IUT Sénart-Fontainebleau, UPEC
MATH 4.2
IUT Info 2A

Florent Madelaine

2022-23

Convertir un automate en une expression régulière

Nous avons vu en TD qu'on pouvait convertir une expression régulière en automate, puisque on peut facilement construire un automate qui reconnaît un seul mot et que l'on sait simuler les opérations de concaténation, somme et étoile.

Aujourd'hui nous allons voir **une méthode permettant de convertir un automate en expression régulière**. On a donc le théorème suivant.

Théorème. *Un langage est reconnaisable par un automate si, et seulement si, il peut être décrit par une expression régulière.*

Schématiquement, on a :

$$\begin{array}{ccc} \text{Automates} & = & \text{Expressions régulières} \\ \| & & \| \\ \{ \text{ langages reconnus par automate } \} & = & \{ \text{ langages décrits par expression régulière } \} \end{array}$$

L'idée

Soit A un automate qu'on veut convertir en expression régulière. On procède en 2 étapes.

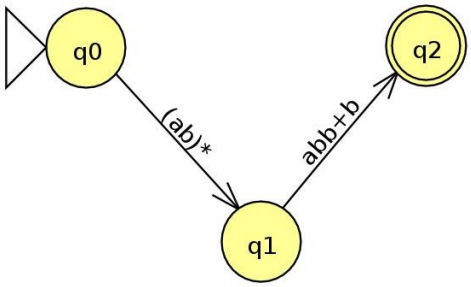
1. On voit A comme un « super automate », dont les transitions sont des langages décrits par des expressions régulières. On transforme progressivement des super-automates jusqu'à obtention d'un super-automate avec 2 états, l'un initial et pas acceptant, l'autre acceptant (les expressions régulières deviennent de plus en plus compliquées mais le nombre d'état diminue).
2. On sait comment déduire l'expression régulière correspondant à un tel super-automate à 2 états.

Super-automate dont les transitions sont des langages

La définition étend celle des automates. Au lieu que les transitions correspondent nécessairement à une lettre, elles correspondent maintenant à un langage qui est décrit par une expression régulière.

Lors d'une exécution dans ce super-automate, on passe d'un état à un autre en lisant non pas une seule lettre mais un mot appartenant au langage décrit par l'expression régulière.

Exemple. On considère le super-automate suivant.



Si le mot en entrée est **abababb**, une exécution possible est de passer de l'état q_0 à l'état q_1 en lisant **ababab** puis de passer de l'état q_1 à l'état q_2 en lisant le reste du mot, c'est-à-dire la lettre **b** et finalement d'accepter.

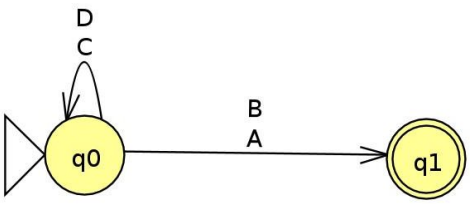
Notez que ce modèle plus général est naturellement **non-déterministe** puisqu'on pourrait aussi faire : q_0 à q_1 en lisant **abab** puis q_1 à q_2 en lisant **abb** et d'accepter. On rappelle qu'un automate non déterministe accepte un mot si il existe une exécution qui accepte. En particulier, si d'autres exécutions rejettent le mot **abababb**, il n'en est pas moins accepté.

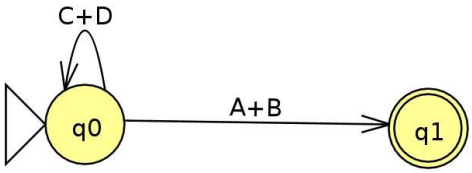
Notez aussi que l'on a maintenant le droit d'avoir des **transitions qui lisent le mot vide** qui est une expression régulière comme une autre. Le mot vide (qu'on note ϵ dans le cours et que JFLAP note λ) fait partie du langage $(ab)*$. Une exécution possible est donc de passer de q_0 à q_1 en lisant ϵ puis de rejeter puisque il n'y a pas de transition possible depuis q_1 (aucun préfixe de **abababb** appartient au langage $abb+b$).

