

INFO-F-302, Cours d'Informatique Fondamentale

Emmanuel Filion
Département d'Informatique
Faculté des Sciences
Université Libre de Bruxelles

Année académique 2023-2024

Automates Finis

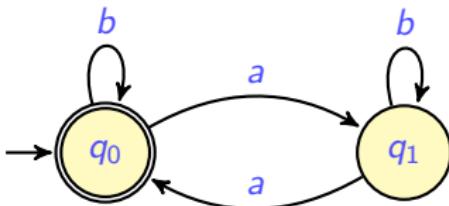
3 Les automates finis

- ▶ modèle de calcul abstrait
- ▶ accepte ou rejette des séquences de symboles (appelés mots)
- ▶ nombre fini d'états
- ▶ nombreuses applications
- ▶ nombreuses extensions

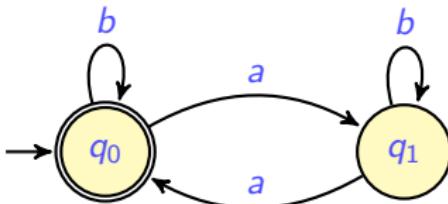
4 Un premier exemple

Un automate fini

- ▶ lit la séquence des lettres de gauche à droite,
- ▶ possède un nombre fini **d'états**,
- ▶ en fonction de l'état courant et de la lettre lue, se déplace vers un autre état,
- ▶ possède un état initial (représenté par une flèche sans source), et des états finaux (ou acceptants, représentés par des doubles cercles)
- ▶ il accepte le mot ssi, lorsqu'il arrive à la fin du mot, il se trouve dans un état final



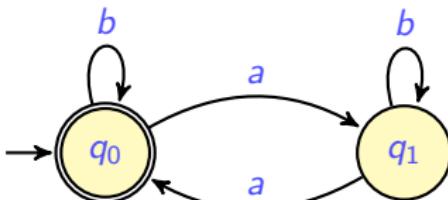
5 Un premier exemple



Exemple d'exécutions (appelées *run* en anglais)

entrée	exécution	accepté
aabaa		
abb		
bbaa		
ϵ		

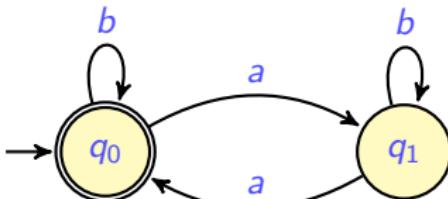
5 Un premier exemple



Exemple d'exécutions (appelées *run* en anglais)

entrée	exécution	accepté
aabaa		oui
abb		
bbaa		
ϵ		

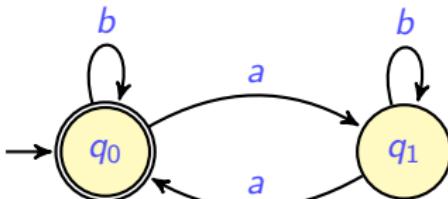
5 Un premier exemple



Exemple d'exécutions (appelées *run* en anglais)

entrée	exécution	accepté
aabaa		oui
abb		non
bbaa		
ϵ		

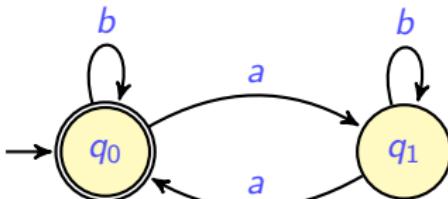
5 Un premier exemple



Exemple d'exécutions (appelées *run* en anglais)

entrée	exécution	accepté
aabaa		oui
abb		non
bbaa		oui
ϵ		

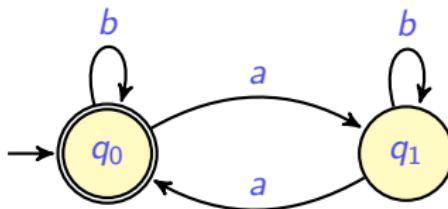
5 Un premier exemple



Exemple d'exécutions (appelées *run* en anglais)

entrée	exécution	accepté
aabaa		oui
abb		non
bbaa		oui
ϵ		oui

6 Un premier exemple



Langage accepté (ou reconnu)

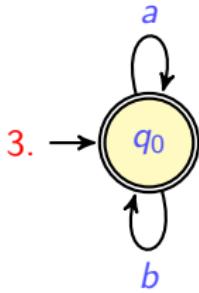
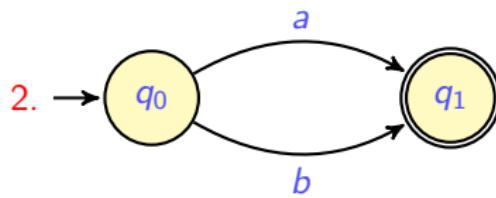
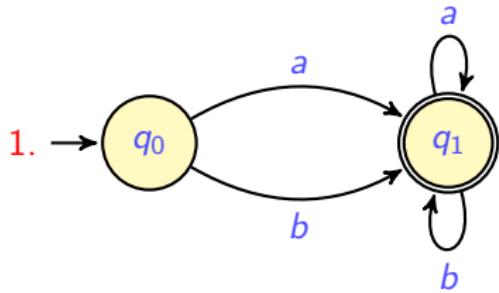
Le langage accepté (ou reconnu) par l'automate A ci-dessous, noté $L(A)$, est l'ensemble des mots qui contiennent un nombre pair de lettres a .

De manière générale, le langage accepté par un automate est l'ensemble des mots pour lesquels il existe une exécution qui les accepte (qui atteint un état acceptant).

Remarque importante On dit qu'un automate *accepte* un langage L s'il accepte tous les mots de L et **n'accepte aucun mot** qui n'est pas dans L .

7 Exercice 1

Quel est l'automate ci-dessous qui accepte l'ensemble des mots sur l'alphabet $\Sigma = \{a, b\}$ qui contiennent au moins une lettre¹.



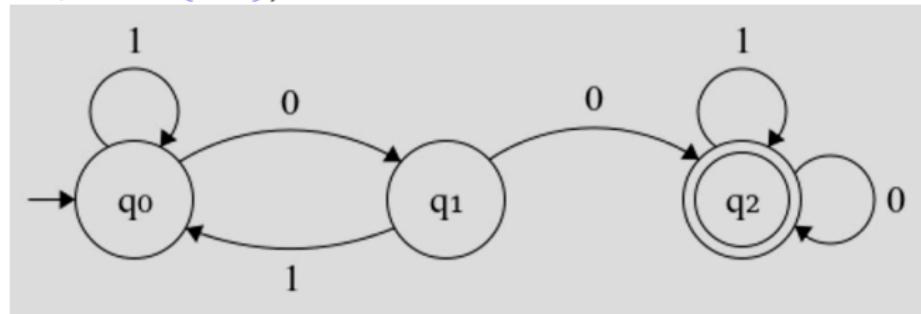
-
1. d'après la remarque précédente, tous les autres mots sont rejétés

8 Réponse

- ▶ La réponse est 1. L'automate rejette uniquement ϵ , le seul mot de longueur 0, et accepte tous les autres.
- ▶ L'automate 2 accepte uniquement les mots de longueur 1, et rejette tous les autres. En effet, si on prend le mot aa par exemple, il n'existe pas d'exécution acceptante sur *tout* le mot aa , car après avoir lu le premier a , l'automate est dans l'état q_1 , mais il n'y a pas de transition possible à partir de q_1 en lisant le deuxième a : le mot est par définition rejeté.
- ▶ L'automate 3 accepte tous les mots, y compris le mot vide ϵ de longueur 0, donc ce n'est pas la réponse.

9 Exercice 2

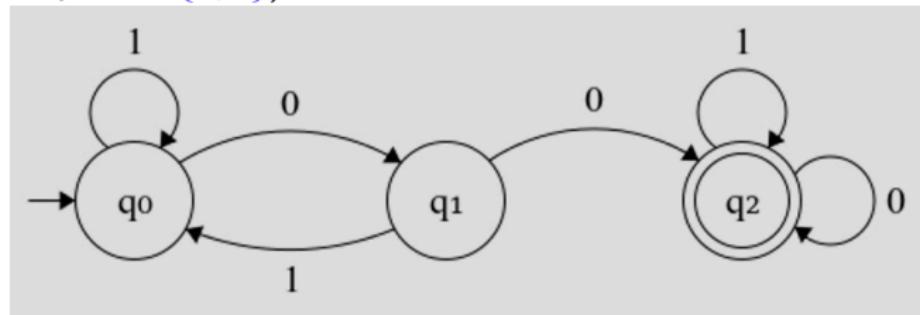
Quel est l'ensemble des mots reconnus par l'automate suivant (sur l'alphabet $\{0, 1\}$) :



-
2. Un mot u est un facteur d'un mot v s'il existe deux mots v_1, v_2 tels que $v = v_1uv_2$. Autrement dit, u est une sous-séquence de lettres consécutives de v .

9 Exercice 2

Quel est l'ensemble des mots reconnus par l'automate suivant (sur l'alphabet $\{0, 1\}$) :



C'est l'ensemble des mots sur l'alphabet $\{0, 1\}$ qui possède 00 pour facteur². En effet, la première fois que cet automate lit un 0 suivi d'un autre 0, il passe dans l'état q_2 , qui est acceptant, et boucle sur lui-même quelque soit la lettre lue, donc le reste du mot sera accepté.

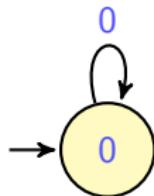
2. Un mot u est un facteur d'un mot v s'il existe deux mots v_1, v_2 tels que $v = v_1uv_2$. Autrement dit, u est une sous-séquence de lettres consécutives de v .

10 Autres exemple : Recherche Textuelle

- ▶ on veut trouver toutes les occurrences de **101** dans une séquence de bits
- ▶ on va construire un automate avec un état spécial tel qu'à chaque fois qu'on entrera dans cet état, c'est qu'on vient de lire **101**.

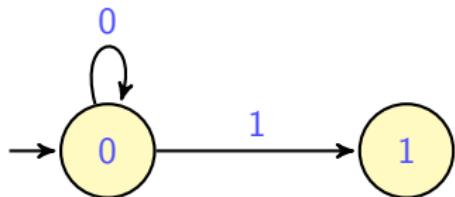
10 Autres exemple : Recherche Textuelle

- ▶ on veut trouver toutes les occurrences de **101** dans une séquence de bits
- ▶ on va construire un automate avec un état spécial tel qu'à chaque fois qu'on entrera dans cet état, c'est qu'on vient de lire **101**.



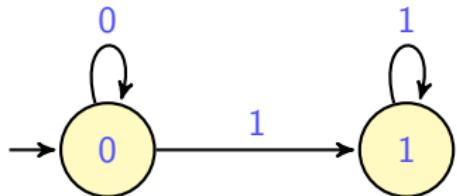
10 Autres exemple : Recherche Textuelle

- ▶ on veut trouver toutes les occurrences de **101** dans une séquence de bits
- ▶ on va construire un automate avec un état spécial tel qu'à chaque fois qu'on entrera dans cet état, c'est qu'on vient de lire **101**.



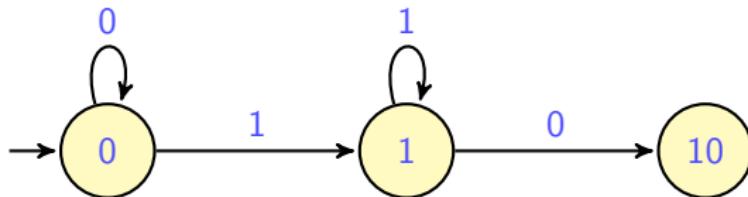
10 Autres exemple : Recherche Textuelle

- ▶ on veut trouver toutes les occurrences de **101** dans une séquence de bits
- ▶ on va construire un automate avec un état spécial tel qu'à chaque fois qu'on entrera dans cet état, c'est qu'on vient de lire **101**.



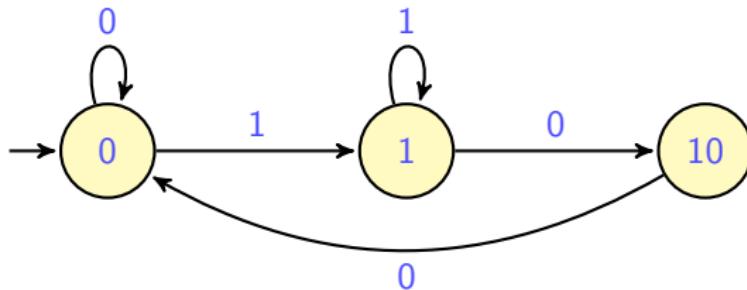
10 Autres exemple : Recherche Textuelle

- ▶ on veut trouver toutes les occurrences de **101** dans une séquence de bits
- ▶ on va construire un automate avec un état spécial tel qu'à chaque fois qu'on entrera dans cet état, c'est qu'on vient de lire **101**.



