Résumé du cours de THLR

Amnézic

2023

Contents

1 Introduction

Un langage est un ensemble de séquences d'objets élémentaires auxquels on associe un sens. Chaque langage dépend d'un ensemble limité de symboles élémentaires.

1.1 Définitions

Alphabet:
Un alphabet est un ensemble Σ de symboles. On appelle les éléments de Σ des lettres. Un alphabet ne peut pas contenir plusieurs fois un même élément.

Mot :

- Un mot w dans un alphabet Σ est une séquence finie (possiblement vide) de lettres. Le mot vide est noté ε.
- L'ensemble Σ^* est l'ensemble des tous les mots sur Σ et l'ensemble $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$

Langage :

- Un langage L sur un alphabet Σ est un ensemble de mots de Σ^*
- L peut être fini ou infini
- $L \subseteq \Sigma^* = L$ est un sous-ensemble de l'ensemble de tous les mots

1.2 Opérations sur les mots :

Pour ces calculs, il faut obligatoirement donné un alphabet de référence. Les opérations «mathématiques »:

- longueur d'un mot : la longueur d'un mot w sur un alphabet Σ, notée $|w|$, est égale au nombre de lettres ($|\varepsilon| = 0$)
- occurrences d'un mot : l'occurrence d'un mot w sur un alphabet Σ et une lettre $a \in \Sigma$, $|w|_a$ correspond au nombre d'occurrences de a dans w
- concaténation : soient deux mots $w_1 = a_1...a_n$ et $w_2 = b_1...b_m$, on définit leur concaténation $w_1 \bullet w_2 = a_1...a_n b_1...b_m$
- propriétés de la concaténation :
  - $|w_1 + w_2| = |w_1| + |w_2|$
  - $\varepsilon \bullet w_1 = w_1 \bullet \varepsilon = w_1$
  - $w_1 \bullet (w_2 \bullet w_3) = (w_1 \bullet w_2) \bullet w_3 \Rightarrow$ associative

- pas commutative

- exponentiation d'un mot : $w^k = w...w$ k fois ($w^0 = \varepsilon$)

Les mots dérivés : (soient w, x, y, z d'un alphabet Σ)

- x est un préfixe de w si w = x • y
- z est un suffixe de w si w = y • z
- y est un facteur de w si w = x • y • z
- ε appartient à Pref(w), Suff(w) et Fact(w)
- inversion d'un mot : soit w = $w_1...w_n$ alors son miroir, noté w^R, est $w_n...w_1$ (si $w = w_R$, alors w est un palindrome)

1.3 Distance d'un mot

Définition :
La distance d'édition $d_e(w_1,w_2)$ entre deux mots w_1 et $w_2 \in \Sigma^*$ est égal au nombre minimal d'insertions et de suppressions lettre par lettre nécessaire pour passer de w_1 à w_2.

Soit E un ensemble, une fonction d : $E^2 \to R_+$ renvoie la distance si elle vérifie les propriétés suivantes ($\forall$x, y, z$\in E$):

- séparation : $d_e(x, y) = 0 \iff$ x = y
- symétrie : d_e(x,y) = d_e(y,x)
- inégalité triangulaire : d_e(x,y) + d_e(y,z) $\geq$ d_e(x,z)

2 Langages décidables

2.1 Définition :

Un langage L sur un alphabet Σ est dit décidable ou récursif s'il existe un algorithme A tel que, pour tout $w \in \Sigma^*$

- if $w \in L$, alors A(w) renvoie true
- sinon, A(w) renvoie false

Un langage L sur un alphabet Σ est dit semi-décidable ou récursivement énumérable s'il existe un algorithme A tel que, pour tout $w \in \Sigma^*$.

- si $w \in L$, alors A(w) renvoie true
- si $w \notin L$, alors A(w) revoie false **ou ne finit jamais**

Note : Si un langage L est récursif, alors il est récursivement énumérable (l'inverse est faux).

