Chapitre 3

Les langages réguliers

Les langages réguliers sont les langages générés par des grammaires de type 3 (ou encore les grammaires régulières). Ils sont acceptés par les automates à états finis. Le terme régulier vient du fait que les mots de tels langages possèdent une formes particulières pouvant être décrites par des expressions simples dites régulières. Ce chapitre introduit ces dernières et établit l'équivalence entre les trois représentations : expression régulière, grammaire régulière et AEF.

1. Les expressions régulières E.R

Définition 3.1 :
Soit X un alphabet quelconque ne contenant pas les symboles $\{*, +, |, ., (,)\}$. Une expression régulière est un mot défini sur l'alphabet $X + \{*, +, |, ., (,)\}$ permettant de représenter un langage régulier de la façon suivante :

— L'expression régulière ε dénote le langage vide ($L = \{\varepsilon\}$);
— L'expression régulière a ($a \in X$) dénote le langage $L = \{a\}$;
— Si r est une expression régulière qui dénote L alors $(r)^*$ (resp. $(r)^+$) est l'expression régulière qui dénote L^* (resp. L^+);
— Si r est une expression régulière dénotant L et s est une expression régulière dénotant L' alors $(r)|(s)$ est une expression régulière dénotant $L + L'$. L'expression régulière $(r).(s)$ (ou simplement $(r)(s)$) dénote le langage $L.L'$.

Les expressions régulières sont également appelées expressions *rationnelles*. L'utilisation des parenthèses n'est pas obligatoire si l'on est sûr qu'il n'y a pas d'ambiguïté quant à l'application des opérateurs $*, +, ., |$. Par exemple, on peut écrire $(a)^*$ ou a^* puisque l'on est sûr que $*$ s'applique juste à a. Par ailleurs, on se convient d'utiliser les priorités suivantes pour les différents opérateurs : 1)$*, +$, 2). et 3)|.

Exemple 3.1 : des expressions régulières

1. a^* : dénote le langage a^n ($n \geqslant 0$);
2. $(a|b)^*$: dénote les mots dans lesquels le symbole a ou b se répètent un nombre quelconque de fois. Elle dénote donc le langage de tous les mots sur $\{a, b\}$;

3. $(a|b)^*ab(a|b)^*$: dénote tous les mots sur $\{a, b\}$ contenant le facteur ab.

On peut également donner la signification suivante aux opérateurs des expressions régulières :

— r^* : signifie répéter r zéro ou plusieurs fois ;
— r^+ : signifie répéter r une ou plusieurs fois ;
— rs : signifie r suivie de s ;
— $r|s$: signifie soit r soit s (soit encore, r ou s).

1..1 Utilisation des expressions régulières

Les expressions régulières sont largement utilisées en informatique. On les retrouve plus particulièrement dans les *shells* des systèmes d'exploitation où ils servent à indiquer un ensemble de fichiers sur lesquels on voudrait appliquer un certain traitement. L'utilisation des expressions régulières en DOS, reprise et étendue par WINDOWS, est très limitée et ne concerne que le caractère "*" qui indique zéro ou plusieurs symboles quelconques ou le caractère "?" indiquant un symbole quelconque. Ainsi, l'expression régulière "f*" indique un mot commençant par f suivi par un nombre quelconque de symboles, "*f*" indique un mot contenant f et "*f*f*" indique un mot contenant deux f. L'expression "f?" correspond à n'importe quel mot de deux symboles dont le premier et f. Ce genre d'expressions régulières s'appelle expressions à *caractères jokers*.

L'utilisation la plus intéressante des expressions régulières est celle faite par UNIX et ses dérivés comme Linux (via la notation POSIX). Les possibilités offertes sont très vastes. Nous les résumons ici :

Expression	Signification
$[abc]$	les symboles a, b ou c
$[^abc]$	aucun des symboles a, b et c
$[a-e]$	les symboles de a jusqu'à e $\{a, b, c, d, e\}$
.	n'importe quel symbole sauf le symbole fin de ligne
$a*$	a se répétant 0 ou plusieurs fois
$a+$	a se répétant 1 ou plusieurs fois
$a?$	a se répétant 0 ou une fois
$a\|bc$	le symbole a ou b suivi de c
$a\{2,\}$	a se répétant au moins deux fois
$a\{,5\}$	a se répétant au plus cinq fois
$a\{2,5\}$	a se répétant entre deux et cinq fois
$\backslash x$	La valeur réelle de x (un caractère spécial)

Exemple 3.2 : des expressions régulières en notation POSIX

— $[^ab]*$: les mots qui ne comportent ni a ni b ;
— $[ab]*$: tous les mots sur $\{a, b\}$;
— $([^a] * a[^a] * a[^a]*)*$: les mots comportant un nombre pair de a ;
— $(ab\{, 4\})*$: en plus de ε, ce sont tous les mots commençant par a où chaque a est suivi

de quatre b au plus.

