CHAPITRE

EXPRESSIONS RÉGULIÈRES ET ÉQUIVALENCES

6

Nos avons étudié dans les chapitres précédents la question d'appartenance d'un mot à un langage formel selon différents points de vue (Ensemble, grammaire et automate). On s'est focalisé sur les langages réguliers ceux qui sont générés par des grammaires régulières (de type 3) et reconnus par des AEFs. Mais le terme régulier vient du fait que les mots de tels langages possèdent une formes particulières pouvant être décrites par des expressions régulières. Nous verrons dans ce chapitre comment employer les expressions régulières comme une 4ème definition donnée à ce type de langages. Nous étudierons par ailleurs l'équivalence/passage entre ces représentations : expressions régulières, grammaires régulières et AEFs.

6.1 Les expressions régulières ERs

Les expressions régulières (rationnelles) sont puissantes et incontournables lorsque vous souhaitez récupérer des informations dans des gros fichiers. Elles sont très utilisées dans des problèmes de recherches scientifiques, notamment en sécurité informatique, en bioinformatique, etc.

6.1.1 Représentation mathématique (Notation algébrique)

Un langage régulier L défini sur l'alphabet X (ne contenant pas $*, +, |, ., (,)$) peut être dénoté ou décrit par une ou plusieurs expressions régulières. Or, une expression régulière ne dénote qu'un seul langage.

Définition 6.1.1 Expression régulière

Une expression régulière (ER) ou rationnelle est un mot (formule) défini sur $X \cup \{*, +, |, ., (,)\}$ et selon laquelle le langage régulier peut se décomposer à l'aide des opérations rationnelles : $*, ., |$.

Inductivement, les langages réguliers sont définis par :

- $\emptyset$ est une ER qui dénote le langage vide.
- ε est une ER qui dénote le langage $\{\varepsilon\}$.
- $\forall a \in X$, a est une ER qui dénote le langage $\{a\}$ (langage fini)
- Si L et L' sont deux langages dénotés respectivement par l'ER r et s, alors on peut déduire que :

ER	Signifie	Dénote
$r\|s$	soit r ou s	$L + L'$
$r.s$ ou rs	r suivie par s	$L.L'$
r^*	r se répéte zéro ou plusieurs fois	L^*
r^+	r se répéte une ou plusieurs fois	L^+

Les expressions rationnelles sont des formules qui simplifient et étendent la notation des langages réguliers

Remarque 6.1.1 Priorités

Il y a un ordre de priorités entre les opérateurs tel que :

1. $*, +,$
2. $.$
3. $|$

Mais on peut utiliser des parenthèses pour éviter tout ambiguïté quant à l'application de ces opérateurs.

Exemple Quelques ERs

— a^* *ou* $(a)^*$ *dénote* $L = \{a^n, n \geq 0\}$.

— $(a|b)^*$ *dénote le langage de tous les mots sur* $\{a, b\}$ *(contenant un nombre quelconque de* a *et de* b*).*

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

Question. Si un langage peut être dénoté par plusieurs ERs, y-a-t-il une ER canonique : la plus simple (la plus courte) pour un langage ?

Propriétés (des ERs équivalentes)

Définition 6.1.2 Expressions régulières équivalentes

Deux expressions régulières sont équivalentes si elles dénotent le même langage.

Voici quelques propriétés (Identités rationnelles) qui expriment certaines équivalences élémentaires et qui permettent de simplifier une ER. Soient : $r, s, t,$ trois ERs quelconques :

— $r.\emptyset = \emptyset.r = \emptyset$
— $r|\emptyset = \emptyset|r = r$
— $\emptyset^* = \varepsilon$
— $r.\varepsilon = \varepsilon.r = r$
— $r|r = r$
— $r|s = s|r$
— $(r|s)|t = r|(s|t)$
— $(r.s).t = s.(s.t)$
— $r.(s|t) = r.s|r.t$
— $(r^*)^* = r^*$
— $r^*.r = r.r^* = r^+$
— $r^+|\varepsilon = r^*$
— $(r|s)^* = (r^*.s^*)^*$
— $(r.s)^*.r = r.(s.r)^*$

Notons que $\equiv$ peut remplacer $=$

Remarque 6.1.2

- En appliquant ces propriétés, il possible de faire des transformations syntaxiques afin de simplifier l'ER tout en préservant le même langage dénoté.
- En revanche, la réduction d'une ER en utilisant directement ces propriétés n'est pas une tâche facile.
- Une méthode efficace la plus utilisée pour vérifier l'équivalence de deux ERs (ou pour trouver une ER canonique) fait plutôt appel à leur transformation en automates finis(voir les sections suivantes)

6.1.2 L'appartenance d'un mot à un langage selon son ER

Nous avons déjà vu comment répondre à la question d'appartenance d'un mot à un langage d'un point de vue ensembliste, grammaticale et par automate. Maintenant, comment est-il possible de répondre à cette question à l'aide d'une ER.

Déduction 6.1.1 vérifier si $w \in L$ d'un point de vue ER

Soit L un langage régulier dénoté par une ER r. Pour montrer qu'un mot $w \in L$, il suffit de trouver une correspondance entre w et r.

Exemple

Soit l'ER $r = a^*b^*$ qui dénote un langage L (de quel langage s'agit-il ?).

Question. le mot $w = aaabb$ appartient-t-il à L ou non ?

Réponse : il faut décomposer w en parties, et essayer d'associer chacune de ces parties à une partie de l'ER (On cherche une correspondance)
$\underbrace{aaa}_{a^*}\underbrace{bb}_{b^*}$ *représente une correspondance (c'est la seule correspondance)*

L'ambigüité dans les ERs

Définition 6.1.3 ER ambiguë

On dit qu'une ER r qui dénote L est ambiguë s'il existe au moins un mot w, pour lequel il existe au moins deux correpodances avec r et qui montrent que $w \in L$

Exemple : ER ambiguë et non ambiguë

— *Reprenons l'ER a^*b^* qui décrit l'ensemble des mots formés par une séquence de zéro ou plusieurs a suivis d'une séquence de zéro ou plusieurs b : Lorsqu'on prend n'importe quel mot de ce langage, il n'y aura qu'une seule correspondance, car la séquence des a et celle des b sont facilement identifiables.*
Donc a^*b^* n'est pas donc pas ambiguë.

— *Prenons un autre exemple d'ER : $r = (a|b)^*a(a|b)^*$ qui dénote le langage des mots contenant le facteur a. $w = abaab$ appartient-t-il à ce langage ?*
La réponse est oui. Pour le prouver, il existe non seulement une mais au moins deux manières (deux correspondances) pour associer ce mot à cette ER,

- *Première correspondance* $\underbrace{ab}_{(a|b)^*}\underbrace{a}_{a}\underbrace{ab}_{(a|b)^*}$
- *Deuxième correspondance* $\underbrace{aba}_{(a|b)^*}\underbrace{a}_{a}\underbrace{b}_{(a|b)^*}$

Donc r est dans ce cas ambiguë.

Lever l'ambiguïté d'une ER

Problème. En cherchant des correspondance, l'ambiguité d'une ER peut poser un problème d'incohérence dans un porgramme.
Par exemple, dans le cas de l'ER précédente $(a|b)^*a(a|b)^*$ lorsqu'on veut comparer la partie à gauche du facteur a à celle qui se trouve à droite dans un mot, on obtiendera deux réponses différentes : Vrai avec la première correspondance et faux avec la deuxième Ce qui est inacceptable dans un programme cohérent.

Remarque 6.1.3

Il faut noter qu'il n y a pas de méthode précise permettant de lever l'ambiguïté d'une ER.

