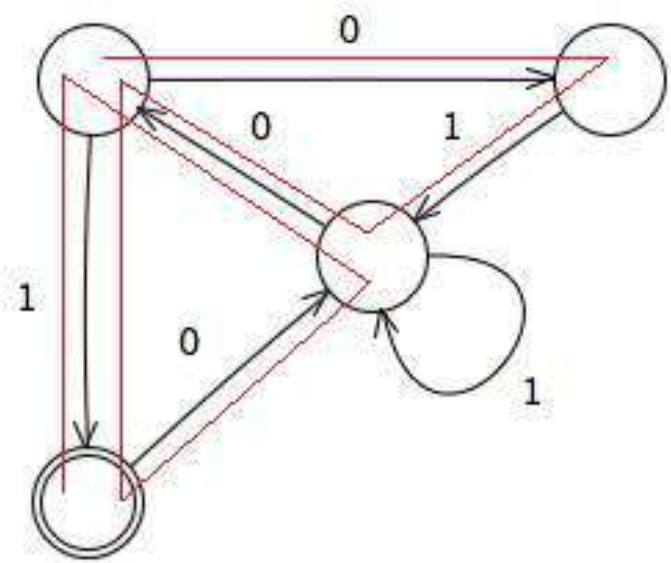
# Il sera symbolisé par le diagramme



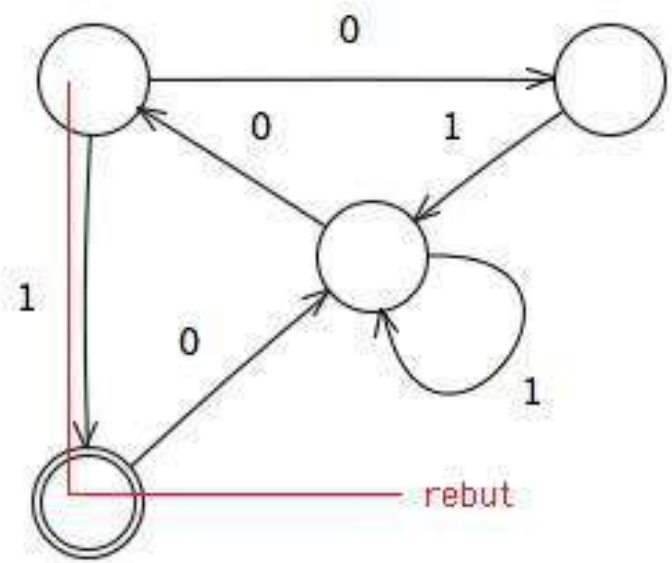
Pour simplifier le diagramme, nous pourrons par la suite ne pas faire figurer dans les cercles les noms des états (ceux-ci n'ayant aucune importance théorique). D'autre part nous conviendrons de ne pas faire figurer les rebuts : nous supposerons comme plus haut que toute transition non définie aboutit à un rebut (il est clair que tout automate ayant plusieurs rebuts est équivalent à l'automate obtenu en identifiant tous ces rebuts en un seul). Ceci conduit à un diagramme simplifié



Sur un tel diagramme, on peut suivre le chemin parcouru lors de la tentative de reconnaissance d'un mot. Par exemple, pour le mot 01010101 on a le chemin suivant (on voit que le mot est reconnu)



Pour un mot de la forme 11 . . . on a le chemin (on voit que le mot n'est pas reconnu)



# Résumé

3 Réduction des automates
- Accessibilité
- Graphes
- Type des graphes
- Exploration des graphes
- Exploration en profondeur
- Exploration en largeur
- Exploration l'aide de piles
- Emonder un graphe

## Définition

*Si $p$ et $p'$ sont deux états d'un automate $\mathcal{A}$ ; on appelle chemin de $p$ à $p'$ toute suite $(p_0, p_1, \ldots, p_n)$ d'états de $\mathcal{A}$ telle que, pour tout $i \in [1, n]$, il existe une transition (soit étiquetée par un élément de A, soit instantanée) de $p_{i-1}$ à $p_i$. On dira qu'un chemin est réussi s'il aboutit à un état final.*

## Définition

*Soit $\mathcal{A}$ un automate d'état initial $p_0$ dont l'ensemble des états finals est $F$. On dit qu'un état $p$ de $\mathcal{A}$ est accessible (resp. coaccessible) s'il existe un chemin de $p_0$ à $p$ (resp. de $p$ à un état final). On dit que $\mathcal{A}$ est accessible (resp. coaccessible) si tout état de $\mathcal{A}$ est accessible (resp. coaccessible). On dit que l'automate est émondé s'il est à la fois accessible et coaccessible.*

## Remarque

Il est clair qu'un état non accessible d'un automate ne fera jamais partie du processus de reconnaissance d'un mot sur l'alphabet $A$ et peut donc être supprimé de l'automate sans changer le langage reconnu. En ce qui concerne un état non coaccessible $p$, si le processus de reconnaissance d'un mot passe par $p$, ce processus ne peut aboutir à la reconnaissance du mot, puisqu'aucun chemin ne va de $p$ à un état final ; les états non coaccessibles peuvent donc être avantageusement remplacés par un unique rebut. Tout automate peut donc être remplacé par un automate émondé, sans changer le langage reconnu.

Il reste à donner un algorithme qui permette, à partir d'un automate, de construire l'ensemble des états accessibles et celui des états coaccessibles. Ensuite la construction de l'automate émondé reconnaissant le même langage sera élémentaire. Soit donc $\mathcal{A}$ un automate, $Q$ l'ensemble de ses états, $p_0$ son état initial, $F$ l'ensemble des états finals. Définissons deux suites $(P_i)_{i\in\mathbb{N}}$ et $(Q_i)_{i\in\mathbb{N}}$ de parties de $P$ par

$$\begin{aligned} P_0 &= \{p_0\},\ Q_0 = F \\ P_{i+1} &= P_i \cup \{p \in Q \mid \text{il existe une transition d'un élément de } P_i \text{ à } p\} \\ Q_{i+1} &= Q_i \cup \{p \in Q \mid \text{il existe une transition de } p \text{ à un élément de } Q_i\} \end{aligned}$$

On montre immédiatement par récurrence que $P_n$ est exactement l'ensemble des états tels qu'il existe un chemin de longueur au plus $n$ de $p_0$ à $p$ et que $Q_n$ est l'ensemble des états $p$ tels qu'il existe un chemin de longueur au plus $n$ de $p$ à un état final, si bien que

## Proposition

*L'ensemble des états accessibles de $\mathcal{A}$ est $\bigcup_{n\in\mathbb{N}} P_n$ et l'ensemble des états coaccessibles de $\mathcal{A}$ est $\bigcup_{n\in\mathbb{N}} Q_n$.*

Mais la suite $P_n$ est une suite croissante dans l'ensemble fini des parties de $Q$. Il existe donc $\mathcal{A} \in \mathbb{N}$ tel que $P_{M+1} = P_M$ auquel cas la définition même montre que $P_{M+2} = P_{M+1} = P_M$, puis par une récurrence triviale que $\forall n \geq M,\ P_n = P_M$ ; on en déduit donc que l'ensemble des états accessibles est $\bigcup_{n\in\mathbb{N}} P_n = P_M$. Le même raisonnement montre que l'ensemble des états coaccessibles est égal à $Q_N$ où $N$ est le premier entier tel que $Q_{N+1} = Q_N$.

L'algorithme de construction de l'automate émondé équivalent à un automate donné peut donc se formuler de la manière suivante :

- Poser $P_0 = \{p_0\}$ et calculer les $P_n$ jusqu'à ce que $P_{M+1} = P_M$
- Supprimer $P \setminus P_M$
- Poser $Q_0 = F$ et calculer les $Q_n$ jusqu'à ce que $Q_{N+1} = Q_N$
- Remplacer $Q \setminus Q_N$ par un unique rebut

La complexité de la construction provient bien entendu de la nécessité d'explorer systématiquement toutes les transitions possibles pour construire $P_{n+1}$ à partir de $P_n$.

# Résumé

3 Réduction des automates
- Accessibilité
- Graphes
- Type des graphes
- Exploration des graphes
- Exploration en profondeur
- Exploration en largeur
- Exploration l'aide de piles
- Emonder un graphe

Il est clair que dans les notions d'accessibilité et de coacessibilité, les étiquettes des transitions sont sans importance, et que seule l'existence d'au moins une transition d'un état $q$ vers un état $q'$ a de l'intérêt. L'automate dans lequel on a effacé les étiquettes des transitions constitue un graphe.

**Définition**

*On appelle graphe (orienté) la donnée d'un ensemble $S$ (les sommets du graphe) et d'une partie $A$ de $S \times S$ (les arêtes du graphe).*

On dispose les sommets dans le plan et on trace une flèche de $s$ à $s'$ si $(s, s') \in A$. On notera alors $s \longrightarrow s'$.

## Définition

*Soit $G$ un graphe, $x$ et $y$ deux sommets du graphe. On appelle chemin de $x$ à $y$ toute suite $s_1, \ldots, s_n$ de sommets du graphe telle que $x \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_N \longrightarrow y$.*

## Définition

*Soit $G$ un graphe, $x$ un sommet du graphe. On appelle composante connexe de $x$ l'ensemble des sommets $y$ de $G$ tels qu'il existe un chemin de $x$ à $y$.*

## Remarque

Il est clair que dans un automate, l'ensemble des sommets accessibles est la composante connexe de l'état initial dans le graphe sous-jacent, alors que l'ensemble des sommets co-accessibles est la réunion des composantes connexes des états finaux dans le graphe opposé (on change le sens des flèches).

# Résumé

3 Réduction des automates
- Accessibilité
- Graphes
- Type des graphes
- Exploration des graphes
- Exploration en profondeur
- Exploration en largeur
- Exploration l'aide de piles
- Emonder un graphe

## Caml

```
type graphe = G of ((int list) vect);;
```

# Résumé

3 Réduction des automates
- Accessibilité
- Graphes
- Type des graphes
- Exploration des graphes
- Exploration en profondeur
- Exploration en largeur
- Exploration l'aide de piles
- Emonder un graphe

Soit $G$ un graphe et $s_0$ un sommet de départ. On cherche à explorer le graphe de manière systématique (à la manière d'un labyrinthe) afin de déterminer la composante connexe de $s_0$. Deux stratégies (au moins) sont possibles

- l'exploration en profondeur : on s'enfonce au maximum dans le labyrinthe, en déposant un petit caillou dans chaque salle visitée ; lorsqu'on aboutit à un cul de sac ou à une salle déjà visitée, on fait marche arrière et on reprend le premier chemin non encore emprunté
- l'exploration en largeur : à partir d'une salle, on visite toutes les salles que l'on peut atteindre en une seule étape puuis on recommence à partir de ces salles

Autrement dit, avec une vision *généalogique*, pour visiter un individu et sa *descendance*

- dans l'exploration en profondeur, on visite d'abord les fils avant les frères
- dans l'exploration en largeur, on commence par visiter les frères avant de visiter les fils et neveux

# Résumé

3 Réduction des automates
- Accessibilité
- Graphes
- Type des graphes
- Exploration des graphes
- Exploration en profondeur
- Exploration en largeur
- Exploration l'aide de piles
- Emonder un graphe

Il suffit de travailler de manière récursive, en marquant les sommets visités et en arrêtant les appels récursifs dès que l'on tombe sur un sommet *cul de sac* ou un sommet déjà marqué. A la fin de l'algorithme, les sommets marqués constituent la composante connexe du sommet initial.

