Automates

Notions de base

Notes de cours, IR1, 2009

Sylvain Lombardy

1 Alphabets, mots, langages

Un *alphabet* A est un ensemble fini de symboles appelés *lettres*.
Un mot est une suite finie de lettres.
On note A^* l'ensemble des mots que l'on peut former avec des lettres de l'alphabet A.
Un *langage* sur un alphabet A est un ensemble (fini ou infini) de mots de A^*.

Exemples :

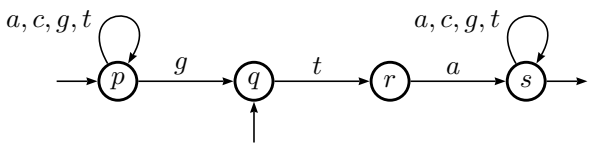
1. L'alphabet A_1 permettant d'écrire les mots usuels comporte environ soixante-dix lettres (minuscules, majuscules, lettres accentuées). L'ensemble des mots du français est un *langage* sur A_1. Le mot *carichon* appartient à A_1^* mais n'est pas un mot français.
2. Les séquences d'ADN se représentent sur un alphabet $A_2 = \{G, A, T, C\}$. Une séquence d'ADN est un mot de A_2^*, mais un mot de A_2^* n'est pas forcément une séquence correcte.
3. On peut prendre comme alphabet A_3 l'ensemble des mots du français (plusieurs centaines de milliers de mots). Un "mot" est alors une suite finie d'éléments de cet alphabet, c'est donc ce qu'on appelle habituellement une phrase. Les phrases correctes sont un langage sur A_3.

2 Automates non déterministes

Définition Un automate (non-déterministe) $\mathcal{A}$ sur un alphabet A est défini par les éléments suivants :
– un ensemble fini d'*états* noté Q ;
– un ensemble fini E de *transitions*, chaque transition étant définie par un état de départ p, un état d'arrivée q et une *étiquette* a appartenant à A ; on note une telle transition (p, a, q).
– un sous-ensemble de Q appelé ensemble d'états *initiaux* noté I ;
– un sous-ensemble de Q appelé ensemble d'états *terminaux* noté T.
Un tel automate est noté $\mathcal{A} = \langle Q, A, E, I, T\rangle$. Ainsi, quand on définit un automate, on donne successivement son ensemble d'états, l'alphabet sur lequel il est défini, son ensemble de transitions, son ensemble d'états initiaux et son ensemble d'états terminaux.

Représentation graphique Un automate est représenté comme un graphe orienté étiqueté : les états sont des ronds (à l'intérieur desquels on écrit le nom de l'état), les transitions sont représentées par des flèches qui partent de l'état de départ et pointent sur l'état d'arrivée, on indique au milieu de la flèche la lettre qui en est l'étiquette. Un état initial est signalé par une petite flèche qui pointe sur l'état, un état terminal par une petite flèche qui en part (et qui pointe dans le vide).

Exemple :



L'automate ci-dessus est l'automate $\mathcal{A} = \langle Q, A, E, I, T\rangle$, avec $Q = \{p, q, r, s\}$, $A = \{a, c, g, t\}$, $E = \{(p, a, p), (p, c, p), (p, g, p), (p, t, p), (p, g, q), (q, t, r), (r, a, s), (s, a, s), (s, c, s), (s, g, s), (s, t, s)\}$, $I = \{p, q\}$ et $T = \{s\}$. On remarque que si plusieurs transitions ont les mêmes états de départ et d'arrivée, on ne dessine qu'une flèche en indiquant les différents étiquettes.

Chemins et langage acceptés Un *chemin* dans l'automate est une suite de transitions consécutives, c'est-à-dire une suite $(p_1, a_1, q_1), (p_2, a_2, q_2), ..., (p_n, a_n, q_n)$ telle que $q_i = p_{i+1}$. L'état de départ du chemin est l'état de départ de la première transition, l'état d'arrivée du chemin est celui de la dernière transition. L'étiquette d'un chemin est le mot $a_1a_2...a_n$ obtenu en concaténant les étiquettes des transitions du chemin.