Solution. Mais pour résoudre ce problème on peut se baser sur des hypothèses d'acceptation afin de rendre l'ER moins ambiguë ou carrément non ambiguë, tout en faisant attention à ne pas changer le langage dénoté.

Exemple **Comment lever l'ambiguïté d'une ER**

Reprenons l'ER ambiguë précédente $(a|b)^*a(a|b)^*$ *:*

— *Une des hypothèses d'acceptation consiste à supposer que le facteur* a *soit le premier* a *rencontré dans un mot, ce qui donne l'ER :* $b^*a(a|b)^*$.

— *Une deuxième hypothèses d'acceptation consiste à supposer que le facteur* a *soit le dernier* a *rencontré dans un mot, ce qui donne l'ER :* $(a|b)^*ab^*$.

6.1.3 Syntaxe pratique pour l'usage des ERs

Les ERs sont largement utilisées en informatique, notamment dans les moteurs de recherche, les éditeurs de texte, les analyses de données, etc. L'intérêt pratique des ERs se manifeste à travers de nombreux outils d'usage courant qui offrent la possibilité d'utiliser ces formules (avec de multiples extensions) pour effectuer essentiellement des recherches de motifs et manipuler des données

Dans les shells des systèmes d'exploitation

- En DOS et WINDOWS :
  L'usage des ERs permet d'indiquer un ensemble de fichiers sur lesquels on voudrait appliquer un certain traitement. Mais cette utilisation est très limitée et basée sur des caractères de remplacement, souvent appelés des caractères "jokers" ou "wildcards" : L'asterisque (*) qui indique zéro ou plusieurs symboles quelconques et le point d'interrogation (?) indiquant un symbole quelconque. Par exemple :

  Exemple

  — *f* : indique une chaîne de caractères commençant par f suivi par zéro ou plusieurs symboles quelconques.*

  — **f* : indique une chaîne de caractères contenant f*

  — *?f* : indique une chaîne de caractères contenant f comme en deuxième position.*

- UNIX (Linux)
  L'utilisation des ERs dans les systèmes UNIX (et ses dérivés comme Linux) est plus intéressante grâce à l'utilitaire **grep** qui permet de rechercher les occurrences d'un mot(ou motif) dans un fichier texte.

  Exemple

  — *grep 'word' file-text*
  Une telle commande imprime toutes les lignes du fichier contenant au moins une occurrence de 'word'. En remplaçant 'word' par une ER, grep imprime toutes les occurrences de tous les mots d'un langage régulier dénoté par cette ER.

  Mais les ERs sont formulées selon la notation **POSIX** qui offre une syntaxe plus riche.

Notation POSIX

La notation POSIX (Portable Operating System Interface for Unix) est une norme définie par l'IEEE (Institute of Electrical and Electronics Engineers) qui spécifie une interface standard pour les systèmes d'exploitation de type Unix et Unix-like. Cette norme englobe plusieurs domaines, notamment la gestion des processus, la gestion des fichiers, etc.

POSIX définit aussi une notation standard pour les expressions régulières utilisées dans les outils de traitement de texte, de recherche et de manipulation de fichiers. Même si elle est restrictive et moins puissante que certaines notations comme celles utilisées dans Perl ou Python, mais elle offre une compatibilité et une portabilité importantes pour les logiciels développés pour les environnements Unix.

Remarque 6.1.4 Quleques illustrations

- La syntaxe POSIX (de grep) est basique mais inclut aussi de nombreuses extensions notationnelles.

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
$a\|bc$	le symbole a ou b suivi de c
.	n'importe quel symbole sauf le symbole de fin de ligne

- Quantificateurs

Expression	Signification
$a*$	a se répéte 0 ou plusieurs fois
$a+$	a se répétant 1 ou plusieurs fois
a ?	a se répétant 0 ou une fois
$a\{n\}$	a se répéte n fois
$a\{n,\}$	a se répéte au moins n fois
$a\{,m\}$	a se répéte au plus m fois
$a\{n,m\}$	a se répéte entre n et m fois

- D'autres extensions

Expression	Signification
^{e}	Le motif e doit apparaître au début d'une ligne
$e\$$	e doit apparaître à la fin d'une ligne
$\backslash n$	la fin d'une ligne
$\backslash x$	La valeur réelle de x (un caractère spécial)
$\backslash d$	n'importe quel chiffre décimal [0-9]
$\backslash D$	pas de chiffre
$\backslash w$	Un caractère alphanumérique
$\backslash s$	espace

- Quant aux caractères spéciaux $+, *, \ldots$ ils sont précédés par caractère d'échappement $\backslash$.

Exemple ERs selon la notion POSIX

1. $[^ab]*$ *: Les mots qui ne comportent ni* a *ni* b*.*
2. $[ab]*$ *: Tous les mots sur*$\{a, b\}$
3. $([^a] * a[^a] * a[^a]*)*$ *: Les mots comportant un nombre pair de* a*.*
4. $(ab\{, 4\})*$ *: Tous les mots commençant par a où chaque a est suivi par au 4* b *au plus (en plus de* ε*).*

6.1.4 Usage pratique des ERs

Les ERs sont très populaires dans la manipulation des données (textuelles) surtout quand il s'agit d'effectuer des opérations de recherche et de remplacement avancées dans les éditeurs de texte et les scripts de traitement de données

Recherche de motifs

Les éditeurs de texte prenant en charge les ERs, tels que Notepad++, Kate, Gedit, Brackets, Atom, Visual Studio Code, et d'autres, permettent d'effectuer des recherches en définissant des motifs de recherche sous forme d'ERs.

Exemple **Recherche de tous les entiers dans un fichier**

Pour chercher tous les entiers dans un fichier (ouvert sur VS code) il suffit de taper dans la barre de recherche (en activant l'option "Expression régulière"). L'ER suivante :

Motif de recherche

$\backslash d+$ *ou* $[0-9]+$

Toutes les correspondances seront mises en surbrillance.

Remplacement

Avec tels éditeurs de texte ou des langages de programmation comme Python, il est possible de définir non seulement des motifs de recherche mais aussi effectuer des remplacements basés sur les ERs.

Exemple **Remplacement**

Si on veut remplacer tous les entiers par le mot "entier", il suffit de fournir deux entrées : le motif de recherche (vu précédemment) et motif de remplacement (dans le champs de remplacement).

Remplacement

entier

En choisissant "Remplacer tout" on peut remplacer toutes les occurrences des entiers par le mot "entier" dans le fichier.

Remplacement contextuel

Les ERs sont aussi utiles dans les remplacements contextuels. En utilisant les parenthèses, il est possible de définir et cibler des groupes (des sous-chaînes) à remplacer/modfier.

Exemple **Miroir d'un entier à deux chiffres**

— *Première entrée :*

Motif de recherche avec des groupes

([0 - 9])([0 - 9])

C'est une ER qui permet de chercher une séquence de deux chiffres. Chaque paire de parenthèses définit un groupe (un chiffre). Ces deux groupes sont respectivement accessibles par \1 et \2.

— *Deuxième entrée :*

Remplacement contextuel

\2\1

Pour obtenir le miroir de ces entiers, il suffit d'inverser l'ordre de chaque séquence rencontrée en utilisant la notation des groupes définis dans la première entrée.

Remarque 6.1.5

— Les exemples précédents montrent que les ERs permettent de faciliter la manipulation des données : Recherche/extraction, remplacement, remplacement contextuel, etc. (Même sur une grande quantité de données).