Utilisation des expressions régulières pour le calcul et la manipulation des données

Dans les éditeurs de texte supportant les expressions régulières (à titre d'exemple : notepad++, kate, gedit, brackets, atom, visual studio, ...), on peut faire une recherche basée sur les expressions régulières. Ces outils supportent aussi le remplacement basé sur les expressions régulières. Par exemple, si on veut remplacer tous les entiers par le mot "integer", on fournit les deux entrées suivantes : $[0-9]+$ et integer.

On peut aussi réaliser des remplacements contextuels via l'utilisation des parenthèses. Par exemple, l'expression régulière $([0-9])([0-9])$ représente une séquence de deux chiffres. Chaque paire de parenthèses correspond à un *groupe*. Ainsi, l'expression précédente renvoie deux groupes (notés respectivement par $\backslash 1$ et $\backslash 2$). Si on veut inverser l'ordre des séquences de deux entiers que l'on rencontre, on fournit alors les deux entrées suivantes : $([0-9])([0-9])$ et $\backslash 2\backslash 1$. Avec ces manipulations, on peut réaliser des calculs très intéressants sur une grande quantité de données.

À présent, nous allons quitter le merveilleux monde des E.R. POSIX, et revenir à ce cours. Nous allons, donc, reprendre la définition des expressions régulières (que l'on qualifiera de mathématiques) données par la section précédentes.

1..2 Expressions régulières ambiguës

Définition 3.2 :

Une expression régulière est dite *ambiguë* s'il existe au moins mot pouvant être mis en correspondance avec l'expression régulière de plusieurs façons.

Cette définition fait appel à la correspondance entre un mot et une expression régulière. Il s'agit, en fait, de l'opération qui permet de dire si le mot appartient au langage décrit par l'expression régulière. Par exemple, prenons l'expression régulière a^*b^*. Soit à décider si le mot aab est décrit ou non par cette expression. On peut écrire :

$$\underbrace{aa}_{a^*}\ \underbrace{b}_{b^*}$$

Ainsi, le mot est décrit par cette E.R. Il n'y a qu'une seule façon qui permet de le faire correspondre. Ceci est valable pour tous les mots de ce langage. L'E.R n'est donc pas ambiguë.

Considérons maintenant l'expression $(a|b)^*a(a|b)^*$ décrivant tous les mots sur $\{a, b\}$ contenant le facteur a. Soit à faire correspondre le mot $abaab$, on a :

$$\begin{aligned} abaab &= \underbrace{ab}_{(a|b)^*}.a.\underbrace{ab}_{(a|b)^*} \\ abaab &= \underbrace{aba}_{(a|b)^*}.a.\underbrace{b}_{(a|b)^*} \end{aligned}$$

Il existe donc au moins deux façons pour faire correspondre aab à l'expression précédente, elle est donc ambiguë.

L'ambiguïté pose un problème quant à l'interprétation d'un mot. Par exemple, supposons que, dans l'expression $(a|b)^*a(a|b)^*$, l'on veut comparer la partie à gauche du facteur a à la partie droite du mot. Selon la méthode de correspondance, le résultat est soit vrai ou faux ce qui est inacceptable dans un programme cohérent.

1..3 Comment lever l'ambiguïté d'une E.R ?

Il n'existe pas une méthode précise pour lever l'ambiguïté d'une E.R. Cependant, on peut dire que cette opération dépend de ce que l'on veut faire avec l'E.R ou plutôt d'une *hypothèse d'acceptation*. Par exemple, on peut décider que le facteur fixe soit le premier a du mot à analyser ce qui donne l'expression régulière : $b^*a(a|b)^*$. On peut également supposer que c'est le dernier a du mot à analyser ce qui donne l'expression régulière $(a|b)^*ab^*$.

2. Les langages réguliers, les grammaires et les AEF

Le théorème suivant établit l'équivalence entre les AEF, les grammaires régulières et les expressions régulières :

Théorème 3.1

(de Kleene) Soient Λ_{reg} l'ensemble des langages réguliers (générés par des grammaires régulières), Λ_{rat} l'ensemble des langages décrits par toutes les expressions régulières et Λ_{AEF} l'ensemble de tous les langages acceptés par un AEF. Nous avons, alors, l'égalité suivante :

$$\Lambda_{reg} = \Lambda_{rat} = \Lambda_{AEF}$$

Le théorème annonce que l'on peut passer d'une représentation à une autre du fait de l'équivalence entre les trois représentations. Par la suite, on verra comment passer d'une représentation à une autre.