Un chemin est *réussi* s'il commence dans un état initial et s'il arrive dans un état terminal. Un tel chemin est parfois appelé *calcul*.

Un mot est *accepté* par l'automate s'il existe un chemin réussi dont il est l'étiquette. Le langage *reconnu* par l'automate est l'ensemble des mots acceptés par l'automate.

Deux automates sont *équivalents* s'ils reconnaissent le même langage.

Exemple : Le mot *cgtag* est accepté par l'automate ci-dessus car il est l'étiquette du chemin réussi suivant :

$$\rightarrow p \xrightarrow{c} p \xrightarrow{g} q \xrightarrow{t} r \xrightarrow{a} s \xrightarrow{g} s \rightarrow$$

Si on regarde attentivement l'automate, on constate que tous les chemins réussis partent des états p ou q et arrivent en s. Les chemins réussis partant de p sont exactement ceux qui sont étiquetés par un mot qui contient gta (on dit que gta est un *facteur* du mot) ; ceux qui partent de q sont étiquetés par des mots qui commencent par ta (on dit que ta est un préfixe du mot). Le langage reconnu par cet automate est donc l'ensemble des mots sur $\{a, c, g, t\}$ qui soit contiennent le facteur gta, soit commencent par le préfixe ta.

Décider si un mot est accepté par un automate L'algorithme qui permet de décider ceci consiste à lire le mot lettre par lettre et à calculer tous les états que l'on peut atteindre à partir d'un état initial en ayant lu ces lettres.

Formellement, soit $w = w_1w_2...w_k$ un mot (w_i est la i-ème lettre du mot) et $\mathcal{A} = \langle Q, A, E, I, T\rangle$ un automate. On veut décider si w est accepté par $\mathcal{A}$. X_i est l'ensemble des états que l'on peut atteindre en ayant lu $w_1w_2...w_i$.

Calcul de X_k :

– $X_0 = I$;

– pour i de 1 à k

    $X_i = \emptyset$

    pour tout p dans X_{i-1}

        pour tout q tel que (p, w_i, q) est dans E

            $X_i = X_i \cup \{q\}$

Le mot w est accepté si et seulement si $X_k \cap T \neq \emptyset$:

– pour tout p dans X_k

    si p est dans T

        retourne VRAI (w est accepté)

    retourne FAUX

La complexité de cet algorithme est $O(kn^2)$, où k est la longueur du mot et n est le nombre d'états de l'automate. En effet, X_i peut contenir n états et il peut y avoir n transitions partant de p étiquetées par w_i.

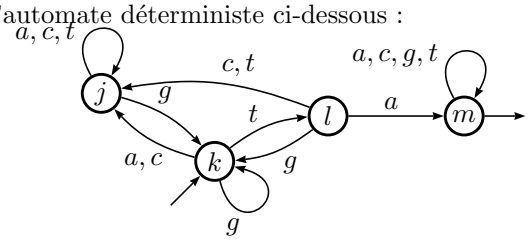
États utiles, automates émondé Un état p d'un automate $\mathcal{A}$ est *utile* s'il existe un chemin réussi passant par p. Si un état n'est pas utile, on peut le supprimer sans modifier le langage reconnu.

Un automate est *émondé* si tous ses états sont utiles. On peut émonder un automate en supprimant tous ses états inutiles.

3 Automates déterministes

Un automate est *déterministe* s'il n'a qu'un seul état initial et si, pour tout état p, toute lettre a, il y a *au plus une* transition partant de p étiquetée par a.

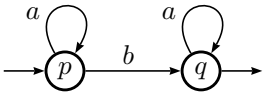
Exemple : L'automate présenté précédemment *n'est pas* déterministe pour deux raisons : d'une part il possède deux états initiaux p et q et d'autre part, il y a deux transition étiquetées par g qui partent de p, l'une restant en p, l'autre arrivant en q. Cet automate est équivalent à l'automate déterministe ci-dessous :



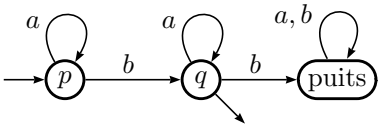
Dans un automate déterministe, si on veut lire un mot donné, à aucun moment on n'a de choix à faire. On doit commencer dans l'unique état initial, et lorsqu'on lit une lettre, il n'y a qu'une seule transition que l'on peut emprunter.