— Les expressions régulières sont prises en charge par beaucoup de langages de programmation (Python et le module 're'). Elles sont employées pour valider des données entrées par les utilisateurs, en garantissent que les données soient correctement formatées : adresses email, mot de passe, numéros de téléphone, codes postaux, etc. (Des illustrations en TP)

6.2 Équivalences entre ER, AEF et grammaire

Selon le mathématicien américain Stephen Cole Kleene, il est possible d'établir une équivalence entre les trois représentations des langages réguliers : ERs, les AEFs et les grammaires régulières et qui permet de passer d'une représentation à une autre.

Théorème 6.2.1 de Kleen

Tout langage régulier, généré par une grammaires régulière, peut être défini par une ER ou AEF. Autrement dit si :

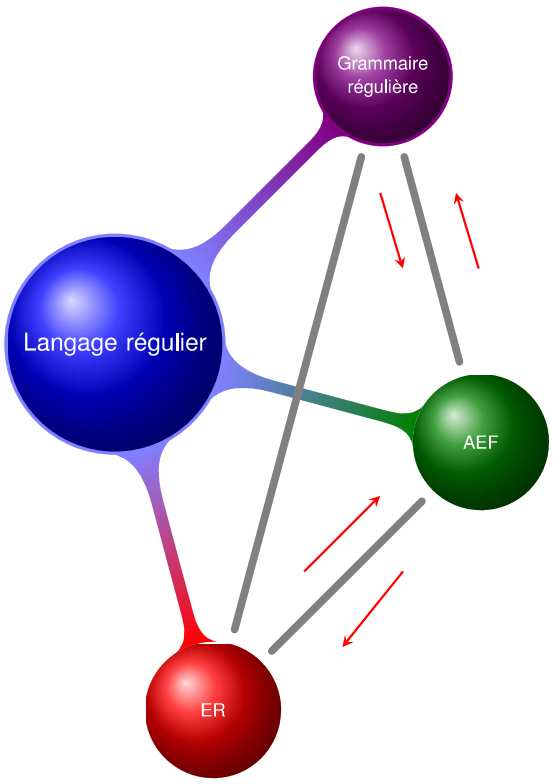
— Λ_{rat} représente l'ensemble des langages réguliers ou rationnels (générés par des grammaires régulières).

— Λ_{reg} représente l'ensemble des langages décrits par toutes les ERs.

— Λ_{AEF} représente l'ensemble des langages acceptés par un AEF.

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

FIGURE 6.1 – Équivalences entre ER, AEF et grammaire



6.2.1 Convertir un automate en une ER

Le passage d'un AEF $A(X, Q, s_0, F, \delta)$ vers une ER (qui décrit le langage accepté) consiste à construire et résoudre un système d'équation.

Remarque 6.2.1

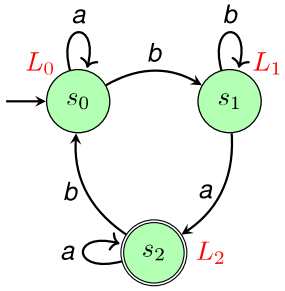
Il faut noter tout d'abord que lorsqu'on se trouve dans un état s_i d'un AEF on est en train de lire un langage L_i. Si on associe à chaque état s_i, un langage L_i, alors trouver le langage accepté par l'automate revient à trouver L_0 étant donné que l'analyse commence à partir de l'état initial s_0.

1. Le système d'équation se construit comme suit :
   - Associer à chaque état $s_i \in Q$ un langage L_i.
   - Si $\delta(s_i, a) = s_j$ on aura l'équation : $L_i = aL_j$
   - Si $s_i \in F$ alors on aura l'équation $L_i = \varepsilon$.
   - Si $L_i = \alpha$ et $L_i = \beta$ alors on écrit : $L_i = \alpha|\beta$.

   Il faut numéroter les équations.
2. Résoudre le système d'équation revient à trouver L_0. Pour y arriver, on va effectuer une série de substitutions tout en utilisant une règle clé 🔑
   ▶ Si on a l'équation $L = \alpha L|\beta$ alors $L = \alpha^*\beta$.
   Cela permet d'éliminer L à droite (Mais si on obtient $L = \alpha L$ alors il y a une faute de raisonnement).

Exemple **Passer d'un AEF vers l'ER**

Essyons de trouver le langage accepté (ER) par l'AEF suivant :



- *Aprés avoir associé à chaque état un lagnage, le système d'équations est le suivant :*
  1. $L_0 = aL_0|bL_1$.
  2. $L_1 = bL_1|aL_2$.
  3. $L_2 = aL_2|bL_0|\varepsilon$.
- *Résolution :*
  1. *L'équation (1) devient :* $L_0 = a^*bL_1$, *en appliquant la règle clé.*
  2. *L'équation (2) devient :* $L_1 = b^*aL_2 = b^*a^+bL_0|b^*a^+$, *en appliquant la règle clé et en remplaçant (3) dans (2)*
  3. *L'équation (3) devient* $L_2 = a^*bL_0|a^*$, *en appliquant la règle clé*

  En remplaçant (2) dans (1) on obtient : $L_0 = a^*b^+a^+bL_0|a^*b^+a^+$. *Il ne nous reste qu'appliquer la règle clé encore une fois pour trouver* $L_0 = (a^*b^+a^+b)^*a^*b^+a^+$.

Remarque 6.2.2

Selon l'ordre d'élimination des variables, il est possible d'obtenir plusieurs ERs relativement différentes associées à un AEF, mais qui dénotent le même langage.

6.2.2 Convertir une ER en un automate

Pour construire un AEF acceptant un langage L à partir de son ER, on va étudier et comparer deux méthodes différentes. La première est basée sur la notion de dérivée et la deuxième est celle de Thompson et qui s'appuie sur les propriétés des langages réguliers.

La méthode des dérivées

Avant d'étudier la construction d'un AEF avec la méthode des dérivées, nous devons aborder quelques notions essentielles liées à la dérivée d'un langage

Définition 6.2.1 Dérivée d'un mot

La dérivée d'un mot w (défini sur l'alphabet X) par rapport à symbole a est définie par : $w||a = v$ si seulement si $w = av$ tel que $a \in X$ et $v \in X^*$

Définition 6.2.2 Dérivée d'un langage

La dérivée d'un langage L par rapport à un mot $z \in X^*$ est notée $L||z = \{u \in X^* | \exists w \in L : w = zu\}$

Exemple La dérivé des lagnages

Soit $L_1 = \{a, ab, aa, ba\}$, $L_2 = \{1, 01, 11\}$, $L_3 = \{a^n | n \geq 0\}$ *:*

- $L_1||a = \{\varepsilon, b, a\}$. $L_1||aa = \{\varepsilon\}$.
- $L_2||0 = \{1\}$. $L_2||1 = \{\varepsilon, 1\}$
- $L_3||a = L_3$.

Remarque 6.2.3 Propriétés des dérivées

- $a_i||a_j = \begin{cases} \varepsilon \text{ si } a_i = a_j \\ \emptyset \text{ si } a_i \neq a_j \end{cases}$
- $L||z = \emptyset \Rightarrow \forall w \in L$, w ne commence pas par z.
- $L||z_1.z_2 = (L||z_1)||z_2$.
- $(\sum_{i=1}^{n} L_i)||z = \sum_{i=1}^{n}(L_i||z)$
- $(L.L')||z = (L||z).L' + f(L).(L'||z) = \begin{cases} (L||z).L' + \emptyset = (L||z).L', f(L) = \emptyset \text{ si } \varepsilon \notin L \\ (L||z).L' + (L'||z), f(L) = \{\varepsilon\} \text{ si } \varepsilon \in L \end{cases}$
- $L^*||z = (L||z)L^*$.