2..1 Passage de l'automate vers l'expression régulière

Soit $A = (X, Q, q_0, F, \delta)$ un automate à états finis quelconque. On désigne par L_i le langage accepté par l'automate si son état initial était q_i. Par conséquent, trouver le langage accepté par l'automate revient à trouver L_0 étant donné que l'analyse commence à partir de l'état initial q_0. L'automate permet d'établir un système d'équations aux langages de la manière suivante :

— si $\delta(q_i, a) = q_j$ alors on écrit : $L_i = aL_j$;

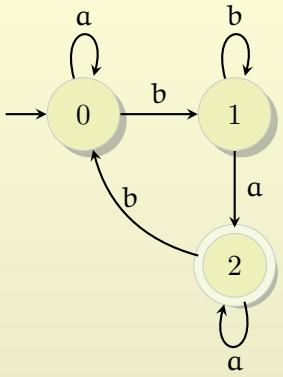
— si $q_i \in F$, alors on écrit : $L_i = \varepsilon$

— si $L_i = \alpha$ et $L_i = \beta$ alors on écrit : $L_i = \alpha|\beta$;

Il suffit ensuite de résoudre le système en précédant à des substitutions et en utilisant la règle suivante : la solution de l'équation $L = \alpha L|\beta$ ($\varepsilon \notin \alpha$) est le langage $L = \alpha^*\beta$ (attention ! si vous obtenez $L = \alpha L$ alors c'est la preuve d'une faute de raisonnement).

Exemple 3.3 : calcul de l'expression régulière

Soit l'automate donné par la figure suivante :



Trouvons le langage accepté par cet automate. Le système d'équations est le suivant :

1) $L_0 = aL_0|bL_1$

2) $L_1 = bL_1|aL_2$

3) $L_2 = aL_2|bL_0|\varepsilon$

Appliquons la deuxième règle sur les équations, on obtient alors après substitutions :

4) $L_0 = a^*bL_1$

5) $L_1 = b^*aL_2 = b^*a^+bL_0|b^*a^+$

6) $L_2 = a^*bL_0|a^*$

En remplaçant 5) dans 4) on obtient : $L_0 = a^*b^+a^+bL_0|a^*b^+a^+$. En appliquant le schéma de résolution donné plus haut, on trouve $L_0 = (a^*b^+a^+b)^*a^*b^+a^+$.

Remarque : Il est possible d'obtenir plusieurs expressions régulières associées à un AEF, selon l'ordre d'élimination des variables. Ceci est dû au fait que l'on peut trouver plusieurs expressions régulières différentes mais qui représentent, en fait, le même langage.

2..2 Passage de l'expression régulière vers l'automate

Il existe deux méthodes permettant de réaliser cette tâche. La première fait appel à la notion de *dérivée* tandis que la deuxième construit un automate comportant des ε-transitions en se basant sur les propriétés des langages réguliers.

Méthode des dérivées

Définition 3.3 : Soit w un mot défini sur un alphabet X. On appelle dérivée de w, par rapport à $a \in X$, le mot $u \in X^*$ tel que $w = au$. On note cette opération par : $u = w||a$.

On peut étendre cette notion aux langages. Ainsi la dérivée d'un langage par rapport à un mot $a \in X$ est le langage $L||a = \{u \in X^* | \exists w \in L : w = au\}$.

Exemple 3.4 : calcul des dérivées

— $L = \{1, 01, 11\}$, $L||1 = \{\varepsilon, 1\}$, $L||0 = \{1\}$.

Propriétés des dérivées

— $(\sum_{i=1}^{n} L_i)||a = \sum_{i=1}^{n}(L_i||a)$

— $(L.L')||a = (L||a).L' + f(L).(L'||a)$ tel que $f(L) = \{\varepsilon\}$ si $\varepsilon \in L$ et $f(L) = \emptyset$ sinon

— $(L^*)||a = (L||a)L^*$

Méthode de construction de l'automate par la méthode des dérivées

Soit r une expression régulière (utilisant l'alphabet X) pour laquelle on veut construire un AEF. L'algorithme suivant donne la construction de l'automate :