Une première simplification : au plus une transition entre 2 états.

On peut toujours s'arranger pour qu'un super-automate ait au plus une transition entre deux états (qui peuvent être le même état dans le cas d'une boucle). Il suffit de faire la **somme des super-transitions**.

Exemple. Ci-dessous, A,B,C et D sont des expressions régulières décrivant des langages quelconques.





Langage vide et mot vide

On autorisera les 2 comme expression régulière. Ils sont très différents.

- ϵ (ou λ en JFLAP) est le mot vide (1 mot accepté).
- $\emptyset$ est le langage vide (aucun mot accepté).

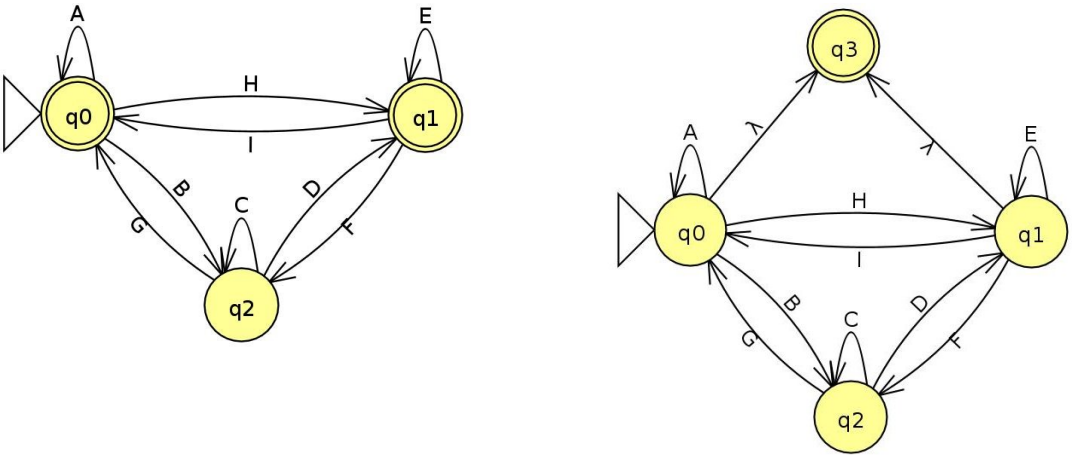
Les deux sont très pratiques pour simplifier le super-automate.

Calculez avec ces 2 langages.

- Notez tout d'abord que : $(\emptyset)^* = (\epsilon)^* = \epsilon$ (l'étoile veut dire répéter 0 ou plusieurs fois, si on répète 0 fois n'importe quoi ça donne ϵ).
- Notez ensuite que si dans $L_1.L_2.L_3.....L_n$ l'un des langage est $\emptyset$ alors le tout donne $\emptyset$.
- Notez finalement que si dans $L_1.L_2.L_3.....L_n$ l'un des langage est ϵ alors on peut simplifier en l'enlevant.

Une seconde simplification : 1 état initial non acceptant et 1 seul état final distinct.

Si on a plusieurs états acceptants, on les transforme en état non-acceptant, on ajoute un nouvel état acceptant et des transitions avec le **mot vide** depuis les anciens états acceptants vers ce nouvel état unique état acceptant.



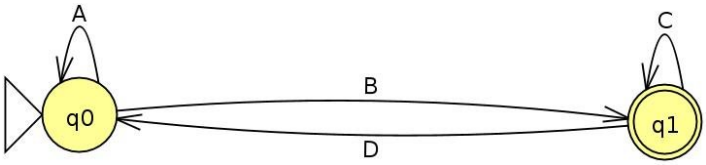
Une troisième simplification : exactement une transition entre 2 états.

On peut toujours s'arranger pour qu'un super-automate ait une et une seule transition entre deux états (qui peuvent être le même état dans le cas d'une boucle). On a vu précédemment comment utiliser la somme pour avoir au plus une transition. D'autre part, si il n'y a pas de transition, on ajoute une super-transition dont l'expression régulière est le **langage vide** $\emptyset$.