Algorithme

Algorithm 5 Construction d'un AEF à partir d'une ER, basée sur les dérivées

Require: Une ER r définie sur X
Ensure: Un AEF A
1: $L_0 = r$
2: **repeat**
3: **for all** $a \in X$ **do**
4: $L_j = L_i||a$, Calculer la dérivée de L_i par rapport à chaque symbole de X
5: **end for**
6: **until** il y a plus de nouveau langage L_j
7: Définir $A(X, Q, L_0, F, \delta)$ tel que
— Chaque langage L_i représente un état dans Q.
— L'état initial correspond au premier langage $L_0 = r$.
— Les états finaux $L_i \in F$ si $\varepsilon \in L_i$.
— Une transition $\delta(L_i, a) = L_j$ pour chaque $L_i||a = L_j$
8: **return** A

Exemple Appliquer la méthde des dérivées

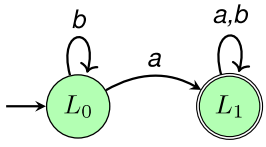
Soit l'ER suivante $(a|b)^*a(a|b)^*$
Commençons par mettre $L_0 = (a|b)^*a(a|b)^*$, *sachant que* $(a|b)||a = (a||a)|(b||a) = \varepsilon$, *et* $(a|b)^*||a = ((a|b)||a).(a|b)^* = (a|b)^*$.

— *Première itération :*
— $L_0||a = [\underbrace{(a|b)^*}_{L}\underbrace{a(a|b)^*}_{L'}]||a = ((a|b)^*a(a|b)^*)|(a|b)^* = (a|b)^* = L_1$.
— $L_0||b = [\underbrace{(a|b)^*}_{L}\underbrace{a(a|b)^*}_{L'}]||b = (a|b)^*a(a|b)^* = L_0$

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

Au bout de la deuxième itération, on remarque qu'il n y a plus de nouveau langage.
Donc on construit l'AEF $A(\{a, b\}, \{L_0, L_1\}, L_0, \{L_1\}, \delta)$ *dont les transitions sont définies par le résultat du calcul précédent :*

État	a	b
L_0	L_1	L_0
L_1	L_1	L_1



Remarque 6.2.4

Il faut noter que cette méthode des dérivées ne s'applique pas uniquement sur les langages réguliers (sur les ERs). Elle s'applique aussi sur tout type de langage. Mais elle est considérée comme un moyen permettant de différencier les langages réguliers des non réguliers.

Théorème 6.2.2 de Brzozowski

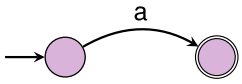
Une expression régulière ER n'a qu'un nombre fini de dérivées distinctes.

La méthode des dérivées est souvent une technique utilisée pour prouver la régularité des langages. Si la méthode des dérivées produit un nombre fini de dérivées distinctes, et donc un nombre fini d'états (AEF) cela montre que le langage en question est régulier.

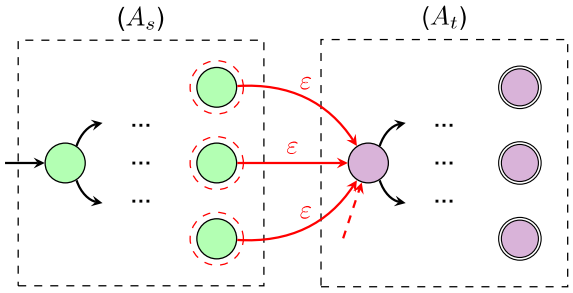
Méthode de Thompson

La méthode de Thompson est une autre méthode permettant de convertir une ER en un automate fini reconnaissant le même langage. Puisque les expressions rationnelles sont formellement définies de manière inductive (récursive), cette méthode s'appuie sur la décomposition de l'ER, en commençant par construire les automates finis reconnaissant les briques de base : $\emptyset, \varepsilon, a \in X$

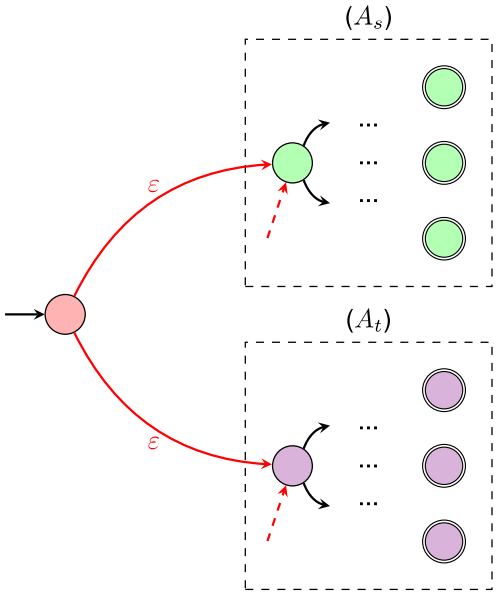
- **Les briques de base :**
  Si $ER = a, a \in X$ ou $ER = \varepsilon$ (une brique de base), on construit l'AEF suivant :

  Si $ER = \emptyset$ alors l'automate précédent ne contient aucune transition.
  À partir de ces automates élémentaires, nous construisons de manière itérative des automates acceptant des ERs plus complexes, selon les opérations utilisées jusqu'à arriver à construire l'AEF acceptant le langage tout en entier.
- **L'automate de la concaténation :**
  Si $ER = st$, tel que A_s accepte s et A_t accepte t, alors L'AEF acceptant le langage dénoté par st est obtenu en reliant avec des ε-transition, les états finaux de A_s (qui ne deviennent plus des états finaux) avec l'état initial de A_t (qui ne l'est plus).

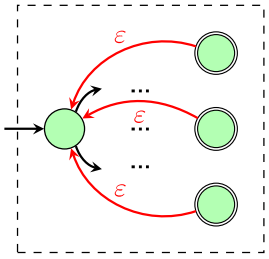
  C'est une mise en série des deux automates A_s et A_t de telle sorte que l'état initial de A_{st} soit celui de A_s, et ses états finaux soient ceux de A_t.
- **L'automate de l'union :**
  Si $ER = s|t$, alors l'automate $A_{s|t}$ acceptant le langage dénoté par $s|t$ est obtenu en créant un nouvel état initial qui va relier avec ε-transitions l'état initial de A_s et celui de A_t. Ces derniers ne deviennent plus des états initiaux.



Tandis que les états finaux de $A_{s|t}$ sont ceux de A_s et de A_t. C'est une mise en parallèle de A_s et A_t.

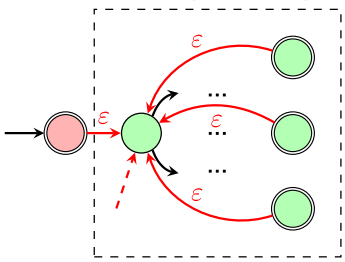
- **L'automate de l'étoile de Kleene :**
  - Si $ER = s^+$ Il suffit de relier les états finaux de A_s à son propre état initial avec ε-transitions.

    Modifier A_s pour accepter s^+



  - Si $ER = s^*$: alors on construit d'abord l'automate acceptant s^+ (vu précédemment). Mais pour s'assurer que ε soit accepté (ce qui distingue la fermeture transitive de Kleen), on doit créer un nouvel état initial qui sera en même temps un état final et le relier avec une ε-transition à l'état initial de A_s et qui ne devient plus un état initial.

    Modifier A_s pour accepter s^*



Déduction 6.2.1

À partir de ces constructions simples, la méthode de Thompson nous permet de construire un automate reconnaissant le langage dénoté par une ER quelconque en la décomposant en composants élémentaires.