Algorithme de construction de l'AEF d'une E.R r par la méthode des dérivées

1 r est un nouveau langage

2 Dériver un nouveau langage disponible par rapport à chaque symbole de X

3 Recommencer l'étape 2 pour chaque nouveau langage obtenu jusqu'à ce qu'il n'y ait plus de nouveaux langages

4 Chaque langage obtenu correspond à un état de l'automate. L'état initial correspond à r. Si ε appartient à un langage obtenu alors l'état correspondant est final

5 **pour** *chaque relation de dérivation* $L_i||a = L_j$ **faire**

6 on crée une transition entre l'état associé à L_i et l'état associé à L_j et on la décore par a

Exemple 3.5 : application de la méthode des dérivées

Considérons le langage $L_0 = (a|b)^*a(a|b)^*$. On sait que :

— $(a|b)||a = \varepsilon$ donc $(a|b)^*||a = (a|b)^*$

Commençons l'application de l'algorithme :

— $L_0||a = [(a|b)^*a(a|b)^*]||a = ((a|b)^*a(a|b)^*)|(a|b)^* = (a|b)^* = L_1$

— $L_0||b = [(a|b)^*a(a|b)^*]||b = (a|b)^*a(a|b)^* = L_0$

— $L_1||a = [((a|b)^*a(a|b)^*)|(a|b)^*]||a = ((a|b)^*a(a|b)^*)|(a|b)^* = (a|b)^* = L_1$

— $L_1||b = [((a|b)^*a(a|b)^*)|(a|b)^*]||b = ((a|b)^*a(a|b)^*)|(a|b)^* = (a|b)^* = L_1$

Il n'y a plus de nouveaux langages, on s'arrête alors. L'automate comporte deux états q_0 (associé à $(a|b)^*a(a|b)^*$) et q_1 (associé à $((a|b)^*a(a|b)^*)|(a|b)^*$), il est donné par la table suivante (l'état initial est q_0 et l'état q_1 est final) :

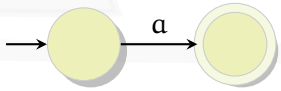
État	a	b
q_0	q_1	q_0
q_1	q_1	q_1

Remarque : Appliquée correctement, la méthode des dérives, bien qu'elle nécessite beaucoup de calcul, permet de construire l'AEF déterministe et minimal du langage. La difficulté de la méthode réside dans la difficulté de décider si deux expressions régulières sont équivalentes.

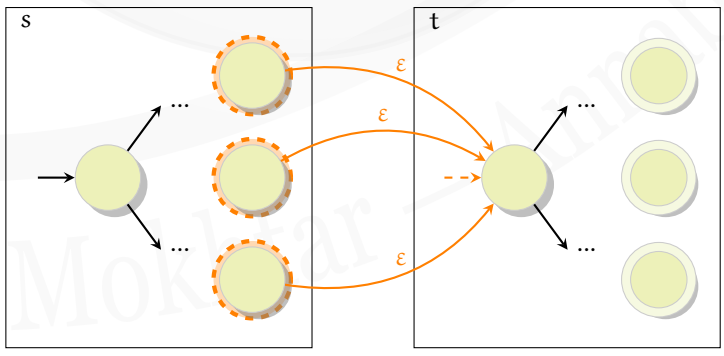
Méthode de Thompson

La méthode de Thompson permet de construire un automate en procédant à la décomposition de l'expression régulière selon les opérations utilisées. Soit r une E.R, alors l'algorithme à utiliser est le suivant :

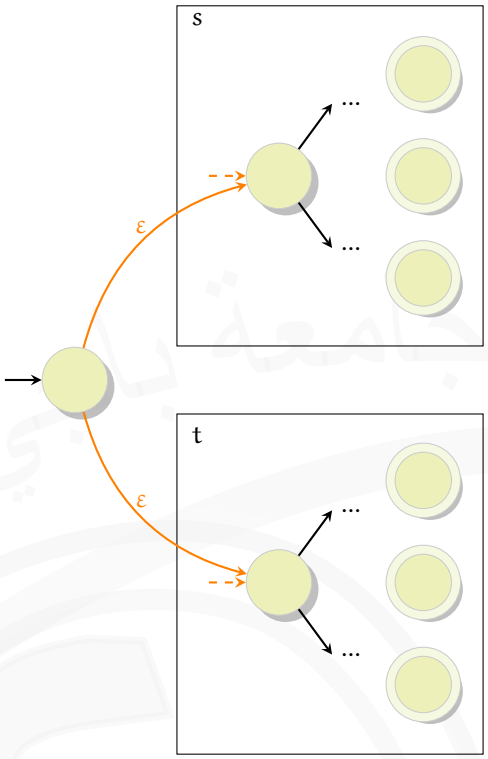
— Si $r = a$ (un seul symbole) alors l'automate est le suivant :