## Caml

```
let explore_en_profondeur (G graphe) depart =
        let n = vect_length graphe in
        let marque = make_vect n false in
        let rec visite i =
              if not marque.(i) then
                 begin
                    marque.(i) <- true;
                    do_list visite graphe.(i);
                  end
         in visite depart; marque;;
```

# Résumé

3 Réduction des automates
- Accessibilité
- Graphes
- Type des graphes
- Exploration des graphes
- Exploration en profondeur
- Exploration en largeur
- Exploration l'aide de piles
- Emonder un graphe

On travaille par couches successives à partir du sommet initial en ajoutant à l'ensemble des sommets rencontrés d'abord les sommets qui sont à une distance 1, puis ceux qui sont à une distance 2, et ainsi de suite jusqu'à ce que plus aucun sommet ne soit ajouté.

## Caml

```
let explore_en_largeur (G graphe) depart =
        let rec reunit = function
                | [] -> []
                | t::r -> union t (reunit r)
        in
        let ajoute_niveau liste =
                let nouveau =
                        reunit (map (function i -> graphe.(i)) liste)
                in
                union liste nouveau
        in let rec itere l1 l2 =
                if list_length l1 = list_length l2
                    then l1
                    else itere l2 (ajoute_niveau l2)
        in itere [] [depart];;
```

# Résumé

3 Réduction des automates
- Accessibilité
- Graphes
- Type des graphes
- Exploration des graphes
- Exploration en profondeur
- Exploration en largeur
- Exploration l'aide de piles
- Emonder un graphe

L'exploration peut également de faire à l'aide d'une pile et d'un tableau de marquage. Lorsqu'on rencontre un sommet non encore marqué, on le marque, on empile ses voisins, puis on dépile un nouveau sommet que l'on visite.

Si l'on utilise une pile LIFO, on visite les fils (que l'on vient tout juste d'empiler) avant de visiter les frères, on obtient un parcours en profondeur.

Si l'on utilise une pile FIFO, on visite les frères avant de visiter les fils, on obtient un parcours en largeur.

A la fin de l'algorithme, les sommets marqués constituent la compsante connexe du sommet initial.

## Caml

```
let explore (G graphe) depart =
      (* exploration en profondeur si pile LIFO,
         en largeur si pile FIFO *)
        let n = vect_length graphe in
        let marque = make_vect n false in
        let (empile,depile,non_vide) = new_pile () in
        let rec empile_liste = function
             | [] -> ()
             | t::r -> empile t; empile_liste r
        in
```

## Caml

```
empile depart;
while non_vide () do
     let x = depile () in
     if not marque.(x) then
        begin
          marque.(x) <- true;
          empile_liste graphe.(x)
         end
done;
marque;;
```

# Résumé

3 Réduction des automates
- Accessibilité
- Graphes
- Type des graphes
- Exploration des graphes
- Exploration en profondeur
- Exploration en largeur
- Exploration l'aide de piles
- Emonder un graphe

## Définition

*On dit qu'un automate est émondé, si tout sommet est à la fois accessible et coaccessible.*

## Remarque

L'exploration en profondeur permet de déterminer les états à la fois accessibles et coaccessibles, autrement dit d'émonder l'automate. En effet, un état est accessible si et seulement si il est atteint au cours du parcours en profondeur, et il est coaccessible si et seulement si, au cours du parcours en profondeur à partir de cet état, on a rencontré un état final ou un état déjà marqué comme coacessible. Il suffit que la fonction de visite renvoie `true` si c'est le cas et `false` sinon.

## Caml

```
let emonde (G graphe) coacc =
        (* emonde un graphe en fonction d'un ensemble coacc
        de sommets acceptants (sous forme de tableau de booleens)
        et d'un sommet initial 0. Modifie l'ensemble coacc pour
        ne garder que les sommets du graphe émondé *)
        let n = vect_length graphe in
        let marque = make_vect n false in
        let rec un_vrai = function
                        (* cherche un true dans une liste de booléens *)
                | [] -> false
                | t :: r -> t || (un_vrai r)
```

## Caml

```
in let rec visite i =
  (* visite et retourne le booleen coaccessible *)
      if not marque.(i) then begin
            marque.(i) <- true;
            let l = map visite graphe.(i) in
            coacc.(i) <- coacc.(i) || (un_vrai l)
          end;
       coacc.(i)
 in
 coacc.(0) <- visite 0; (* pas tres utile *)
 for i = 0 to (n-1) do
       coacc.(i) <- coacc.(i) && marque.(i)
done;;  (* accessible et coaccessible *)
```

# Résumé

4 Automates finis non déterministes
- Notion d'automate non déterministe
- Déterminisation d'un automate
- Complexité de la déterminisation
- Reconnaissance par un AFND

Les automates que nous avons considérés précédemment ont un comportement complètement déterminé par le mot qu'ils ont à examiner ; on dit qu'ils sont déterministes. Pour des raisons de commodité et en particulier pour pouvoir définir certaines opérations de base sur les automates, nous allons généraliser les automates en leur adjoignant un comportement a priori aléatoire, ou plutôt quantique.

Nous allons autoriser l'automate, lorsqu'il est dans un état $p$ et qu'il lit un caractère $c$, à passer dans un état quelconque d'un certain sous-ensemble $\delta(p, c)$. Si nous notons $\Delta = \{(p, c, q) \in Q \times A \times Q \mid q \in \delta(p, c)\}$, nous avons également $\delta(p, c) = \{q \in Q \mid (p, c, q) \in \Delta\}$, si bien que la donnée de $\Delta$ est équivalente à celle de $\delta : Q \times A \to \mathcal{P}(Q)$ (comme précédemment, lorsqu'une transition n'est pas définie, on l'envoie dans un rebut). Si bien que l'on peut poser :

## Définition

*On appelle automate fini non déterministe (ou plus simplement automate non déterministe, AND) la donnée*

- *d'un alphabet $A$*
- *d'un ensemble fini (et non vide) $Q$ appelé ensemble des états de l'automate*
- *d'un élément $p_0$ de $Q$ appelé l'état initial ou état de départ*
- *d'une partie $F$ de $Q$ appelée l'ensemble des états finaux ou états acceptants*
- *d'une partie $\Delta$ de $Q \times A \times Q$ appelée ensemble des transitions de l'automate*

On peut représenter un automate non déterministe par un diagramme sur lequel figureront tous les états de la machine ; ces états seront reliés par des flèches étiquetées par les éléments de l'alphabet comme ci dessus, la relation $(p, a, q) \in \Delta$ étant symbolisée par le même sous diagramme que d'habitude. La différence par rapport au diagramme d'un automate déterministe est que d'un même état peuvent partir plusieurs flèches ayant la même étiquette.

Partons maintenant d'un état $p$ et lisons un mot $w = c_1 \dots c_n$. Nous pouvons parcourir plusieurs chemins dans l'automate suivant la transition choisie à chaque étape. Il existe donc plusieurs suites $p, q_1, \dots, q_n$ d'états telles que

$$p \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_2 \longrightarrow \cdots \xrightarrow{c_n} q_n$$

Nous noterons $\overline{\delta}(p, w) \subset Q$ l'ensemble des états $q_n$ qui peuvent être atteints de cette manière. Ceci nous permet d'étendre la fonction $\delta : Q \times A \longrightarrow \mathcal{P}(Q)$ en une fonction $\overline{\delta} : Q \times A^* \longrightarrow \mathcal{P}(Q)$. Nous poserons $\overline{\Delta} = \{(p, w, q) \in Q \times A^* \times Q \mid q \in \overline{\delta}(p, w)\}$ si bien que l'on a également $\delta(p, w) = \{q \in Q \mid (p, w, q) \in \overline{\Delta}\}$. Nous écrirons encore $p \xrightarrow{w} q$ à la place de $(p, w, q) \in \overline{\Delta}$.

## Définition

*On dit que le mot $w$ est reconnu par l'automate non déterministe $\mathcal{A}$ s'il existe un état final dans $\overline{\delta}(p_0, w)$.*

## Remarque

Il faut donc considérer qu'un AND est plus un automate quantique qu'un automate probabiliste. Il ne choisit pas un chemin au hasard, mais au contraire examine d'un seul coup tous les chemins possibles : si au moins l'un de ces chemins aboutit à un état final, il déclare qu'il a reconnu le mot.

# Résumé

4 Automates finis non déterministes
- Notion d'automate non déterministe
- Déterminisation d'un automate
- Complexité de la déterminisation
- Reconnaissance par un AFND

On peut se demander si les automates non déterministes sont plus puissants, du point de vue des langages reconnus, que les automates non déterministes. Il n'en est rien, d'après le théorème suivant :

**Théorème**

*Pour tout automate non déterministe $\mathcal{A}$, il existe un automate déterministe $\mathcal{A}_d$ équivalent : un mot de $A^*$ est reconnu par $\mathcal{A}$ si et seulement si il est reconnu par $\mathcal{A}_d$.*

## Démonstration

Supposons que $\mathcal{A} = (A, Q, p_0, F, \Delta)$ avec $\Delta \subset Q \times A \times Q$ et comme précédemment soit $\delta : Q \times A \to \mathcal{P}(Q)$ définie par $\delta(p, c) = \{q \in Q \mid (p, c, q) \in \Delta\}$. Quitte à ajouter un état rebut, nous pouvons supposer que $\delta$ est bien définie sur $Q \times A$ tout entier. Nous allons poser $Q' = \mathcal{P}(Q)$ (ensemble des parties de $Q$), $p'_0 = \{p_0\}$, $F' = \{X \subset Q \mid X \cap F \neq \emptyset\}$ et $\delta' : Q' \times A \to Q'$ définie par $\delta'(X, c) = \bigcup_{x \in X} \delta(x, c)$ (pour $X \subset Q$). Considérons alors $\mathcal{A}_d = (A, Q', p'_0, F', \delta')$ ; il s'agit d'un automate déterministe. Nous allons montrer qu'il reconnaît exactement les mêmes mots que $\mathcal{A}$. Nous noterons $\overline{\delta}$ et $\overline{\delta'}$ les extensions respectives de $\delta$ et $\delta'$ aux mots de $A^*$

## Lemme

*Soit* $w \in A^*$ *et* $X \subset Q$. *Alors* $\overline{\delta'}(X, w) = \bigcup_{x \in X} \overline{\delta}(x, w)$

## Démonstration

nous allons procéder par récurrence sur la longueur $n$ de $w$. Si $n = 0$, $w$ est le mot vide $\varepsilon$ et on a $\overline{\delta}(x, \varepsilon) = \{x\}$ et $\overline{\delta'}(X, \varepsilon) = X$ ; comme bien évidemment $X = \bigcup_{x \in X} \{x\}$, l'égalité est vérifiée. Si $n = 1$, $w$ est réduit à un caractère $c$ et par définition $\delta'(X, c) = \bigcup_{x \in X} \delta(x, c) = \bigcup_{x \in X} \overline{\delta}(x, w)$. Supposons le résultat démontré pour les mots de longueur $n - 1$ et soit $w = c_1 \dots c_n$. On a vu que si $w' = c_2 \dots c_n$, alors

$$\forall p \in Q,\ \overline{\delta}(p, w) = \bigcup_{x \in \delta(p, c_1)} \overline{\delta}(x, w')$$

## Démonstration (suite)

On a donc, si $X \subset Q$

$$\overline{\delta'}(X, w) = \overline{\delta'}(\delta'(X, c_1), w') = \bigcup_{y \in \delta'(X, c_1)} \overline{\delta}(y, w')$$

par l'hypothèse de récurrence. Mais $\delta'(X, c_1)$ est l'ensemble des $y \in Q$ tels qu'il existe une transition dans $\mathcal{A}$ du type $xa \xrightarrow{c_1} y$, avec $x \in X$ ; donc $\bigcup_{y \in \delta'(X, c_1)} \overline{\delta}(y, w')$ est l'ensemble des $q \in Q$ tels qu'il existe une suite de transitions $x \xrightarrow{c_1} y \xrightarrow{w'} q$, avec $x \in X$, $y \in Q$, c'est à dire $x \xrightarrow{w} q$ ; cet ensemble est donc $\bigcup_{x \in X} \overline{\delta}(x, w)$, ce qui démontre le résultat.

