CALCULABILITE ET DECIDABILITE

5. Automates

5-1 La notion d'état ou de configuration

La notion d'*automate* est une bonne approximation de notre projet de représenter la notion d'algorithme de la manière la plus générale possible. On sait en effet que si on analyse un petit programme tel que le suivant :

```
(a et b sont des entiers positifs ou nuls)
x := a
y := b
z := 0
tant que y ≠ 0 faire:
     début
          z := z + x
          y := y - 1
     fin
afficher z
```

on décrira son éxécution par une suite d'états, depuis un *état initial* (défini lors de la phase d'initialisation) jusqu'à un *état final* (si l'algorithme termine bien!) qui contient la valeur recherchée de z. Dans cet exemple, on peut identifier chaque état au couple des valeurs prises par les variables y et z. Avec a = 5 et b = 6 par exemple, on obtient la suite d'états:

$e_0 = (6, 0)$; $e_1 = (5, 5)$; $e_2 = (4, 10)$; $e_3 = (3, 15)$; $e_4 = (2, 20)$; $e_5 = (1, 25)$; $e_6 = (0, 30)$

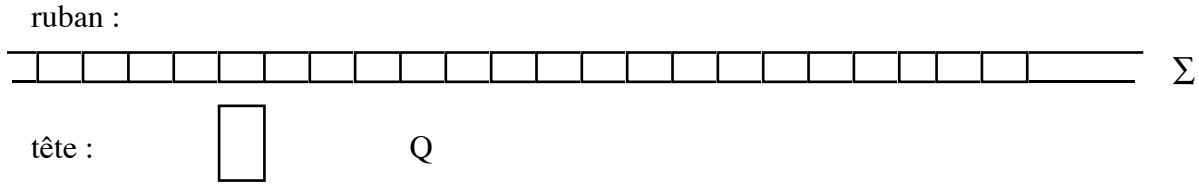
Les e_i sont appelés *états* (computationnels) ou *configurations* (on peut en effet dire que ce sont des configurations de la machine à des moments donnés). Soit E l'ensemble de tous les états possibles pour une machine donnée (cet ensemble est définissable, on peut même dénombrer ses éléments en tenant compte de la taille de la mémoire, du nombre des registres et de la taille de ces derniers). Formellement, un algorithme peut se représenter par *une fonction de transition* T de E dans E qui associe à tout état de E son état suivant dans l'éxécution de l'algorithme. Evidemment, le nombre d'états possibles en général est trop grand pour qu'on puisse définir n'importe quel algorithme de cette façon. Les états par lesquels passe l'éxécution d'un algorithme quelconque ne sont donc pas prédéfinis avant l'écriture de l'algorithme: ils sont calculés au cours de son éxécution (ce calcul faisant partie de cette dernière). Il y a pourtant une classe d'algorithmes simples et utilisables dans de nombreuses circonstances pour lesquels l'ensemble des états – très réduit – est prédéfini, l'algorithme consistant simplement alors à donner la fonction de transition. Ces algorithmes sont appelés **automates à états finis** (on devrait d'ailleurs plutôt dire: "à nombre fini d'états"). Ils ont un lien simple avec notre précédent propos ("reconnaître un langage") en ce qu'ils sont utilisables pour reconnaître une certaine classe de langages, lesquels sont justement... les langages réguliers!

5-2 Automates à états finis déterministes (AEFD)

Considérons donc un ensemble non vide d'états prédéfinis, noté Q, parmi ces états: un *état initial* q_0, et un sous ensemble non vide d'états (éventuellement réduit à un singleton) Q_f appelés *états finaux*. Représentons la mémoire d'une machine par un ruban (potentiellement illimité), sur lequel peuvent être écrits des symboles (en général 0 et 1, mais nous verrons qu'on peut travailler avec des rubans sur lesquels sont écrits des

paquets de 0 et de 1, qu'on peut alors interpréter comme les symboles d'un alphabet arbitraire). Cette petite machine est dotée d'une *tête de lecture* autorisée à lire le ruban case par case, celui-ci se déplaçant toujours dans la même direction. En fonction de son état computationnel de départ et de ce qu'elle lit sur le ruban, cette tête de lecture adopte un nouvel état (éventuellement le même). Elle s'arrêtera (et avec elle le déplacement du ruban) lorsqu'elle sera arrivé à un état final. Ceci est la description d'un automate à états finis. Nous appellerons *représentation concrète de l'automate* cette manière de décrire son fonctionnement.