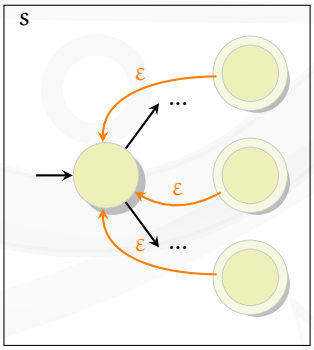
— Si $r = st$ alors il suffit de trouver l'automate A_s qui accepte s et l'automate A_t qui accepte t. Il faudra, ensuite relier chaque état final de A_s à l'état initial de A_t par une ε-transition. Les états finaux de A_s ne le sont plus et l'état initial de A_t ne l'est plus :



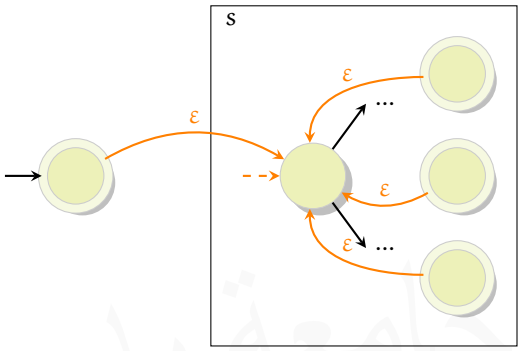
— Si $r = s|t$ alors il suffit de créer un nouvel état initial et le relier avec des ε-transitions aux états initiaux de A_s et A_t qui ne le sont plus :



— Si $r = s^+$ alors il suffit de relier les états finaux de A_s par des ε-transitions à son propre état initial :

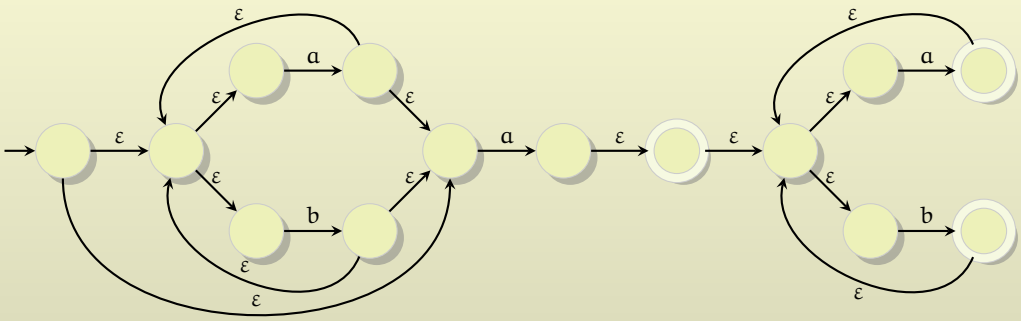


— Si $r = s^*$ alors il suffit de relier les états finaux de A_s par des ε-transitions à son état initial, créer un nouvel état initial et le relier à l'ancien état initial par une ε-transition. Enfin, le nouvel état initial est également final (dans certaines conditions, on peut trouver d'autres constructions possibles) :



Exemple 3.6 : application de la méthode de Thompson

Soit à construire l'automate du langage $(a|b)^*a(a|b)^*$, la méthode de Thompson donne l'automate suivant :



2..3 Passage de l'automate vers la grammaire

Du fait de l'équivalence des automates à états finis et les grammaires régulières, il est possible de passer d'une forme à une autre. Le plus facile étant le passage de l'automate vers la grammaire. Le principe de correspondance entre automates et grammaires régulières est très intuitif : il correspond à l'observation que chaque transition dans un automate produit exactement un seul symbole (au plus), de même que chaque dérivation dans une grammaire régulière *normalisée* (un concept qui sera présenté plus bas).

Soit $A = (X, Q, q_0, F, \delta)$ un AEF, la grammaire qui génère le langage accepté par A est $g = (V, N, S, R)$:

— $V = X$;

— On associe à chaque état de Q, un non-terminal. Ceci permet d'avoir autant de non-terminaux qu'il existe d'états dans A ;

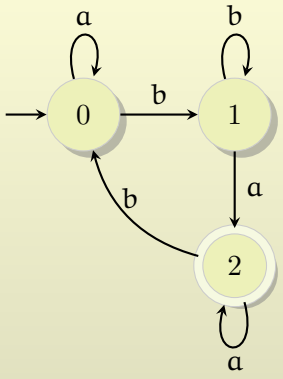
— L'axiome S est le non-terminal associé à l'état initial q_0 ;

— Soit A le non-terminal associé à q_i et B le non-terminal associé à q_j, si $\delta(q_i, a) = q_j$ alors la grammaire possède la règle de production : $A \to aB$;

— Si q_i est final et A est le non-terminal associé à q_i alors la grammaire possède la règle de production : $A \to \varepsilon$.