## Démonstration

Alors un mot est reconnu par $\mathcal{A}_d$ si et seulement si $\overline{\delta'}(p'_0, w) \in F'$, soit encore $\bigcup_{x \in \{p_0\}} \overline{\delta}(x, w) \cap F \neq \emptyset$, c'est à dire $\overline{\delta}(p_0, w) \cap F \neq \emptyset$ ce qui signifie exactement qu'il est reconnu par $\mathcal{A}$.

Tout ceci peut s'écrire facilement en Caml

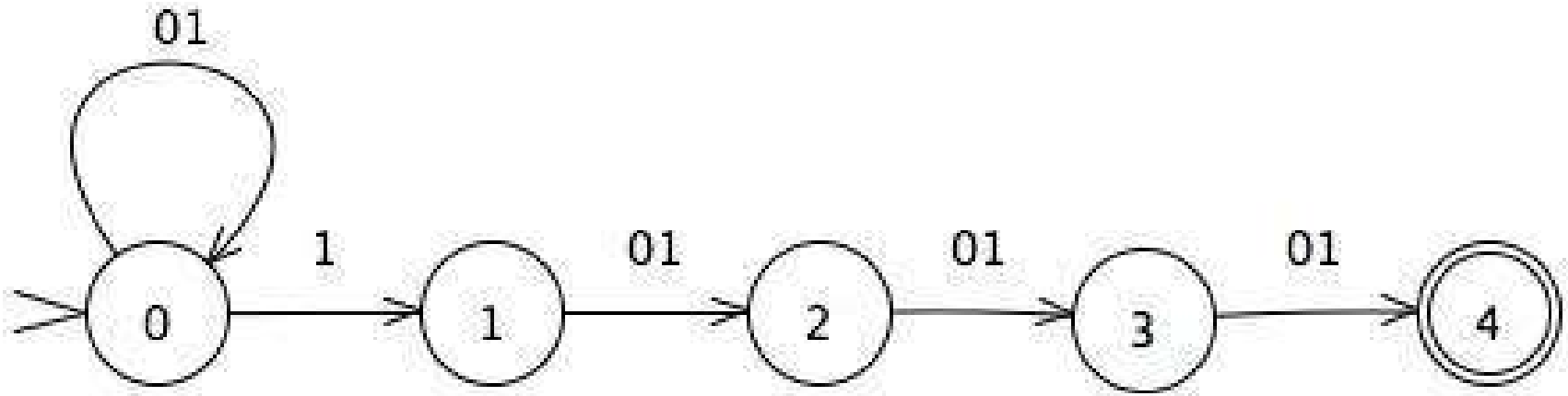
## Caml

```
#let determinise (delta: 'a * 'b -> 'a list) = function
          (liste,c) -> reunion (map (function a -> delta (a,c)) liste)
where rec reunion = function
             |  [] -> []
             |  t::r -> union t (reunion r);;
determinise : ('a * 'b -> 'a list) -> 'a list * 'b -> 'a list = <fun>
```

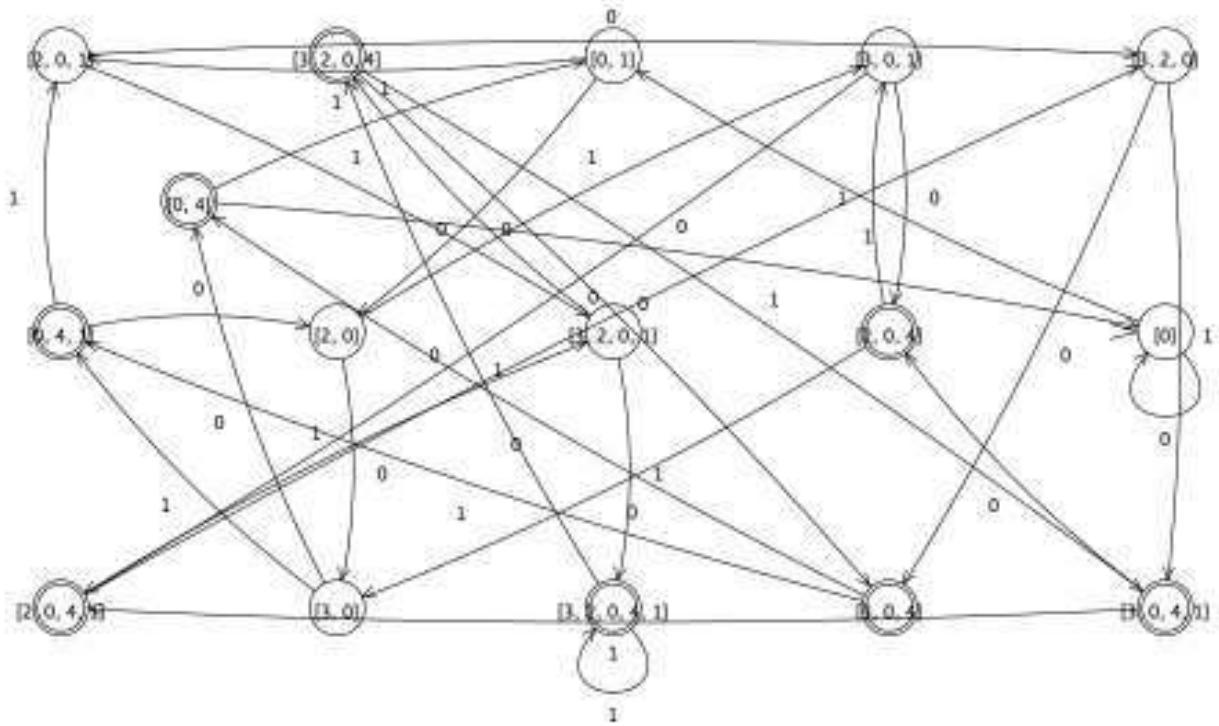
# Résumé

4 Automates finis non déterministes
- Notion d'automate non déterministe
- Déterminisation d'un automate
- Complexité de la déterminisation
- Reconnaissance par un AFND

La déterminisation d'un automate présente un intérêt essentiellement théorique. En effet, si l'on part d'un automate non déterministe dont l'ensemble $Q$ des états possède $n$ éléments, l'automate déterministe associé utilise $\mathcal{P}(Q)$ comme ensemble d'états, qui possède $2^n$ éléments. Ceci est illustré par l'exemple de l'automate ci-dessous qui reconnait le langage $A^*1A^3$ sur l'alphabet $\{0, 1\}$

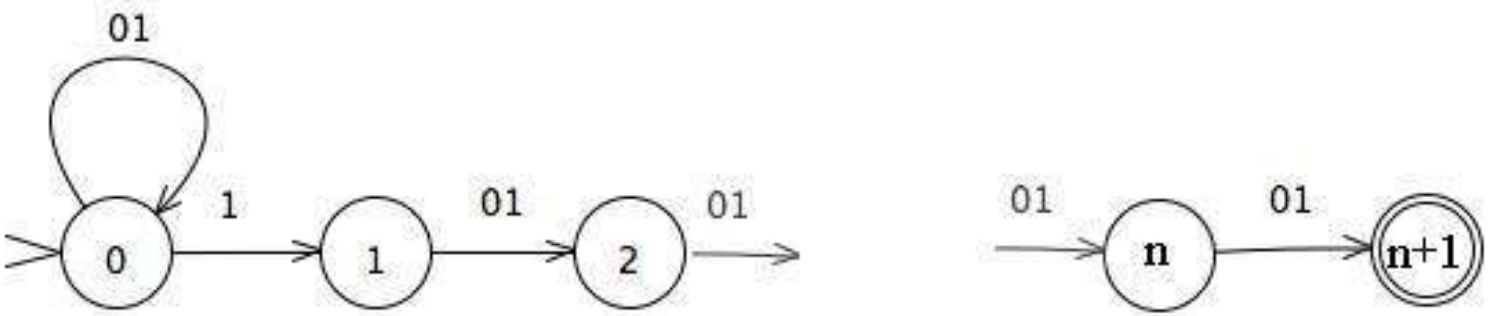


et dont l'automate déterministe associé est représenté ci-dessous (seuls les états accessibles sont représentés).




On est passé de 5 à 16 états.

Plus généralement, on montre facilement que si on déterminise l'automate à $n+2$ états qui reconnait le langage $A^*1A^n$,




on obtient un automate déterministe à $2^{n+1}$ états, ce qui aboutit à une croissance exponentielle de l'occcupation mémoire (sans compter le temps de calcul).

# Résumé

4 Automates finis non déterministes
- Notion d'automate non déterministe
- Déterminisation d'un automate
- Complexité de la déterminisation
- Reconnaissance par un AFND

Puisque la déterminisation complête de l'automate n'est pas réaliste, nous nous contenterons lors de la reconnaissance d'un mot, de construire au fur et à mesure les états utiles de l'automate déterministe : à partir d'une partie $X$ de $Q$, lorsque nous lisons le caractère $a$, nous passons dans le nouvel état $\delta'(X,x) = \bigcup_{p\in X} \overline{\delta}(p,w)$. Il s'agit alors d'opérations purement ensemblistes pour lesquelles la structure de liste est tout à fait adaptée.

Bien entendu, il ne faut plus utiliser la fonction `assoc` qui ne renvoie que la première association dans une liste d'association, mais in nous faut construire une fonction qui renvoie toutes les associations. C'est le rôle de notre fonction `cherche_tous` dans le code ci-après.

## Caml

```
type etat = {final: bool; transitions: (char*int) list};;

type AFND = AFND of etat vect;;

let reconnait (AFND v) w =
    let rec reunit = function
                (* reunit une liste de listes *)
          | [] -> []
          | t::r -> union t (reunit r)
    and cherche_tous a  = function
              (* recherche toutes les associations *)
          | [] -> []
          | (x,y)::r -> if x=a then y::(cherche_tous a r)
else cherche_tous a r
    and cherche_bon = function
                      (* recherche un état final *)
          |  [] -> false
          |  t::r -> v.(t).final || cherche_bon r
```

## Caml

```
    in
    let delta partie c =
        let f = function
                  x -> cherche_tous c v.(x).transitions in
        reunit (map f partie)
    in let rec delta_barre partie = function
           | [] -> partie
           | t::r -> delta_barre (delta partie t) r
in
    let dernier = delta_barre [0] w
    in cherche_bon dernier;;
```

# Résumé

5 Transitions instantanées
- Automates à transitions instantanées
- Suppression des transitions instantanées
- Reconnaissance par un AFND$\varepsilon$

La notion d'automate non déterministe a déjà introduit un certain effet *quantique* dans la reconnaissance des langages, puisque ces automates explorent tous les chemins possibles *à la fois*. Nous allons persévérer dans cette direction en introduisant l'analogue d'un *effet tunnel* : nous allons admettre que l'automate peut changer d'état alors qu'il ne lit aucun caractère, autrement dit alors qu'il lit le mot vide $\varepsilon$ ; un tel changement d'état, sera appelé une transition instantanée, ou encore une $\varepsilon$-transition.