Représentation concrète d'un automate à états finis :



Donnons-en une définition formelle.

Définition : on appelle **automate à états finis déterministe** la donnée de :
- un ensemble fini non vide Q (états),
- un élément q_0 de Q (état initial),
- un sous-ensemble non vide Q_f de Q (états finaux),
- un alphabet terminal $\sum_T$
- une fonction de transition $\tau : Q \times \sum_T \rightarrow Q$

5-3 Configuration et dérivabilité d'une configuration dans un AEFD

Tout instant d'un calcul est caractérisé par: un état et la suite des symboles qui restent à lire sur le ruban à cet instant. Nous appellerons donc *configuration* tout élément de $Q \times \sum_T^*$. Nous admettons qu'au départ, la tête de lecture est positionnée sur la première case à gauche du ruban: elle a donc à traiter toute la suite des symboles marqués sur le ruban (donc tout un mot w de $\sum^*$), et de plus elle est dans l'état initial q_0. Nous appelons donc *configuration initiale* le couple: (q_0, w).

Définition : une configuration (q', w') *suit immédiatement* une configuration (q, w), ce que nous écrirons:

$$(q, w) \vdash (q', w')$$

si et seulement si :

- $w = \alpha.w'$

et

- $q' = \tau(q, \alpha)$

Définition : étant donné un automate M, une configuration (q', w') est *dérivable* à partir d'une configuration (q, w) si et seulement si il existe une suite finie[1] de configurations (s_0, w_0), (s_1, w_1), (s_2, w_2), ..., (s_k, w_k) telle que:

$$(s_0, w_0) = (q, w) \vdash (s_1, w_1) \vdash (s_2, w_2) \vdash .... \vdash (s_k, w_k) = (q', w')$$

On écrira : $(q, w) \vdash_M^* (q', w')$

Définition : un mot w est dit *accepté* par un automate M si et seulement si on a:

$(q_0, w) \vdash_M^* (q_f, \varepsilon)$ où q_0 est l'état initial et q_f un état final

[1] donc éventuellement vide (k = 0)

Définition : le langage L accepté par un automate M est l'ensemble de tous les mots acceptés par M: $L(M) = \{w; (q_0, w) \vdash_M^* (q_f, \varepsilon)\}$

Exemples:

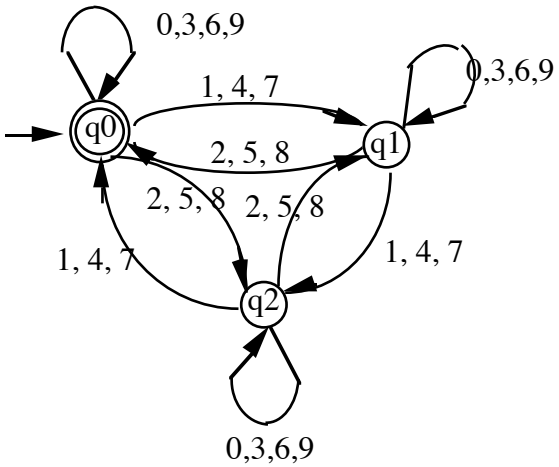
1) soit $M = < \sum, Q, q_0, Q_f, \tau >$ l'automate déterministe défini par :

$\sum = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$Q = \{q_0, q_1, q_2\}$
$Q_f = \{q_0\}$

τ :	0, 3, 6, 9	1, 4, 7	2, 5, 8
q_0	q_0	q_1	q_2
q_1	q_1	q_2	q_0
q_2	q_2	q_0	q_1

Remarque: on peut représenter tout Automate à Etats Finis par un **diagramme**.