L'expression régulière d'un super-automate à 2 états

Quand on a un super-automate à 2 états seulement, dont l'un est initial mais pas acceptant et l'autre est acceptant, on peut facilement déduire l'expression régulière.

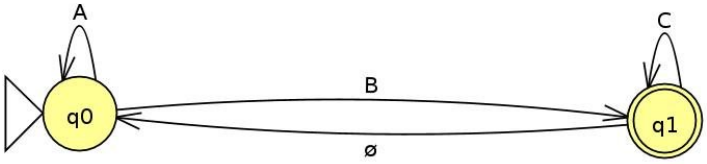
Le super-automate



a pour expression régulière

$$(A^*BC^*D)^*A^*BC^*$$

Lorsque certains langages sont $\emptyset$ l'expression régulière est plus simple. Le super-automate



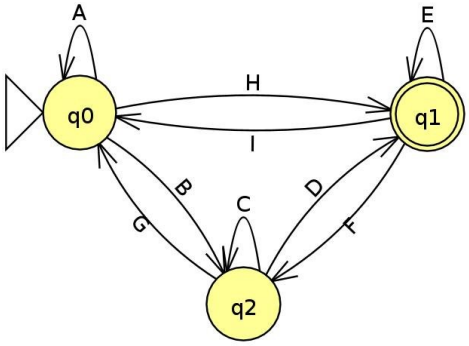
a pour expression régulière

$$A^*BC^*$$

Le cœur de la méthode : supprimer un état non initial non acceptant.

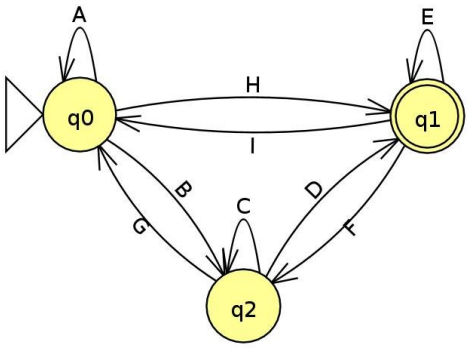
Avec les 3 premières méthodes, on a maintenant un automate qui a tous les arcs possibles, un état initial non acceptant et un unique état final. Si il n'y a pas d'autres états, on a terminé (voir ci-dessus). Sinon, on va supprimer un état en ajoutant dans le reste du super-automate des expressions régulières représentant les **détours** qu'on aurait pu faire en passant par cet état qu'on supprime.

Illustrons le principe sur un exemple : nous cherchons à éliminer l'état q_2.



Pour aller de q_0 à q_0 :

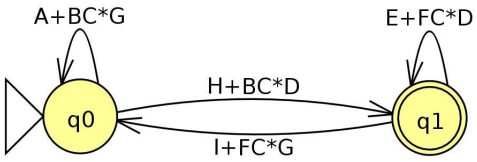
- (**sans détour**) on peut prendre la super-transition boucle dont l'expression régulière est A ; ou bien,
- (**avec détour par** q_2) on peut passer par l'état q_2 et revenir ce qui donne BC^*G.



Pour aller de q_1 à q_0 :

- (**sans détour**) on peut prendre la super-transition dont l'expression régulière est I ; ou bien,
- (**avec détour par** q_2) on peut passer par l'état q_2 avant d'aller à q_0 ce qui donne FC^*G.

On peut donc se passer de l'état q_2 si on ajoute pour chaque transition directe n'empruntant pas q_2, l'expression régulière correspondant au détour par q_2.



La méthode détaillée.

1. Première simplification : au plus une transition entre 2 états (avec $+$)
2. Seconde simplification : au plus un état acceptant distinct de l'état initial (avec ϵ)
3. Troisième simplification : exactement une transition entre 2 états (avec $\emptyset$)
4. Cœur de la méthode : on supprime l'un après l'autre tous les états sauf 2 en tenant compte de détours (ajout d'expressions régulières décrivant les détours).
5. Fin : on lit l'expression régulière sur le super-automate à 2 états.

À vous de jouer

Exercice 1. On considère l'expression régulière suivante, où les lettres majuscules représentent des langages :

$$(A^*BC^*D)^*A^*BC^*$$

Simplifiez cette expression pour chaque hypothèse ci-dessous.