Automate complet et fonction de transition Un automate déterministe est *complet* si, pour tout état p, toute lettre a, il y a *exactement une* transition partant de p étiquetée par a.

Exemple : L'automate déterministe suivant, sur l'alphabet $\{a, b\}$ n'est pas complet :



En effet, l'état q ne comporte pas de transition sortante étiquetée par b. En ajoutant un état inutile (puits), on peut toujours rendre un automate déterministe complet :



Si $\mathcal{A} = \langle Q, A, E, \{i\}, T\rangle$ est un automate déterministe complet, on peut définir la *fonction de transition* δ, qui, étant donné un état p et une lettre a , donne le *successeur* de p par a, c'est-à-dire l'unique état q tel que (p, a, q) est une transition de $\mathcal{A}$: $\delta(p, a) = q$. On peut alors noter $\mathcal{A} = (Q, A, \delta, i, T)$.

Décider si un mot est accepté par un automate Cet algorithme, comme le précédent consiste à lire le mot à accepter lettre par lettre. Soit $w = w_1w_2...w_k$ un mot et $\mathcal{A} = (Q, A, \delta, i, T)$. On veut décider si w est accepté par $\mathcal{A}$.

– $p_0 = i$

– pour i de 1 à k

$p_i = \delta(p_{i-1}, w_i)$

– si p_k est terminal alors retourner VRAI sinon retourner FAUX

Cet algorithme est linéaire dans la longueur k du mot : $O(k)$. On le voit, il est beaucoup plus rapide de décider si un mot est accepté par un automate si celui-ci est déterministe.

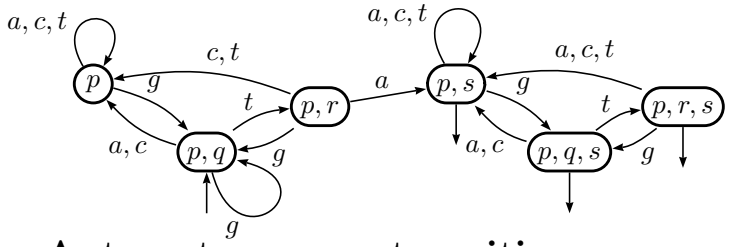
Déterminisation Il existe un algorithme qui permet de transformer n'importe quel automate en automate déterministe équivalent.

Soit $\mathcal{A} = \langle Q, A, E, I, T\rangle$ un automate non déterministe. On calcul $\mathcal{D}$ le déterminisé de $\mathcal{A}$. Chaque état de $\mathcal{D}$ correspond à un *ensemble* d'états de $\mathcal{A}$. On veut construire $\mathcal{D}$ de sorte que lorsqu'on lit un mot w à partir de l'état initial, on arrive dans un état qui correspond à l'ensemble des états de $\mathcal{A}$ dans lesquels on peut arriver à partir d'un état initial de $\mathcal{A}$ en lisant w.

Formellement, l'état initial de $\mathcal{D}$ est (correspond à) l'ensemble I. Si X est un état initial, pour chaque lettre a, il y a une transition de X à Y étiquetée par a, où $Y = \{q \mid \exists p \in X, (p, a, q) \in E\}$. L'état X est terminal si $X \cap T \neq \emptyset$.

En pratique, on construit le déterminisé de façon incrémentale. Au départ, il n'y a que l'état initial et on ajoute les états au fur et à mesure qu'on en a besoin.

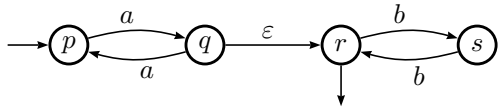
Exemple : Le déterminisé du premier automate est :



4 Automates avec ε-transitions

Il est parfois commode d'autoriser dans les automates des transitions particulières appelées ε-transitions. Ces transitions n'ont pas d'étiquette, ou, pour être plus exact, leur étiquette est le mot vide. Ainsi, l'étiquette d'un chemin comportant des ϵ-transition est le mot formé par les lettres qui étiquettent les transitions du chemin qui ne sont pas des ε-transitions.