Remarque 6.2.5

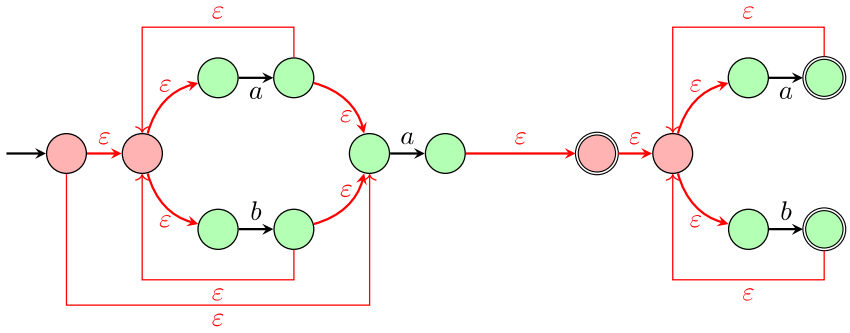
- Il existe une autre variante de construction de Thomposon selon laquelle, on a un état initial qui

n'est pas final et un unique état final sans transition sortante. Ceci donnera deux fois plus d'états que de symboles (autres que la concaténation) dans une ER. Cette construction n'est pas pris en charge dans ce cours.

- Il existe par ailleurs d'autres méthodes de construction comme celle de Glushkov permettant de décomposer de manière efficace les ERs ne contenant pas ε.

Exemple **Application de la méthode de Thompson**

Reprenons l'ER précédente : $(a|b)^*a(a|b)^*$. Mais cette fois, nous allons construire l'automate acceptant le langage dénoté en appliquant la méthode de Thompson.



Comparaison

Déduction 6.2.2 **Méthode des dérivées Vs. Méthode de Thompson**

Pour convertir une ER en un AEF, nous avons vu et appliqué deux méthodes : la méthode de Thompson et celle des dérivées sur le même langage. Cela nous permettra de déduire la comparaison suivante :

Méthode des dérivées	Méthode de Thompson
- Plus Complexe et nécessite beaucoup de calcul	- Plus simple à appliquer et à programmer
- Elle produit l'AEF déterministe et minimal du langage	- Elle produit un AEF non-déterministe avec beaucoup d'états (il y a d'autres méthodes plus complexes qui produisent moins d'états)
- Difficulté de décider si deux ERs sont équivalentes	
- S'applique sur tout type de langage	- S'applique sur les langages réguliers (nécessite une ER)

6.2.3 Convertir un automate en une grammaire

Après avoir vu comment passer de l'AEF vers l'ER et vice-versa, il est temps maintenant d'étudier le lien entre un AEF et une grammaire. À partir d'un AEF $A(X, Q, s_0, F, \delta)$ on peut définir une grammaire G régulière et normalisée de telle sorte que $L(G) = L(A)$.

Définition 6.2.3 **Grammaire normalisée**

$G = (V, N, S, R)$ est une grammaire régulière (à droite), dite normalisée si toutes les règles de production prenent l'une des formes suivantes :

— $A \to aB, A, B \in N, a \in V$ ce qui veut dire que $|a| = 1$.

— $A \to \varepsilon, A \in N$.

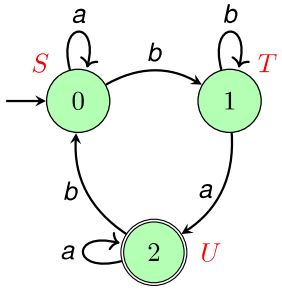
Obtenir une grammaire régulière à droite

Maintenant pour définir G à partir de A :

- $V = X$.
- N : Chaque état de Q est associé à un non-terminal de telle sorte que $card(Q) = card(N)$.
- S est associé à l'état inital s_0.
- R :
  - Chaque transition $\delta(s, a) = s'$ va nous donner une règle $A \to aB$ tel que $A \in N$ est déjà associé à s et $B \in N$ est déjà associé à s'.
  - Pour chaque état final $s \in F$, associé à un non-terminal $A \in N$ on ajoute la règle $A \to \varepsilon$.

? Question. et les transitions $\delta(s, \varepsilon) = s'$?

Exemple **Trouver la grammaire qui génère le langage accepté par cet AEF**



Après avoir associé un non-terminal à chaque état de cet automate A, nous définissons formellement la grammaire $G = (\{a, b\}, \{S, T, U\}, S, R)$ qui génère $L(A)$ de telle sorte que les règles R soient définies comme suit :

- $\delta(0, a) = 0 \Rightarrow S \to aS$
- $\delta(0, b) = 1 \Rightarrow S \to bT$
  On a deux règles ayant le même non-terminal S à gauche donc on peut écrire : $S \to aS|bT$
- $\delta(1, b) = 1 \Rightarrow T \to bT$
- $\delta(1, a) = 2 \Rightarrow T \to aU$
  Donc on a : $T \to bT|aU$
- $\delta(2, b) = 0 \Rightarrow U \to bS$
- $\delta(2, a) = 2 \Rightarrow U \to aU$
- *U est un état final ce qui donne $U \to \varepsilon$*
  Donc on a : $U \to bS|aU|\varepsilon$

? Question. La méthode ci-dessus permet de définir une grammaire régulière à droite. Mais est-il possible de trouver à partir d'un AEF une grammaire $G = (V, N, S, R)$ régulière à gauche, dont toutes les règles prennent la forme suivante : $A \to Ba$ tel que $A, B \in N, a \in V$ ou $A \to \varepsilon$?

Obtenir une grammaire régulière à gauche

Selon la méthode précédente, il n'est pas possible de déduire directement une grammaire régulière à gauche à partir de l'AEF A. Mais il existe une construction astucieuse qui se fait selon les étapes suivantes :

1. Construire l'AEF A^R acceptant L^R
2. Trouver la grammaire G^R régulière à droite à partir de A^R et qui génère L^R.
3. Appliquer le miroir aux parties droites (aB) de toutes les règles de G^R, ce qui donne la grammaire G régulière à gauche qui produit le langage miroir $(L^R)^R$ (voir le chapitre des grammaires).

▶ Donc G sera une grammaire régulière à gauche qui génère $L(A)$.

6.2.4 Convertir une grammaire en un automate

Construire un AEF $A(X, Q, s_0, F, \delta)$ à partir d'une grammaire régulière (à droite), normalisée $G = (V, N, S, R)$ revient à définir :

- $X = V$.
- Q : Chaque non-terminal de N est associé à un état de telle sorte que $card(N) = card(Q)$.
- s_0 : L'état initial est associé à l'axiome.
- δ : Les transtions sont définies comme suit :
  - — Chaque règle $A \to aB$ nous permet d'ajouter à l'auotmate une transition $\delta(s_A, a) = s_B$, tel que s_A est associé à A, s_B est associé à B et $a \in V$. Mais, si $a = \varepsilon$ alors la règle $A \to B$ est convertie en une ε-transition de s_A vers s_B
  - — Chaque règle $A \to \varepsilon$ permet de considérer l'état s_A comme état final.

Remarque 6.2.6

Pour appliquer cette construction, il faut que la grammaire soit normalisée.

Question. Mais si ce n'est pas le cas, que faut-t-il-faire ?

Il faut tout d'abord passer par une normalisation de la grammaire.

Normalisation d'une grammaire règulière

Nous nous intéressons à la normalisation d'une grammaire régulière à droite non normalisée.