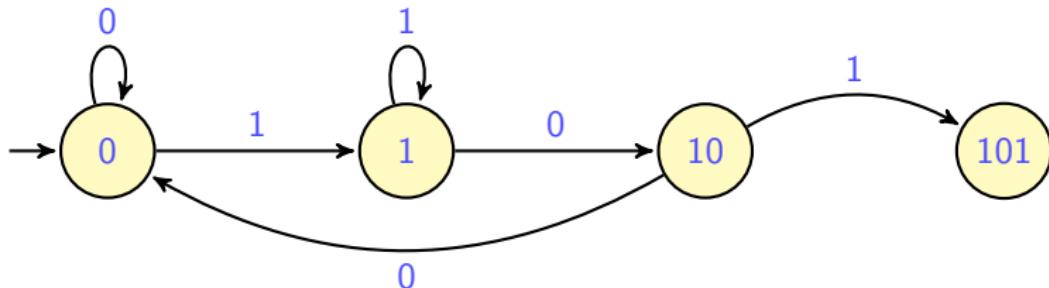
10 Autres exemple : Recherche Textuelle

- ▶ on veut trouver toutes les occurrences de **101** dans une séquence de bits
- ▶ on va construire un automate avec un état spécial tel qu'à chaque fois qu'on entrera dans cet état, c'est qu'on vient de lire **101**.



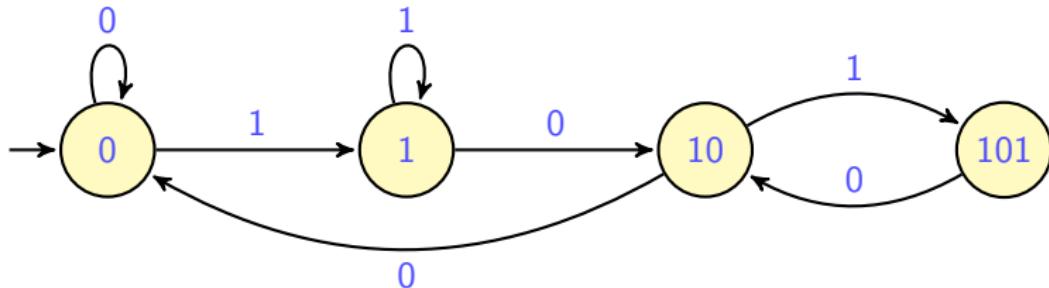
10 Autres exemple : Recherche Textuelle

- ▶ on veut trouver toutes les occurrences de **101** dans une séquence de bits
- ▶ on va construire un automate avec un état spécial tel qu'à chaque fois qu'on entrera dans cet état, c'est qu'on vient de lire **101**.



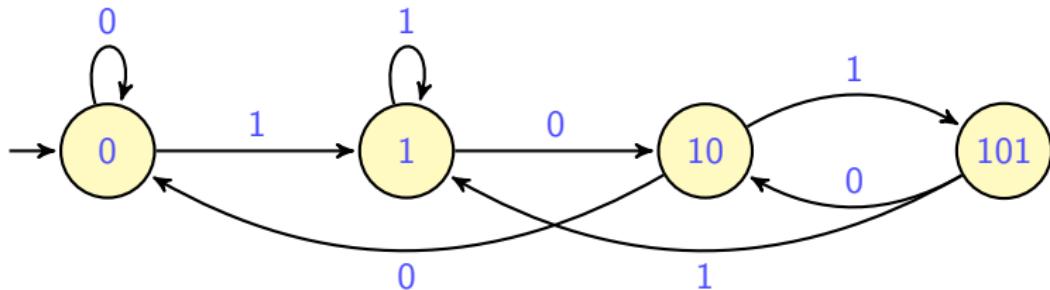
10 Autres exemple : Recherche Textuelle

- ▶ on veut trouver toutes les occurrences de **101** dans une séquence de bits
- ▶ on va construire un automate avec un état spécial tel qu'à chaque fois qu'on entrera dans cet état, c'est qu'on vient de lire **101**.



10 Autres exemple : Recherche Textuelle

- ▶ on veut trouver toutes les occurrences de **101** dans une séquence de bits
- ▶ on va construire un automate avec un état spécial tel qu'à chaque fois qu'on entrera dans cet état, c'est qu'on vient de lire **101**.



11 Quelques applications

- ▶ nombreuses applications en informatique théorique
- ▶ modélisation de protocoles réseau
- ▶ analyse de texte, recherche textuelle
- ▶ compilation (analyseur lexicaux)
- ▶ preuve automatique de programmes (théorie du model-checking)
- ▶ ...

12 Avantages/Inconvénients des automates

Avantages

- ▶ classe de programmes simples mais ayant de nombreuses applications
- ▶ bien compris, nombreux résultats, nombreux algorithmes pour les analyser
- ▶ robustes : beaucoup de caractérisations (expressions rationnelles, logique, régularité, algébrique ...), clôture par opérations Booléennes

13 Avantages/Inconvénients des automates

Inconvénients

- ▶ peu puissants
- ▶ peu compacts

14 Dans ce cours ...

- ▶ définition, exemples
- ▶ opérations Booléennes
- ▶ minimisation ?
- ▶ expressions rationnelles ?

15 Langages

- ▶ un alphabet est un ensemble fini, que l'on notera Σ
- ▶ ses éléments sont appelés lettres ou symboles
- ▶ un mot est une suite finie de symboles, ϵ dénotant le mot vide.
- ▶ l'ensemble des mots sur Σ est noté Σ^*
- ▶ un langage L est un sous-ensemble de Σ^* , i.e. $L \subseteq \Sigma^*$

15 Langages

- ▶ un alphabet est un ensemble fini, que l'on notera Σ
- ▶ ses éléments sont appelés lettres ou symboles
- ▶ un mot est une suite finie de symboles, ϵ dénotant le mot vide.
- ▶ l'ensemble des mots sur Σ est noté Σ^*
- ▶ un langage L est un sous-ensemble de Σ^* , i.e. $L \subseteq \Sigma^*$

Exemple : avec $\Sigma = \{a, b\}$, l'ensemble $\{a^n \mid n \text{ est un entier pair}\}$ est un langage.

16 Définition

Définition

Un automate fini³ A sur un alphabet Σ est un 4-uplet (Q, q_0, F, δ) où

- ▶ Q est un ensemble fini d'éléments appelé *états*,
- ▶ $q_0 \in Q$ est appelé état *initial*,
- ▶ $F \subseteq Q$ est un ensemble d'états dits *finaux* (ou acceptants),
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est une fonction (pas nécessairement totale) appelée *fonction de transition*.

16 Définition

Définition

Un automate fini³ A sur un alphabet Σ est un 4-uplet (Q, q_0, F, δ) où

- ▶ Q est un ensemble fini d'éléments appelé *états*,
- ▶ $q_0 \in Q$ est appelé état *initial*,
- ▶ $F \subseteq Q$ est un ensemble d'états dits *finaux* (ou acceptants),
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est une fonction (pas nécessairement totale) appelée *fonction de transition*.

Lorsqu'il lit une entrée $w \in \Sigma^*$, l'automate A commence son exécution dans l'état q_0 , et suit la fonction de transition δ , qui lui indique, en fonction de l'état où il se trouve et de la lettre qu'il lit, l'état suivant où il doit aller (si la fonction n'est pas définie, alors l'automate s'arrête et le mot n'est pas accepté). Si, à la fin de son exécution, il atteint un état final, alors le mot est accepté, sinon il est rejeté.

3. Par la suite, on dira simplement automate

17 Exécution

Définition

Une exécution de A est une suite finie $e = p_0\sigma_1p_1\sigma_2\dots p_{n-1}\sigma_np_n$ ($n \geq 0$) telle que :

- ▶ $p_0 = q_0$
- ▶ pour tout $i \in \{0, \dots, n\}$, $p_i \in Q$
- ▶ pour tout $i \in \{1, \dots, n\}$, $\sigma_i \in \Sigma$
- ▶ pour tout $i \in \{0, \dots, n-1\}$, $\delta(p_i, \sigma_{i+1})$ est définie et vaut p_{i+1} .

On dit l'exécution e est une exécution sur le mot $\sigma_1\dots\sigma_n$, et que e est acceptante si l'état atteint est final, i.e. $p_n \in F$.

Remarque : la suite q_0 est une exécution sur le mot vide ϵ . Elle est acceptantessi q_0 est final.

18 Langage accepté par un automate fini

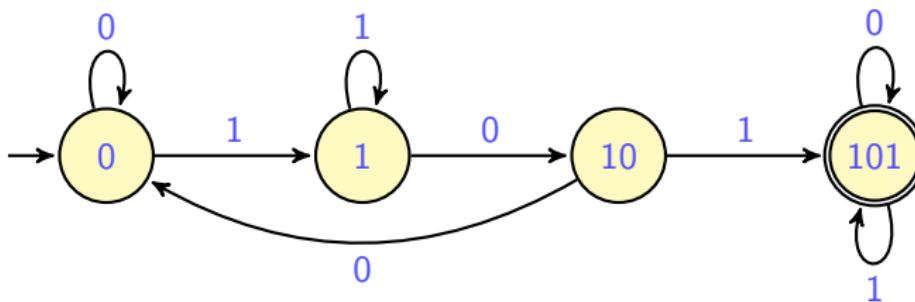
Définition

Etant donné un automate A , le langage accepté (ou reconnu) par A , noté $L(A)$, est l'ensemble des mots pour lesquels il existe une exécution acceptante de A .

$$L(A) = \{w \in \Sigma^* \mid \text{il existe une exécution acceptante de } A \text{ sur } w\}$$

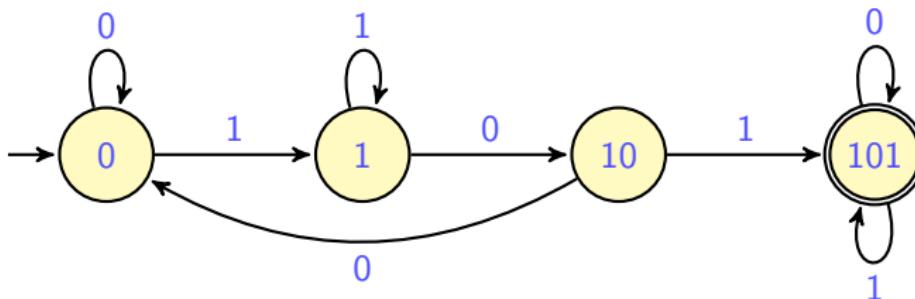
19 Exemples

$$\Sigma = \{0, 1\}.$$



19 Exemples

$$\Sigma = \{0, 1\}.$$

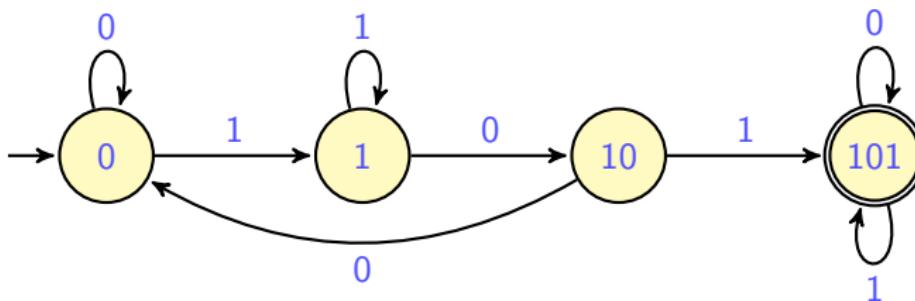


Question

Quel est le langage accepté par cet automate ?