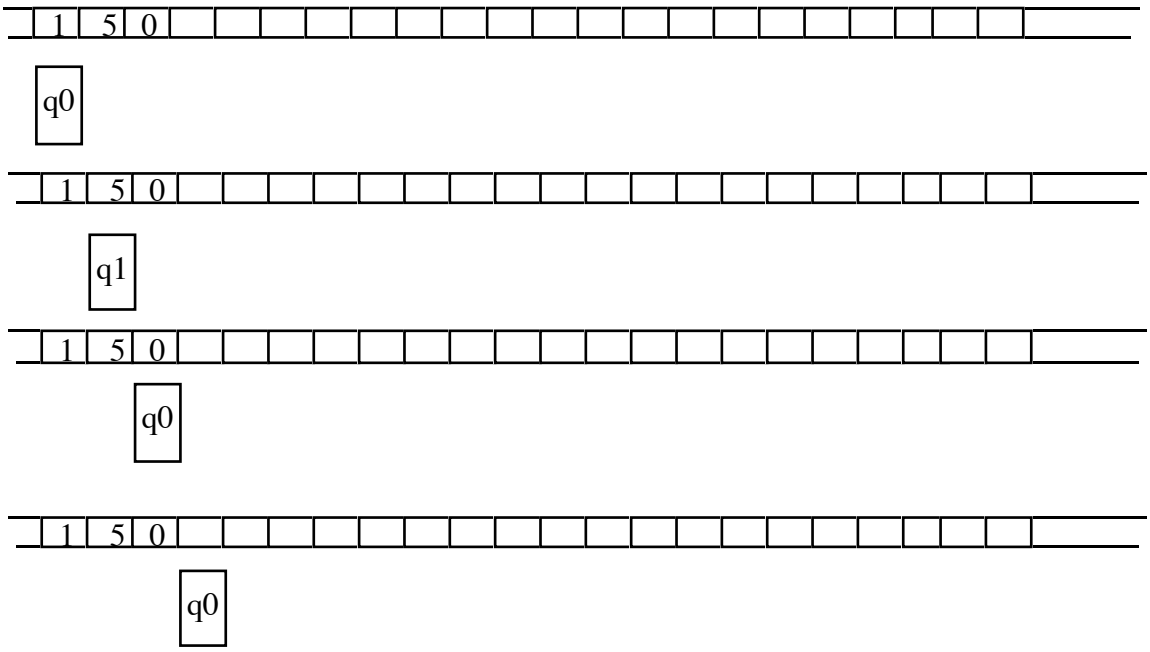
Diagramme de transition de cet automate :



Démontrons que le mot '150' est accepté par M:

début : $(q_0, 150)$
comme on a : $\tau(q_0, 1) = q_1$, on a: $(q_0, 150) \vdash (q_1, 50)$, puis:
$(q_1, 50) \vdash (q_0, 0) \vdash (q_0, \varepsilon)$ donc '150' est accepté.

Représentation concrète de cette acceptation :



En revanche: $(q_0, 136) \vdash (q_1, 36) \vdash (q_1, 6) \vdash (q_1, \varepsilon)$, comme q_1 n'est pas un état final, '136' n'est pas accepté.

Question : comment caractériser ici l'ensemble des mots (= le langage) accepté par M?

2) Trouver un automate à états finis pour chacun des langages suivants:

$L_1 = \{0^p 1^n;\ p, n \geq 0\}$;

$L_2 = \{a^p b^k c^m;\ p, k, m \geq 0\}$;

L_3 = l'ensemble des mots sur {a, b} ayant deux 'a' consécutifs ou deux 'b' consécutifs.

5-4 Automates à Etats Finis Non-Déterministes

Dans ce paragraphe, nous introduisons la notion de non-déterminisme, c'est-à-dire le fait qu'un calcul laisse place à un choix possible à certaines de ses étapes. Evidemment, d'un point de vue strictement informatique, cela n'est pas très désirable! Toutefois, comme nous le verrons plus loin, nous pouvons sans risque nous permettre cette "fantaisie" dans le cas des automates à Etats Finis puisque nous démontrerons qu'en fait, tout automate non-déterministe peut être rendu déterministe. L'intérêt alors du non-déterminisme est qu'il nous permet quelquefois d'avoir une description plus simple de certains langages. Les Automates à Etats Finis Non-Déterministes (AEFND) autorisent:

- plusieurs transitions correspondant à une même lettre dans chaque état (autrement dit une fonction de transition t non plus de $Q \times \sum_T$ dans Q mais de $Q \times \sum_T$ dans $\wp(Q)$),
- des transitions sur le mot vide (ie sans avancer sur le mot d'entrée)
- des transitions sur des mots de longueur supérieure à 1.