On peut voir les langages comme des ensembles en probabilités, il partage d'ailleurs certaines propriétés:

- $\overline{(\overline{A})} = \mathrm{A}$
- $\mathrm{A} \cup \overline{A} = \Sigma^*$
- $\overline{(A \cup B)} = \overline{A} \cap \overline{B}$
- $\overline{(A \cap B)} = \overline{A} \cup \overline{B}$
- $\mathrm{A} \cap (\mathrm{B} \cup \mathrm{C}) = (\mathrm{A} \cup \mathrm{B}) \cap (\mathrm{A} \cup \mathrm{C})$
- $\mathrm{A} \cup (\mathrm{B} \cap \mathrm{C}) = (\mathrm{A} \cup \mathrm{B}) \cap (\mathrm{A} \cup \mathrm{C})$

2.2 Préfixes, facteurs et suffixes

Soit L un langage sur Σ, le préfixe d'un langage est l'ensemble de tous les mots qui sont les préfixes d'au moins un mot de L. De manière plus formelle on le définit comme ceci :

$$\mathrm{Pref(L)} = \{\ \mathrm{w} \mid \mathrm{x} \in \mathrm{L},\ \mathrm{w} \in \mathrm{Pref(x)}\ \}$$

Note : cette définition s'applique aussi aux facteurs et aux suffixes

2.3 Étoile de Kleen :

Pour tout langage L, on a les propriétés suivantes :

- $\varepsilon \in L^*$, même si $\varepsilon \notin L$
- $\varepsilon \in L^+$ ssi $\varepsilon \in L$
- $\emptyset^* = \varepsilon$
- $\Sigma^* = \Sigma^*$

3 Expressions régulières

3.1 Les bases

Les symboles des motifs de langages :

- + : ou → a + b = a ou b
- "(" et ")" : but uniquement syntaxique
- (x)* : répétition finie (potentiellement 0 fois) de x
- pour écrire un ensemble assez instinctif, on peut remplacer les ... par - → {A,...,Z} = [A-Z]
- $x^? = x + \varepsilon$: la lettre x est optionnelle
- $x^+ = xx^*$: x est présent au moins une fois

3.2 Définition

Syntaxe d'expression régulière
L'ensemble Reg_Σ sur un alphabet finie Σ qui ne contient pas les symboles réservés (ceux utilisés plus haut) est défini inductively par :

- atomes : $\emptyset$, ε et a ($\forall a \in \Sigma$) sont des expressions régulières
- règles : si e_1 et e_2 sont des expressions régulières, alors $e_1 + e_2$, $e_1 e_2$ et e_1^* aussi
- profondeur : (pas compris)

Note: ces règles ne s'appliquent pas pour les intersections et les unions

- If $e = \emptyset$, $\mathcal{L}(e) = \emptyset$.
- If $e = \varepsilon$, $\mathcal{L}(e) = \{\varepsilon\}$.
- If $e = a$ for $a \in \Sigma$, $\mathcal{L}(e) = \{a\}$.
- If $e = e_1 e_2$, $\mathcal{L}(e) = \mathcal{L}(e_1) \cdot \mathcal{L}(e_2)$.
- If $e = e_1 + e_2$, $\mathcal{L}(e) = \mathcal{L}(e_1) \cup \mathcal{L}(e_2)$.
- If $e = e_1^*$, $\mathcal{L}(e) = \mathcal{L}(e_1)^*$.

Figure 1: Sémantique d'expressions régulières

Langages rationnels

L'ensemble Rat_Σ des langages définis sur un alphabet Σ est défini ainsi :

$$\text{Rat} = \{\text{L} \subseteq \Sigma^* | \exists\ \text{e} \in Reg_\Sigma,\ \text{L} = \mathfrak{L}(\text{e})\}$$

- Un langage rationnel est un langage décidable.
- Un même langage peut être décrit par une infinité d'expressions régulières.
  Si $\mathfrak{L}(e_1) = \mathfrak{L}(e_2) \iff e_1 \equiv e_2$

$$\begin{gathered}
e\emptyset \equiv \emptyset e \equiv \emptyset \\
e\varepsilon \equiv \varepsilon e \equiv e \\
e^{**} \equiv e^* \equiv e^* e^* \\
\emptyset^* \equiv \varepsilon \\
\varepsilon^* \equiv \varepsilon \\
e + f \equiv f + e \\
e + \emptyset \equiv e \\
e(f + g) \equiv ef + eg \\
(e + f)g \equiv eg + fg
\end{gathered}$$

Figure 2: Équivalences d'expressions régulières

4 Automates

4.1 Définition

Un automate fini $A(Q, \Sigma, \delta, I, F)$ est composé de 5 éléments :