Exemple :



Le chemin réussi

$$\rightarrow p \xrightarrow{a} q \xrightarrow{\varepsilon} r \xrightarrow{b} s \xrightarrow{b} r \rightarrow$$

est étiqueté par le mot abb. Cet automate reconnaît les mots formés d'un bloc de a de longueur impaire puis d'un bloc de b de longueur paire.

Suppression des ε-transitions On peut toujours calculer un automate équivalent sans ε-transition.

Pour cela, il faut d'abord calculer, pour chaque état, les ε-succeseurs de chaque état p, c'est-à-dire l'ensemble des états q tels qu'il existe un chemin formé uniquement de ε-transitions partant de p arrivant en q. On note cet ensemble $\mathsf{Succ}_\varepsilon(p)$.

L'algorithme est ensuite le suivant :

–pour tout état p

pour tout état q dans $\mathsf{Succ}_\varepsilon(p)$

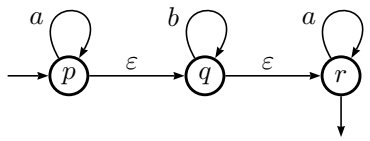
si q est final, rendre p final

pour toute transition (q, a, r) partant de q

créer la transition (p, a, r) (sauf si elle existe).

–Supprimer les ε-transitions.

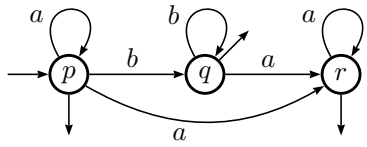
Exemple :



On calcule les ε-successeurs de chaque état :

$$\mathsf{Succ}_\varepsilon(p) = \{q, r\}, \qquad \mathsf{Succ}_\varepsilon(q) = \{r\}, \qquad \mathsf{Succ}_\varepsilon(r) = \emptyset.$$

On obtient l'automate non déterministe suivant :



Attention, pour appliquer la plupart des algorithmes (déterminisation, intersection,...) il faut d'abord supprimer les ε-transitions s'il y en a.

5 Opérations sur les langages

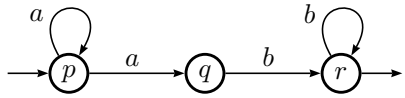
On considère un automate $\mathcal{A} = \langle Q, A, E, I, T\rangle$ qui reconnaît un langage L et un automate $\mathcal{A}' = \langle Q', A, E', I', T'\rangle$ (sur le même alphabet) qui reconnaît un langage L'.

On va voir quelles opérations sur les langages L et L' peuvent être représentées par des automates.

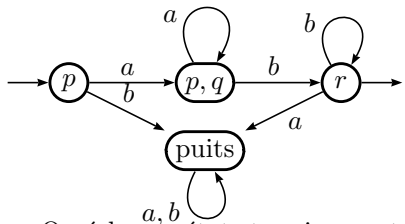
Complémentation On cherche à construire un automate $\overline{\mathcal{A}}$ qui reconnaisse le langage $\overline{L}$ des mots qui *ne sont pas* dans L.

On calcul $\mathcal{D}$ l'automate déterminisé *complet* à partir de $\mathcal{A}$. Un mot w est dans L si et seulement si lorsqu'on le lit dans $\mathcal{D}$, on arrive à un état terminal. Donc il est dans $\overline{L}$ si et seulement si lorsqu'on le lit dans $\mathcal{D}$, on arrive à un état non terminal. Le langage $\overline{L}$ est donc reconnu par l'automate $\overline{\mathcal{D}}$, obtenu à partir de $\mathcal{D}$ en échangeant états terminaux et non terminaux.