19 Exemples

$$\Sigma = \{0, 1\}.$$



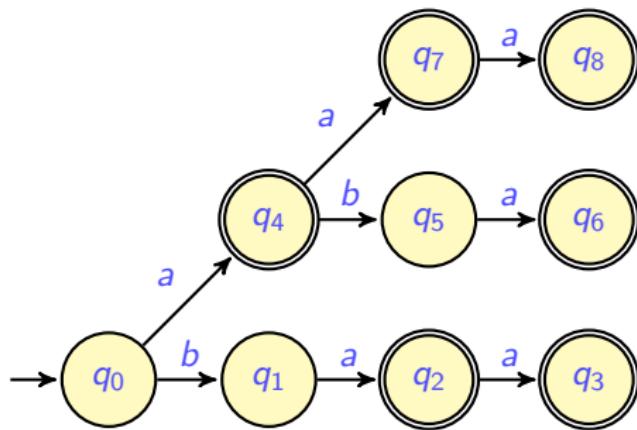
Question

Quel est le langage accepté par cet automate ?

$$L(A) = \{w \in \{0, 1\}^* \mid w \text{ contient le facteur } 101\}$$

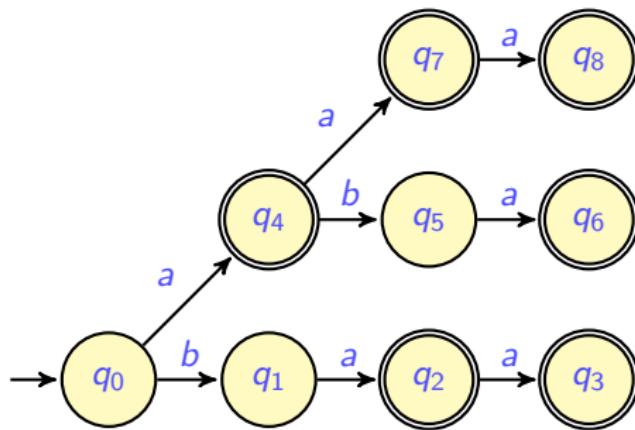
20 Exemples

$$\Sigma = \{a, b\}.$$



20 Exemples

$$\Sigma = \{a, b\}.$$

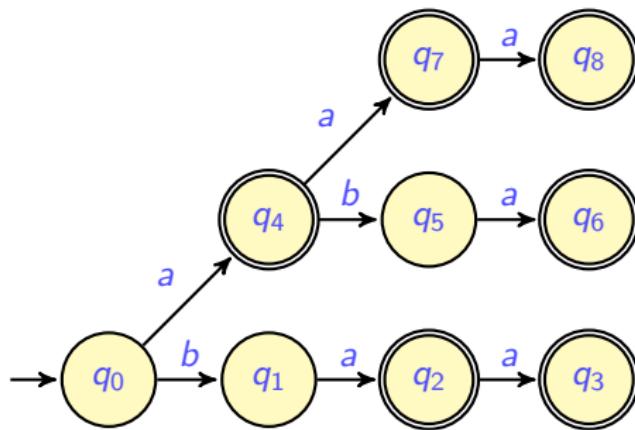


Question

Quel est le langage accepté par cet automate ?

20 Exemples

$$\Sigma = \{a, b\}.$$



Question

Quel est le langage accepté par cet automate ?

$$L(A) = \{a, ba, aa, aba, baa, aaa\}$$

21 Complétion d'un automate

Définition

Un automate A est dit *complet* si sa fonction de transition est totale.

Exemple : l'automate précédent n'est pas complet, alors que l'automate qui accepte les mots contenant 101 est complet.

21 Complétion d'un automate

Définition

Un automate A est dit *complet* si sa fonction de transition est totale.

Exemple : l'automate précédent n'est pas complet, alors que l'automate qui accepte les mots contenant 101 est complet.

Lemme

On peut toujours transformer un automate A en un automate B complet qui accepte le même langage, i.e. tel que $L(A) = L(B)$.

21 Complémentation d'un automate

Définition

Un automate A est dit *complet* si sa fonction de transition est totale.

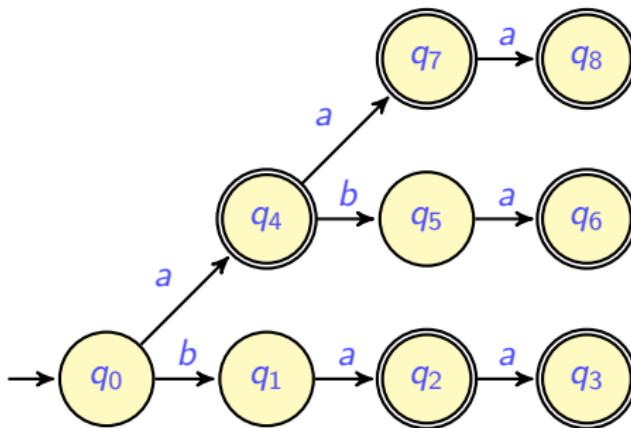
Exemple : l'automate précédent n'est pas complet, alors que l'automate qui accepte les mots contenant 101 est complet.

Lemme

On peut toujours transformer un automate A en un automate B complet qui accepte le même langage, i.e. tel que $L(A) = L(B)$.

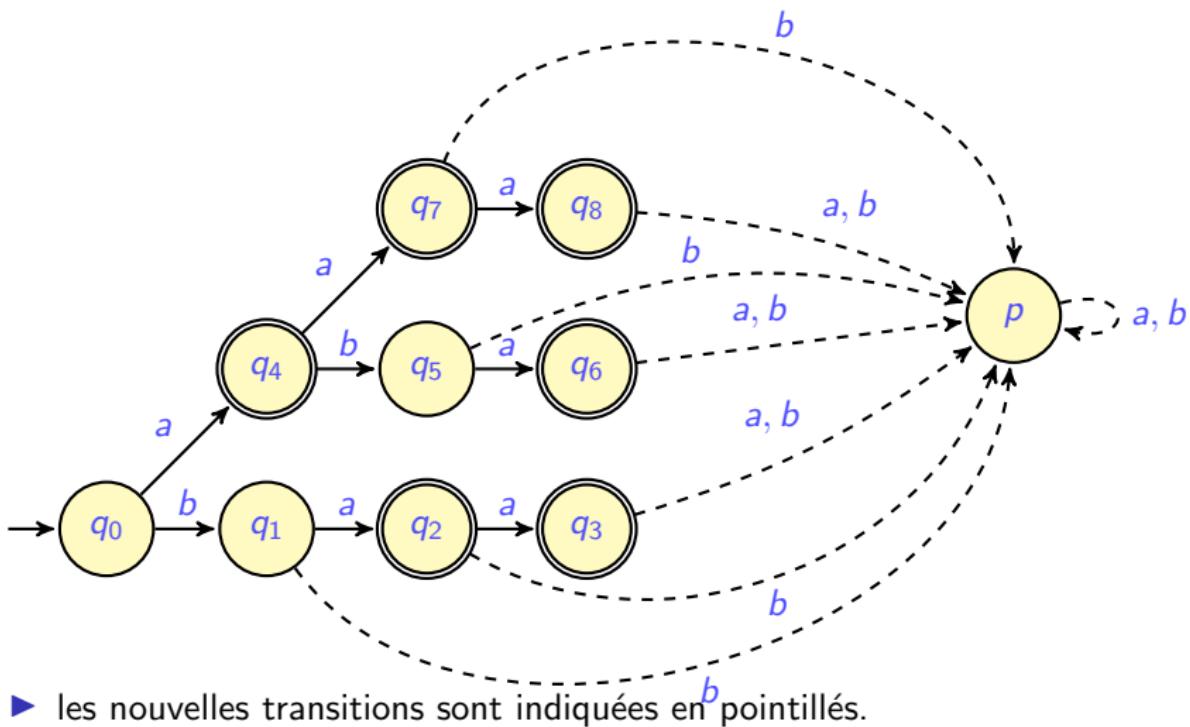
Idée de la preuve : ajouter un état supplémentaire (appelé état puits) non final et ajouter les transitions manquantes vers cet état.

22 Complémentation d'un automate : Exemple



- les nouvelles transitions sont indiquées en pointillés.

22 Complémentation d'un automate : Exemple



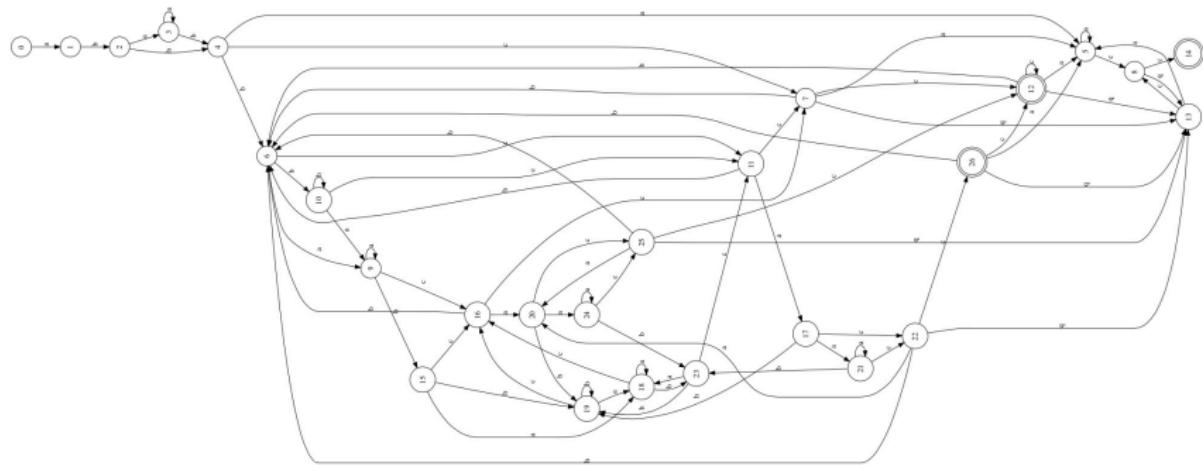
23 Test du vide

Problème

Le problème VIDE est le suivant.

- ▶ **ENTREE** : un automate A sur un alphabet Σ ,
- ▶ **SORTIE** : est-ce que $L(A) = \emptyset$?

24 Test du vide : comment feriez-vous avec l'automate suivant ?



25 Test du vide

Définition (Etats atteignables)

Soit $A = (Q, q_0, F, \delta)$ un automate sur un alphabet Σ . Un état $q \in Q$ est dit *atteignable* s'il existe un mot $w \in \Sigma^*$ et une exécution de A sur w qui se termine en q .

Remarque : l'ensemble des états atteignables d'un graphe peut être calculer en temps $O(n + m)$ par des algorithmes de graphe classiques (parcours en largeur par exemple), où n est le nombre d'états de l'automate, et m le nombre de transitions.

25 Test du vide

Définition (Etats atteignables)

Soit $A = (Q, q_0, F, \delta)$ un automate sur un alphabet Σ . Un état $q \in Q$ est dit *atteignable* s'il existe un mot $w \in \Sigma^*$ et une exécution de A sur w qui se termine en q .

Remarque : l'ensemble des états atteignables d'un graphe peut être calculer en temps $O(n + m)$ par des algorithmes de graphe classiques (parcours en largeur par exemple), où n est le nombre d'états de l'automate, et m le nombre de transitions.

Théorème

Etant donné un automate A avec n états m transitions, on peut tester en $O(n + m)$ si $L(A) = \emptyset$.

25 Test du vide

Définition (Etats atteignables)

Soit $A = (Q, q_0, F, \delta)$ un automate sur un alphabet Σ . Un état $q \in Q$ est dit *atteignable* s'il existe un mot $w \in \Sigma^*$ et une exécution de A sur w qui se termine en q .

Remarque : l'ensemble des états atteignables d'un graphe peut être calculer en temps $O(n + m)$ par des algorithmes de graphe classiques (parcours en largeur par exemple), où n est le nombre d'états de l'automate, et m le nombre de transitions.

Théorème

Etant donné un automate A avec n états m transitions, on peut tester en $O(n + m)$ si $L(A) = \emptyset$.

Algorithme

1. calculer l'ensemble des états atteignables R de A
2. tester si $R \cap F = \emptyset$.

26 Opérations Booléennes sur les langages

Définition (Complément)

Le complément d'un langage $L \subseteq \Sigma^*$ est le langage, noté \bar{L} , défini par
 $\bar{L} = \{w \in \Sigma^* \mid w \notin L\} = \Sigma^* \setminus L$.

Exemple : si L est l'ensemble des mots sur $\{a, b\}$ qui contiennent au moins un a , alors \bar{L} est l'ensemble des mots qui contiennent au moins deux a , ou pas de a .