Définition : on appelle **automate à états finis non déterministe** la donnée de :
– un ensemble fini non vide Q (états),
– un élément q_0 de Q (état initial),
– un sous-ensemble non vide Q_f de Q (états finaux),

– un alphabet terminal $\sum$

– une fonction de transition $\tau : Q \times \sum^* \rightarrow \wp(Q)$, ou encore (ce qui revient au même) une relation ternaire Δ incluse dans $Q \times \sum^* \times Q$

Définition : une configuration (q', w') suit immédiatement une configuration (q, w), ce que nous écrirons:

$(q, w) \vdash (q', w')$

si et seulement si :

- $w = uw'$ $\quad (u \in \sum^*)$

et

- $(q, u, q') \in \Delta$

Exemple : soit M l'automate non déterministe défini par:

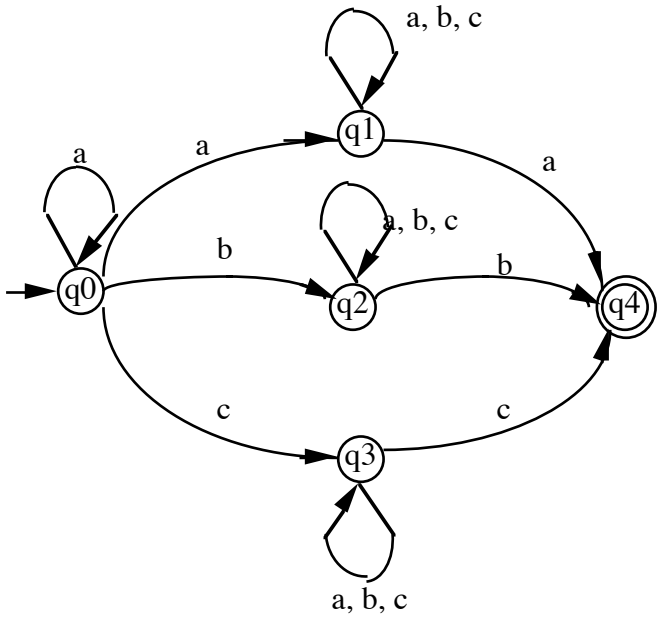
$\sum = \{a, b, c\}$
$Q = \{q_0, q_1, q_2, q_3, q_4\}$
$Q_f = \{q_4\}$

τ :

	a	b	c
q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_3\}$
q_1	$\{q_1, q_4\}$	$\{q_1\}$	$\{q_1\}$
q_2	$\{q_2\}$	$\{q_2, q_4\}$	$\{q_2\}$
q_3	$\{q_3\}$	$\{q_3\}$	$\{q_3, q_4\}$
q_4	Ø	Ø	Ø

les mots 'bab', 'aca' sont acceptés par cet automate mais pas, par exemple le mot 'cbbca'. (En fait, le langage accepté par cet autiomate est l'ensemble des mots dont la dernière lettre est une lettre qui figure déjà dans le mot).

Diagramme de transition :



5-5 Elimination du non-déterminisme

Une question qui se pose est: les automates non déterministes acceptent-ils plus de langages que les automates déterministes? La réponse est non, puisque comme il a été suggéré plus haut, à tout AEFND correspond un AEFD qui accepte le même langage. C'est ce que nous allons maintenant démontrer.

Théorème : *pour tout langage L accepté par un AEFND, il existe aussi un AEFD qui accepte L.*

Démonstration : Soit M_1 un automate non-déterministe et M_2 l'automate déterministe à construire. Il y a deux possibilités permises par les AEFND qui ne le sont pas par les AEFD : d'une part l'étiquetage des arcs de transition par des mots quelconques (y compris vide) de $\sum^*$, d'autre part le choix entre plusieurs états à partir de certaines configurations. Considérons d'abord la première source d'indéterminisme. Evidemment, si un arc de transition est étiqueté par un mot w tel que $|w|>1$, il est aisé de le remplacer par une suite d'arcs, avec l'introduction d'autant nouveaux états que nécéssaire. Soit par exemple: $w = x_1x_2...x_n$, on introduira $n-1$ états supplémentaires: q_1, ..., q_{n-1} et on remplacera toute transition (q_I, w, q_0) par une suite de transitions:
(q_I, x_1, q_1), (q_1, x_2, q_2), ... , (q_{n-1}, x_n, q_0).

En ce qui concerne les transitions vides, étant donnée une transition (q, ε, q'), nous formerons un seul état avec tous les états accessibles à partir de q par des transitions vides.