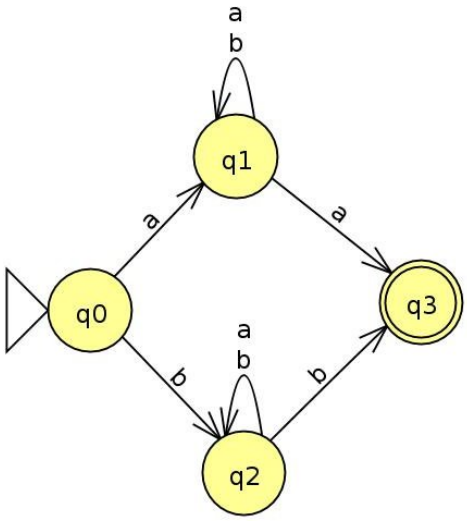
1. $A=\emptyset$
2. $D=\emptyset$
3. $B=\emptyset$
4. $A=\epsilon, B=\epsilon, D=\emptyset$.

1. Puisque $A=\emptyset$, on a $A^*=\epsilon$. Donc $(A^*BC^*D)^*A^*BC^*=(\epsilon BC^*D)^*\epsilon BC^*=(BC^*D)^*BC^*$
2. Puisque $D=\emptyset$, on a $(A^*BC^*D)=\emptyset$ et donc $(A^*BC^*D)^*=\epsilon$. Finalement on a $(A^*BC^*D)^*A^*BC^*=\epsilon A^*BC^*=A^*BC^*$.
3. $(A^*BC^*D)^*A^*BC^*=(A^*\emptyset C^*D)^*A^*\emptyset C^*=\emptyset$ (concaténation avec au moins un langage vide est vide).
4. $(A^*BC^*D)^*=\epsilon$ puisque D étant $\emptyset$, la partie entre parenthèses donne $\emptyset$. Donc $(A^*BC^*D)^*A^*BC^*=A^*BC^*=\epsilon^*\epsilon C^*=C^*$.

Réclame

JFLAP permet d'effectuer la conversion automate vers expression régulière par la méthode que nous venons de présenter. Pour cela il faut aller dans le menu `Convert` et choisir `Convert FA to RE` et de suivre les indications.

Exercice 2. Convertissez l'automate ci-dessous en expression régulière.

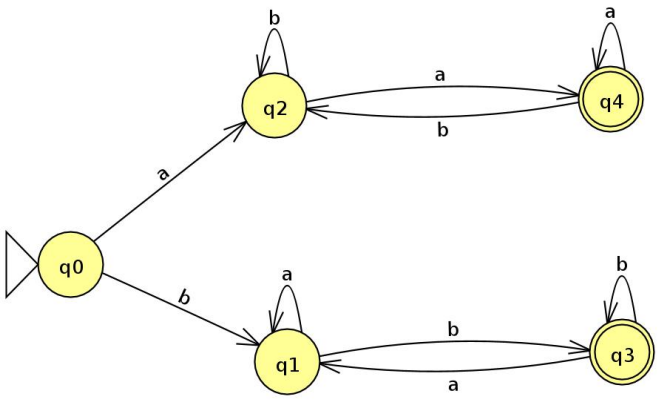


Avec notre méthode on trouve

$$a(b+a)^*a+b(b+a)^*b$$

(utilisez JFLAP pour bien comprendre les étapes).

Exercice 3. Convertissez l'automate ci-dessous en expression régulière.

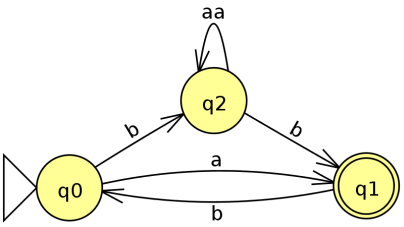


Avec notre méthode on trouve

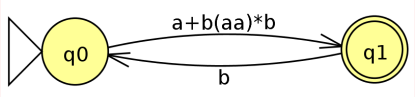
$$ba^*b(b+aa^*b)^*+ab^*a(a+bb^*a)^*$$

(utilisez JFLAP pour bien comprendre les étapes).