Définition (Union, Intersection)

Soient $L_1, L_2 \subseteq \Sigma^*$ deux langages. L'union et l'intersection de L_1 et L_2 sont définies respectivement par :

$$\begin{aligned} L_1 \cup L_2 &= \{w \in \Sigma^* \mid w \in L_1 \text{ ou } w \in L_2\} \\ L_1 \cap L_2 &= \{w \in \Sigma^* \mid w \in L_1 \text{ et } w \in L_2\} \end{aligned}$$

27 Clôture des automates par opérations Booléennes

Théorème

Soient A, A_1, A_2 des automates finis sur un alphabet Σ . Il existe des automates A_c, U et I tels que :

$$\begin{aligned} L(A_c) &= \overline{L(A)} \\ L(U) &= L(A_1) \cup L(A_2) \\ L(I) &= L(A_1) \cap L(A_2) \end{aligned}$$

Dans la suite, nous allons montrer comment construire A_c, U et I à partir de A, A_1 et A_2 .

28 Clôture par complément

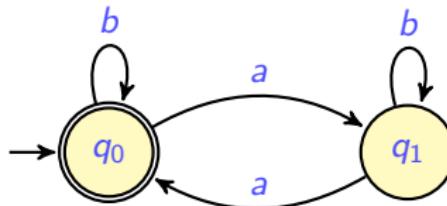
- ▶ Si $A = (Q, q_0, F, \delta)$ est complet, alors il suffit de prendre
 $A_c =$

28 Clôture par complément

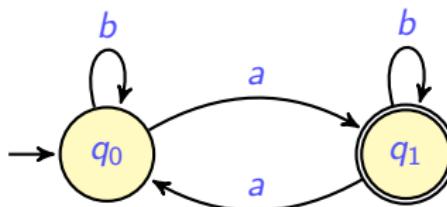
- Si $A = (Q, q_0, F, \delta)$ est complet, alors il suffit de prendre $A_c = (Q, q_0, Q \setminus F, \delta)$. Si A n'est pas complet, alors il faut le compléter avant.

28 Clôture par complément

- ▶ Si $A = (Q, q_0, F, \delta)$ est complet, alors il suffit de prendre $A_c = (Q, q_0, Q \setminus F, \delta)$. Si A n'est pas complet, alors il faut le compléter avant.
- ▶ Par exemple, si $A =$

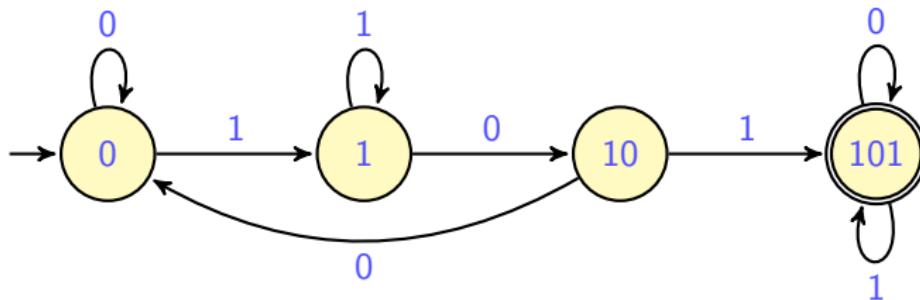


- ▶ alors $A_c =$

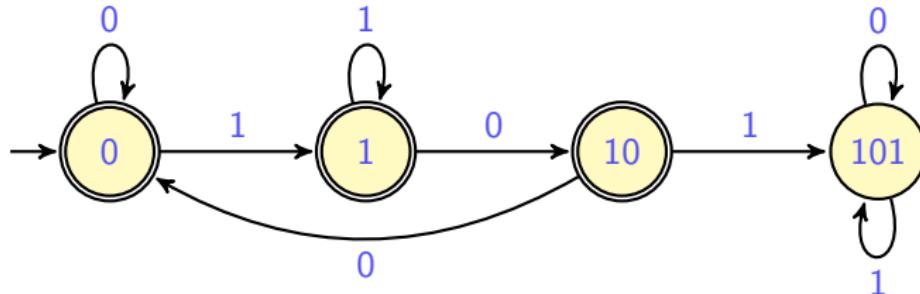


29 Clôture par complément

► Par exemple, si $A =$



► alors $A_c =$



30 Clôture par complément

Exercice

Donner un exemple d'automate $A = (Q, q_0, F, \delta)$ non complet sur un alphabet Σ , tel que l'automate $A_c = (Q, q_0, Q \setminus F, \delta)$ ne satisfait pas $L(A_c) = \overline{L(A)}$.

30 Clôture par complément

Exercice

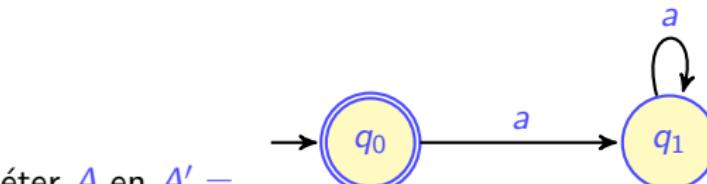
Donner un exemple d'automate $A = (Q, q_0, F, \delta)$ non complet sur un alphabet Σ , tel que l'automate $A_c = (Q, q_0, Q \setminus F, \delta)$ ne satisfait pas $L(A_c) = \overline{L(A)}$.

On prend $\Sigma = \{a\}$ et $A =$

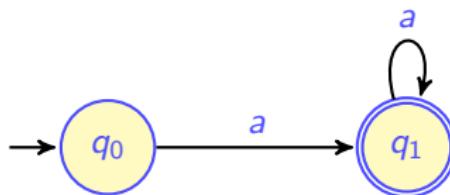
► $L(A) = \{\epsilon\}$, $L(B) = \emptyset \neq \overline{\{\epsilon\}}$



Il faut d'abord compléter A en $A' =$



Et on obtient $A_c =$



31 Produit d'Automates

On appellera *pré-automate* sur Σ un triplet (Q, q_0, δ) , où Q est un ensemble fini, $q_0 \in Q$, et $\delta : Q \times \Sigma \rightarrow Q$ est une fonction.

Définition

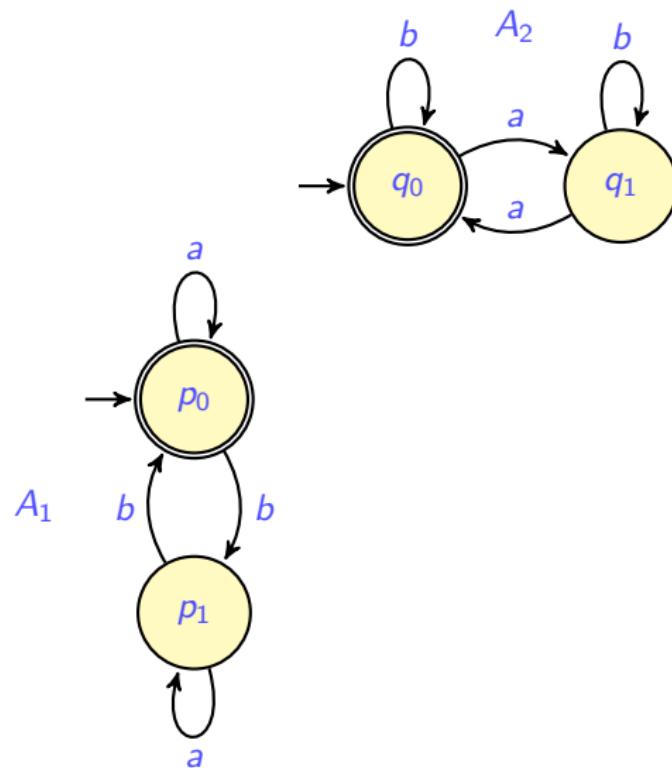
Soient $A_1 = (Q_1, q_0^1, F_1, \delta_1)$ et $A_2 = (Q_2, q_0^2, F_2, \delta_2)$ deux automates sur Σ . Le produit de A_1 et A_2 , noté $A_1 \otimes A_2$, est le pré-automate défini par :

$$A_1 \otimes A_2 = (Q_1 \times Q_2, (q_0^1, q_0^2), \delta_{12})$$

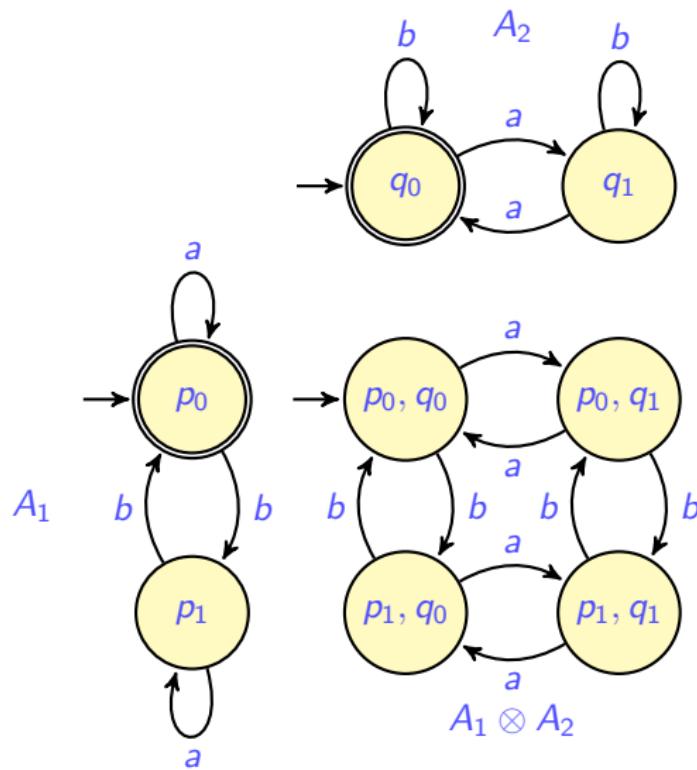
où, pour tout $(q_1, q_2) \in Q_1 \times Q_2$, pour tout $\sigma \in \Sigma$,

$$\delta_{12}((q_1, q_2), \sigma) = \begin{cases} \text{indéfini} & \text{si } \delta_1(q_1, \sigma) \text{ est indéfinie} \\ \text{indéfini} & \text{si } \delta_2(q_2, \sigma) \text{ est indéfinie} \\ (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) & \text{sinon.} \end{cases}$$

32 Exemple de produit



32 Exemple de produit



33 Exemple de produit

- ▶ $L(A_1)$ est l'ensemble des mots qui contiennent un nombre pair de b
- ▶ $L(A_2)$ est l'ensemble des mots qui contiennent un nombre pair de a

33 Exemple de produit

- ▶ $L(A_1)$ est l'ensemble des mots qui contiennent un nombre pair de b
- ▶ $L(A_2)$ est l'ensemble des mots qui contiennent un nombre pair de a
- ▶ toute exécution de $A_1 \otimes A_2$ sur un mot w simule en parallèle l'exécution de A_1 sur w , ainsi que celle de A_2 .
 - ▶ exemple avec $w = abba$
 - ▶ sur A_1 , on a $e_1 = p_0 a p_0 b p_1 b p_0 a p_0$
 - ▶ sur A_2 , on a $e_2 = q_0 a q_1 b q_1 b q_1 a q_0$
 - ▶ sur $A_1 \otimes A_2$, on a $(p_0, q_0)a(p_0, q_1)b(p_1, q_1)b(p_0, q_1)a(p_0, q_0)$.

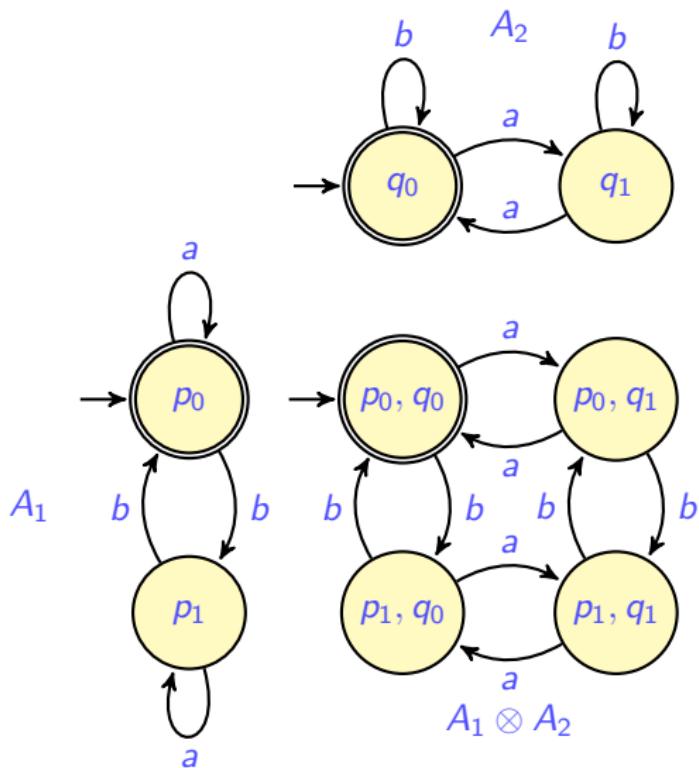
33 Exemple de produit

- ▶ $L(A_1)$ est l'ensemble des mots qui contiennent un nombre pair de b
- ▶ $L(A_2)$ est l'ensemble des mots qui contiennent un nombre pair de a
- ▶ toute exécution de $A_1 \otimes A_2$ sur un mot w simule en parallèle l'exécution de A_1 sur w , ainsi que celle de A_2 .
 - ▶ exemple avec $w = abba$
 - ▶ sur A_1 , on a $e_1 = p_0 a p_0 b p_1 b p_0 a p_0$
 - ▶ sur A_2 , on a $e_2 = q_0 a q_1 b q_1 b q_1 a q_0$
 - ▶ sur $A_1 \otimes A_2$, on a $(p_0, q_0)a(p_0, q_1)b(p_1, q_1)b(p_0, q_1)a(p_0, q_0)$.
- ▶ comment définir, à partir du produit, un automate qui accepte $L(A_1) \cap L(A_2)$? et $L(A_1) \cup L(A_2)$?