Envisageons maintenant la deuxième source d'indéterminisme. Selon l'idée déjà suggérée pour l'élimination des transitions vides, nous allons remplacer les états de M_1 par des *ensembles* d'états. Autrement dit, si, à partir d'un état q, on peut aller, en lisant un symbole α, aussi bien vers q_1 que vers q_2, nous regrouperons q_1 et q_2 en un seul état et nous dirons que, à partir de q, en lisant α, on va (de façon déterministe) vers l'état $\{q_1,q_2\}$. Evidemment, nous tenons compte des transitions vides. Autrement dit, nous désignons par E(q) l'ensemble d'états accessibles à partir de q par une transition vide. L'état initial, dans M_2, est désormais $E(q_0)$. Puis, nous construisons inductivement l'ensemble des états de M_2 de la façon suivante. Soit $s(q_0, x)$ l'ensemble des états de M_1 auxquels on accède à partir d'un état de $E(q_0)$ par une transition étiquetée x (où $x\in\sum$), soit $E(q_0,x)$ l'ensemble obtenu à partir de $s(q_0, x)$ en ajoutant tous les états de M_1 auxquels on accède à partir d'un état de $E(q_0,x)$ par une transition vide, on rajoute $E(q_0,x)$ à M_2 si $E(q_0,x) \neq \emptyset$, ainsi que la transition $(E(q_0), x, E(q_0,x))$. Et on continue de cette manière: pour tout état Q_j figurant déjà dans l'ensemble des états de M_2, et tout terminal x, on fabrique un état E(Qj, x) de la même façon. On arrête cette construction quand il n'y a plus de nouvel état créé. Ce moment arrive nécessairement puisque le nombre d'états de M_2 est borné par le nombre de parties de l'ensemble des états de M_1 (donc un nombre nécessairement fini).

Reste à définir les états finaux de M_2. Un mot est accepté dans M_1 ssi on arrive à une configuration (q_f, ε) où q_f est un état final de M_1. Cela correspond alors à un chemin dans M_2 arrivant dans un état qui contient un état final de M_1. Donc sont finaux dans M_2 tous les états qui contiennent au moins un état final de M_1.

Selon cette construction, il est clair que M_2 est déterministe.

Il reste à prouver que $L(M_1) = L(M_2)$.

1- <u>$L(M_2)$ inclus dans $L(M_1)$</u> :

soit w accepté par M_2: alors on a:

$(E(q_0), w) \vdash (\mathbf{q}_1, w_1) \vdash ... \vdash (\mathbf{q}_k, w_k) \vdash (\mathbf{q}_f, \varepsilon)$, avec $\mathbf{q}_f \in Q'_f$.

D'après la définition de $Q2_f$, il existe $p \in \mathbf{q}_f \cap Q1_f$. D'où:

si $(\mathbf{q}_k, w_k) \vdash (\mathbf{q}_f, \varepsilon)$, alors il existe $q_k \in \mathbf{q}_k$ et $p \in \mathbf{q}_f \cap Q_f$ tels que $\tau(q_k, w_k) = p$.

Donc $(q_k, w_k) \vdash_M^* (p, \varepsilon)$. Le même raisonnement est alors possible pour passer de w_{k-1} à w_k.

Remarque: rigoureusement, on ferait une démonstration par récurrence sur la longueur k de la chaîne des transitions.

2- L(M) inclus dans L(M') : en exercice !

Exemple : soit $M = < \{a, b\}, \{q_0, q_1, q_2\}, q_0, \{q_2\}, \tau >$ avec τ définie par :

τ :	a	b
q_0	$\{q_0, q_1\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\{q_0\}$
q_2	$\{q_0\}$	$\{q_1, q_2\}$

Q' contient tous les sous-ensembles sauf Ø. Et on a:

$\tau'(\{q_0\}, a) = \{q_0, q_1\}$, $\tau'(\{q_0\}, b) = \{q_2\}$, $\tau'(\{q_1\}, a) = \{q_1\}$, $\tau'(\{q_1\}, b) = \{q_0\}$, $\tau'(\{q_2\}, a) = \{q_0\}$, $\tau'(\{q_2\}, b) = \{q_1, q_2\}$, puis:

$\tau'(\{q_0, q_2\}, a) = \tau(q_0, a) \cup \tau(q_2, a) = \{q_0, q_1\}$

etc.

Les états finaux sont : $\{\{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$.

Exercice : éliminer le non-déterminisme de l'automate donné plus haut en exemple.

5- Langages réguliers et automates à états finis

Nous allons prouver dans ce qui suit que toute expression régulière E définit un automate à états finis M_E qui accepte le langage qu'elle désigne et que, réciproquement, tout automate M accepte un langage L qui peut se représenter par une expression régulière E_M.