- un ensemble fini Q d'états : un état est un noeud du graphe
- un alphabet fini Σ
- un ensemble de transitions $\delta \subseteq Q \times \Sigma \times Q$: une transition est un triplé, et δ est un ensemble de triplés
- un ensemble d'états initiaux I
- un ensemble d'états finaux/terminaux/acceptants F

Un automate A accepte un mot $w \in \Sigma$ s'il existe un chemin depuis un état initial jusqu'à un état final nommé par w.
Le langage $L(A) = \{\text{w} \in \Sigma^* | A \text{ accepte w}\}$ d'un automate A est l'ensemble des tous les mots dans Σ^* accepté par A. On dit que $\mathfrak{L}(A)$ est reconnu par A.

4.2 Propriétés d'un automate :

- Deux automates A_1 et A_2 sont dit équivalents si $\mathfrak{L}\ (A_1) = \mathfrak{L}(A_2)$.
- Un automate A est dit déterministe (et appelé DFA, NFA si non déterministe) si
  - il a un unique état initial
  - pour chaque lettre a dans Σ et chaque état dans Q de A, il y a **au plus** une transition sortante nommée a

5 Automates et expressions régulières

5.1 Complétude :

Définitions :

- complétude : un automate A est dit complet si $\forall\ a \in \Sigma$ et $\forall$ état q de A, il existe depuis q au moins une transition sortant nommé a
- lemme : si un automate A est complet, alors pour tous les mots $w \in \Sigma^*$ il y a **au moins** une transition nommé w

À tout automate incomplet fini, on peut lui associer un automate complet, notamment grâce aux états puits. Exemple :

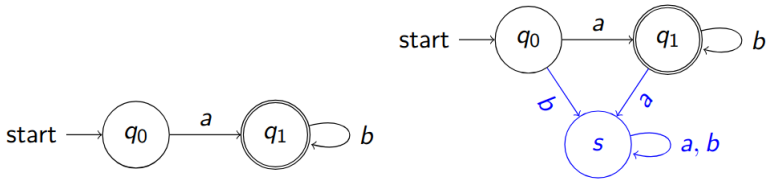


Figure 3: Automate incomplet (à gauche) et son équivalent complet (à droite)

5.2 Transitions spontanées :

Pour tout automate fini, on utilise une transition par lettre, mais on peut utiliser des ε (mot vide) transitions (aka transitions spontanées), ce qui permet de prendre des transitions sans lire de lettre.
Contrairement aux automates, les algorithmes qui utilisent des ε-transitions multiples à partir d'un même état, donc par définition non-déterministe. Ces automates sont appelés *automate fini non déterministe avec ε-transitions* ou en plus court ε-NFA.

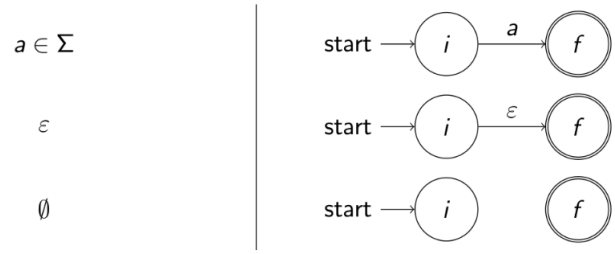
5.3 Algorithme de Thompson :

On peut généraliser ces automates aux Regex :
Théorème de Kleene :

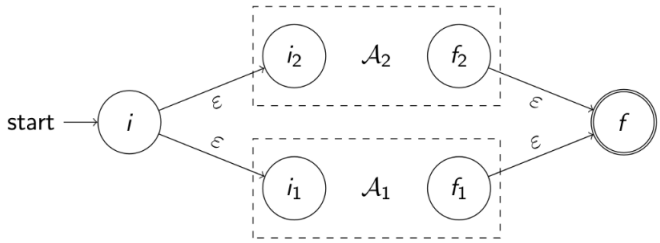
> Soit une expression régulière e $\in \mathrm{Reg}_\Sigma$, il existe un ε-NFA A sur l'alphabet Σ tel que $L(\mathrm{e}) = L(A)$.