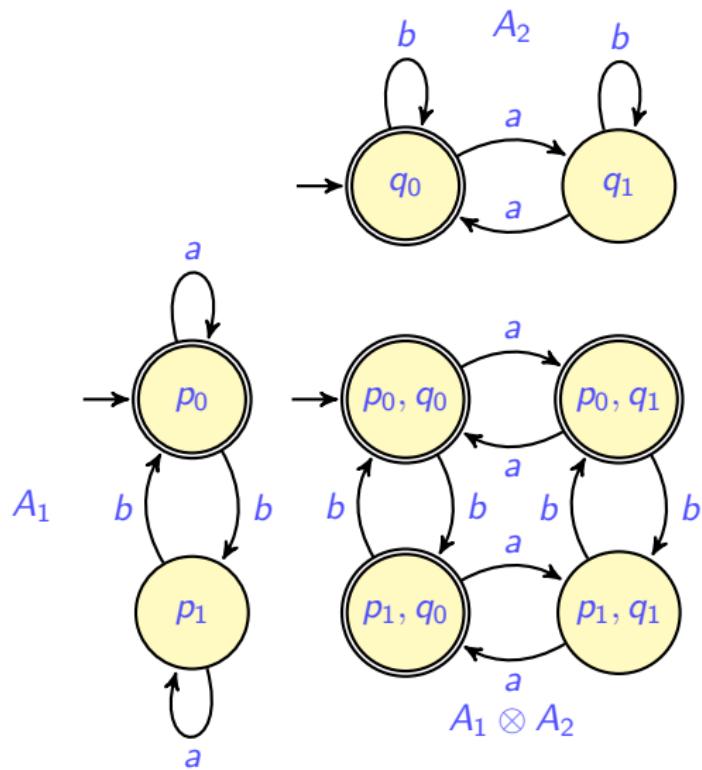
33 Exemple de produit

- ▶ $L(A_1)$ est l'ensemble des mots qui contiennent un nombre pair de b
- ▶ $L(A_2)$ est l'ensemble des mots qui contiennent un nombre pair de a
- ▶ toute exécution de $A_1 \otimes A_2$ sur un mot w simule en parallèle l'exécution de A_1 sur w , ainsi que celle de A_2 .
 - ▶ exemple avec $w = abba$
 - ▶ sur A_1 , on a $e_1 = p_0 a p_0 b p_1 b p_0 a p_0$
 - ▶ sur A_2 , on a $e_2 = q_0 a q_1 b q_1 b q_1 a q_0$
 - ▶ sur $A_1 \otimes A_2$, on a $(p_0, q_0)a(p_0, q_1)b(p_1, q_1)b(p_0, q_1)a(p_0, q_0)$.
- ▶ comment définir, à partir du produit, un automate qui accepte $L(A_1) \cap L(A_2)$? et $L(A_1) \cup L(A_2)$?
- ▶ pour avoir $L(A_1) \cap L(A_2)$, il suffit de prendre $F_{\cap} = F_1 \times F_2$ pour états finaux de $A_1 \otimes A_2$,
- ▶ pour avoir $L(A_1) \cup L(A_2)$, il suffit de prendre $F_{\cup} = (F_1 \times Q_2) \cup (Q_1 \times F_2)$ pour états finaux de $A_1 \otimes A_2$.

34 Clôture par intersection



35 Clôture par union



36 Clôture par union et intersection

Théorème

Soient $A_1 = (Q_1, q_0^1, F_1, \delta_1)$ et $A_2 = (Q_2, q_0^2, F_2, \delta_2)$ deux automates. Soit $A_1 \otimes A_2 = (Q_1 \times Q_2, (q_0^1, q_0^2), \delta_{12})$ le pré-automate produit.

- Si A_1 et A_2 sont complets⁴, et
 $U = (Q_1 \times Q_2, (q_0^1, q_0^2), (F_1 \times Q_2) \cup (Q_1 \times F_2), \delta_{12})$, alors

$$L(U) = L(A_1) \cup L(A_2).$$

- Si $I = (Q_1 \times Q_2, (q_0^1, q_0^2), F_1 \times F_2, \delta_{12})$, alors

$$L(I) = L(A_1) \cap L(A_2).$$

La preuve est laissée en exercice.

4. sinon on les complète avant de faire le produit

37 Inclusion et Equivalence

- ▶ on peut se servir des résultats précédents pour définir un automate permettant de tester si deux automates définissent le même langage (équivalence)

37 Inclusion et Equivalence

- ▶ on peut se servir des résultats précédents pour définir un automate permettant de tester si deux automates définissent le même langage (équivalence)

Définition

Soient A_1 et A_2 deux automates sur un alphabet Σ . On dit que A_1 et A_2 sont équivalents si $L(A_1) = L(A_2)$.

37 Inclusion et Equivalence

- ▶ on peut se servir des résultats précédents pour définir un automate permettant de tester si deux automates définissent le même langage (équivalence)

Définition

Soient A_1 et A_2 deux automates sur un alphabet Σ . On dit que A_1 et A_2 sont équivalents si $L(A_1) = L(A_2)$.

Théorème

Etant donnés deux automates A_1 et A_2 , il est décidable en temps polynomial si $L(A_1) \subseteq L(A_2)$, et si $L(A_1) = L(A_2)$.

38 Inclusion et Equivalence

Idée de l'algorithme ?

38 Inclusion et Equivalence

Idée de l'algorithme ? $L(A_1) = L(A_2)$ ssi $L(A_1) \subseteq L(A_2)$ et $L(A_2) \subseteq L(A_1)$. On peut donc se concentrer sur l'inclusion. Si les automates A_1 et A_2 ne sont pas complets, on commence par les compléter. Ensuite, on a :

$$L(A_1) \subseteq L(A_2) \text{ ssi } L(A_1) \cap \overline{L(A_2)} = \emptyset$$

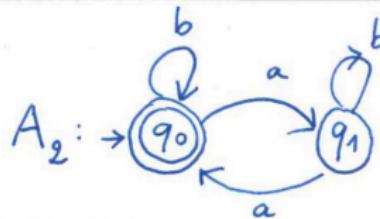
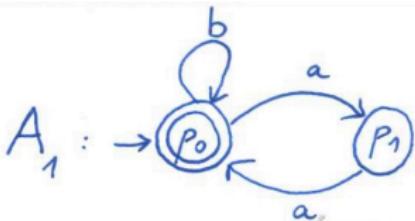
38 Inclusion et Equivalence

Idée de l'algorithme ? $L(A_1) = L(A_2)$ ssi $L(A_1) \subseteq L(A_2)$ et $L(A_2) \subseteq L(A_1)$. On peut donc se concentrer sur l'inclusion. Si les automates A_1 et A_2 ne sont pas complets, on commence par les compléter. Ensuite, on a :

$$L(A_1) \subseteq L(A_2) \text{ ssi } L(A_1) \cap \overline{L(A_2)} = \emptyset$$

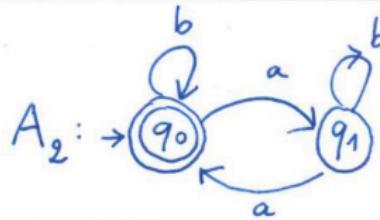
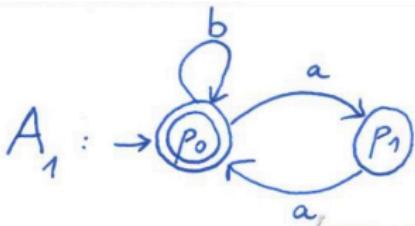
1. construire A_c tel que $L(A_c) = \overline{L(A_2)}$ (temps polynomial)
2. construire I tel que $L(I) = L(A_1) \cap L(A_c)$ avec le produit $A_1 \otimes A_c$ (temps polynomial)
3. tester le vide de I (temps polynomial)

39 Exemple

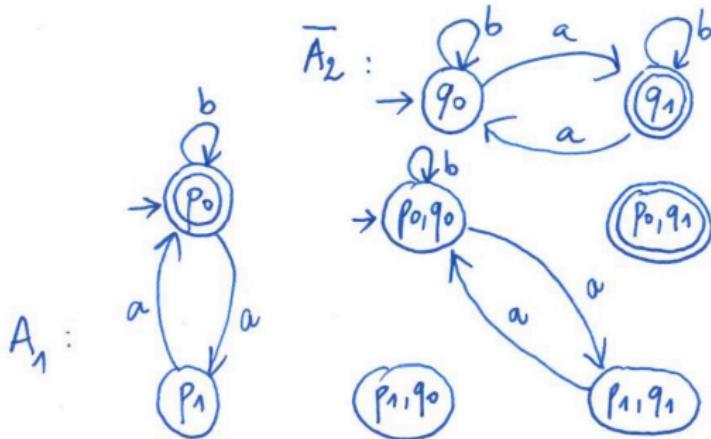


$$\boxed{L(A_1) \subseteq L(A_2) ?}$$

39 Exemple



$$L(A_1) \subseteq L(A_2) ?$$



QUESTION

Soient A_1 et A_2 deux automates complets.

On considère le produit $A_1 \otimes A_2$. Associer les opérations ensemblistes aux définitions d'états acceptants (pour une paire $(q_1, q_2) \in Q_1 \times Q_2$) :

union $L(A_1) \cup L(A_2)$

$q_1 \in F_1$ et $q_2 \in F_2$

intersection $L(A_1) \cap L(A_2)$

$q_1 \in F_1$ ou $q_2 \notin F_2$

déférence symétrique $L(A_1) \Delta L(A_2)^*$

$q_1 \in F_1$ ou $q_2 \in F_2$

déférence 1 $L(A_1) \setminus L(A_2)$

$(q_1 \in F_1 \text{ et } q_2 \notin F_2)$ ou $(q_1 \notin F_1 \text{ et } q_2 \in F_2)$

déférence 2 $L(A_2) \setminus L(A_1)$

$q_1 \notin F_1$ et $q_2 \in F_2$

$q_1 \in F_1$ et $q_2 \notin F_2$

*

$$E \Delta F = (E \cup F) \setminus (E \cap F)$$

METHODE PLUS DIRECTE POUR TESTER L'EQUIVALENCE

Pour tester l'équivalence, on pourra se passer de tester la double inclusion, en construisant un automate pour la différence symétrique et en testant son vide.

$$L(A_1) = L(A_2) \text{ ssi } L(A_1) \subseteq L(A_2) \text{ et } L(A_2) \subseteq L(A_1)$$

$$\text{ssi } L(A_1) \setminus L(A_2) = \emptyset \text{ et } L(A_2) \setminus L(A_1) = \emptyset$$

$$\text{ssi } L(A_1) \setminus L(A_2) \cup L(A_2) \setminus L(A_1) = \emptyset$$

$$\text{ssi } L(A_1) \Delta L(A_2) = \emptyset$$



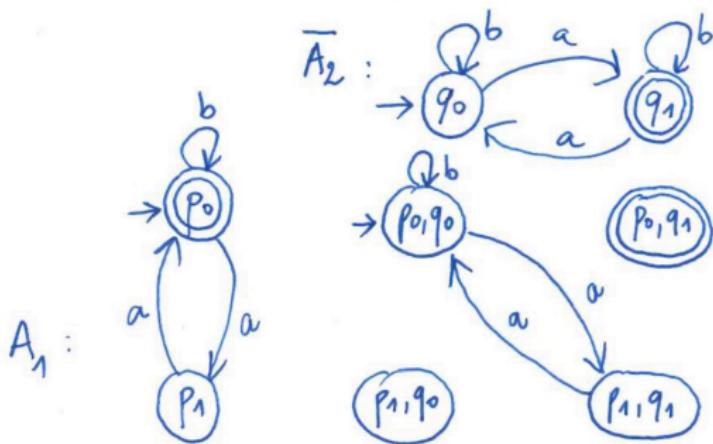
METHODE

① Faire l'automate produire (attention à compléter les automates avant)

② Une paire (q_1, q_2) est acceptante si $q_1 \in F_1$ et $q_2 \notin F_2$, ou $q_1 \notin F_1$ et $q_2 \in F_2$

③ tester le vide. Retourner "équivalents" ssi le langage est vide.