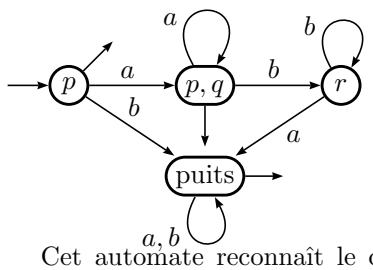
Exemple :



L'automate ci-dessus reconnaît les mots formés d'un bloc non vide de a suivi d'un bloc non vide de b. On calcule le déterminisé complet :



On échange états terminaux et non terminaux :



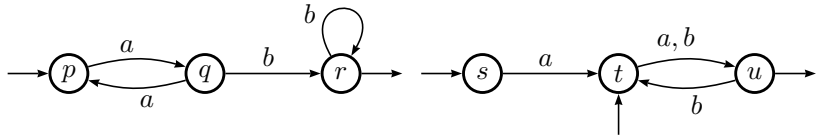
Cet automate reconnaît le complémentaire du langage de départ. On voit qu'il est crucial d'utiliser un automate complet.

Intersection de langages On veut calculer un automate qui reconnaisse le langage $L \cap L'$, c'est-à-dire le langage des mots qui sont reconnus à la fois par $\mathcal{A}$ et $\mathcal{A}'$. On construit pour cela le

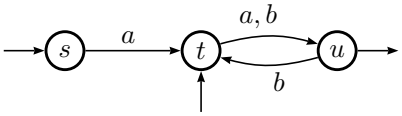
produit des deux automates dans lequel chaque chemin correspond à une paire formée d'un chemin de chacun des deux automates étiquetés par le même mot.

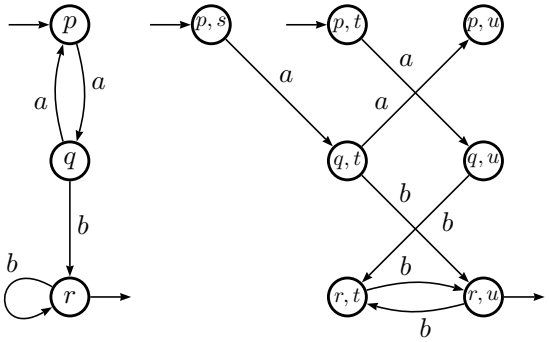
Soit $\mathcal{P} = \langle R, A, F, J, U\rangle$ l'automate défini par :
– les états de $\mathcal{P}$ sont des pairs d'états formées d'un état de $\mathcal{A}$ et d'un état de $\mathcal{A}'$;
– l'ensemble J des états initiaux est formé des paires (p, p'), où p est initial dans $\mathcal{A}$ et p' est initial dans $\mathcal{A}'$;
– pour tout état (p, p'), pour toute lettre a, s'il y a une transition (p, a, q) dans $\mathcal{A}$ et une transition (p', a, q') dans $\mathcal{A}'$, il y a un état (q, q') dans $\mathcal{P}$ et une transition de (p, p') à (q, q') étiquetée par a ;
– un état (p, p') est terminal si et seulement si les états p et p' sont terminaux.

Exemple : Soit $\mathcal{A}$ et $\mathcal{A}'$ les deux automates suivants :



Pour calculer le produit, on place les automates en ligne, le premier verticalement, le second horizontalement.





On peut remarquer que le produit de deux automates déterministes est un automate déterministe.

Union de langages On veut calculer un automate qui reconnaisse le langage $L \cup L'$, c'est-à-dire le langage des mots qui sont reconnus par $\mathcal{A}$ ou par $\mathcal{A}'$ (ou par les deux).

La solution consiste tout simplement à placer les deux automates côte à côte. On obtient alors l'automate $\mathcal{U} = \langle Q \cup Q', A, E \cup E', I \cup I', T \cup T'\rangle$.

Produit de concaténation Le produit des deux langages L et L' est le langage $L.L'$ des mots w formés d'un mot u de L et d'un mot v de L'.