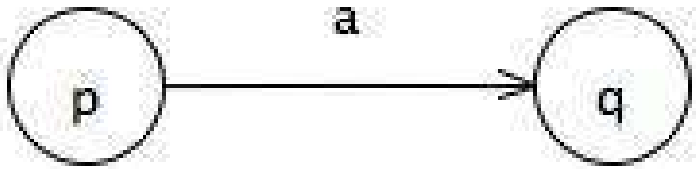
Nous verrons en effet que les transitions instantanées facilitent la construction d'un certain nombre d'opérations fondamentales sur les automates.

## Définition

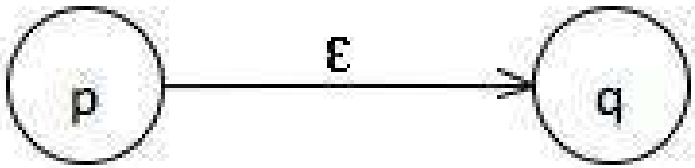
*On appelle automate fini non déterministe à transitions instantanées (ou automate non déterministe à $\varepsilon$-transitions, AND$\varepsilon$) la donnée*

- *d'un alphabet $A$*
- *d'un ensemble fini (et non vide) $Q$ appelé ensemble des états de l'automate*
- *d'un élément $p_0$ de $Q$ appelé l'état initial ou état de départ*
- *d'une partie $F$ de $Q$ appelée l'ensemble des états finaux ou états acceptants*
- *d'une partie $\Delta$ de $Q \times A \times Q$ appelée ensemble des transitions de l'automate*
- *d'une partie $I$ de $Q \times Q$ appelée ensemble des transitions instantanées de l'automate*

On peut représenter un automate non déterministe à transitions instantanées par un diagramme sur lequel figureront tous les états de la machine ; ces états seront reliés par des flèches étiquetées par les éléments de l'alphabet comme ci dessus, ou par le mot vide $\varepsilon$, la relation $(p, a, q) \in \Delta$ étant symbolisée par le




sous-diagramme la relation $(p, q) \in I$ étant




symbolisée par

Soit $w = c_1 \dots c_n$ un mot sur l'alphabet $A$. Nous allons définir récursivement la relation $p \overset{w}{\mapsto} q$ (la notation $\mapsto$ permet de distinguer les transitions *complétées* des transitions usuelles) de la façon suivante :

- si $w = \varepsilon$ (c'est à dire $n = 0$), on a $p \overset{\varepsilon}{\mapsto} q$ si et seulement si,
  - soit $q = p$,
  - soit il existe $q_1, \dots, q_p$ ($p \geq 0$) tels que $(p, q_1), (q_1, q_2), \dots, (q_p, q)$ soient toutes des transitions instantanées
- si $w = a \in A$ (c'est à dire si $n = 1$), on a $p \overset{a}{\mapsto} q$ si et seulement si il existe $q_1, q_2 \in Q$ tels que $p \overset{\varepsilon}{\mapsto} q_1$, $q_1 \overset{a}{\to} q_2$ et $q_2 \overset{\varepsilon}{\mapsto} q$
- si $w = c_1 \dots c_n$ (avec $n \geq 2$) et si $w' = c_1 \dots c_{n-1}$, alors $p \overset{w}{\mapsto} q$ signifie qu'il existe $q' \in Q$ tel que $p \overset{w'}{\mapsto} q' \overset{c_n}{\mapsto} q$

**Définition**

*On dit qu'un mot $w$ est reconnu par l'automate non déterministe à transitions instantanées $\mathcal{A}$ s'il existe un état final $q \in F$ tel que $p_0 \overset{w}{\mapsto} q$.*

# Résumé

5 Transitions instantanées
- Automates à transitions instantanées
- Suppression des transitions instantanées
- Reconnaissance par un AFND$\varepsilon$

L'introduction des transitions instantanées, qui est une commodité pour la suite de la théorie, n'ajoute rien à la puissance de reconnaissance des automates. C'est ce que nous nous proposons de montrer.

A chaque état $p \in Q$, nous pouvons associer l'ensemble des états $q$ tels que $p \overset{\varepsilon}{\mapsto} q$ (c'est à dire l'ensemble des états auxquels on peut accéder à partir de $p$ par une succession de transitions instantanées, ensemble auquel nous adjoindrons $p$ lui-même). C'est donc l'ensemble des états que l'on peut atteindre à partir de $p$ en lisant le mot vide.

## Définition

*Soit $\mathcal{A}$ un automate non déterministe à transitions instantanées et p un état de $\mathcal{A}$ ; on appelle clôture instantanée de p le sous ensemble constitué de p et de tous les états q de $\mathcal{A}$ tels que $p \overset{\varepsilon}{\mapsto} q$.*

## Théorème

*Pour tout automate non déterministe à transitions instantanées $\mathcal{A}$, il existe un automate déterministe $\mathcal{A}'$ équivalent, c'est à dire qu'un mot de $A^*$ est reconnu par $\mathcal{A}$ si et seulement si il est reconnu par $\mathcal{A}'$.*

## Démonstration

soit $\mathcal{A} = (A, Q, p_0, F, \Delta, I)$ un automate non déterministe à transitions instantanées, avec $\Delta \subset Q \times A \times Q$ et $I \subset Q \times Q$. Pour chaque $p \in Q$, nous disposons de la clôture instantanée de $p$, que nous noterons $\mathrm{cl}(p)$ ; pour toute partie $U$ de $Q$, on peut alors définir $\mathrm{cl}(U)$ comme $\mathrm{cl}(U) = \bigcup_{p \in U} \mathrm{cl}(p)$ ; on a $U \subset \mathrm{cl}(U)$. Nous poserons alors $\mathcal{A}' = (A, Q', p_0', F', \delta')$ avec

- $Q' = \{\mathrm{cl}(U) \mid U \subset Q\}$
- $p_0' = \mathrm{cl}(p_0)$
- $F' = \{U \in Q' \mid U \cap F \neq \emptyset\}$
- $\delta'(U, a) = \{q \in Q \mid \exists p \in U,\ p \overset{a}{\mapsto} q\}$

## Démonstration (suite)

Pour justifier cette définition, il suffit de remarquer que si $V = \delta'(U, a) = \{q \in Q \mid \exists p \in U,\ p \overset{a}{\mapsto} q\}$, on a $V = \mathrm{cl}(V) \in Q'$ ; en effet il est clair que $V \subset \mathrm{cl}(V)$ et inversement, si $q \in \mathrm{cl}(V)$, il existe $q' \in V$ tel que $q' \overset{\varepsilon}{\mapsto} q$ ; pour ce $q'$, il existe $p \in U$ tel que $p \overset{a}{\mapsto} q$, mais on a alors $p \overset{a}{\mapsto} q \overset{\varepsilon}{\mapsto} q'$, soit encore $p \overset{a}{\mapsto} q$ et donc $q \in V$.
Soit alors $w = c_1 \dots c_n$ un mot reconnu par $\mathcal{A}'$ ; on a donc une suite $U_0, \dots, U_n$ de $Q'$ telle que $U_0 = p'_0 = \mathrm{cl}(p_0)$, $U_{i+1} = \delta(U_i, c_i)$ et $U_n \cap F \neq \emptyset$ ; soit alors $q_n \in U_n \cap F$ ; il existe $q_{n-1} \in U_{n-1}$ tel que $q_{n-1} \overset{c_n}{\mapsto} q_n$, puis $q_{n-2} \in U_{n-2}$ tel que $q_{n-2} \overset{c_{n-1}}{\mapsto} q_{n-1}$, et ainsi de suite jusqu'à $q_0 \in U_0$ tel que

$$q_0 \overset{c_1}{\mapsto} q_1 \overset{c_2}{\mapsto} \cdots \overset{c_{n-1}}{\mapsto} q_{n-1} \overset{c_n}{\mapsto} q_n$$

## Démonstration (suite)

Mais comme $q_0$ appartient à $\mathrm{cl}(p_0)$, on a $p_0 \overset{\varepsilon}{\mapsto} q_0 \overset{c_1}{\mapsto} q_1$ et donc $p_0 \overset{c_1}{\mapsto} q_1$ ; ceci montre que

$$p_0 \overset{c_1}{\mapsto} q_1 \overset{c_2}{\mapsto} \cdots \overset{c_{n-1}}{\mapsto} q_{n-1} \overset{c_n}{\mapsto} q_n \in F$$

et donc $w$ est reconnu par $\mathcal{A}$.
Inversement, supposons que $w = c_1 \dots c_n$ est reconnu par $\mathcal{A}$ ; on a alors $p_0 \overset{w}{\mapsto} q$ avec $q \in F$. Il existe donc $q_1, \dots, q_n \in Q$ tels que

$$p_0 \overset{c_1}{\mapsto} q_1 \overset{c_2}{\mapsto} \cdots \overset{c_{n-1}}{\mapsto} q_{n-1} \overset{c_n}{\mapsto} q_n = q \in F$$

Si on définit alors $U_0 = \mathrm{cl}(p_0)$ puis $U_{i+1} = \delta'(U_i, c_i)$, la définition de $\delta'$ et une récurrence évidente montrent que $\forall i \geq 1,\ q_i \in U_i$ ; en particulier $q = q_n \in F \cap U_n$, donc $U_n \in F'$ et $w$ est reconnu par $\mathcal{A}'$.

# Résumé

5 Transitions instantanées
- Automates à transitions instantanées
- Suppression des transitions instantanées
- Reconnaissance par un AFND$\varepsilon$

## Caml

```
type etat = {final: bool; transitions: (char*int) list;
                instantanees: int list};;

type AFNDe = AFNDe of etat vect;;

let reconnait (AFNDe v) w =
    let rec reunit = function
                  (* reunit une liste de listes *)
          | [] -> []
          | t::r -> union t (reunit r)
    and cherche_tous a  = function
                (* recherche toutes les associations *)
          | [] -> []
          | (x,y)::r -> if x=a then y::(cherche_tous a r)
                        else cherche_tous a r
```

## Caml

```
and cloture l =
     let rec ajoute_instantanees l =
          reunit (
                map (function x -> v.(x).instantanees)) l
    and aux l1 l2 =
       if list_length l1 = list_length l2 then l1
                      else aux l2 (ajoute_instantanees l2)
    in aux [] l
and cherche_bon = function (* recherche un état final *)
      |  [] -> false
      |  t::r -> v.(t).final || cherche_bon r
in
```

## Caml

```
    let delta partie c =
        let f = function
                  x -> cherche_tous c v.(x).transitions in
        cloture (reunit (map f partie))
    in let rec delta_barre partie = function
           | [] -> partie
           | t::r -> delta_barre (delta partie t) r
in
    let dernier = delta_barre (cloture [0]) w
    in cherche_bon dernier;;
```

# Résumé

6 Langages reconnaissables
- Automate minimal
- Concaténation de mots, préfixes, suffixes
- Opérations sur les langages
- Traduction des opérations sur les langages
- Langages rationnels
- Expressions régulières
- Expressions régulières et automates

## Définition

*Soit $A$ un alphabet, $L$ un langage de $A^*$, $u \in A^*$. On note $u^{-1}L = \{v \in A^* \mid uv \in L\}$, appelé le résiduel de $L$ par rapport à $u$.*

## Définition

*La relation sur $A^*$ définie par*

$$u \sim_L v \iff u^{-1}L = v^{-1}L$$

*est une relation d'équivalence sur $A^*$ appelée la relation d'équivalence associée à $L$.*

On utilisera par la suite les deux lemmes suivants dont la vérification est élémentaire :

## Lemme

*Si $L$ est reconnu par un automate déterministe $\mathcal{A}$ d'état initial $p_0$, de fonction de transition $\delta$ et d'états finals $F$, alors*

$$u^{-1}L = \{v \in A^* \mid \delta(\delta(p_0, u), v) \in F\}$$

## Lemme

*Si* $u \in A^*$, $a \in A$, *alors* $(ua)^{-1}L = a^{-1}(u^{-1}L)$.