Les Regex sont définies "inductively".
A la manière d'un automate, on commence par créer un état initial i distinct d'un état final f. Voici les différents motifs pour la création d'un ε-NFA : (avec (a,b)$\in \Sigma^2$ et (u_1,u_2)$\in (\Sigma^*)^2$)

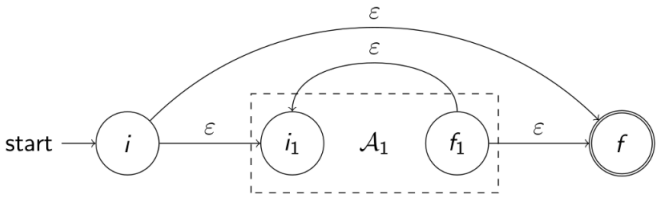
- début et fin de l'automate :



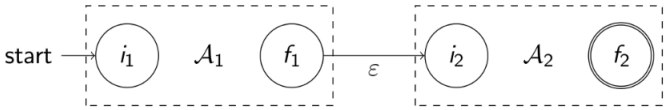
- $e_1 + e_2$:



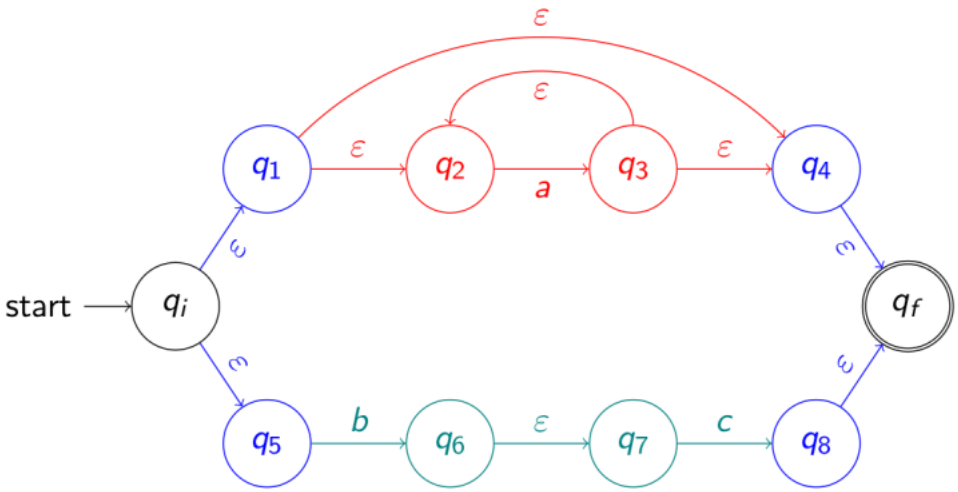
- e_1^* :



- $e_1 \bullet e_2$:



- par exemple, à partir de la Regex a* + bc sur l'alphabet Σ = {a,b,c} on obtient l'algorithme de Thompson suivant :



- Soit e ∈ Reg_Σ une Regex et n le nombre d'occurences de {ε,∅,+,*, le nombre de lettres dans Σ} (on exclue les parenthèses). Le nombre d'états est alors égale à 2n.

  À partir de l'automate obtenu par l'algorithme de Thompson, on réduit à 0 les ε-transitions, en reliant les différents états nécessaires à l'état initial. Puis on modifie les derniers états en les modifiant par des états finaux.

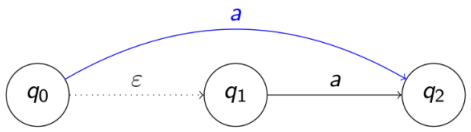
6 Simplification des ε-NFA

6.1 Élimination des transitions

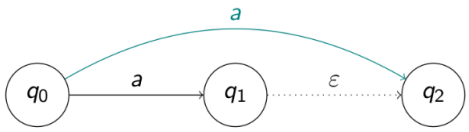
C'est la partie du cours la plus compliquée, il y aura un certain nombre de termes techniques. Une bonne connaissance du cours vu jusqu'ici est absolument nécessaire à la compréhension de cette partie.