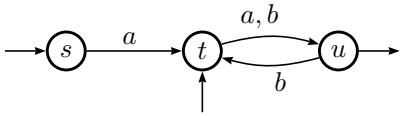
Le plus simple pour calculer un automate qui reconnaisse $L.L'$ est d'utiliser les ε-transitions. L'automate $\mathcal{C} = \langle Q \cup Q', A, E \cup E' \cup T \times I', I, T'\rangle$ est formé de la juxtaposition des automates $\mathcal{A}$ et $\mathcal{A}'$; on place une ε-transition entre chaque état terminal de $\mathcal{A}$ et chaque état initial de $\mathcal{A}'$, puis seuls les états initiaux de $\mathcal{A}$ restent initiaux, seuls les états terminaux de $\mathcal{A}'$ restent terminaux.

Étoile de Kleene – Itération Pour tout langage L, on peut définir L^n, la n-ième puissance de L, qui consiste à faire n fois le produit de concaténation de L par lui-même. L^n est donc le lanage des mots qui peuvent être découpés en n parties appartenant chacune au langage L. Par convention (et afin d'en faire un élément neutre pour la multiplication), L^0 est le langage qui contient uniquement le mot vide (c'est-à-dire le mot de longueur nulle).

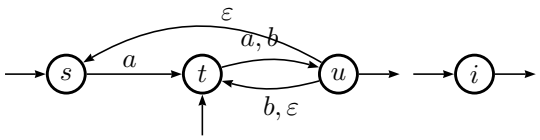
L'étoile de L notée L^* est l'union (infinie) de toutes les puissances de L ; c'est le langage des mots qui peuvent être découpées en plusieurs parties appartenant chacune au langage L.

Si l'automate $\mathcal{A}$ reconnaît le langage L, on obtient un automate (avec ε-transitions) reconnaissant L^* en reliant chaque état terminal de $\mathcal{A}$ à chaque état terminal de $\mathcal{A}$ par une ε-transition et en ajoutant un état initial et final.

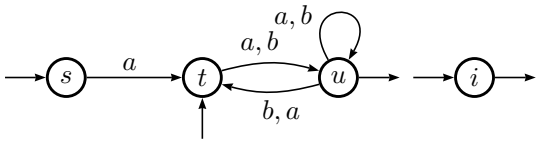
Exemple : Considérons l'automate ci-dessous :



Pour construire un automate avec ε-transitions qui reconnaisse l'étoile du langage, on relie l'état terminal aux états initiaux :



On peut ensuite supprimer les ε-transitions :



6 Automate des positions

Pour calculer l'automate des positions d'une expression, on commence par numéroter chaque occurence de lettre.

Exemple : $E = (a \cup ba)^*(\varepsilon \cup b)$ devient $\bar{E} = (a_1 \cup b_2 a_3)^*(\varepsilon \cup b_4)$.

On dira que cette expression a 4 positions : $\{1, 2, 3, 4\}$ et que la lettre en position 3, par exemple, est un a.

On calcule sur l'expression les fonctions Null, First, Last et Follow.

Null est une fonction à valeur booléenne qui indique si le langage représenté par une expression contient le mot vide ; elle est inductivement définie par :

$$\begin{aligned}
\mathsf{Null}(\emptyset) &= \text{False} \\
\mathsf{Null}(\varepsilon) &= \text{True} \\
\mathsf{Null}(a_p) &= \text{False} \\
\mathsf{Null}(E \cup F) &= \mathsf{Null}(E) \text{or} \mathsf{Null}(F) \\
\mathsf{Null}(EF) &= \mathsf{Null}(E) \text{and} \mathsf{Null}(F) \\
\mathsf{Null}(E^*) &= \text{True}
\end{aligned}$$

First est une fonction qui retourne un ensemble de positions. Elle indique où sont situées les lettres qui peuvent apparaître comme première lettre d'un mot décrit par l'expression.

$$
\begin{aligned}
\mathsf{First}(\emptyset) = \mathsf{First}(\varepsilon) &= \emptyset \\
\mathsf{First}(a_p) &= \{p\} \\
\mathsf{First}(E \cup F) &= \mathsf{First}(E) \cup \mathsf{First}(F) \\
\mathsf{First}(EF) &= \begin{cases} \mathsf{First}(E) \cup \mathsf{First}(F) & \text{si } \mathsf{Null}(E) = \text{True} \\ \mathsf{First}(E) & \text{si } \mathsf{Null}(E) = \text{False} \end{cases} \\
\mathsf{First}(E^*) &= \mathsf{First}(E)
\end{aligned}
$$