## Démonstration

En effet

$$\begin{aligned}(ua)^{-1}L &= \{w \in A^* \mid uaw \in L\} = \{w \in A^* \mid aw \in u^{-1}L\} \\ &= \{w \in A^* \mid w \in a^{-1}(u^{-1}L)\} = a^{-1}(u^{-1}L)\end{aligned}$$

On a la caractérisation suivante des langages reconnus par automates :

## Théorème

*Un langage $L$ est reconnu par un automate si et seulement si il y a un nombre fini de classes d'équivalence modulo $L$ (de résiduels de $L$) dans $A^*$.*

## Démonstration

Supposons que $L$ est reconnu par l'automate $\mathcal{A}$ d'état initial $p_0$, de fonction de transition $\delta$ et d'états finals $F$. A tout mot $u \in A^*$, on peut associer $f(u) = \delta(p_0, u) \in Q$ et on a donc $u^{-1}L = \{v \in A^* \mid \delta(f(u), v) \in F\}$ si bien que $f(u) = f(u') \Rightarrow u^{-1}L = u'^{-1}L$, ou encore que $u^{-1}L \neq u'^{-1}L \Rightarrow f(u) \neq f(u')$. On en déduit qu'il y a au plus autant de résiduels que d'éléments dans l'image de $f$ qui est contenue dans $Q$. Comme $Q$ est fini, il en est de même de l'ensemble des résiduels.

## Démonstration (suite)

Réciproquement, supposons que le nombre des résiduels de $L$ soit fini ; prenons pour $Q$ l'ensemble des résiduels, pour $p_0$ le résiduel de $L$ par le mot vide (c'est à dire $L$ lui même), pour états finals les résiduels de $L$ par les éléments de $L$ (c'est à dire ceux qui contiennent le mot vide $\varepsilon$) et comme fonction de transition $\delta(u^{-1}L, a) = a^{-1}(u^{-1}L) = (ua)^{-1}L$, $\mathcal{A}_0$ l'automate ainsi obtenu. Une récurrence évidente sur la longueur de $w \in A^*$ montre que $\delta(u^{-1}L, w) = w^{-1}(u^{-1}L) = (uw)^{-1}L$, si bien que

$$\begin{aligned} w \in L &\iff w^{-1}L \in F \iff (\varepsilon w)^{-1}L \in F \iff \delta(\varepsilon^{-1}L, w) \in F \\ &\iff w \text{ reconnu par } M_0 \end{aligned}$$

Donc $L$ est le langage reconnu par $\mathcal{A}_0$.

## Remarque

La première partie de la démonstration montre que tout automate déterministe reconnaissant le langage $L$ possède au moins autant d'états qu'il existe de résiduels de $L$ dans $A^*$, autrement dit autant d'états que $\mathcal{A}_0$. L'automate $\mathcal{A}_0$ est donc un automate à nombre minimal d'états reconnaissant $L$. On peut montrer que tout automate reconnaissant $L$ et ayant autant d'état que $\mathcal{A}_0$ est isomorphe à $\mathcal{A}_0$, ce qui montre que l'automate $\mathcal{A}_0$ est unique à isomorphisme près. Pour cette raison, $\mathcal{A}_0$ est appelé l'automate déterministe minimal reconnaissant $L$. La même notion n'existe pas pour les automates non déterministes.

## Remarque

Il existe un algorithme, dit de minimisation d'automates, permettant de construire, à partir de n'importe quel automate déterministe $\mathcal{A}$, un automate isomorphe à $\mathcal{A}_0$. Soit $\mathcal{A}$ un automate d'état initial $p_0$, de fonction de transition $\delta$ et d'états finals $F$. Pour $p \in Q$, on pose $L(q) = \{u \in A^* \mid \delta(q,u) \in F\}$ et $L_n(q) = \{u \in A^* \mid \delta(q,u) \in F \text{ et } |u| \leq n\}$. On introduit des relations d'équivalences sur $Q$ en posant

$$p \sim p' \iff L(p) = L(p') \quad \text{et } p \sim_n p' \iff L_n(p) = L_n(p')$$

## Remarque

L'automate minimal admet alors comme états l'ensemble des classes d'équivalence pour la relation $\sim$, comme état initial la classe de $p_0$, comme états finals les classes des éléments de $F$ et une fonction de transition qui à la classe de $p$ associe la classe $\delta_\sim(p, a)$ de $\delta(p, a)$ (encore faut-il vérifier que tout ceci est bien défini). Pour construire la relation d'équivalence $\sim$ on construit successivement les classes d'équivalence pour les relations $\sim_n$ en remarquant que $\sim_N = \sim_{N+1} \Rightarrow \forall n \geq N \sim_n = \sim_N \Rightarrow \sim_N = \sim$. Les vérifications sont laissées au lecteur.

# Résumé

6 Langages reconnaissables
- Automate minimal
- Concaténation de mots, préfixes, suffixes
- Opérations sur les langages
- Traduction des opérations sur les langages
- Langages rationnels
- Expressions régulières
- Expressions régulières et automates

## Définition

*Soit $A$ un alphabet, $w_1 = c_1 \dots c_m$ et $w_2 = c'_1 \dots c'_n$ deux mots sur $A$. On appelle concaténation de $w_1$ et $w_2$ le mot $w = w_1 w_2 = c_1 \dots c_m c'_1 \dots c'_n$.*

## Remarque

Il est clair que cette opération de concaténation est associative et que le mot vide $\varepsilon$ est un élément neutre à gauche et à droite. On dit encore que $A^*$ est un monoïde. De manière plus savante, on peut définir $A^*$ comme le monoïde libre construit sur l'alphabet $A$.

## Définition

*Soit $A$ un alphabet, $w$ et $w'$ deux mots de $A^*$. On dit que $w$ est un préfixe (resp. suffixe) de $w'$ s'il existe un mot $w''$ tel que $w' = ww''$ (resp. $w = w''w$).*

## Remarque

Il s'agit clairement de deux relations d'ordre : réflexives, antisymétriques et transitives. Le mot vide est préfixe et suffixe de tout mot.

# Résumé

## 6 Langages reconnaissables

- Automate minimal
- Concaténation de mots, préfixes, suffixes
- Opérations sur les langages
- Traduction des opérations sur les langages
- Langages rationnels
- Expressions régulières
- Expressions régulières et automates

Considérons un alphabet $A$. Comme les langages sont tout simplement des parties de $A^*$, nous disposons de trois opérations ensemblistes évidentes sur les langages : la complémentation, la réunion et l'intersection. Si $L$ est un langage, nous noterons $\neg L = A^* \setminus L$ le complémentaire de $L$ dans $A^*$. Si $L$ et $M$ sont deux langages, nous noterons $L \mid M$ (ou encore $L \cup M$ ou $L + M$) la réunion des langages $L$ et $M$. Enfin nous noterons $L \cap M$ l'intersection des langages $L$ et $M$.

L'opérateur de concaténation des mots, va nous permettre de définir un produit de concaténation sur les langages.

## Définition

*Soit $L$ et $M$ deux langages sur l'alphabet $A$. On appelle produit de concaténation de $L$ et $M$ le langage noté $L \cdot M$ (ou encore $LM$) défini par $LM = \{ww' \mid w \in L \text{ et } w' \in M\}$.*

## Remarque

Ce produit de concaténation est visiblement associatif. Il est distributif (à gauche et à droite) par rapport à la réunion.

On définit alors la puissance $n$-ième d'un langage $L$ par récurrence de la manière suivante

- $L^o = \{\varepsilon\}$
- $L^{n+1} = L.L^n = L^n.L$

## Définition

*Soit L un langage sur l'alphabet A. On appelle étoile de L le langage $L^*$ défini par* $L^* = \bigcup_{n\in\mathbb{N}} L^n$. *On appelle étoile stricte de L le langage $L^+$ défini par*

$$L^+ = \bigcup_{n\geq 1} L^n = L.L^* = L^*.L$$

# Résumé

6 Langages reconnaissables
- Automate minimal
- Concaténation de mots, préfixes, suffixes
- Opérations sur les langages
- Traduction des opérations sur les langages
- Langages rationnels
- Expressions régulières
- Expressions régulières et automates

Soit $\mathcal{A}$ un automate. Nous pouvons lui associer un nouvel automate $\neg\mathcal{A}$ ayant même état initial, mêmes transitions mais dont les états acceptants sont exactement les états non acceptants de $\mathcal{A}$. Le résultat suivant est alors évident :

## Théorème

*Si $\mathcal{A}$ est un automate déterministe et si $L$ est le langage reconnu par $\mathcal{A}$, alors le langage reconnu par $\neg\mathcal{A}$ est $\neg L$.*

## Remarque

Le fait que l'automate soit supposé déterministe est essentiel. Si l'automate n'est pas un automate déterministe, il est nécessaire de prendre au préalable un automate déterministe équivalent. Ceci rend la *négation* d'un automate infaisable dans la pratique (complexité exponentielle).

Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ deux automates. Nous pouvons définir un nouvel automate $\mathcal{A}_1 \times \mathcal{A}_2$ de la manière suivante :

- son ensemble d'états est $Q_1 \times Q_2$ (où $Q_i$ désigne l'ensemble des états de $\mathcal{A}_i$
- il possède un état initial $q = (q_1, q_2)$ où $q_1$ est l'état initial de $\mathcal{A}_1$ et $q_2$ celui de $\mathcal{A}_2$
- on a $(p_1, p_2) \xrightarrow{a} (p'_1, p'_2)$ si et seulement si on a $p_1 \xrightarrow{a} p'_1$ et $p_2 \xrightarrow{a} p'_2$
- son ensemble d'états acceptant est soit $F = F_1 \times F_2$, soit $F' = (F_1 \times Q_2) \bigcup (Q_1 \times F_2)$.