En reprenant le dernier automate obtenu (le dernier automate de la page précédente), on remarque qu'il y a beaucoup d'états qu'un automate déterministe aurait pu éviter. On va alors procéder à des *éliminations arrière des ε-transitions* et des *éliminations avant des ε-transitions*

- élimination arrière des transitions spontanées :



- élimination avant des transitions spontanées :



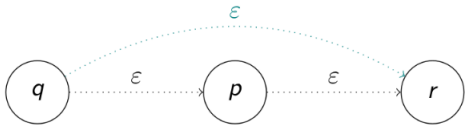
On définit les ε-fermeture de la manière suivante :

ε-fermeture d'un état :

La ε-fermeture $\varepsilon^A_{forward}$(q) d'un état q d'un automate A est l'ensemble $\{p \in Q | q \xrightarrow{\text{epsilon}}{}^*_A \}$
Note : q $\in \varepsilon^A_{forward}$ (q) car on peut atteindre q à partir de q en ne lisant rien du tout.

Théorème :

Si p $\in \varepsilon^A_{forward}$(q) and r $\in \varepsilon^A_{forward}$(p) alors r $\in \varepsilon^A_{forward}$(q).



6.2 Algorithme du point fixe itératif

1. Pour chaque état, on regarde où mènent les ε-transitions.
2. Pour tous les états, on rajoute où mènent les ε-transitions depuis les états trouvés à l'étape précédente.
3. On réitère l'étape précédente jusqu'à n'avoir plus rien à rajouter.

Dans l'algorithme du point fixe, on ne considère que les ε-transitions.
Exemple :

States	Step 0	Step 1	Step 2	Step 3
0	0 1 5	0 1 5 2 4	0 1 5 2 4 9	0 1 5 2 4 9
1	1 2 4	1 2 4 9	1 2 4 9	1 2 4 9
2	2	2	2	2
3	3 2 4	3 2 4 9	3 2 4 9	3 2 4 9
4	4 9	4 9	4 9	4 9
5	5	5	5	5
6	6 7	6 7	6 7	6 7
7	7	7	7	7
8	8 9	8 9	8 9	8 9
9	9	9	9	9

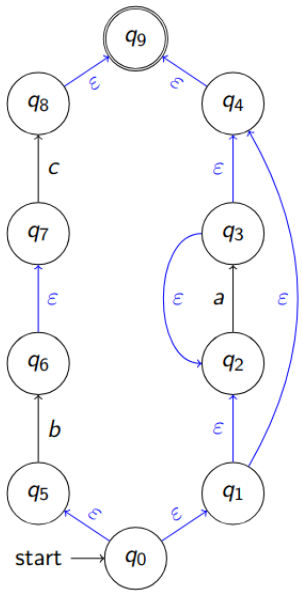


Figure 4: Exemple de l'algorithme du point fixe

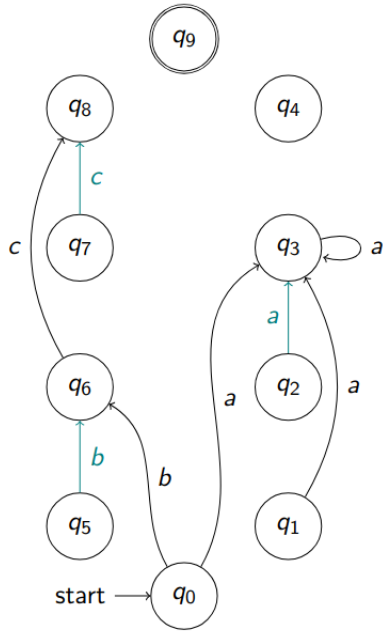


Figure 5: Simplification de l'automate après application de l'algorithme du point fixe

Explication de l'image ci-dessus :

1. on prend l'automate initial auquel on enlève toutes les ε-transitions.
2. à partir de l'état initial, on remonte les ε-transitions à l'aide du tableau.
3. on relie chaque état à l'état étiqueté par une lettre à partir de chaque état atteint à partir des ε-transitions depuis l'état initial.
4. on réitère les étapes précédentes en utilisant les derniers états utilisés comme des états initiaux.
5. quand on à finit l'algorithme, on met l'état initial de l'automate initial en état acceptant.

Note : Ceci est un automate $simplifie$ (ce n'est pas le nom officiel, juste pour spécifier qu'il ne contient pas les ε-transitions mais qu'il n'est pas encore émondé, ce qui sera vu dans la suite du cours).