40 Exemple



$L(A_1) \subseteq L(A_2)$?

On peut vérifier sur le produit ci-dessus⁵ que $L(A_1) \cap \overline{L(A_2)} = \emptyset$, donc on a bien $L(A_1) \subseteq L(A_2)$.

5. Les transitions sortant de (p_0, q_1) et (p_1, q_0) ne sont pas dessinées car ces paires d'états ne sont pas atteignables, il est donc inutile de les considérer pour le test du vide

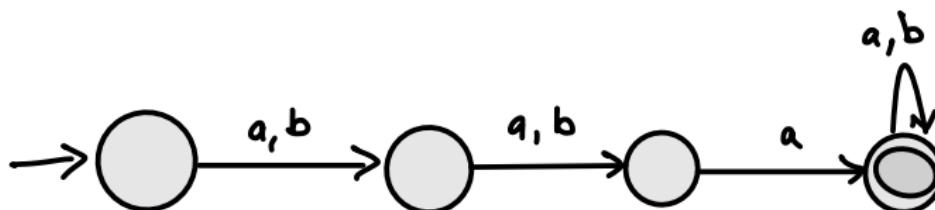
Automates non-déterministes

42 Exemple (facile)

Prenons l'alphabet $\Sigma = \{a, b\}$. Donner un automate qui accepte

$$L_3 = \{u \in \Sigma^* \mid |u| \geq 3 \text{ et } u[3] = a\}$$

i.e. l'ensemble des mots de longueur au moins 3 dont la troisième lettre est *a*.

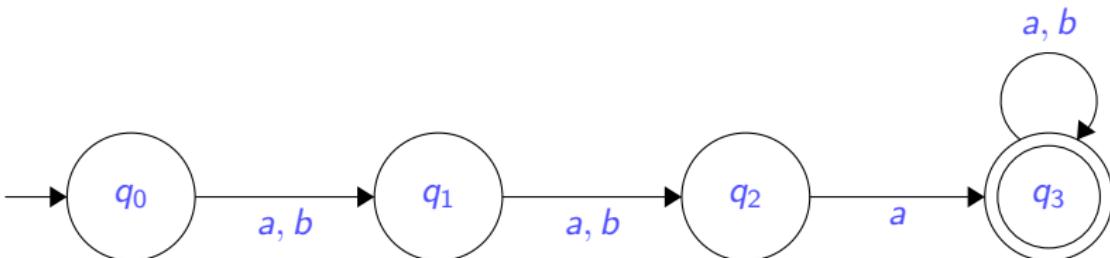


42 Exemple (facile)

Prenons l'alphabet $\Sigma = \{a, b\}$. Donner un automate qui accepte

$$L_3 = \{u \in \Sigma^* \mid |u| \geq 3 \text{ et } u[3] = a\}$$

i.e. l'ensemble des mots de longueur au moins 3 dont la troisième lettre est a .



43 Exemple (moins facile)

Prenons l'alphabet $\Sigma = \{a, b\}$. Donner un automate qui accepte

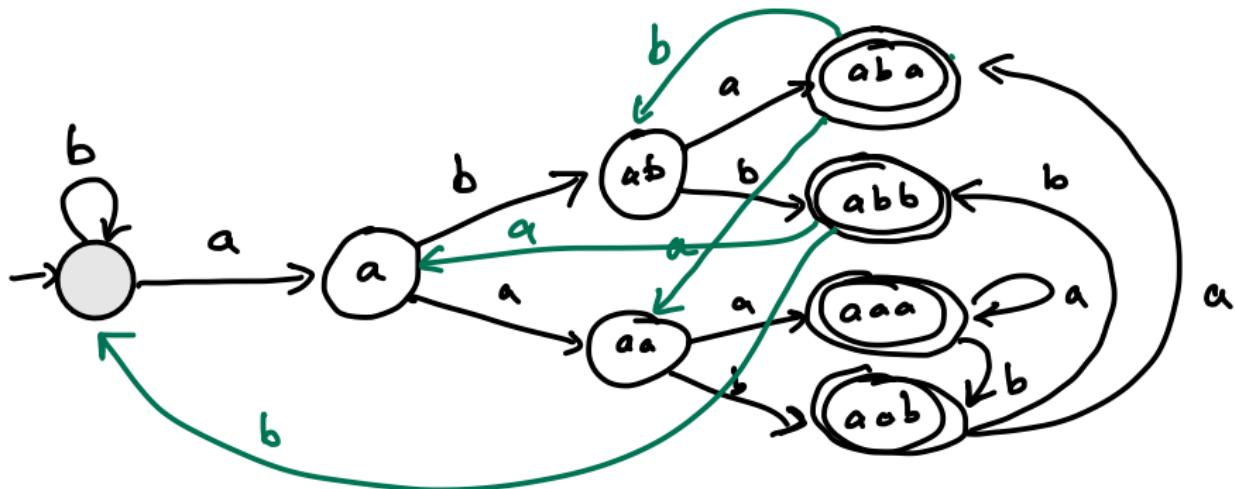
$$L_3 = \{u \in \Sigma^* \mid |u| \geq 3 \text{ et } u[|u| - 3] = a\}$$

i.e. l'ensemble des mots de longueur au moins 3 dont la troisième lettre **en partant de la fin** est **a**.

$a a b \ a \underline{a} b b$

$\underline{a} a a$

$b b b \underline{a} b a$

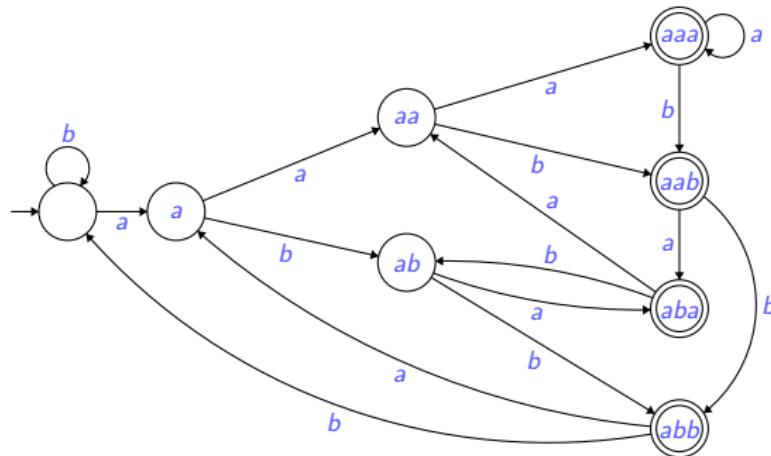


43 Exemple (moins facile)

Prenons l'alphabet $\Sigma = \{a, b\}$. Donner un automate qui accepte

$$L_3 = \{u \in \Sigma^* \mid |u| \geq 3 \text{ et } u[|u| - 3] = a\}$$

i.e. l'ensemble des mots de longueur au moins 3 dont la troisième lettre **en partant de la fin** est **a**.

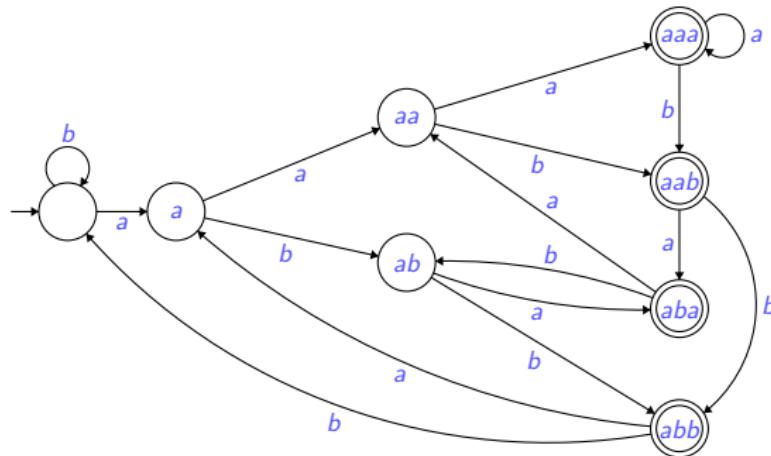


43 Exemple (moins facile)

Prenons l'alphabet $\Sigma = \{a, b\}$. Donner un automate qui accepte

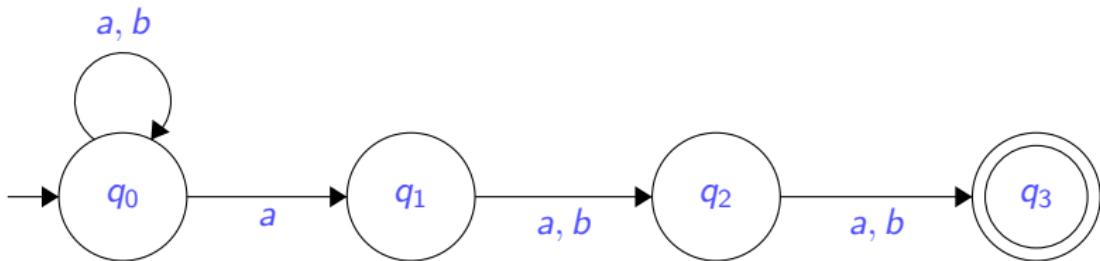
$$L_3 = \{u \in \Sigma^* \mid |u| \geq 3 \text{ et } u[|u| - 3] = a\}$$

i.e. l'ensemble des mots de longueur au moins 3 dont la troisième lettre en partant de la fin est **a**.



Généralement, si on veut vérifier que la n -ième lettre en partant de la fin est **a**, alors il faut $O(2^n)$ états (à cause de la lecture de gauche à droite).

Le même exemple avec un automate non-déterministe

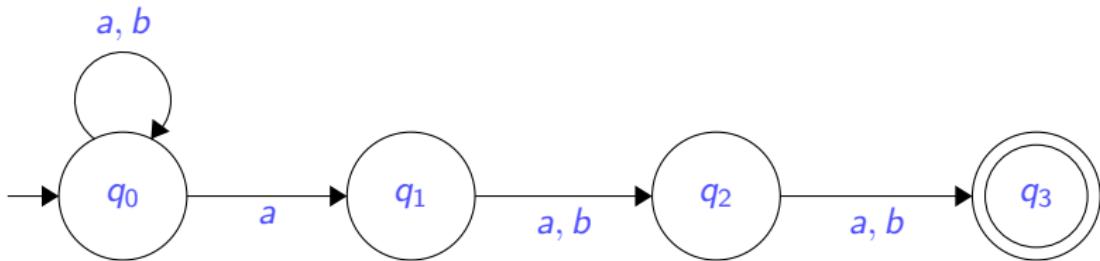


Sur un mot, il existe plusieurs exécutions : pour accepter le mot, au moins une doit être acceptante. Sur le mot *abaab*, on a les exécutions suivantes :

$$\begin{array}{ccccccccc}
 q_0 & \xrightarrow{a} & q_0 & \xrightarrow{b} & q_0 & \xrightarrow{a} & q_0 & \xrightarrow{a} & q_0 \\
 q_0 & \xrightarrow{a} & q_0 & \xrightarrow{b} & q_0 & \xrightarrow{a} & q_0 & \xrightarrow{b} & q_0 \\
 q_0 & \xrightarrow{a} & q_0 & \xrightarrow{b} & q_0 & \xrightarrow{a} & q_1 & \xrightarrow{b} & q_2 \\
 q_0 & \xrightarrow{a} & q_1 & \xrightarrow{b} & q_2 & \xrightarrow{a} & q_3 & \xrightarrow{b} & ?
 \end{array}$$

La troisième est acceptante, donc le mot est accepté.