Last est une fonction qui retourne un ensemble de positions. Elle indique où sont situées les lettres qui peuvent apparaître comme dernière lettre d'un mot décrit par l'expression.

$$
\begin{aligned}
\mathsf{Last}(\emptyset) = \mathsf{Last}(\varepsilon) &= \emptyset \\
\mathsf{Last}(a_p) &= \{p\} \\
\mathsf{Last}(E \cup F) &= \mathsf{Last}(E) \cup \mathsf{Last}(F) \\
\mathsf{Last}(EF) &= \begin{cases} \mathsf{Last}(E) \cup \mathsf{Last}(F) & \text{si } \mathsf{Null}(F) = \text{True} \\ \mathsf{Last}(F) & \text{si } \mathsf{Null}(F) = \text{False} \end{cases} \\
\mathsf{Last}(E^*) &= \mathsf{Last}(E)
\end{aligned}
$$

Last est une fonction qui a en argument une expression et une position et qui retourne un ensemble de positions. Elle indique où sont situées les lettres qui peuvent suivre celle dont la position est donnée en argument dans un mot décrit par l'expression.

$$
\begin{aligned}
\mathsf{Follow}(\emptyset, p) = \mathsf{Follow}(\varepsilon, p) &= \emptyset \\
\mathsf{Follow}(a_q, p) &= \emptyset \\
\mathsf{Follow}(E \cup F, p) &= \begin{cases} \mathsf{Follow}(E,p) & \text{si } p \text{ est une position de } E \\ \mathsf{Follow}(F,p) & \text{si } p \text{ est une position de } F \\ \emptyset & \text{sinon} \end{cases} \\
\mathsf{Follow}(EF, p) &= \begin{cases} \mathsf{Follow}(E,p) & \text{si } p \text{ est une position de } E \text{ et } p \notin \mathsf{Last}(E) \\ \mathsf{Follow}(E,p) \cup \mathsf{First}(F) & \text{si } p \in \mathsf{Last}(E) \\ \mathsf{Follow}(F,p) & \text{si } p \text{ est une position de } F \\ \emptyset & \text{sinon} \end{cases} \\
\mathsf{Follow}(E^*, p) &= \begin{cases} \mathsf{Follow}(E,p) \cup \mathsf{First}(E) & \text{si } p \text{ est une position de } E \\ \emptyset & \text{sinon} \end{cases}
\end{aligned}
$$

Pour l'expression $\bar{E} = (a_1 \cup b_2 a_3)^*(\varepsilon \cup b_4)$, on obtient $\mathsf{Null}(E) = \text{True}$, $\mathsf{First}(E) = \{1,2,4\}$, $\mathsf{Last}(E) = \{1,3,4\}$, et

p	1	2	3	4
$\mathsf{Follow}(E,p)$	$\{1,2,4\}$	$\{3\}$	$\{1,2,4\}$	$\emptyset$

Définition 1 *Soit E une expression rationnelle et $[1;n]$ l'ensemble des positions de cette expressions. L'automate des positions de E est $\mathcal{A} = \langle \{i\} \cup [1;n], A, E, \{i\}, T \rangle$, où*

$$
\begin{aligned}
E = &\{(i,a,p) \mid p \in \mathsf{First}(E) \textit{ et la lettre en position } p \textit{ est } a\} \\
&\cup \{(p,a,q) \mid q \in \mathsf{Follow}(E,p) \textit{ et la lettre en position } q \textit{ est } a\} \\
T = &\begin{cases} \{i\} \cup \mathsf{Last}(E) & \textit{si } \mathsf{Null}(E) = \text{True} \\ \mathsf{Last}(E) & \textit{si } \mathsf{Null}(E) = \text{False} \end{cases}
\end{aligned}
$$