7 Pruning et déterminisme

7.1 Pruning :

Le NFA généré par l'algorithme de Thompson est compliqué, même avec les les suppressions arrières des ε-transistions. On va donc supprimer les états et transitions qui ne servent à rien.

Utilité des états : (soit q un état de l'automate A)

- q est dit *accessible* s'il peut être atteint à partir d'un état initial
- q est dit *co-accessible* si à partir de q, on peut atteindre un état acceptant
- q est dit *utile* s'il est accessible **et** co-accessible.

Si un état n'est ni accessible, ni co-accessible (donc par définition, pas utile), on dit qu'il est *inutile*.

Ces états permettent d'introduire la notion de *pruning* (émondage en français) d'un automate.

Soient A un automate et A' l'automate obtenu à partir de la suppression des états inutiles de A. On dit que A est équivalent à A'.

Ce théorème permet de donner l'algorithme de pruning:

1. trouver les états accessibles (par un parcours profondeur ou largeur) depuis les états initiaux
2. trouver les états co-accessibles (on "remonte" les flèches à partir des états acceptants)
3. ne garder que les états accessibles et co-accessibles

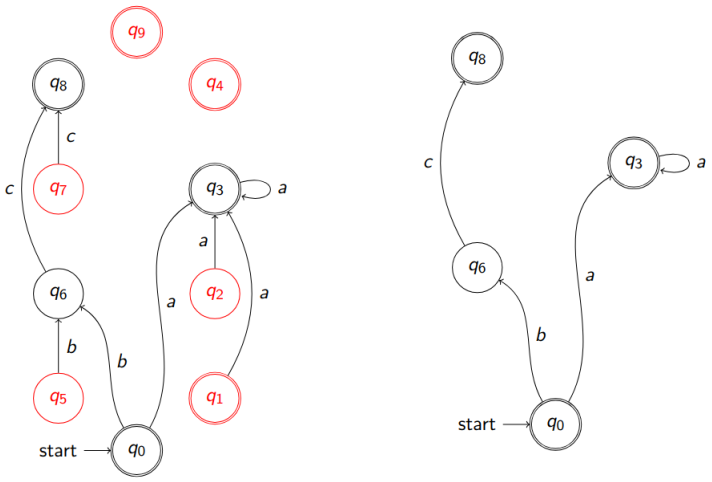


Figure 6: Automate simplifié (gauche) et son équivalent émondé (droite)

7.2 Déterminisme

Petit rappel :

> Un automate est dit déterministe si pour toute lettre a et état q, il existe au plus une transition sortant de q étiqueté par a.
> Un automate est dit complet si pour toute lettre a et état q, il existe au moins une transition sortant de q étiqueté par a.

Théorème

> $\forall$ A NFA sur un alphabet Σ, $\exists$ un équivalent DFA A' sur Σ, qui peut contenir au maximum 2^n états.

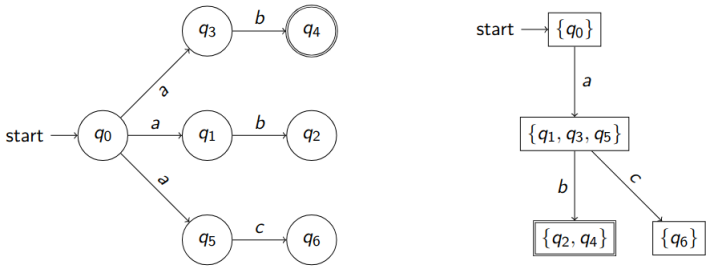


Figure 7: Un NFA et son équivalent DFA

7.3 Comment déterminiser un NFA ?

Un NFA est un automate non déterministe, c'est-à-dire qu'il y a au moins un état qui possède deux arêtes distinctes portées par la même lettre. Pour passer un NFA à un DFA, il faut suivre les étapes suivantes:

1. faire un tableau avec autant de colonnes que de lettres dans l'alphabet étudié
2. pour la première ligne, on commence par l'état initial (ou les états initiaux), et on note les états accessibles via la lettre de chaque colonne
3. on réitère l'opération avec les états trouvés à l'étape précédent
4. on crée tous les états qui sont dans la colonne
5. on les relie grâce au tableau

8 Lemme de pompage

Théorème :

> Tous les langages rationnels sont décidables. La contraposée n'est pas toujours vraie.