5-1 Automate associé à une expression régulière

Théorème : *pour toute expression régulière E, il existe un automate à états finis qui accepte le langage qu'elle définit.*

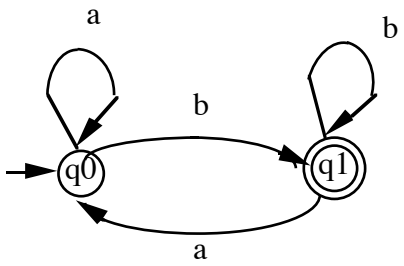
Démonstration : Soit une expression régulière E. On a donné précédemment (§3-1-3) la définition inductive des expressions régulières. Il nous faut donc la suivre pour définir un automate associé à une expression régulière. Cas de base: à Ø, on peut associer un automate M ayant un état initial mais pas d'état final (i-e: $Q_f = Ø$), à ε, on peut associer une transition vide d'un état initial q_0 vers un état final q_f (ou bien un automate consistant simplement en un seul état, qui est à la fois initial et final). A une lettre $x \in \sum$, on peut associer une transition étiquetée x allant de q_0 à q_f. Schéma inductif: soit M_1 et M_2 les automates associés à α et à β, qui sont deux expressions régulières. Pour $\alpha \cup \beta$: ajouter un état initial q_0, relié par une transition vide aux états initiaux respectifs de M_1 et M_2. Les états finaux du nouvel automate sont la réunion de ceux de M_1 et de ceux de M_2. Pour $\alpha\beta$: relier tout état final de M_1 à l'état initial de M_2 par une transition vide. Les états finaux sont ceux de M_2. Pour $(\alpha)^*$, nous introduisons une transition vide de chaque état final de l'automate M associé à α vers son état initial, afin d'introduire autant de boucles

que nécessaire. Evidemment, l'automate obtenu peut être simplifié, et en particulier rendu déterministe grâce au théorème précédent.

5-2 Expression régulière associée à un automate

Théorème : *tout langage accepté par un automate à états finis peut être défini par une expression régulière.*

Démonstration : par induction sur le diagramme d'un automate.
L'idée est de trouver une méthode permettant d'énumérer tous les chemins possibles sur le diagramme de transition d'un automate. Prenons l'exemple de l'automate suivant:



exemples de mots acceptés :
b ; ab ; aaaaaab ; aaaabbbbb; bbbbbaaaaab ; bbbbbbaaaaabbbbbbaaaaabbbbb ; babababab etc.

Intuitivement, il semble possible de dire que le langage accepté est dénoté par l'expression régulière: a*b(a*b)*, mais comment y arriver avec méthode?

Une façon de faire est la suivante:

Soit un automate M avec un ensemble d'états Q que nous renumérotons 1, 2, ..., n : $\{q_1, q_2,..., q_n\}$. Définissons l'ensemble R(i, j, k) par : l'ensemble des mots permettant de passer de l'état qi à l'état qj en passant seulement par les états $\{q_1, ..., q_{k-1}\}$. Ainsi, dans notre exemple, a-t-on :

$R(1, 1, 1) = \{\varepsilon, a\}$

car pour passer de l'état q_1 (ex-q_0) à lui-même en ne passant par aucun autre état, on ne peut que: soit lire ε (aucune transition n'est prise), soit lire a. De même:

$R(1, 2, 1) = \{b\}$

car pour aller de q_1 à q_2 sans passer par un autre état, on est obligé de passer par l'arc étiqueté b,

$R(1, 1, 2) = \{e, a, aa, aaa, ...\}$ autrement dit : a*
$R(1, 2, 2) = \{b, ab, aab, ...\}$ autrement dit : a*b
$R(2, 1, 1) = \{a\}$ (e.r. : a)
$R(2, 1, 2) = \{a, aa, aaa, ...\} = aa^*$
$R(2, 2, 1) = \{b\}$ (e.r; : b)
$R(2, 2, 2) = \{\varepsilon, b, ab, a...ab, ...\} = \varepsilon + a^*b$

Il existe un moyen de construire inductivement ces ensembles:

• **lorsque k = 1**: (il s'agit des chemins ne transitant par aucun état), on a:

$R(i, j, 1) = \{w; (q_i, w, q_j) \in \Delta\}$ si $i \neq j$,
$= \{\varepsilon\} \cup \{w; (q_i, w, q_i) \in \Delta\}$ si $i = j$