Algorithme

Algorithm 6 Normalisation d'une grammaire régulière à droite

Require: Une grammaire régulière à droite non-normalisée $G = (V, N, S, R)$
Ensure: Une grammaire régulière à droite normalisée G'

for Chaque règle de R **do**
  if la règle est sous forme : $A \to wB, (A, B \in N, w \in V^*, |w| > 1)$ **then**
    repeat
      ajouter un nouveau non-terminal B' à N.
      éclater $A \to wB$ en deux règles $A \to aB'$ et $B' \to uB$ tel que : $w = au$ et $a \in V$.
    until obtenir une règle où B est précédé par un seul terminal.
  end if
  if la règle est sous forme $A \to w, (A \in N, w \in V^*, |w| \geq 1)$ **then**
    repeat
      ajouter un nouveau non-terminal B' à N.
      éclater $A \to w$ en deux règles $A \to aB'$ et $B' \to u$ tel que : $w = au$ et $a \in V$.
    until obtenir une règle contenant seulement ε à droite.
  end if
  Ajouter les nouvelles règles normalisées à R.
end for
return G'

Exemple **Normaliser la grammaire suivante**

$G = (\{a, b\}, \{S, T\}, S, \{S \to aabS|bT, T \to aS|bb\})$.
Les règles concernées par la transformation sont : $S \to aabS, T \to bb$

— *éclater* $S \to aabS$ *en*

- $S \to aS_1$
- $S_1 \to abS$. *Cette dernière est également éclatée en :*

- $S_1 \to aS_2$
- $S_2 \to bS$

— *éclater* $T \to bb$ *en :*

- $T \to bT_1$.
- $T_1 \to b$. *Cette dernière est également éclatée en :*
  - $T_1 \to bT_2$
  - $T_2 \to \varepsilon$

Trouver l'AEF acceptant $L(G)$ *? ?*

Déduction 6.2.3 Faire le lien entre une grammaire et une ER

Pour trouver une grammaire (qui ne sera pas forcément régulière) à partir d'une ER, on peut appliquer un algorithme similaire à celui de Thomposon. Suivant la même idée qui consiste à décomposer une ER qui dénote un langage L, on commence par construire les grammaires qui génèrent les briques élémentaires, puis on recompose jusqu'à arriver à une grammaire qui génère L.

Exemple Déduire une grammaire à partir d'une ER

Soit $ER = (a|b)^*$. *Commençons par :*

— *La grammaire qui génère* $\{a\}$ *:* $G_1 = (\{a\}, \{S\}, S, \{S \to a\})$

— *La grammaire qui génère* $\{b\}$ *:* $G_2 = (\{b\}, \{S'\}, S', \{S' \to b\})$.

— ...

Compléter la construction jusqu'à atteindre la grammaire qui génère $(a|b)^*$ *(?).*

6.3 Propriétés des langages réguliers

6.3.1 Fermeture et Stabilité

Rappelons nous le théorème de la fermeture d'un langage (vu dans le chapitre des grammaires) qui nous annonce que l'application des opération régulières (union, concaténation, fermeture de Kleen) préserve le type de ou des langages. La classe des langages réguliers n'échappera pas à cette règle. Nous pouvons ainsi déduire :

Déduction 6.3.1 Fermeture (Langages rationnellement clos)

- Etant donné que $a \in X$ représente un langager régulier, tous les langages finis sont réguliers. Car ils qui se déduisent en réalité des singletons sur lequels on applique un nombre fini d'opérations : d'union et de concaténation.
- En utilisant les ERs, nous avons vu que les langages composés par des opérations régulières : $*, ., |$ sont inductivement réguliers. Alors on dit que l'ensemble des langages rationnels est rationnellement clos (par rapport à ces opérations).

Ce dernier point est important, car il fournit une première méthode pour prouver qu'un langage est régulier car il suffit de trouver une ER qui dénote ce langage.

Déduction 6.3.2 Stabilité

Les opérations que nous avons étudiées sur les AEFs ainsi que l'équivalence entre AEF et ER, nous ont montré que les langages réguliers restent stables dans le sens où ils sont à chaque fois reconnaissables par des automates finis (non seulement par rapport aux opérations régulières), C'est-à-dire :

Si L_1 et L_2 sont deux langages réguliers, acceptés deux AEFs A_1, A_2 et dénotés par r et s respectivement, alors :

- $L_1 + L_2$ reste régulier car il est accepté par l'AEF $A_{r|s}$ obtenu par la construction d'un automate appliquée sur l'ER $r|s$.
  Même si l'union est considérée comme opération régulière, l'union infinie des langages réguliers ne donnent pas forcément un langage régulier (La stabilité ici dépend du nombre fini d'opérations)
- $L_1.L_2$ reste régulier car il est accepté par l'AEF $A_{r.s}$ obtenu par la construction d'un automate appliquée sur l'ER $r.s$.
- $L_1^*, (L_1^+)$ reste régulier car il est accepté par un AEF obtenu par la construction d'un automate appliquée sur l'ER $r^*, (r^+)$.
- L_1^R reste régulier car il est accepté par l'AEF A^R obtenu en inversant le sens des arcs (voir les opérations sur les AEFs dans le chapitre précédent).
- $L_1 \cap L_2$ reste régulier car il est accepté par un AEF obtenu en calculant le produit $A_1 \times A_2$ (voir les opérations sur les AEFs dans le chapitre précédent).
- $\overline{L_1}$ reste régulier car il est accepté par l'AEF obtenu en inversant le statut final/ non-final des états de la version déterministe et complète de A (voir les opérations sur les AEFs dans le chapitre précédent).

6.3.2 La régularité d'un langage

Il existe différents critères permettant de démontrer la régularité d'un langage. Mais certains ne sont pas suffisants pour affirmer qu'un langage est régulier.

Conditions nécessaires et suffisantes

- Tout langage fini est régulier
- S'il existe une grammaire régulière qui génère un langage L alors L est régulier (Mais ça ne veut pas dire que L est fini)
- Selon le théorème Brzozowski (avec la méthode des dérivés) il suffit de trouver un AEF qui accepte un langage L pour affirmer que L est régulier.
  Sinon, la méthode des dérivées produit un nombre infini de dérivées pour tout langage non régulier.

Exemple **Les dérivées pour montrer ou non la régularité d'un langage**

Nous allons appliquer la méthode des dérivées sur le langage $L = \{a^n b^n | n \geq 0\}$ par rapport à un symbole a.

- $L_0||a = \{a^{n-1}b^n | n \geq 1\} = L_1$ *un nouveau langage.*
- $L_1||a = \{a^{n-2}b^n | n \geq 2\} = L_2$ *un nouveau langage.*
- ...
- $L_i||a = \{a^{n-(i+1)}b^n | n \geq i+1\} = L_{i+1}$ *encore un autre langage.*

Tant que $n \geq 0$ est un entier quelconque, il n'y a aucune garantie que les opérations de dérivation s'arrêtent.
Cela produit un nombre infini d'états, et par conséquent, le langage L n'est pas régulier (il est en réalité algébrique).

Théorème 6.3.1 de Myhill-Nerode

Un langage L défini sur X est régulier si et seulement si le nombre de ses dérivées par rapport aux mots de X^* est fini.

- Il suffit aussi de trouver une ER qui dénote un langage pour dire qu'il régulier.
  Les propriétés de fermeture sont également un moyen pour montrer qu'un langage est régulier.

Remarque 6.3.1

Etant donné que si un langage L est règulier, alors il existe une ER. Alors, réciproquement, si L est dénoté par une ER, alors il est lui-même régulier. Ceci se montre par induction sur le nombre d'opérations pour définir l'ER.