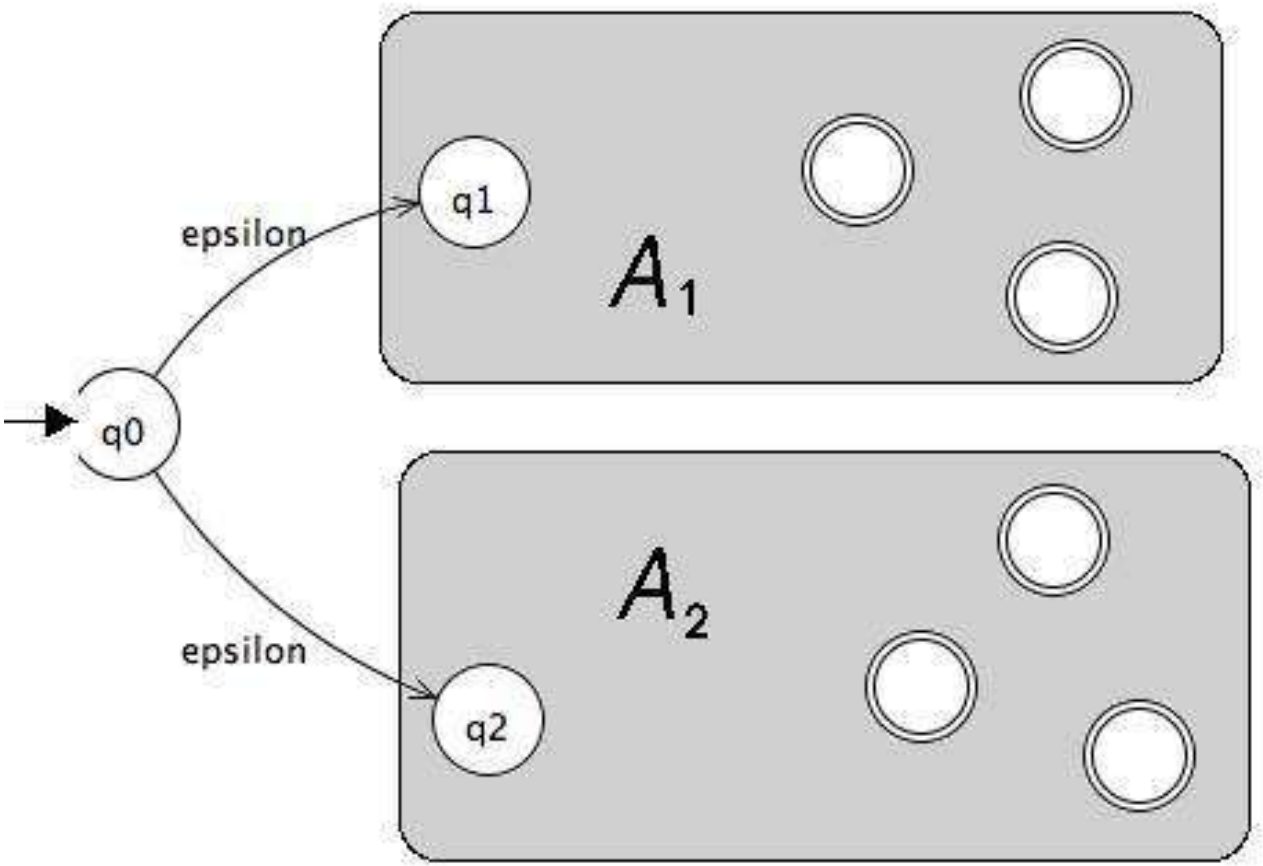
Dans le premier cas où $F = F_1 \times F_2$, on a $(\mathcal{A}_1 \times \mathcal{A}_2) = (\mathcal{A}_1) \bigcap (\mathcal{A}_2)$. Dans le deuxième cas, et dans la mesure où $\mathcal{A}_1$ et $\mathcal{A}_2$ sont complets, on a $(\mathcal{A}_1 \times \mathcal{A}_2) = (\mathcal{A}_1) \bigcup (\mathcal{A}_2)$.

## Remarque

De nouveau, la croissance du nombre d'états et la difficulté de construire les transitions, rend cette construction peu réaliste dans la pratique. On

Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ deux automates dont les ensembles d'états sont supposés disjoints (ceci n'est en aucune façon une limitation puisque les noms des états n'ont aucune importance), admettant des états initiaux $p_1$ et $p_2$. Nous pouvons définir un nouvel automate $\mathcal{A}_1 + \mathcal{A}_2$ de la manière suivante :

- il possède un état initial $p_0$ qui n'est ni un état de $\mathcal{A}_1$, ni un état de $\mathcal{A}_2$
- les seules transitions issues de $p_0$ sont des $\varepsilon$-transitions vers $p_1$ et $p_2$
- les autres transitions de $\mathcal{A}_1 + \mathcal{A}_2$ sont les transitions de $\mathcal{A}_1$ et celles de $\mathcal{A}_2$
- les états finaux de $\mathcal{A}_1 + \mathcal{A}_2$ sont les états finaux de $\mathcal{A}_1$ et ceux de $\mathcal{A}_2$


q0
epsilon
q1
$A_1$
epsilon
q2
$A_2$

## Théorème

*Si $L_i$ est le langage reconnu par $\mathcal{A}_i$ ($i = 1, 2$), alors le langage reconnu par $\mathcal{A}_1 + \mathcal{A}_2$ est $L_1 \mid L_2$.*

## Démonstration

soit $L$ le langage reconnu par $\mathcal{A}_1 + \mathcal{A}_2$. Soit $w \in L$ ; la première transition est forcément une transition instantanée de $p_0$ soit vers $p_1$, soit vers $p_2$, disons par exemple $p_1$. Comme il n'existe aucune transition d'un état de $\mathcal{A}_1$ vers un état de $\mathcal{A}_2$, tous les états parcourus par la suite sont des états de $\mathcal{A}_1$ et en particulier l'état final est un état final de $\mathcal{A}_1$. Ceci montre que le mot $w$ est reconnu par $\mathcal{A}_1$ et que donc $L \subset L_1 \mid L_2$. Inversement, si $w$ est reconnu par $L_1$, il suffit d'ajouter à la succession de transitions qui permet de le reconnaître la $\varepsilon$-transition de $p_0$ vers $p_1$ pour le faire reconnaître par $\mathcal{A}_1 + \mathcal{A}_2$, si bien que $L_1 \mid L_2 \subset L$. Donc $L = L_1 \mid L_2$.

Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ deux automates (dont les ensembles d'états sont toujours supposés disjoints). Nous pouvons définir un nouvel automate $\mathcal{A}_1 * \mathcal{A}_2$ par

$$\mathcal{A}_1 * \mathcal{A}_2 = \neg((\neg\mathcal{A}_1) + (\neg\mathcal{A}_2))$$

**Théorème**

*Si $L_i$ est le langage reconnu par $\mathcal{A}_i$ ($i = 1, 2$), alors le langage reconnu par $\mathcal{A}_1 * \mathcal{A}_2$ est $L_1 \cap L_2$.*

**Démonstration**

Evident puisque $L_1 \cap L_2 = \neg((\neg L_1) \cup (\neg L_2))$.

Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ deux automates (dont les ensembles d'états sont toujours supposés disjoints) d'états initiaux respectifs $p_1$ et $p_2$. Nous pouvons définir un nouvel automate $\mathcal{A}_1\mathcal{A}_2$ de la manière suivante :

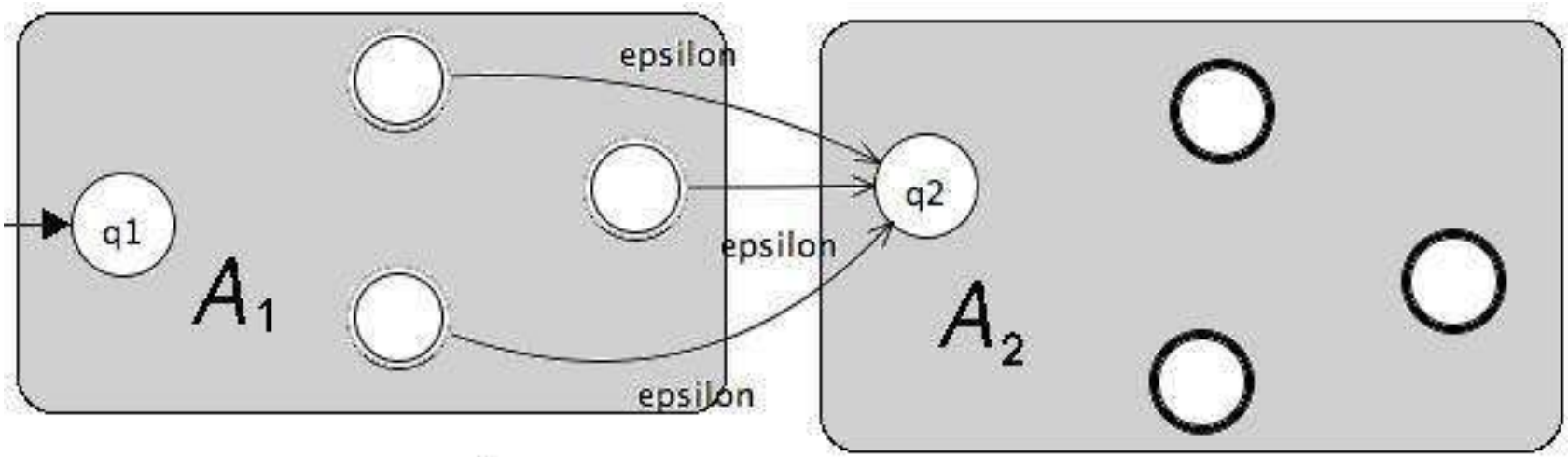
- son état initial est $p_1$
- on construit une $\varepsilon$-transition de chaque état final de $A_1$ vers $p_2$
- les autres transitions de $\mathcal{A}_1 + \mathcal{A}_2$ sont les transitions de $\mathcal{A}_1$ et celles de $\mathcal{A}_2$
- les états finaux de $\mathcal{A}_1\mathcal{A}_2$ sont les états finaux de $\mathcal{A}_2$

## Théorème

*Si $L_i$ est le langage reconnu par $\mathcal{A}_i$ ($i = 1, 2$), alors le langage reconnu par $\mathcal{A}_1\mathcal{A}_2$ est le langage $L_1L_2$, concaténation de $L_1$ et $L_2$.*

## Démonstration

soit $L$ le langage reconnu par $\mathcal{A}_1\mathcal{A}_2$. Soit $w \in L$ ; comme il n'existe aucune transition d'un état de $\mathcal{A}_2$ vers un état de $\mathcal{A}_1$, que l'état de départ est dans $A_1$ et que l'état final est dans $A_2$, tous les états parcourus sont pour les premiers dans $A_1$, pour les suivants dans $A_2$. Ceci donne une décomposition du mot $w$ sous la forme $w_1w_2$. Les seules transitions d'un état de $A_1$ vers un état de $A_2$ sont des transitions instantanées d'un état acceptant de $A_1$ vers $p_2$, ce qui montre que $w_1$ est reconnu par $A_1$. Mais alors $w_2$ est évidemment reconnu par $A_2$ (puisque l'on repart de $p_2$) et donc $w = w_1w_2 \in L_1L_2$, soit $L \subset L_1L_2$. L'inclusion en sens inverse est évidente. Le tout est parfaitement décrit par le dessin suivant


epsilon
q1
$A_1$
epsilon
epsilon
q2
$A_2$

Soit $\mathcal{A}$ un automate. Nous pouvons définir un nouvel automate $\mathcal{A}^*$ de la manière suivante :