• **lorsque k > 1** : les mots qui permettent de passer de q_i à q_j en ne transitant que par les états de l'ensemble $\{q_1, ..., q_k\}$ (donc: les éléments de R(i, j, k+1)) se décomposent en :

– ceux qui permettent de passer de q_i à q_j en ne transitant que par les états de l'ensemble $\{q_1, ..., q_{k-1}\}$ (donc: les éléments de R(i, j, k)),

– ceux qui permettent de passer de q_i à q_k, puis de q_k à q_k un certain nombre de fois et finalement de q_k à q_j, en ne transitant que par les états de l'ensemble$\{q_1, ...,q_{k-1}\}$ autrement dit:

$$R(i, j, k+1) = R(i, j, k) \cup R(i, k, k)R(k, k, k)^*R(k, j, k)$$

Le langage accepté par M est alors:

$$L(M) = \bigcup_{q_j \in Q_f} R(1, j, n+1)$$

En appliquant cette méthode à notre exemple, on voit qu'on construirait automatiquement le tableau suivant:

	k = 1	k = 2
R(1, 1, k)	ε + a	(ε + a) + (ε + a)(ε + a)*(ε + a)
R(1, 2, k)	b	b + (ε + a)(ε + a)*b
R(2, 1, k)	a	a + a(ε + a)*(ε + a)
R(2, 2, k)	ε + b	(ε + b) + a(ε + a)*b

Finalement, L(M) = R(1, 2, 3) = [b + (ε + a)(ε + a)*b] + [b + (ε + a)(ε + a)*b]
[(ε + b) + a(ε + a)*b]*[(ε + b) + a(ε + a)*b]

Expression qiui peut évidemment se simplifier grâce aux identités remarquables sur les expressions régulières:

chaque terme peut d'abord se simplifier :

b + (ε + a)(ε + a)*b = (ε + a)(ε + a)*b (car b est déjà contenu dans (ε + a)(ε + a)*b)
= [(ε + a)*b] + [a(ε + a)*b] = a*b + aa*b = a*b

(ε + b) + a(ε + a)*b = ε + b + aa*b = ε + a*b

donc L(M) = a*b + a*b(ε + a*b)*(ε + a*b) = a*b(a*b)*

6- Langages de type 3 et automates à états finis

6-1 Grammaire associée à un AEF

Théorème : *tout langage accepté par un automate à états finis est engendré par une grammaire de type 3 étendu.*

Démonstration : soit M = <Q, ∑, q_0, Q_f, Δ> un AEF sans transition vide ni transition étiquetée par un mot de plus d'une lettre. Construisons une grammaire de la façon suivante :

V_N est un ensemble en bijection avec Q,
V_T = ∑,
S (l'axiome) est l'élémént de VN associé à q0,

à tout triplet (q_i, a, q_j) associons la règle :

X_{qi} ----> aX_{qj} (où X_q désigne le non-terminal associé à q)

de plus, si $q_j \in Q_f$, ajoutons la règle:

X_{qi} ----> a

et si $q_0 \in Q_f$, ajoutons la règle:

X_{q0} ----> ε

La grammaire obtenue est de type 3. Soit $w \in L(M)$, alors il existe une suite de configurations (q_0, w_0), ..., (q_k, w_k) représentant l'acceptation de w, telle que:

$w_0 = w$; $w_k = \varepsilon$, $q_k \in Q_f$

Procédons par induction sur la longueur de l'acceptation d'un mot.

Hypothèse d'induction : pour toute acceptation ((q_0, w_0), ..., (q_m, ε)) telle que $m \leq k$, il existe une dérivation de w_0 sous le non-terminal X_{q0} de la grammaire G construite par le procédé précédent.

Cas de base :
si k=1, alors ou bien w consiste en une lettre a et on a une transition (q_0, a, q_1) et $q_1 \in Q_f$. Il existe donc dans la grammaire, une règle : X_{q0} ----> a. En utilisant cette règle à partir du symbole X_{q0} on obtient : X_{q0} |--* a = w, ou bien w = ε et $q_0 \in Q_f$, auquel cas, on a la règle X_{q0} ----> ε, qui permet de dériver w = ε.