Exemple 3.7 : déduction de la grammaire régulière à droite

Soit l'AEF suivant :



La grammaire générant le langage accepté (en associant S à 0, T à 1 et U à 2) est : $(\{a, b\}, \{S, T, U\}, S, \{S \to aS|bT, T \to bT|aU, U \to aU|bS|\varepsilon\})$.

La forme engendrée par cet algorithme correspond à ce qu'on appelle grammaire régulière à droite (du fait que le non-terminal, dans la partie droite des règles, se trouve le plus à droite). En réalité, il existe une autre forme des grammaires de type 3, dite grammaire régulière à gauche.

Définition 3.4 : Soit $G = (V, N, S, R)$ une grammaire. G est dite régulière à gauche si toutes les règles de production sont de la forme $A \to B\alpha$ ou $A \to \alpha$ tel que $A, B \in N$ et $\alpha \in V^*$. Une grammaire régulière à gauche génère un langage régulier.

Notons ici qu'une grammaire est soit régulière à droite, soit à gauche, sinon elle n'est pas régulière. En d'autres termes, si on trouve des formes régulières à droite et à gauche, alors la grammaire n'est pas du type 3 (elle est de type 2).

Il n'est toutefois pas possible de déduire la grammaire régulière à gauche à partir de l'AEF Pour cela, il faut une petite gymnastique. Commençons d'abord par noter que si l'on prend une grammaire régulière droite à laquelle on applique le miroir sur toutes les parties droites des règles de production, on obtient une grammaire régulière à gauche générant le langage miroir. On peut alors proposer l'algorithme suivant. Soit A un AEF acceptant un langage L :

1. Construire l'AEF A^R acceptant le langage miroir (voir le chapitre 2 pour la méthode de construction).
2. Déduire la grammaire régulière à droite de L^R à partir de A^R.
3. Appliquer le miroir aux parties droites de toutes les règles de la grammaire obtenue. Le résultat représente la grammaire régulière à gauche générant L.

Notons, par le passage, que l'on s'intéresse généralement à une forme normalisée des grammaires régulières. Celle-ci est définie comme suit : soit $G = (V, N, S, R)$ une grammaire régulière à droite, elle est dite normalisée si toutes les règles de production sont de l'une des formes suivantes :

— $A \to aB$, $A, B \in N$, $a \in V$ $(|a| = 1)$;

— $A \to \varepsilon$, $A \in N$

2..4 Passage de la grammaire vers l'automate

D'après la section précédente, il existe une forme de grammaires régulières pour lesquelles il est facile de construire l'AEF correspondant. En effet, soit $G = (V, N, S, R)$ une grammaire régulière à droite, si toutes les règles de production sont de la forme : $A \to aB$ ou $A \to B$ $(A, B \in N, a \in V \cup \{\varepsilon\})$ alors il suffit d'appliquer l'algorithme suivant :

1. Associer un état à chaque non-terminal de N ;
2. L'état initial est associé à l'axiome ;
3. Pour chaque règle de production de la forme $A \to \varepsilon$, l'état q_A est final ;
4. Pour chaque règle $A \to aB$ alors créer une transition partant de q_A vers l'état q_B en utilisant l'entrée a ;
5. Pour chaque règle $A \to B$ alors créer une ε-transition partant de q_A vers l'état q_B.

Cependant, cet algorithme ne peut pas s'appliquer aux grammaires non normalisées. On peut néanmoins transformer toute grammaire régulière à droite à la forme normalisée. L'algorithme de transformation est le suivant. Soit $G = (V, N, S, R)$ une grammaire régulière :

1. Pour chaque règle de la forme $A \to wB$ $(A, B \in N$ et $w \in V^*)$ tel que $|w| > 1$, créer un nouveau non-terminal B' et éclater la règle en deux : $A \to aB'$ et $B' \to uB$ tel que $w = au$ et $a \in V$. Il faut recommencer la transformation jusqu'à obtenir des règles où le non-terminal à droite est précédé d'un seul non-terminal.
2. Pour chaque règle de la forme $A \to w$ $(A \in N$ et $w \in V*)$ tel que $|w| \geqslant 1$, créer un nouveau non-terminal B' et éclater la règle en deux : $A \to aB'$ et $B' \to u$ tel que $w = au$ et $a \in V$. Il faut recommencer la transformation jusqu'à obtenir une règle avec ε dans la partie droite.