Le même exemple avec un automate non-déterministe



Sur le mot *abbab*, on a les exécutions suivantes :

$$\begin{array}{ccccccccc} q_0 & \xrightarrow{a} & q_0 & \xrightarrow{b} & q_0 & \xrightarrow{b} & q_0 & \xrightarrow{a} & q_0 \\ q_0 & \xrightarrow{a} & q_0 & \xrightarrow{b} & q_0 & \xrightarrow{b} & q_0 & \xrightarrow{a} & q_1 \\ q_0 & \xrightarrow{a} & q_1 & \xrightarrow{b} & q_2 & \xrightarrow{b} & q_3 & \xrightarrow{a} & ? \end{array}$$

Aucune n'est acceptante \rightarrow le mot est rejeté.

Automate non-déterministe : définition

En rouge : ce qui change par rapport aux automates finis (déterministes)

Définition

Un automate fini non-déterministe (AFN) A sur un alphabet Σ est un 4-uplet (Q, q_0, F, Δ) où

- ▶ Q est un ensemble fini d'éléments appelé *états*,
- ▶ $q_0 \in Q$ est appelé état *initial*,
- ▶ $F \subseteq Q$ est un ensemble d'états dits *finaux* (ou acceptants),
- ▶ $\Delta \subseteq Q \times \Sigma \times Q$ est une relation appelée *relation de transition*.

47 Exécution, exécution acceptante et langage accepté

Définition

Une exécution de A est une suite finie $e = p_0\sigma_1p_1\sigma_2\dots p_{n-1}\sigma_np_n$ ($n \geq 0$) telle que :

- ▶ $p_0 = q_0$
- ▶ pour tout $i \in \{0, \dots, n\}$, $p_i \in Q$
- ▶ pour tout $i \in \{1, \dots, n\}$, $\sigma_i \in \Sigma$
- ▶ pour tout $i \in \{0, \dots, n-1\}$, $(p_i, \sigma_{i+1}, p_{i+1}) \in \Delta$.

On dit l'exécution e est une exécution sur le mot $\sigma_1\dots\sigma_n$, et que e est *acceptante* si l'état atteint est final, i.e. $p_n \in F$.

Définition (Langage accepté)

Etant donné un AFN A , le langage accepté (ou reconnu) par A , noté $L(A)$, est l'ensemble des mots pour lesquels il existe une exécution acceptante de A .

$$L(A) = \{w \in \Sigma^* \mid \text{il existe une exécution acceptante de } A \text{ sur } w\}$$

Questions

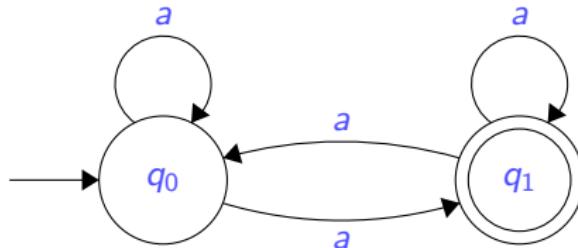
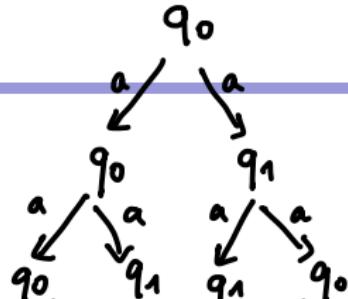
1. pour un AFN à n états, combien d'exécutions au plus sur un mot de taille m ?

Questions

1. pour un AFN à n états, combien d'exécutions au plus sur un mot de taille m ? exponentiel : n^m

Questions

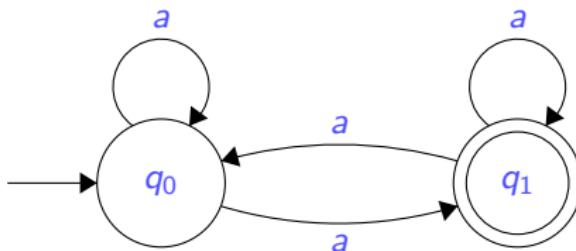
1. pour un AFN à n états, combien d'exécutions au plus sur un mot de taille m ? exponentiel : n^m



A chaque fois, on a le choix entre rester dans l'état ou changer d'états \Rightarrow sur $aa\dots a$ de longueur m , il y a 2^m exécutions.

Questions

1. pour un AFN à n états, combien d'exécutions au plus sur un mot de taille m ? exponentiel : n^m

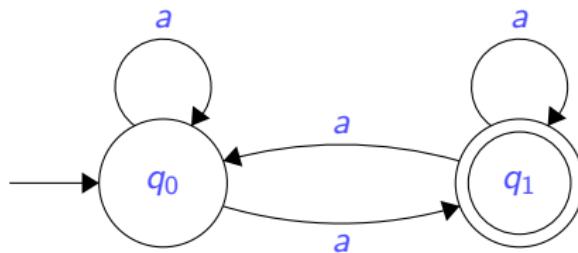


A chaque fois, on a le choix entre rester dans l'état ou changer d'états \Rightarrow sur $aa\dots a$ de longueur m , il y a 2^m exécutions.

2. Etant donné un AFN A et un mot u , quel est la complexité de décider si $u \in L(A)$?

Questions

1. pour un AFN à n états, combien d'exécutions au plus sur un mot de taille m ? exponentiel : n^m

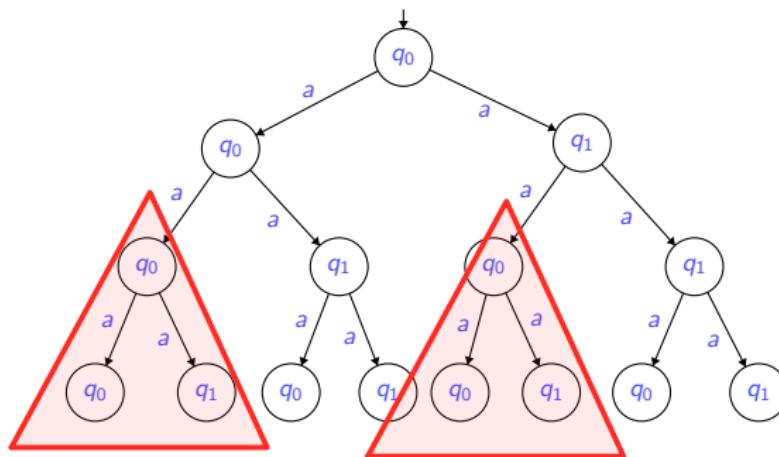


A chaque fois, on a le choix entre rester dans l'état ou changer d'états \Rightarrow sur $aa\dots a$ de longueur m , il y a 2^m exécutions.

2. Etant donné un AFN A et un mot u , quel est la complexité de décider si $u \in L(A)$? polynomiale, voyons maintenant pourquoi.

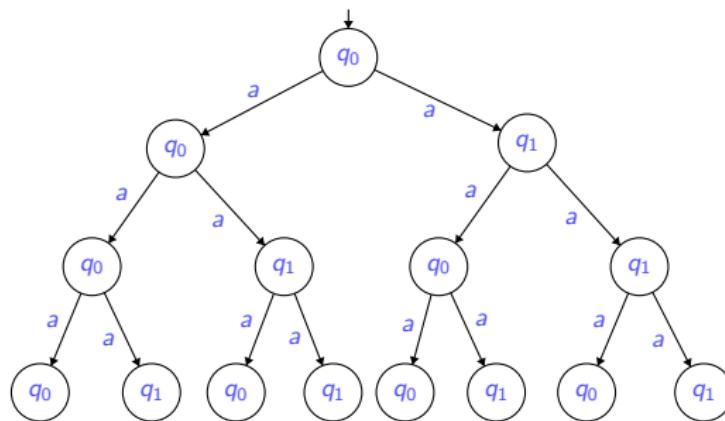
Arbre des exécutions

Toutes les exécutions peuvent être représentées par un arbre, prenons l'exemple de l'automate précédent sur le mot *aaa*, cela donne :



Arbre des exécutions

Toutes les exécutions peuvent être représentées par un arbre, prenons l'exemple de l'automate précédent sur le mot *aaa*, cela donne :

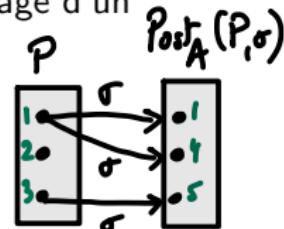


Remarque A un même niveau dans l'arbre, tous les sous-arbres enracinés aux mêmes états sont identiques \Rightarrow lorsqu'on lit un préfixe, ce qui est important, c'est l'ensemble des états atteints et non comment on les a atteints !

Test d'appartenance au langage

- ▶ on va exploiter l'idée précédente pour avoir une manière efficace de tester qu'un mot u sur un alphabet Σ appartient au langage d'un AFN $A = (Q, q_0, F, \Delta)$ donné, i.e. si $u \in L(A)$.
- ▶ Soit $P \subseteq Q$ et $\sigma \in \Sigma$, on note

$$\text{Post}_A(P, \sigma) = \{p \mid \exists p' \in P \cdot (p', \sigma, p) \in \Delta\}$$



l'ensemble des états qu'on peut atteindre à partir des états de P en lisant σ .

- ▶ on obtient l'algorithme suivant :

$\text{TEST}(A, P, u) =$

case $u = \epsilon$: **return** $P \cap F \neq \emptyset$

case $u = \sigma v$ ($\sigma \in \Sigma$) : **return** $\text{TEST}(A, \text{Post}_A(P, \sigma), v)$

- ▶ Lemme : $u \in L(A)$ ssi $\text{TEST}(A, \{q_0\}, u)$.

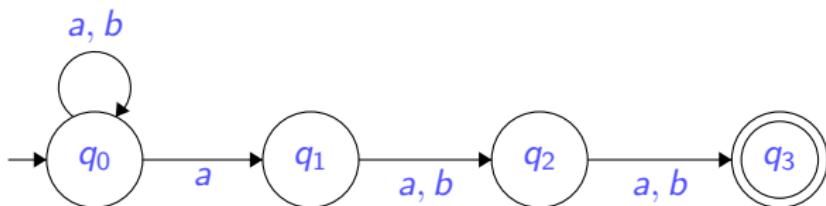
Test d'appartenance au langage

Théorème

Etant donné un AFN A avec n transitions et un mot u de longueur m ,
tester si $u \in L(A)$ peut se faire en temps $O(|u|.m)$.

Pour la preuve, on prend l'algorithme précédent, et il suffit de remarquer
que calculer $\text{Post}_A(P, \sigma)$ se fait en temps $O(m)$.

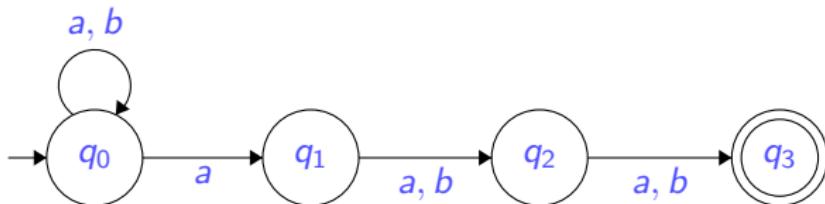
Test d'appartenance au langage : exemple



Exécutons l'algorithme sur le mot *abaab* :

q_0	\xrightarrow{a}	q_0	\xrightarrow{b}	q_0	\xrightarrow{a}	q_0	\xrightarrow{a}	q_0	\xrightarrow{b}	q_0
q_0	\xrightarrow{a}	q_0	\xrightarrow{b}	q_0	\xrightarrow{a}	q_0	\xrightarrow{a}	q_1	\xrightarrow{b}	q_2
q_0	\xrightarrow{a}	q_0	\xrightarrow{b}	q_0	\xrightarrow{a}	q_1	\xrightarrow{a}	q_2	\xrightarrow{b}	q_3
q_0	\xrightarrow{a}	q_1	\xrightarrow{b}	q_2	\xrightarrow{a}	q_3	\xrightarrow{a}	?		
ensemble atteint	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$
				$= Post_A(\{q_0\}, a)$	$= Post_A(\{q_0, q_1\}, b)$	$= Post_A(\{q_0, q_1\}, b)$	$= Post_A(\{q_0, q_1, q_2\}, a)$	$= Post_A(\{q_0, q_1, q_2\}, a)$	$= Post_A(\{q_0, q_1, q_2\}, a)$	$= Post_A(\{q_0, q_1, q_2\}, b)$

Test d'appartenance au langage : exemple



Exécutons l'algorithme sur le mot *abaab* :

q_0	$\xrightarrow{a} q_0$	$\xrightarrow{b} q_0$	$\xrightarrow{a} q_0$	$\xrightarrow{a} q_0$	$\xrightarrow{a} q_0$	$\xrightarrow{b} q_0$
q_0	$\xrightarrow{a} q_0$	$\xrightarrow{b} q_0$	$\xrightarrow{a} q_0$	$\xrightarrow{a} q_1$	$\xrightarrow{a} q_1$	$\xrightarrow{b} q_2$
q_0	$\xrightarrow{a} q_0$	$\xrightarrow{b} q_0$	$\xrightarrow{a} q_1$	$\xrightarrow{a} q_2$	$\xrightarrow{a} q_2$	$\xrightarrow{b} q_3$
q_0	$\xrightarrow{a} q_1$	$\xrightarrow{b} q_2$	$\xrightarrow{a} q_3$	\xrightarrow{a} ?		
ensemble atteint	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_2, q_3\}$
	$= Post_A(\{q_0\}, a)$					
	$= Post_A(\{q_0, q_1\}, b)$					
	$= Post_A(\{q_0, q_2\}, a)$					
	$= Post_A(\{q_0, q_1, q_3\}, a)$					
	$= Post_A(\{q_0, q_1, q_2\}, a)$					
	$= Post_A(\{q_0, q_1, q_2\}, b)$					

⇒ l'ensemble final $\{q_0, q_2, q_3\}$ contient un état acceptant donc le mot est accepté.