Lemme de pompage :

> Soit un langage rationnel L $\in \mathrm{Rat}_\Sigma$, il existe un niveau de pompage n_0 tel que pour tout mot w$\in$L de longueur n$\geq n_0$, il existe 3 mots x,y et z $\in \Sigma^*$ tel que w = xyz, y $\neq \varepsilon$, and $L(xy^*z) \subseteq$L

En français, si un automate à n états distincts accepte un mot de longueur n' tel que n<n', alors il y a forcément un cycle dans l'automate. Le seuil de pompage n est le nombre tel qu'il existe au moins un mot de longueur n tel qu'au moins un état soit visité 2 fois.
Théorème :

> Les langages du type L = $\{a^n b^m \mid n,m \in \mathbf{N}\}$, sur l'alphabet $\Sigma=\{a,b\}$, est décidable mais non rationnel.

Un langage rationnel est reconnaissable par un algorithme qui utilise une quantité constante de données qui ne dépend pas de la taille de l'input.

9 Minimisation des automates

Le déterminisme d'un automate d'une regex créé un très grand nombre d'états. Dans l'ordre on a :

1. expression régulère : n états
2. ε-NFA : 2n états
3. NFA : 2n états
4. NFA émondé : $\leqslant$ 2n états
5. DFA : $\leqslant 2^{2n}$ états

États indistinguables:

> Deux états q_1 et q_2 d'un DFA A sont dits **indistinguables** si pour tout mot $w \in \Sigma^*$, de q_1, A accepte w ssi, depuis q_2, A accepte w. On dit que $L_{q_1}(A) = L_{q_2}(A)$.

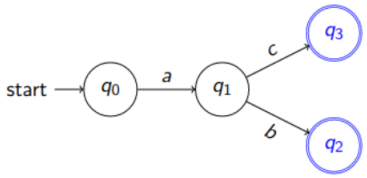


Figure 8: À partir de cet automate:

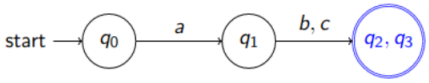


Figure 9: on peut donner l'automate équivalent ci-dessus:

Théorème de Myhil-Nerode :

> Soit un langage rationnel L, il existe un unique DFA A avec un nombre minimal d'états tel que $L(A)$ = L. On dit que cet automate est l'automate canonique de L.

Ainsi, si deux états p_1 et p_2 d'un DFA A sont indistinguables et $p_1 \xrightarrow{a} q_1$ et $p_2 \xrightarrow{a} q_2$, alors q_1 et q_2 sont eux aussi indistinguables.

Par exemple, dans cette automate :

On regarde pour chaque état quels sont les autres états de l'automate qui se comportent de la même manière. Puis on peut les regrouper pour minimiser l'automate. (section à compléter, pas compris la fin du ppt)

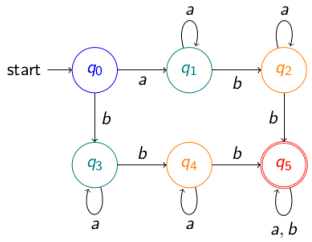


Figure 10: Chaque classe d'équivalence est de la même couleur

9.1 Algorithme de Moore

Algorithme :

- Par définition, les états initiaux et acceptant sont dans deux classes différentes
- On vérifie si deux états appartenant à la même classe ont les mêmes successeurs via une même lettre donnée
- Si ce n'est pas le cas, on divise la classe en deux
- On réitère l'algorithme jusqu'à ce que les classes soient stables

Pour appliquer l'algorithme

1. on fait un tableau avec autant de colonnes que de lettres dans l'alphabet étudié
2. on prend une première ligne avec tous les états sauf les acceptants
3. on écrit dans le tableau à quels états on peut accéder en fonction des lettres
4. s'il y a minimum un état auquel on peut accéder avec minimum deux lettres différentes à un autre état, on recrée le tableau en coupant les états indistaguables trouvés.
5. on réitère l'opération jusqu'à qu'il n'y ait plus aucun conflit

Pour plus de précisions, veuillez vous référez aix slides 11 à 17 du cours sur la minimisation des automates (cours 9).