Exemple 3.8 : transformation de grammaire vers la forme normalisée

Soit la grammaire régulière $G = (\{a, b\}, \{S, T\}, S, \{S \to aabS|bT, T \to aS|bb\})$. Le processus de transformation est le suivant :

— Éclater $S \to aabS$ en $S \to aS_1$ et $S_1 \to abS$;
— Éclater $S_1 \to abS$ en $S_1 \to aS_2$ et $S_2 \to bS$;
— Éclater $T \to bb$ en $T \to bT_1$ et $T_1 \to b$;
— Éclater $T_1 \to b$ en $T_1 \to bT_2$ et $T_2 \to \varepsilon$

Remarque : Il est possible de déduire une grammaire (qui ne sera pas forcément de type 3) à partir de l'expression régulière en utilisant un algorithme semblable à celui de Thompson. L'étudiant est invité à revoir le dernier exercice de la première série de TD pour une première idée de l'algorithme de construction.

3. Propriétés des langages réguliers

3..1 Stabilité par rapport aux opérations sur les langages

Les langages réguliers sont stables par rapport aux opérations de l'union, l'intersection, le complément, la concaténation et la fermeture de Kleene. La démonstration de ce résultat est très simple. Soient L et M deux langages réguliers désignés respectivement par les E.R r et s et respectivement acceptés par les automates A_r et A_s. Étant donné l'équivalence entre les langages réguliers et les AEF, nous avons :

— $L + M$ est régulier : l'AEF correspondant s'obtient en utilisant l'algorithme de construction de l'automate d'acceptation de l'E.R $r|s$;

— $L \cap M$ est régulier : l'AEF correspondant s'obtient en calculant le produit de A_r et A_s (voir le chapitre précédent) ;

— $\overline{L}$ est régulier : l'AEF correspondant s'obtient en rendant l'automate A_r déterministe et complet puis en inversant le statut final/non final des états (voir le chapitre précédent) ;

— L^R est régulier : l'AEF correspondant s'obtient en inversant le sens des arcs dans A_r et en inversant le statut initial/final des états (voir le chapitre précédent) ;

— L.M est régulier : l'AEF correspondant s'obtient en utilisant l'algorithme de construction de l'automate de $r.s$;

— L^* (resp. L^+) est régulier : l'AEF correspondant s'obtient en utilisant l'algorithme de construction de l'automate de r^* (resp. r^+) ;

Revenons un peu sur la stabilité des langages réguliers par rapport à l'union. Le résultat donné ci-haut montre que l'union finie de langages réguliers représente forcément un langage régulier. Ce qui signifie que tout langage fini est régulier. Cependant, l'union infinie de langages réguliers peut être de n'importe quel type (tout dépend des langages en question).

3..2 Les langages réguliers et la méthode des dérivées

Nous avons déjà vu deux méthodes de construction des AEF à partir de l'E.R : méthode des dérivées et méthode de Thompson. La première, nécessitant beaucoup de calculs, a le mérite de produire l'AEF déterministe et minimal du langage. La deuxième, très simple à appliquer et à programmer, a l'inconvénient de produire un AEF non-déterministe et comportant beaucoup d'états (notons qu'il existe des méthodes concurrentes à celle de Thompson, plus complexes mais produisant moins d'états de d'ε-transitions).

Si la méthode de Thompson requiert une expression régulière pour fonctionner, celle des dérivées ne le requiert pas car elle peut être appliquée à tout langage. On s'interrogera néanmoins sur le critère d'arrêt de cette méthode, le théorème suivant nous sera d'une grande aide (bien sûr, on suppose que la méthode des dérivées est convenablement appliquée) :

Théorème 3.2

La méthode des dérivées produit un nombre fini d'états pour tout langage régulier, et un nombre infini d'états pour tout langage non régulier.

Ainsi, la méthode des dérivées nous offre un premier moyen permettant de différencier les langages réguliers des non réguliers.

Exemple 3.9 : montrer que le langage $\{a^n b^n | n \geqslant 0\}$ n'est pas régulier

Soit le langage $L_0 = \{a^n b^n | n \geqslant 0\}$, appliquons-lui la méthode des dérivées. Nous avons alors :

- $L_0 || a = \{a^{n-1} b^n | n \geqslant 1\} = L_1$
- $L_1 || a = \{a^{n-2} b^n | n \geqslant 2\} = L_2$
- ...
- $L_i || a = \{a^{n-(i+1)} b^n | n \geqslant i+1\} = L_{i+1}$

On voit alors que les opérations de dérivations ne s'arrêtent jamais, produisant ainsi une infinité d'états. On en déduit que L_0 n'est pas régulier (en fait, il est algébrique).