Lemme de la pompe

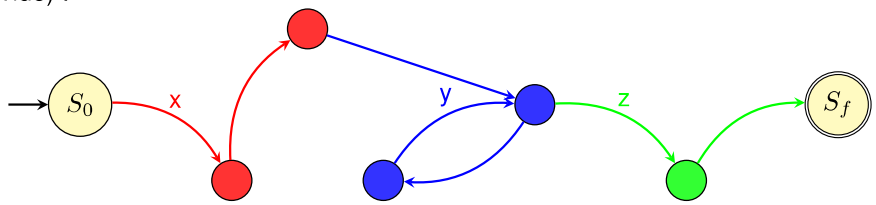
Le lemme de la pompe ou lemme de pompage (on l'appelle aussi lemme de l'étoile ou d'itération) permet de mieux comprendre la régularité d'un langage régulier. En revanche, il présente une condition nécessaire mais pas suffisante pour qu'un langage soit régulier.

Lemme 6.3.1 de la pompe ou de l'étoile

Soit L un langage régulier infini défini sur l'alphabet X. $\exists n \in N$ (un entier), tel que pour tout mot $w \in L$, si $|w| \geq n$ alors :

- $\exists x, y, z \in X^*$ et $|y| \geq 1$ tels que $w = xyz$
- $|xy| \leq n$
- $xy^iz \in L, i \geq 0$ (pomper y)

Pour faciliter la compréhension de ce lemme, il suffit d'imaginer un AEF A acceptant L (car L est régulier) avec un certain nombre fini d'états qui est n. Si le mot $w = xyz \in L$ (donc A accepte w) dont la longueur $|w| \geq n$ il existe au moins deux états sur le chemin d'acceptation qui sont "égaux" (il y a au moins une boucle non vide) :



Exemple Un langage régulier vérifiant le critère de la pompe

Soit le langage régulier suivant $L = a^*b^* = \{a^pb^q | p, q \geq 0\}$. *Allons voir si* L *vérifie le critère de la pompe, pour* $n = 1$. *Prenons un mot* $w = a^pb^q$ *tel que* $|w| = p + q \geq 1$.
Pour identifier le y *ainsi que* x *et* z *on peut envisager deux décompositions*
Dans un premier cas : $p > 0$

- $y = a, x = a^{p-1}, z = b^q$.
- *On remarque que tout mot* $xy^iz = a^{p-1+i}b^q \in L$

Dans un deuxième cas : $p = 0$

- $y = b, x = \varepsilon, z = b^{q-1}$.
- *On remarque que tout mot* $xy^iz = a^pb^{q-1+i} \in L$

Donc le critère de la pompe est vérfié.

Remarque 6.3.2

Le lemme de l'étoile (de la pompe) est quasiment utilisé pour prouver qu'un langage donné n'est pas régulier en raisonnant par l'absurde

Exemple Pour montrer l'irrégularité d'un langage en raisonnant par l'absurde avec le lemme de la pompe

Montrons que $L = \{a^nb^n | n \geq 0\}$ *n'est pas régulier.*

- *Hypothèse : supposons que L est réguiler :*
  L est réguiler $\Rightarrow$ L satisfait le lemme de la pompe et donc reconnaissable par un AEF ayant un certain nombre d'états n comme on l'a vu ci-dessus.
  Donc si on prend un mot $w = a^n b^n$ il devrait vérifier le lemme de la pompe mais il faut identifer le facteur y. Mais il y a plusieurs décomposition possibles :
  - *Cas 1 :*
    - $\underbrace{a \dots a}_{x} \underbrace{a^m}_{y} \underbrace{a \dots ab \dots b}_{z}$ *sachant que* $m \geq 1$ *car* $y \neq \varepsilon$
    - *Si* $i = 2$*, selon le lemme le mot* $xyyz = a^{n+m}b^n \in L$ *(contradiction)*
  - *Cas 2 :*
    - $\underbrace{a \dots ab \dots b}_{x} \underbrace{b^m}_{y} \underbrace{b \dots b}_{z}$
    - *Si* $i = 2$*, selon le lemme le mot* $xyyz = a^n b^{n+m} \in L$ *(contradiction)*
  - *Cas 3 :*
    - $\underbrace{a \dots a}_{x} \underbrace{a^k b^m}_{y} \underbrace{b \dots b}_{z}$ *,* m *et* k *ne peuvent pas être nuls en même temps*
    - *Si* $i = 2$*, selon le lemme le mot* $xyyz = a^n b^m a^k b^n \in L$ *(contradiction)*

Selon tous les cas étudiés, nous obtenons une contradiction, ce qui veut dire que l'hypothèse est fausse. Ainsi, L n'est pas régulier.

Remarque 6.3.3

- Le lemme de l'étoile affirme l'irrégularité d'un langage L (s'il n'est pas vérfié) mais ce n'est pas une condition nécessaire pour affrimer la régularité d'un langage.
  Si un langage L vérifie le lemme de la pompe, cela ne permet pas d'affirmer que L est régulier.
- Il existe néanmoins une version nécessaire et suffisante de ce lemme permettant d'affirmer si un langage est régulier ou non.

Exemple un langage vérifiant le lemme de la pompe mais il n'est pas régulier

Prenons un langage arbitraire non régulier $L_1 \subset b^*$. Le langage $L = a^+ L_1 | b^*$. vérifie le lemme de la pompe (par exemple pour $n = 1$).

6.4 Série d'exercices de TD N°4

Exercice 1 : Définir un langage dénoté par une ER

Pour chacune des ERs suivantes, donnez une description (ou une définition ensembliste) du langage dénoté ainsi que la notation POSIX de l'ER :

1. $11(1|0)00$
2. $(aaa)^*$
3. $(11)^*(0|1)00$
4. $a(a|b)^*(b|\varepsilon)$
5. $(a|ab)^*$
6. $a(a|b)^*(aa|bb)^+$
7. $(a|ab|abb)^*$

Exercice 2 : Trouver l'ER d'un langage

Donnez une ER en notation POSIX qui dénote chacun des langages suivants :

1. *Langage des mots définis sur $\{a, b, c, d\}$ de longueur 2 commençant par a.*
2. *Les mot définis sur $\{a, b, c\}$ contenant le facteur a^5*
3. *Les nombres binaires commençant par 1 et se terminant par 1 ou 10*
4. *Les nombres entiers multiples de 5 (en base 10).*
5. *Les codes postaux de la wilaya d'Annaba*
6. *Les identifiants en C.*

Exercice 3 : Convertir une ER en AEF (en montrant la régularité d'un langage)

Sur une plateforme en ligne, nous voulons traiter et valider des codes utilisateur définis sur $\{x, y, z\}$.

1. *En appliquant la construction de Thompson trouvez l'AEF correpondant à l'ER qui dénote tous les codes commençant par x.*
2. *En appliquant la méthode des dérivées, comment peut-on montrer que le langage en question est régulier ?*

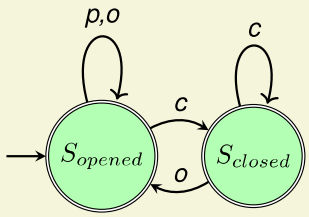
Exercice 4 : Convertir une ER en AEF

Pour chacun des langages suivants, trouvez l'ER et puis construisez l'AEF correspondant en appliquant la méthode de Thompson :

1. *Tous les mots sur $\{a, b\}$ commençant par un symbole différent de leur dernier symbole.*
2. *Les codes-barres définis sur $\{0, 1, 2\}$ contenant au moins deux 0.*
3. *Les codes-barres définis sur $\{0, 1, 2\}$ contenant au plus deux 0.*
4. *Les séquences ADN définies sur $\{A, G, T\}$ qui ne doivent pas contenir le codon AAG*

Exercice 5 : Controleur d'une porte automatique

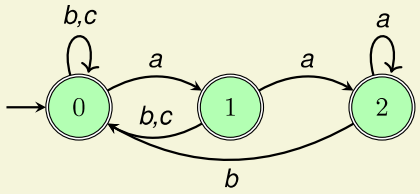
Nous avons un simple contrôleur de porte automatique dont le fonctionnement est défini par l'AEF suivant



Le contrôleur reçoit 3 types de signaux $\{p, o, c\}$: p : corps détecté devant la porte, o boutton 'open' appuyé, c : bouton 'close' appuyé.