Exercice 4. On considère le « super-automate » ci-dessous. Supprimez l'état q_2 pour obtenir un super-automate équivalent à deux états, puis donnez l'expression régulière correspondant à cet automate.



Le seul « détour » permis par l'état q_2 est celui allant de q_0 à q_1. Lorsqu'on tient compte de son effet, on aboutit à l'automate ci-dessous, qui représente la terminaison de la méthode.



Il reste à en déduire l'expression régulière correspondante. L'expression générale permettant de passer de l'état q_0 à l'état q_1 est $a+b(aa)^*b$. En lisant un b à la suite de cette expression, on obtient une boucle sur l'état q_0. On peut répéter cette boucle un nombre indéfini de fois, avant de conclure par une transition vers q_1. Ainsi, l'expression raégulière finale est :

$$L=\big((a+b(aa)^*b)b\big)^*(a+b(aa)^*b)$$

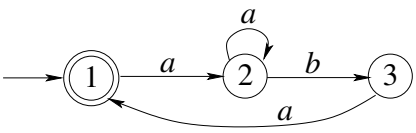
Exercice 5. Pour un langage L donné par plusieurs automates, obtient on une seule expression régulière ?

Non. les deux exercices précéedents définissent le même langage puisque le second est la version déterministe du premier. Or on trouve $a(b + a)^*a + b(b + a)^*b$ pour le premier et $ba^*b(b+aa^*b)^*+ab^*a(a+bb^*a)^*$ pour le second.

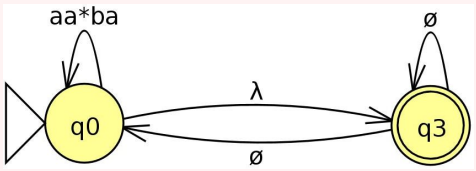
Exercice 6. Est-ce-que les super-automates dont les transitions sont des expressions régulières permettent de reconnaître des langages qu'on ne peut pas reconnaître avec les automates normaux ?

Non. D'une part, la méthode permettant de transformer un automate en expression régulière fonctionne pour les super-automates. D'autre part, on a vu en cours que pour toute expression régulière, on peut construire un automate (normal) qui reconnaît le langage décrit par l'expression régulière. On peut donc transformer un super-automate en automate normal.

Exercice 7. Déterminer le langage reconnu par l'automate suivant.

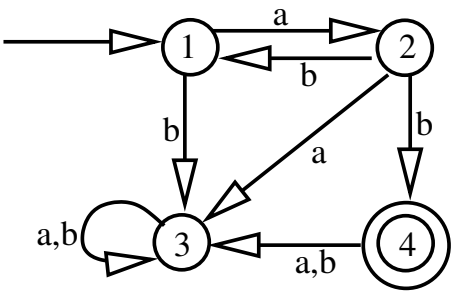


Avec notre méthode, on obtient finalement :

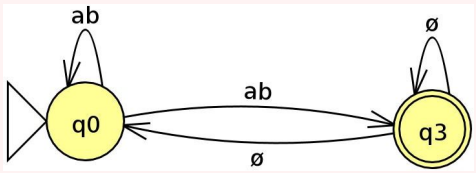


$(aa^*ba)^*$

Exercice 8. Déterminer le langage reconnu par l'automate suivant.



Avec notre méthode, on obtient finalement :

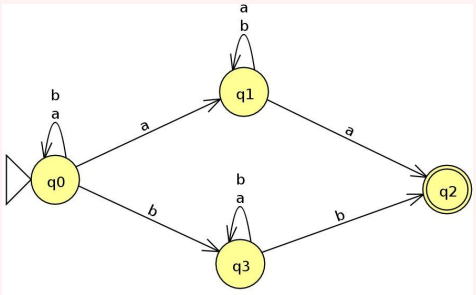


$$(ab)*ab$$

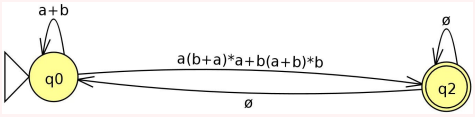
Exercice 9. Soit le langage sur l'alphabet $\Sigma = \{a,b\}$ constitué des mots dont la derniére lettre apparaît au moins une fois dans la partie initiale (*i.e.* dans le mot privé de sa dernière lettre).