Ainsi, l'automate présenté plus haut, après algorithme de Moore, ressemblera à ça:

Classes	a	b
$A = \{0\}$	1	3
$B = \{1, 3\}$	1 3	2
$C = \{2, 4\}$	2 4	5
$D = \{5\}$	5	5

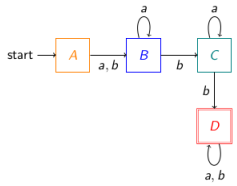


Figure 11: Automate après algorithme de Moore

10 Propriétés des langages rationnels

10.1 Algorithme de Brzozowski-McCluskey

L'algorithme de Brzozowski-McCluskey consiste à transformer un automate avec plusieurs transitions à un automate avec un seul état initial, un seul état final et une seule transition entre les deux états. Cette transition doit contenir l'expression régulière qui doit être égale au langage initial. L'état initial ne doit pas avoir de transitions entrantes et l'état final ne doit pas avoir de transitions sortantes. Au final, l'automate doit avoir exactement cette structure :

Figure 12: e doit être équivalent au langage initial

Cet algorithme a pour conséquence le théorème suivant :

> Soit un automate A sur un alphabet Σ, il existe une expression régulière e $\in \mathrm{Reg}_\Sigma$ tel que $L(A) = L(\mathrm{e})$, c'est-à-dire que e est **équivalent** à A.

Pour se faire, voici les différentes étapes :

1. tout d'abord, s'il y a plusieurs états initiaux, on met en place la structure suivante : (figure 13)
2. enlever chaque pont possible
3. enlever tous les états possibles
4. faire ça jusqu'à qu'on obtienne un état initial, un état final et une regex

Si un état initial possède une entrée ou un état final une sortie, on rajoute une ε-transitions et ils deviennent état initial/final.

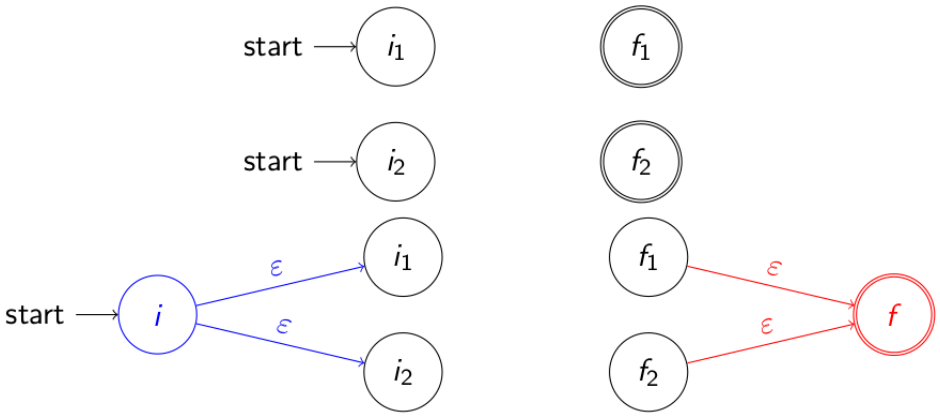


Figure 13: On passe de la configuration en haut à celle en bas

10.2 Stabilité des propriétés des langages rationnels

On considère un langage L rationnel (L$\in$Rat$_\Sigma$):

- $\overline{L} \in$ Rat$_\Sigma$.
- Pref(L), Fact(L) et Suff(L) $\in$Rat$_\Sigma$
- L^R (langage miroir) $\in$Rat$_\Sigma$
- Si L$_1$, L$_2$ $\in$Rat$_\Sigma$, alors L$_1\cap$L$_2\in$Rat$_\Sigma$
- on peut savoir si L $= \oslash$ ou non (si on peut accéder à un état acceptant depuis l'état initial)
- soit L'$\in$Rat$_\Sigma$, on peut savoir si L = L' ou non

11 Précisions à partir des questions de QCM

- L'étoile de Kleene d'un langage fini peut être infini.
- Les langages rationnels sont stables par étoile de Kleene, concaténation et union.
- Tout langage rationnel vérifie le lemme de pompage mais tout langage qui vérifie le lemme de pompage n'est pas rationnel.
- L'inclusion ne préserve pas la rationalité.
- Soit L un langage ne contenant pas ε. ε appartient à L^* mais pas à L^+.
- Tout langage rationnel est décidable mais tout langage décidable n'est pas rationnel.
- Un automate qui contient des ε-transitions n'est pas déterministe.