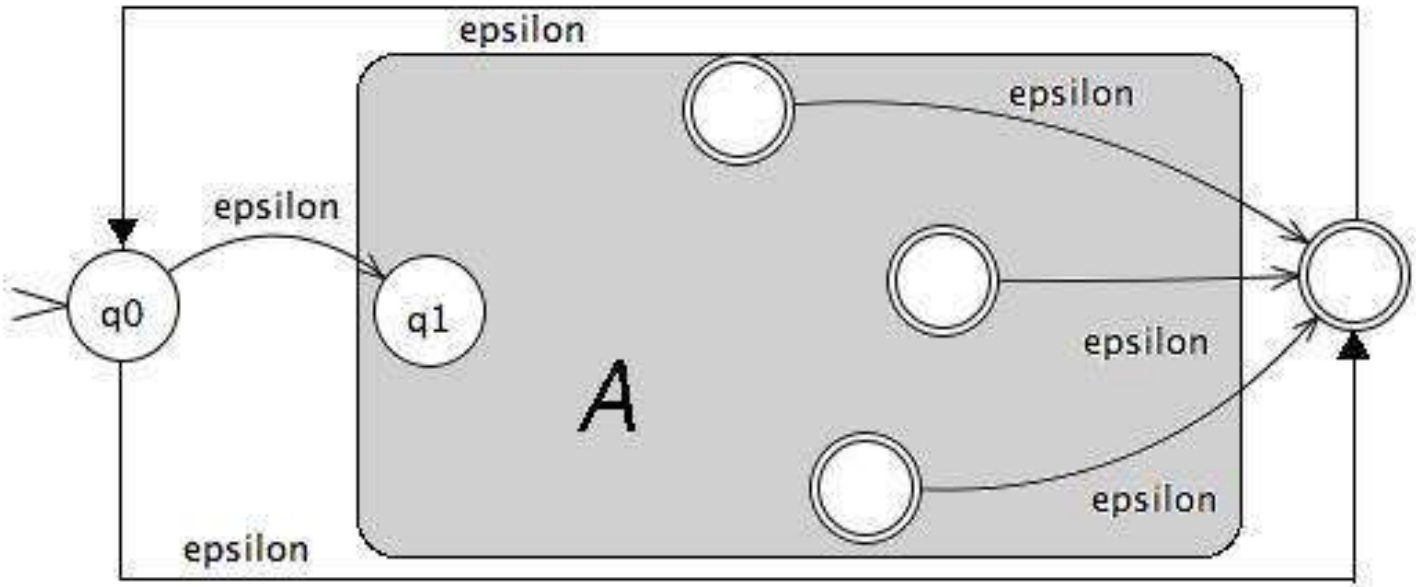
- il possède un unique état initial et un unique état final qui ne sont pas des états de $\mathcal{A}$ ; les autres états sont ceux de $A$
- on construit une $\varepsilon$-transition de l'état initial de $\mathcal{A}^*$ vers l'état initial de $A$ et une autre de l'état initial de $\mathcal{A}^*$ vers l'état final de $\mathcal{A}^*$ (pour reconnaître le mot vide)
- on ajoute une transition instantanée de chaque état final de $A$ vers l'état final de $\mathcal{A}^*$ et une transition instantanée de l'état final de $\mathcal{A}^*$ vers l'état initial de $A^*$ (qui permet de boucler)
- les autres transitions de $\mathcal{A}^*$ sont les transitions de $\mathcal{A}$.

## Théorème

*Si L est le langage reconnu par $\mathcal{A}$ , alors le langage reconnu par $\mathcal{A}^*$ est le langage $L^*$, étoile du langage L.*

## Démonstration

la transition instantanée de l'état initial vers l'état final permet de reconnaître le mot vide. En ce qui concerne les autres mots, une simple récurrence sur $n$ permet de montrer que les mots reconnus par l'automate $\mathcal{A}^*$ en empruntant exactement $n$ fois la nouvelle $\varepsilon$-transition de l'état final de $\mathcal{A}^*$ vers son état initial sont les mots de $L^{n+1}$, ce qui montre le résultat.


epsilon
epsilon
epsilon
q0
q1
A
epsilon
epsilon
epsilon

# Résumé

6 Langages reconnaissables
- Automate minimal
- Concaténation de mots, préfixes, suffixes
- Opérations sur les langages
- Traduction des opérations sur les langages
- Langages rationnels
- Expressions régulières
- Expressions régulières et automates

## Définition

*Soit A un alphabet. On appelle ensemble des langages rationnels sur l'alphabet A la partie de l'ensemble des langages sur A définie inductivement par*

- *tout langage fini est rationnel*
- *si $L_1$ et $L_2$ sont deux langages rationnels, alors $L_1 \mid L_2$ et $L_1 L_2$ sont rationnels*
- *si L est un langage rationnel, alors $L^*$ est un langage rationnel.*

## Théorème

*Un langage sur un alphabet A est rationnel si et seulement si il est reconnaissable par un automate.*

## Démonstration

Un langage fini est évidemment reconnaissable par un automate admettant deux états, l'un initial et l'autre final, avec exactement une suite de transitions pour chaque élément du langage et en utilisant les constructions du paragraphe précédent. De plus on vient de voir que si $L_1$ et $L_2$ sont deux langages reconnaissables, alors $L_1 \mid L_2$ et $L_1 L_2$ sont reconnaissables et que si $L$ est un langage reconnaissable, alors $L^*$ est un langage reconnaissable. Donc l'ensemble des langages reconnaissables contient l'ensemble des langages rationnels.

## Démonstration (suite)

Il nous reste à montrer que tout langage reconnaissable est rationnel. Soit donc $L$ un langage reconnu par un automate $\mathcal{A} = (Q, p_0, F, \delta)$ que l'on peut supposer déterministe. Pour $i, j \in Q$ et $P \subset Q$ soit $L_{i,j}(P)$ l'ensemble des mots sur $A^*$ permettant d'aller de $i$ à $j$ en ne passant que par des états intermédiaires dans $P$ :

$$L_{i,j}(P) = \{w = a_1 \dots a_n \mid i \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \dots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} j, \quad q_1, \dots, q_{n-1} \in P\}$$

Par convention, si $i = j$, on imposera à $L_{i,j}(P)$ de contenir le mot vide $\varepsilon$. On va montrer par récurrence sur le cardinal de $P$ que le langage $L_{i,j}(P)$ est rationnel. On en déduira le résultat puisque $L(\mathcal{A}) = \bigcup_{p \in F} L_{p_0,p}(Q)$.

## Démonstration (suite)

Si $|P| = 0$, alors $P$ est l'ensemble vide ; de plus $L_{i,j}(P) = \{a \mid i \xrightarrow{a} j\}$ si $i \neq j$ et $L_{i,i}(P) = \{\varepsilon\} \cup \{a \mid i \xrightarrow{a} i\}$, qui sont des langages finis, donc rationnels. Supposons donc le résultat montré lorsque $|P| = n - 1$ et supposons que $|P| = n$. Soit $k \in P$ et $P' = P \setminus \{k\}$ (de cardinal $n - 1$). Soit $i, j \in Q$. Un chemin de $i$ à $j$ peut soit ne pas passer par $k$, soit être la concaténation d'un chemin de $i$ à $k$ ne passant pas par $k$, d'un certain nombre (éventuellement nul) de cycles ayant $k$ comme origine et comme extrémité et d'un chemin de $k$ à $j$ ne passant pas par $k$, si bien que

$$L_{i,j}(P) = L_{i,j}(P') \mid (L_{i,k}(P')L_{k,k}(P')^* L_{k,j}(P'))$$

qui est bien un langage rationnel puisque les $L_{p,q}(P')$ sont rationnels. Ceci achève la démonstration.

**Méthode matricielle** La démonstration du théorème fournit un véritable algorithme (appelé algorithme de Mac-Naughton-Yamada) de construction du langage reconnu par un automate. Supposons en effet les états numérotés de 1 à $n$, prenons $P_k = \{1, \ldots, k\}$ et $P'_k = P_{k-1} = \{1, \ldots, k-1\}$. Nous poserons par convention $L_{i,j}^{(k)} = L_{i,j}(P_k)$. Les formules ci dessus s'écrivent alors sous forme *matricielle*

$$L_{i,j}^{(0)} = \{a \mid i \xrightarrow{a} j\} \text{ si } i \neq j \text{ et } L_{i,i}^{(0)} = \{\varepsilon\} \cup \{a \mid i \xrightarrow{a} i\}$$

$$\forall k \in [1,n],\ \forall i,j \in [1,n],\ L_{i,j}^{(k)} = L_{i,j}^{(k-1)} + L_{i,k}^{(k-1)} \left(L_{k,k}^{(k-1)}\right)^* L_{k,j}^{(k-1)}$$

ce qui permet un calcul de $L(\mathcal{A}) = \bigcup_{k \in F} L_{1,k}^{(n)}$ si on attribue à l'état initial le numéro 1. Bien entendu la forme du résultat final (en général très compliquée) est très sensible à l'ordre de numérotation des états.

**Méthode par résolution d'équations** Une autre méthode pour calculer le langage reconnu par un automate fini est une méthode par résolution d'équations. Pour tout $p$ dans $Q$, appelons $L_p$ l'ensemble des mots qui sont étiquettes d'un chemin de $p$ à un état final de l'automate

$$L_p = \{w \in A^* \mid \exists q \in F, p \xrightarrow{w} q\}$$

On a évidemment $L(\mathcal{A}) = L_{p_0}$ si $p_0$ est l'état initial de l'automate. Mais un mot $w$ appartient à $L_p$

- soit si $w = \varepsilon$ et $p \in F$
- soit si $w = aw'$ avec $p \xrightarrow{a} q$ et $w' \in L_q$

Autrement dit

$$\forall p \in Q,\ L_p = \bigcup_{q \in Q} A_{p,q} L_q + \delta_{p,F}$$

où $A_{p,q} = \{a \in A \mid p \xrightarrow{a} q\}$ et $\delta_{p,F} = \varepsilon$ si $p \in F$, $\delta_{p,F} = \emptyset$ sinon. Ceci amène à résoudre un système de $n$ équations *linéaires* à $n$ inconnues.

Ce système est résoluble par une méthode de susbstitutions successives (de la même manière que l'on résoud un système algébrique ordinaire) à condition de savoir résoudre un système de la forme $X = KX + L$ où $K$ et $L$ sont deux parties de $A^*$. Or on a

## Lemme

*Soit $K$ et $L$ deux parties de $A^*$ telles que $\varepsilon \notin K$. Alors l'équation $X = KX + L$ a pour unique solution $X = K^*L$.*

## Démonstration

Posons $X = K^*L$. On a alors $K^* = \varepsilon + KK^*$ si bien que

$$X = (\varepsilon + KK^*)L = L + KK^*L = L + KX$$

ce qui montre que $X$ est solution.

## Démonstration (suite)

Inversement soit $X$ une solution. On a $L \subset L + KX = X$, puis $KL \subset KX \subset X$ et par une récurrence évidente $K^n L \subset X$ pour tout $n \geq 0$, ce qui montre que $K^*L = \bigcup_{n=0}^{+\infty} K^n L \subset X$. Supposons que $X \neq K^*L$ et soit $w$ un mot de longueur minimal de $X \setminus K^*L$. Alors $w \in X = L + KX$ et $w \notin L$. On en déduit que $w \in KX$, autrement dit que $w = w_1 w_2$ avec $w_1 \in K$ et $w_2 \in X$. Mais puisque $\varepsilon \notin K$, on a $|w_2| < |w|$, et comme $w$ est un mot de longueur minimal de $X \setminus K^*L$, on a $w_2 \in K^*L$. Mais alors $w = w_1 w_2 \in K^*L$, c'est absurde. Donc $X = K^*L$.