1. Trouver un automate A non déterministe à 4 états reconnaissant L.
2. Convertissez l'automate A en expression régulière.
3. Déterminiser l'automate A en un automate A'.
4. Convertissez l'automate A' en expression régulière.

1. L'automate non-déterministe suivant reconnaît le langage demandé.

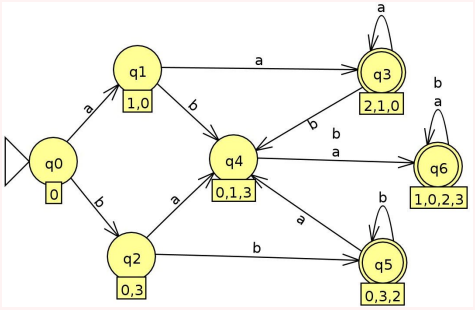


2. Avec notre méthode, on obtient finalement :

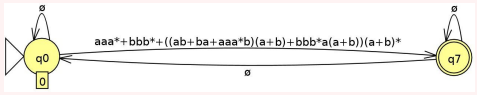


$$(a+b)^*(a(b+a)^*a+b(a+b)^*b)$$

3. On peut déterminiser l'automate A avec la méthode des super-états (ensemble d'états, voir cours et TD de la semaine 2). On peut aussi le faire avec JFLAP.



4. Avec notre méthode, on obtient finalement :



$$aaa^*+bbb^*+((ab+ba+aaa^*b)(a+b)+bbb^*a(a+b))(a+b)^*$$

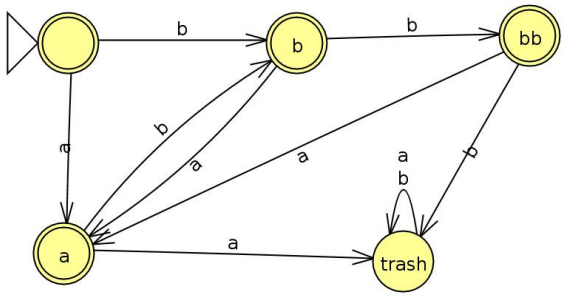
Comme dans un exercice précédent, on remarque qu'on obtient pas forcément la même expression régulière.

Exercice 10. Soit l'alphabet $\Sigma=\{a,b\}$.

1. Donner un automate déterministe qui reconnaît les mots ne contenant ni le facteur a^2 ni le facteur b^3.
2. Donner un automate déterministe qui reconnaît les mots de taille multiple de 3 ne contenant pas b^3.
3. Donner l'expression rationnelle pour chacun des deux points précédents.

1. indication. Il faut faire un automate pour les automates contenant aa et un automate pour bbb puis faire la somme en standardisant au préalable.

   Ensuite on déterminise l'automate. On obtient un automate qui est complet (sinon on doit ajouter un état poubelle). On peut donc prendre son complémentaire en intervertissant les états acceptants et non-acceptants (attention seulement si l'automate est déterministe complet). Pour plus de détails vous reportez au TD de cette semaine. Ce n'était pas demandé mais on peut même minimiser l'automate obtenu. En fait on pouvait presque deviner directement ce dernier automate qui est représenté ci-dessous (l'état a veut dire, je viens de lire un a, l'état bb je viens de lire 2 b etc).



2. Indication. Faire un automate qui accepte les multiples de trois et un automate n'acceptant pas de mots contenant bbb (pour le précédent, on transforme d'abord en automate déterministe complet et ensuite on échange acceptant et non-acceptant). Ensuite on fait le produit avec l'automate pour les mots multiple de 3 (sur le même principe que lorsqu'on veut savoir si 2 automates sont équivalents).
3. JFLAP trouve l'expression rationnelle suivante pour l'automate minimal de la question 1. $\lambda+b(\lambda+b)+(a+b(a+ba))(b(a+ba))^*(\lambda+b(\lambda+b))$