— *Déduisez l'ER correpondante sans passer (par le système d'équation) et puis construisez l'AEF non-déterministe équivalent (qui accepte le même langage) en appliquant la méthode Thompson.*

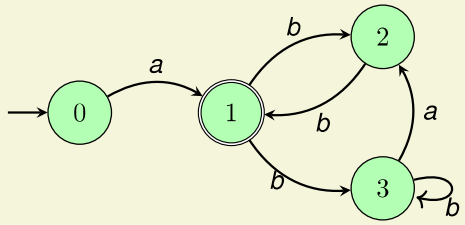
Exercice 6 : Convertir un AEF en une ER, Ambiguité, Correspondance, Fermeture des langages réguliers



1. *Trouvez l'ER qui dénote le langage L accepté par cet AEF. Que représente ce langage ?*
2. *Montrez que L reste stable (rationnellement clos) par rapport à son complément $\overline{L}$.*
3. *Déduisez l'ER qui dénote $\overline{L}$*
4. *En utilisant cette dernière ER, montrez que le mot $bcaacbbaac$ appartient à $\overline{L}$ Que peut-on dire sur l'ambiguïté de cet ER ?*

Exercice 7 : Passage de l'AEF vers : ER/ Grammaire règulière

1. *Trouvez le langage accepté par cet AEF.*



2. *Trouvez la grammaire régulière qui génère. ce langage*

Exercice 8 : Convertir une grammaire régulière en AEF, Normalisation

1. *Trouvez l'AEF A acceptant le langage $L(G)$, tel que $G = (\{0,1\}, \{S,T\}, S, \{S \rightarrow 01S|0S|1T|\varepsilon, T \rightarrow 10T|1T|0S|\varepsilon\})$*
2. *Déduisez l'ER qui dénote L (sans passer nécessairement par l'AEF).*

6.5 Série de TP N°4

L'objectif de cette série d'exercices de travaux pratiques est de manipuler les expressions régulières ERs, soit en programmant avec ou bien comme un moyen pour manipuler des données dans un fichier. Pour y arriver, on aura besoin de :

Prérequis

- *Des notions sur la recherche et le remplacement des motifs en utilisant un éditeur de texte supportant les ERs.*
- *Syntaxe pratique permettant de manipuler les expressions régulières (notation POSIX et quelques extensions).*
- *Notions de programmation liées aux ERs avec le module 're' en Python.*

Exercice 1: Définir des motifs (groupes) de recherche

Chargez le fichier data.txt (qui sera l'objet de notre manipulation) dans un éditeur de votre choix. Le plus recommandé : Visual Studio Code (VSCode) ou Notepad++, Sublime Text, Atom, etc.
Effectuez une recherche en proposant à chaque fois une ER permettant de trouver les motifs suivants :

1. Les nombres de 4 chiffres.
2. Les mots commençant par une majuscule.
3. Les mots se trouvant à la fin d'une phrase.
4. Les mots se trouvant entre parenthèses (un seul mot à la fois)
5. Les phrases qui commencent par une lettre majuscule et ne contenant aucun chiffre.
6. Les mots alphanumériques word qui se répètent deux fois tel que word :word (Les deux occurrences sont séparées par deux points)

Note : Si vous utilisez VSCode, n'oubliez pas d'activer l'option "ER" et "Respecter la casse" sur la barre de recherche.

Exercice 2: Modification (Remplacement et groupes de remplacement)

Avec le même fichier précédent, utilisez votre éditeur pour appliquer les modifications suivantes :

1. Toutes les séquences de caractères (y compris les parenthèses imbriquées) qui sont entre parenthèses seront placées entre crochets
2. Toute phrase doit être écrite sur une ligne séparée
3. Lorsqu'on a une séquence commençant par un mot w (composé de lettres alphabétiques) suivi par un nombre, suivi par le même mot w on va la remplacer par w suivi par le nombre uniquement en supprimant la deuxième occurrence de w.

Note : Avec VSCode, la désignation des groupes de remplacement se fait par $1, $2, etc. au lieu $\backslash$ 1, $\backslash$ 2, etc.

Quelques notions sur: Module 're' en Python

Le module 're' (regular expression ou regex) permet d'utiliser des expressions régulières avec Python tout en offrant une panoplie de fonctions utiles, dont certaines sont brièvement expliquées ci-dessous :

- La fonction $search(regex, string)$ permet de rechercher un pattern (motif) sous forme d'une regex, au sein d'une chaîne de caractères. Si le motif est retrouvé dans la chaine, elle renvoie un objet du type SRE_Match.

```
1 cours = "theorie des langages"
```

```
if (re.search("langages", cours)):
    print("OK")
```

- La fonction $match(regex, string)$ fonctionne sur le modèle de $search$. La différence est qu'elle renvoie un objet du type SRE_Match seulement lorsque la regex correspond au début de la chaîne de caractères (à partir du premier caractère).

```
obj1 = re.match("a+","aabc")
obj1.group(0) # renvoie 'aa' la premiere occurrence
obj2 = re.match("ba+","aabc") # renvoie None car il n y a pas de correspondance
```

- La fonction $fullmatch(regex, string)$ renvoie un objet du type SRE_Match si et seulement si regex correspond exactement au string.

```
result = re.fullmatch("a+bc","aabc")# renvoie l'objet SRE_Match
print(re.fullmatch("a+","aabc")) # renvoie None
```

- La fonction $findall(regex, string)$ permet chercher et récupérer toutes chaînes correspondantes à regex sous forme d'une liste (au lieu de chercher la correspondance en commençant du début).

```
print(re.findall("a+","aabaaacbcaaba")) #--> ['aa', 'aaa', 'aa', 'a']
```

Exercice 3: Extraire des informations depuis un fichier

En chargeant le fichier 'datamail.txt', écrivez une fonction permettant d'extraire toutes les adresses e-mail citées dans ce fichier. Affichez ces adresses dans une liste associée à leur numéro de ligne.

Function

Header

```
def email_finder(file_name):
```

Test

```
def email_finder(datamail.txt) #Result -->
# Line nbr: [adress 1, adress 2, ...]
```

Seulement les caractères alphanumériques sont autorisés dans les adresses cherchées.

Exercice 4: Manipuler des fichiers

On considère un fichier dans lequel sont stockées les données d'électrocardiogramme (ECG) d'un ensemble de patients. Chaque ligne doit contenir un numéro de patient et une série d'entiers positifs représentant l'amplitude du signal électrique cardiaque (mesurées en millivolts 'mV') du patient à des intervalles de temps réguliers sous la forme suivante : $Num_{passion} : val_1, val_2, \ldots, val_n$ tel que : $Num_{passion}, val_i \in N$

1. Étant donné que certaines lignes sont mal formatées (contiennent des erreurs), écrivez une fonction qui nous permet de nettoyer ce fichier en gardant uniquement les lignes correctement écrites (Imprimez ces lignes).