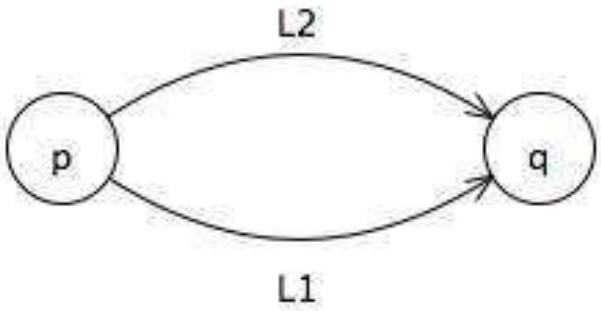
**Méthode par réduction** Enfin nous citerons une troisième méthode, la méthode par réduction. Nous admettrons des automates généralisés, dont les transitions sont étiquetées par des langages (ou comme nous le verrons plus tard par des expressions régulières qui décrivent ces langages) au lieu d'être étiquetées par des caractères de l'alphabet. Nous noterons alors $p \overset{w}{\mapsto} q$ s'il existe $p_1, \ldots, p_{n-1} \in Q$, $w_1, \ldots, w_n \in A^*$ et des langages $L_1, \ldots, L_n$ tels que

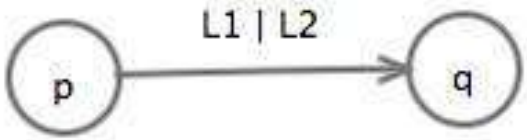
$$\forall i \in [1, n],\ w_i \in L_i, \quad w = w_1 \ldots w_n, \quad p \overset{L_1}{\to} p_1 \overset{L_2}{\to} p_2 \cdots \overset{L_{n-1}}{\to} p_{n-1} \overset{L_n}{\to} q$$

Soit alors un automate (déterministe ou non) $\mathcal{A} = (Q, p_0, F, \Delta)$. On commence par lui ajouter deux états $\alpha\ \beta$ et on remplace $\mathcal{A}$ par $\mathcal{A}' = (Q \cup \{\alpha, \beta\}, \alpha, \{\beta\}, \Delta')$ dont l'état initial est $\alpha$, le seul état final $\beta$ et où l'ensemble des transitions est obtenu en ajoutant à $\Delta$ une transition $\alpha \overset{\varepsilon}{\to} p_0$ et des transitions $q \overset{\varepsilon}{\to} \beta$ pour chaque $q \in F$. On considère maintenant que les transitions de $\mathcal{A}'$ sont étiquetées par des langages (pour le moment réduits à des singletons). Il est clair que $\mathcal{A}'$ reconnaît le même langage que $\mathcal{A}$.

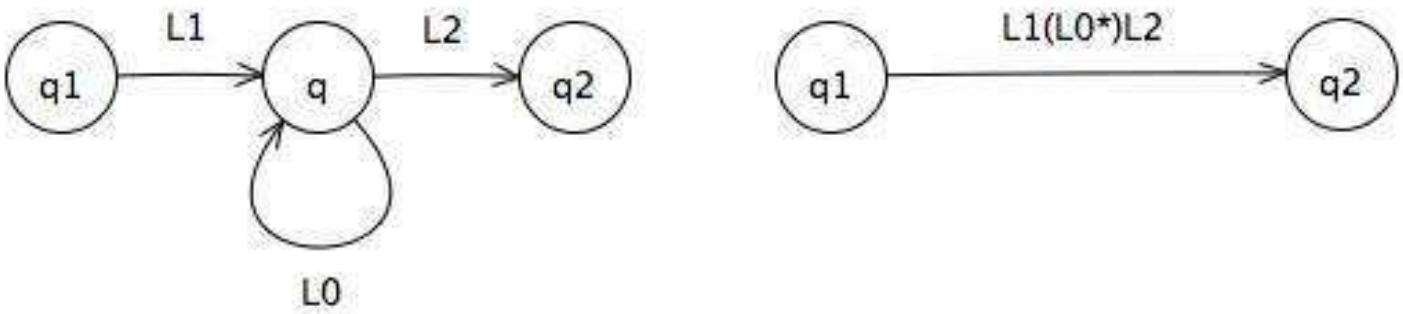
Nous allons ensuite effectuer des réductions successives sur l'automate $\mathcal{A}'$ en supprimant à chaque fois soit une transition, soit un état distinct de $\alpha$ et $\beta$.

Le premier type d'opérations consiste à remplacer un couple de transitions $p \xrightarrow{L_1} q$ et $p \xrightarrow{L_2} q$ par la transition $p \xrightarrow{L_1 \cup L_2} q$. On obtient un nouvel automate qui reconnaît le même langage.



Le deuxième type va consister à supprimer un état $q \in Q$. Soit $L_0$ le langage tel que $q \xrightarrow{L_0} q$ ($L_0$ est réduit à $\varepsilon$ s'il n'existe pas de transition de $q$ à $q$).Pour tout couple d'états $q_1, q_2 \in Q \cup \{\alpha, \beta\}$ tel que $q_1 \xrightarrow{L_1} q \xrightarrow{L_2} q_2$, on définit une nouvelle transition $q_1 \xrightarrow{L_1 L_0^* L_2}$. Ensuite on supprime $q$ et toutes les transitions qui y aboutissent ou qui en partent. On obtient un nouvel automate qui reconnaît le même langage (facile).




Après un nombre fini d'opérations, on aboutit à un automate $\alpha \xrightarrow{L} \beta$ ; et $L$ est bien entendu le langage reconnu par l'automate.

# Résumé

6 Langages reconnaissables
- Automate minimal
- Concaténation de mots, préfixes, suffixes
- Opérations sur les langages
- Traduction des opérations sur les langages
- Langages rationnels
- Expressions régulières
- Expressions régulières et automates

Il arrive fréquemment que l'on veuille tester si une chaîne de caractères a une forme déterminée. Par exemple :

- tester si une chaîne commence par un *a*, se termine par un *z* tout en contenant un *m*
- tester si une chaîne contient comme sous-chaîne *abc*
- tester si une chaîne ne contient que des chiffres avec éventuellement un point décimal

Pour cela nous allons introduire la notion d'expression régulière. La syntaxe d'une expression régulière sera une version légèrement réduite de celle qui est utilisée par certaines commandes de systèmes d'exploitation. Une expression régulière pourra être

- soit un caractère alphabétique ou numérique
- soit un caractère joker : le point d'interrogation ? ou le point usuel .
- soit la concaténation de deux expressions régulières : $m = m_1 m_2$
- soit une expression régulière suivi d'une étoile : $m = m_1 *$
- soit une expression régulière suivie d'un symbole d'addition : $m = m_1 +$
- soit plusieurs expressions régulières séparées par des barres verticales $m = m_1 \mid \cdots \mid m_n$

La signification (ou sémantique) de ces expressions régulières sera la suivante (dans l'ordre de priorités décroissantes) :

- une expression régulière réduite à un caractère alphanumérique reconnaît uniquement les chaînes à un seul élément réduites à ce caractère
- une expression régulière réduite au caractère joker $\boxed{.}$ reconnaît toute chaîne à un seul caractère
- une expression régulière $m = m_1*$ reconnaît toute chaîne qui est la concaténation de 0, 1 ou plusieurs chaînes, chacune étant reconnue par $m_1$
- une expression régulière $m = m_1+$ reconnaît toute chaîne qui est la concaténation de 1 ou plusieurs chaînes, chacune étant reconnue par $m_1$
- une expression régulière $m = m_1 m_2$ reconnaît toute chaîne $s = s_1 s_2$ qui est la concaténation de deux chaînes $s_1$ et $s_2$, la première reconnue par $m_1$ et la deuxième par $m_2$
- une expression régulière $m = m_1 \mid \ldots \mid m_n$ reconnaît les chaînes reconnues par l'une au moins des expressions régulières $m_i$

Les parenthèses permettront comme d'habitude de résoudre les difficultés liées aux priorités.

De cette manière, on a par exemple

- une chaîne commence par un *a* et se termine par un *z*, tout en contenant un *m*, si et seulement si elle est reconnue par le motif `a.*m.*z`
- une chaîne contient comme sous-chaîne *abc* si et seulement si elle est reconnue par le motif `.*abc.*`
- une chaîne ne contient que des chiffres si et seulement si elle est reconnue par le motif $(0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)*$

## Théorème

*Les langages rationnels sont exactement les langages reconnus par des expressions régulières.*

## Démonstration

Il est clair que l'ensemble des langages reconnus par des expressions régulières contient les mots à une lettre, et est stable par concaténation, réunion et étoile. A partir des mots à une lettre et de la concaténation, on montre que cet ensemble contient les singletons, puis par réunion contient les langages finis. Par définition de l'ensemble des langages rationnels, l'ensemble des langages reconnus par des expressions régulières contient l'ensemble des langages rationnels. Mais inversement, on montre immédiatement par récurrence sur la longueur d'une expression régulière, que le langage reconnu par une expression régulière est rationnel (il est obtenu par réunion, concaténation et étoile à partir de langages réduits à un mot de longueur 1). Par conséquent, les deux ensembles sont égaux.

# Résumé

6 Langages reconnaissables
- Automate minimal
- Concaténation de mots, préfixes, suffixes
- Opérations sur les langages
- Traduction des opérations sur les langages
- Langages rationnels
- Expressions régulières
- Expressions régulières et automates

Nous allons associer à chaque motif, un automate (de Thompson) chargé de reconnaître si une certaine chaîne est reconnue par ce motif. La construction est évidente :

- l'automate chargé de reconnaître un unique caractère possède un état initial, un état final et une unique transition étiquetée par ce caractère
- l'automate chargé de reconnaître le motif $m_1 m_2$ est la concaténation $\mathcal{A} = \mathcal{A}_1 \mathcal{A}_2$ des automates chargés de reconnaître $m_1$ et $m_2$
- l'automate chargé de reconnaître le motif $m_1 \mid \ldots \mid m_n$ est l'automate $\mathcal{A}_1 \mid \ldots \mid \mathcal{A}_n$, si $\mathcal{A}_i$ est chargé de reconnaître $m_i$
- l'automate chargé de reconnaître le motif $m*$ (resp. $m+$) est l'automate $\mathcal{A}^*$ (resp. $\mathcal{A}^+$)

Les autres automates s'en déduisent facilement puisque le motif *point* reconnaît les mêmes chaînes que le motif $a \mid \ldots \mid z \mid A \mid \ldots \mid Z \mid 0 \mid \ldots \mid 9$ (sans parler des caractères accentués).

A chaque motif, nous pouvons associer un certain arbre syntaxique lié aux différents opérateurs sur les motifs : l'étoile et le plus qui sont unaires, la concaténation et la barre verticale qui sont binaires. Ceci conduit à poser :

## Caml

```
#type arbre_motif =
          | Feuille_car of char
          | Feuille_point
          | Etoile of arbre_motif
          | Plus of arbre_motif
          | Concat of (arbre_motif*arbre_motif)
          | Barre of (arbre_motif*arbre_motif);;
Type arbre_motif defined.
```

La construction de l'automate associé à un motif réduit à un caractère est la suivante :

## Caml

```
#let autom_car a =
    let e1=nouvel_etat () and e2=nouvel_etat () in
    e1.initial=true; e2.final=false;
    ajoute_transition e1 a e2;
    {etat_initial=e1; etat_final=e2};;
```

L'automate associé au point peut alors s'écrire

## Caml

```
#let caracteres =
     let rec aux_alpha i =
         if i>26 then [] else
              (char_of_int (64+i))::(char_of_int (96+i))::aux_alpha (i+1)
     and aux_num i =
         if i>9 then aux_alpha 1 else (char_of_int (48+i))::aux_num (i+1)
     in aux_num 0 ;;
let autom_point () =
     reunit_liste(map autom_car caracteres);;
```

La construction de l'automate associé à une expression régulière d'arbre donné peut alors s'écrire :

## Caml

```
#let rec arbre_en_automate = function
      |  Feuille_car c -> autom_car c
      |  Feuille_point -> autom_point ()
      |  Concat (gauche,droite) ->
            concatene (arbre_en_automate gauche)
                      (arbre_en_automate droite)
      |  Etoile arbre -> etoile (arbre_en_automate arbre)
      |  Plus arbre -> etoile_stricte (arbre_en_automate arbre)
      |  Barre (gauche,droite) ->
            reunit (arbre_en_automate gauche)
                   (arbre_en_automate droite);;
arbre_en_automate : arbre_motif -> automate = <fun>
```