Pas d'induction :
si k>1 : alors par hypothèse d'induction, le mot w_1 peut être engendré sous X_{q1} dans la grammaire G: cf X_{q1} |--* w_1, comme on passe de q_0 à q_1 par une transition (q_0, a, q_1) où a est la lettre telle que $w = aw_1$, on a par définition de la grammaire, la règle :

X_{q0} ----> aX_{q1}

et donc la dérivation:

X_{q0} |-- aX_{q1} |--* $aw_1 = w$

– **Exercice** : on démontrera la réciproque, c'est-à-dire la grammaire construite n'engendre QUE les mots acceptés par l'automate (induction sur la longueur d'une dérivation).

6-2 AEF associé à une grammaire de type 3

Théorème : *tout langage engendré par une grammaire de type 3 étendu est accepté par un automate à états finis.*

Démonstration: il s'agit de la construction inverse. Soit G une grammaire de type 3, avec éventuellement une règle S ----> ε. Définissons un AEF M de la manière suivante :

Q en bijection avec V_N,
$\sum = V_T$,
q_0 est l'élément de Q associé à S,
à toute règle de grammaire X ----> aY, on associe le triplet (q_X, a, q_Y), où
q_Z désigne l'état associé bijectivement à Z,

à toute règle de grammaire X ----> a, on associe un triplet (q_X, a, q_f) où $q_f \in Q_f$,

si la règle S ----> ε figure dans la grammaire, on pose que $q_0 \in Q_f$.

Alors, soit w un mot engendré par G, prouvons qu'il est accepté par M.

Hypothèse d'induction : tout mot w dérivé à partir d'un non-terminal X dans G avec une longueur de dérivation ≤ n est accepté par l'automate M à partir de l'état q_X.

Cas de base: n = 1 : $w \in V_T$ et il existe une règle X ----> w, alors l'automate possède la transition (q_X, w, q_f) où $q_f \in Q_f$, ce qui prouve que w est accepté par M à partir de q_X, l'acceptation est:

(q_X, w) ; (q_f, ε)

De même, si w = ε et si w est dérivé à partir de S, alors il y a la règle S ----> ε dans la grammaire, ce qui entraîne que q_0 est final, d'où l'acceptation de ε = w par M.
Pas d'induction : soit w engendré sous un non-terminal Y avec une longueur de dérivation n+1, alors la première règle appliquée est: Y ----> aX et w = aw' et w' est engendré sous X, avec une dérivation de longueur nécessairement inférieure ou égale à n. Donc par hypothèse d'induction, il y a une acceptation de w' à partir de l'état q_X, ce qui s'écrit:

(q_X, w') |--* (q_f, ε)

Or, la règle Y ----> aX entraîne la présence dans l'automate de la transition (q_Y, a, q_X), d'où l'acceptation:

(q_Y, w) |-- (q_X, aw) |--* (q_f, ε)

autrement dit: w est accepté à partir de l'état q_Y.

Exercice : prouver la réciproque, à savoir que M n'accepte QUE les mots engendrés par G.

7- Exercices :

7-1 :

– Trouver une grammaire de type 3 engendrant le langage accepté par l'automate du §3,
– Trouver une expression régulière dénotant ce même langage.

7-2 : Trouver un Automate à Etats Finis Déterministe acceptant le langage accepté par l'expression régulière : 0(10)*,

– idem pour le langage des mots sur {0, 1} qui contiennent au moins une fois exactement quatre '0' qui se suivent,

– trouver une expression régulière, la plus simple possible, qui dénote ce même langage.

7-3 : Voici un problème bien connu:

> *Un homme (H) est sur la rive gauche d'un fleuve, avec un loup (L), une chèvre (B) et un chou (C). Il veut transporter tout ce beau monde de l'autre côté. Pour cela il dispose d'une barque, mais la taille de celle-ci ne permet pas de contenir plus d'un objet ou animal en plus de sa personne. Il peut faire autant de traversées qu'il désire, dans un sens ou dans l'autre mais à chacune, il ne transporte qu'un objet ou animal. S'il laisse le loup et la chèvre seuls sur une rive, le loup ne manquera pas de manger la chèvre, et s'il laisse la chèvre et le chou, la chèvre bouffera le chou!*

Trouver une solution à ce problème qui évite ces deux incidents fâcheux, sous la forme d'un Automate à Etats Finis qui accepte seulement les mots correspondant à une solution. A chuque voyage, dans un sens ou dans l'autre, on associe une lettre: h si l'homme voyage seul, l s'il voyage avec le loup, b avec la chèvre et c avec le chou. {h, b, l, c} est donc l'alphabet de l'automate.