3..3 Lemme de la pompe

Dans cette section, nous présentons d'autres critères permettant d'affirmer si un langage est régulier ou non. Rappelons d'abord que tout langage fini est un langage régulier. À présent, nous allons annoncer un critère dont la vérification permet de juger si un langage n'est pas régulier. Il s'agit en fait d'une condition nécessaire et non suffisante pour qu'un langage soit régulier. La démonstration de ce résultat sort du cadre du cours.

Proposition 3.1

Soit L un langage régulier infini défini sur l'alphabet X. Il existe alors un entier n tel que pour tout mot $w \in L$ et $|w| \geqslant n$, il existe $x, z \in X^*$ et $y \in X^+$ tels que :

- $w = xyz$;
- $|xy| \leqslant n$;
- $xy^i z \in L$ pour tout $i > 0$.

Il est à noter que cette condition est nécessaire et non suffisante. Il existe, en effet, des langages non réguliers vérifiant ce critère. Il existe néanmoins une condition nécessaire et suffisante que l'on va énoncer après avoir présenté un exemple d'utilisation du lemme de la pompe.

Exemple 3.10 : application du lemme de la pompe sur un langage régulier

Soit le langage $L = \{a^k b^l | k, l \geqslant 0\}$ (ou encore $a^* b^*$, il s'agit donc d'un langage régulier). Prenons $n = 1$ et vérifions le critère de la pompe. Soit un mot $w = a^k b^l$ $(k + l \geqslant 1)$. Si $k > 0$ alors il suffit de prendre $x = a^{k-1}, y = a$ et $z = b^l$, ainsi tout mot de la forme $xy^i z = a^{k+i-1} b^l$ appartient au langage. Si $k = 0$ alors il suffit de prendre $x = \varepsilon, y = b$ et $z = b^{l-1}$ pour vérifier le critère de la pompe.

Exemple 3.11 : application du lemme de la pompe sur un langage non régulier

Soit le langage $L = \{a^k b^k | k \geqslant 0\}$[a]. Supposons que le langage est régulier et appliquons le lemme de la pompe. Supposons que l'on a trouvé n qui vérifie le critère, ceci implique que pour tout mot $w \in L$, on peut le décomposer en trois sous-mots x, y et z tel que : $|xy| \leqslant n$, $y \neq \varepsilon$ et $xy^i z \in L$. Considérons le mot $a^{n+1} b^{n+1}$, toute décomposition de ce mot produit les mots $x = a^j$, $y = a^l$ et $z = a^{n+1-j-l} b^{n+1}$, $l > 0$ (si xy contient n symboles alors x et y ne contiennent que des a étant donné qu'il y a $(n + 1)$ a). Maintenant, le lemme de la pompe stipule que $xy^i z \in L$ pour tout $i > 0$ donc tout mot de la forme $a^j a^{il} a^{n+1-j-l} b^{n+1} = a^{(i-1)l+n+1} b^{n+1}$ appartient à L. Pour $i = 2$ la borne inférieure de $(i - 1)l$ est 1, ce qui signifie que le nombre de a est différent du nombre de b. Contradiction. Donc, le langage $a^k b^k$ n'est pas régulier.

a. Attention, l'expression $a^k b^k$ n'est pas régulière.

Exemple 3.12 : le lemme de la pompe peut être satisfait par un langage non régulier

Soit maintenant soit $L \subset b^*$ un langage non régulier arbitraire. Le langage :

$$a^+ L | b^*$$

satisfait le lemme de la pompe. Il suffit de prendre avec les notations du lemme, $n = 1$. Cet exemple illustre donc le fait que le lemme précédent ne constitue pas une condition nécessaire pour décider de la régularité d'un langage.

Il existe néanmoins une version nécessaire et suffisante du lemme de la pompe pour savoir, à coup sûr, si un langage est régulier.

Proposition 3.2

Soit L un langage infini défini sur l'alphabet X. L est régulier si et seulement s'il existe un entier $n > 0$ tel que pour tout mot $w \in X^*$ et $|w| \geqslant n$, il existe $x, z \in X^*$ et $y \in X^+$ tels que :

— $w = xyz$;

— $\forall u \in X^* : wu \in L \Leftrightarrow xy^i zu \in L$.