Test du vide

Théorème

Etant donné un AFN A avec n états et m transitions, on peut décider en temps $O(n + m)$ si $L(A) = \emptyset$.

L'algorithme est exactement le même que pour les automates finis déterministes.

Automates déterministes versus automates non-déterministes

- ▶ tout langage accepté par un automate fini (déterministe) peut être accepté par un automate fini non-déterministe.
- ▶ est-ce que l'inverse est vrai ?

Automates déterministes versus automates non-déterministes

- ▶ tout langage accepté par un automate fini (déterministe) peut être accepté par un automate fini non-déterministe.
- ▶ est-ce que l'inverse est vrai ? oui, c'est ce qu'énonce le théorème suivant.

Automates déterministes versus automates non-déterministes

- ▶ tout langage accepté par un automate fini (déterministe) peut être accepté par un automate fini non-déterministe.
- ▶ est-ce que l'inverse est vrai ? oui, c'est ce qu'énonce le théorème suivant.

Théorème

Soit L un langage sur un alphabet Σ . L est accepté par un automate fini si et seulement si il est accepté par un automate fini non-déterministe.

- ▶ Nous allons démontrer le sens AFN \rightarrow AF.
- ▶ étant donné un ANF A , nous allons montrer comment construire un AF (déterministe) B équivalent (i.e. qui accepte le même langage)

Construction des sous-ensembles

- ▶ l'idée est la même que pour tester l'appartenance au langage : l'automate déterministe B qui simule l'automate non-déterministe A calcule le sous-ensemble d'états atteints.
- ▶ les états de B seront donc des sous-ensembles d'états de A
- ▶ à partir d'un sous-ensemble $P \subseteq Q$, en lisant un lettre σ , B va vers l'état $\text{Post}_A(P, \sigma)$.
- ▶ les états acceptants de B sont les sous-ensembles qui contiennent un état acceptant de A

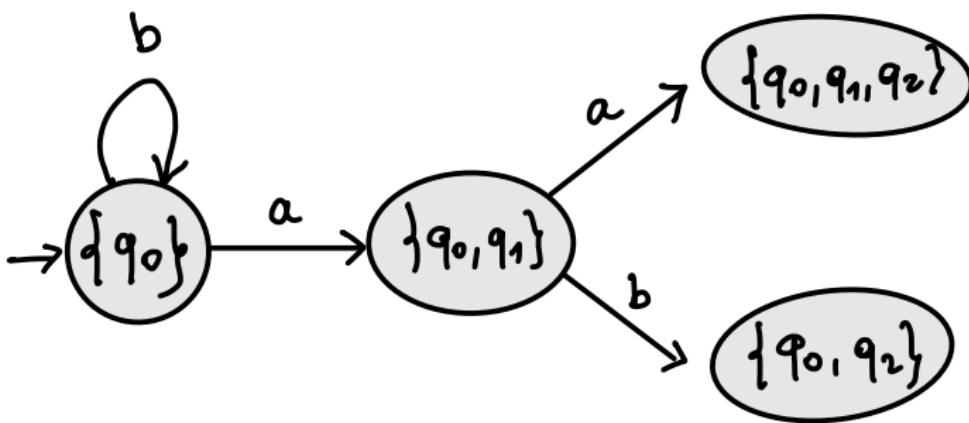
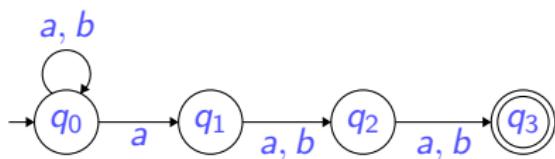
Construction des sous-ensembles

- ▶ l'idée est la même que pour tester l'appartenance au langage : l'automate déterministe B qui simule l'automate non-déterministe A calcule le sous-ensemble d'états atteints.
- ▶ les états de B seront donc des sous-ensembles d'états de A
- ▶ à partir d'un sous-ensemble $P \subseteq Q$, en lisant un lettre σ , B va vers l'état $\text{Post}_A(P, \sigma)$.
- ▶ les états acceptants de B sont les sous-ensembles qui contiennent un état acceptant de A

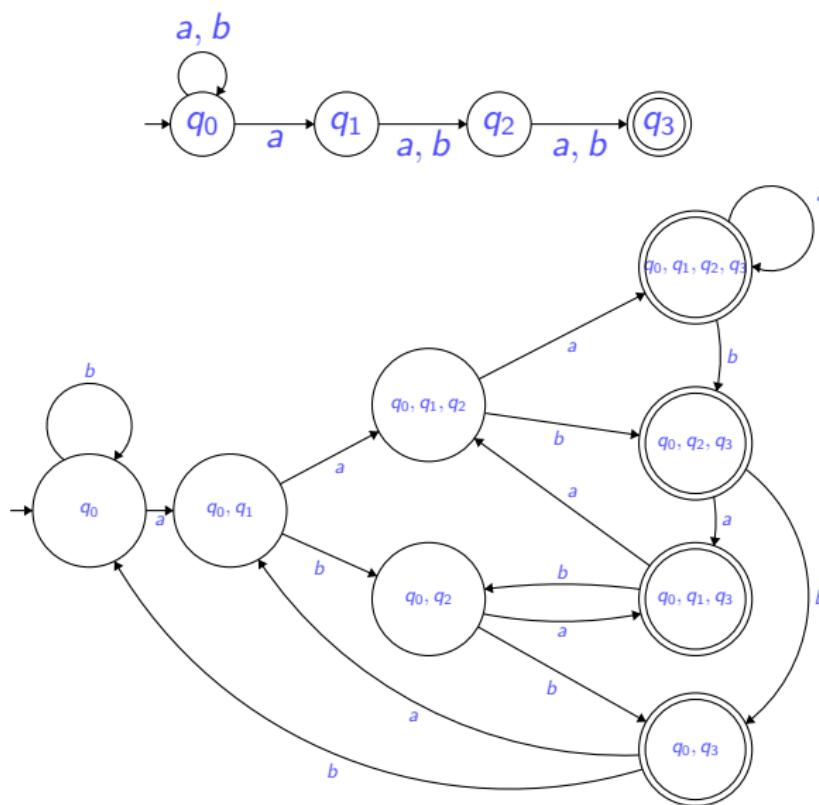
Formellement :

- ▶ $A = (Q, q_0, F, \Delta)$ sur un alphabet Σ
- ▶ alors $B = (2^Q, \{q_0\}, F' := \{P \subseteq Q \mid P \cap F \neq \emptyset\}, \delta := \text{Post}_A)$.

Exemple



Exemple



Complexité

- ▶ l'automate B construit a exponentiellement plus d'états que A
- ▶ le "saut" exponentiel est inévitable en général
- ▶ on peut montrer que pour tout $n \geq 0$, le plus petit automate fini déterministe acceptant le langage L_n des mots de longueur au moins n dont la n ème lettre en partant de la fin est a , sur l'alphabet $\{a, b\}$, a $2^{O(n)}$ états.
- ▶ le plus petit AFN pour L_n a $O(n)$ états
- ▶ on dit que les AFN sont *exponentiellement plus succincts* que les AF

Expressions Rationnelles

59 Expressions Rationnelles

Définition

Une expression rationnelle E sur un alphabet Σ est une expression qui respecte la grammaire suivante :

$$E ::= \epsilon \mid a \mid \emptyset \mid (E + E) \mid (E.E) \mid E^*$$

pour tout $a \in \Sigma$.

59 Expressions Rationnelles

Définition

Une expression rationnelle E sur un alphabet Σ est une expression qui respecte la grammaire suivante :

$$E ::= \epsilon \mid a \mid \emptyset \mid (E + E) \mid (E.E) \mid E^*$$

pour tout $a \in \Sigma$.

Exemple : $((a + b)^*.a)$

60 Autres opérations sur les langages

Afin de donner une sémantique en terme de langage aux expressions rationnelles, nous définissons des nouvelles opérations :

Définition

Soient $L, L_1, L_2 \subseteq \Sigma^*$ trois langages. Alors

- ▶ $L_1 \cdot L_2 = \{u_1 u_2 \mid u_1 \in L_1 \wedge u_2 \in L_2\}$ (on écrira aussi $L_1 L_2$)
- ▶ $L^* = \{u_1 \dots u_k \mid k \geq 0, u_i \in L \text{ pour tout } i \in \{1, \dots, k\}\}$ (en particulier $\epsilon \in L^*$ avec $k = 0$)

60 Autres opérations sur les langages

Afin de donner une sémantique en terme de langage aux expressions rationnelles, nous définissons des nouvelles opérations :

Définition

Soient $L, L_1, L_2 \subseteq \Sigma^*$ trois langages. Alors

- $L_1 \cdot L_2 = \{u_1 u_2 \mid u_1 \in L_1 \wedge u_2 \in L_2\}$ (on écrira aussi $L_1 L_2$)
- $L^* = \{u_1 \dots u_k \mid k \geq 0, u_i \in L \text{ pour tout } i \in \{1, \dots, k\}\}$ (en particulier $\epsilon \in L^*$ avec $k = 0$)

Exemples : Si $L_1 = \text{PAIR}$, alors $L_1 L_1 = L_1$. Si $L_1 = \{a, b\}$ et $L_2 = \{a, bb\}$, alors $L_1 L_2 = \{aa, abb, ba, bbb\}$. Si $L = \{a\}$, alors $L^* = \{a^n \mid n \geq 0\}$.

61 Sémantique des expressions rationnelles

Définition

La sémantique d'une expression rationnelle E sur Σ est donnée par un langage, noté $L(E)$, défini inductivement par :

- ▶ $L(\epsilon) = \{\epsilon\}$,
- ▶ $L(a) = \{a\}$ pour tout $a \in \Sigma$,
- ▶ $L(\emptyset) = \emptyset$,
- ▶ $L(E_1 + E_2) = L(E_1) \cup L(E_2)$,
- ▶ $L(E_1.E_2) = L(E_1).L(E_2)$,
- ▶ $L(E^*) = L(E)^*$.

61 Sémantique des expressions rationnelles

Définition

La sémantique d'une expression rationnelle E sur Σ est donnée par un langage, noté $L(E)$, défini inductivement par :

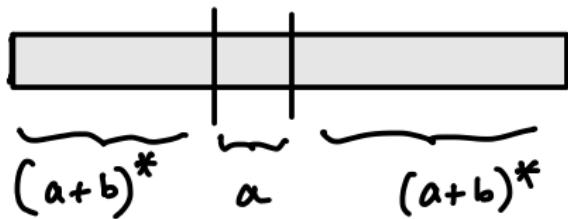
- ▶ $L(\epsilon) = \{\epsilon\}$,
- ▶ $L(a) = \{a\}$ pour tout $a \in \Sigma$,
- ▶ $L(\emptyset) = \emptyset$,
- ▶ $L(E_1 + E_2) = L(E_1) \cup L(E_2)$,
- ▶ $L(E_1 \cdot E_2) = L(E_1) \cdot L(E_2)$,
- ▶ $L(E^*) = L(E)^*$.

On pourra omettre les parenthèses dans les expressions régulières lorsque cela est possible.

62 Exemples

Sur $\Sigma = \{a, b\}$.

► $L((a + b)^* a (a + b)^*)$ est



62 Exemples

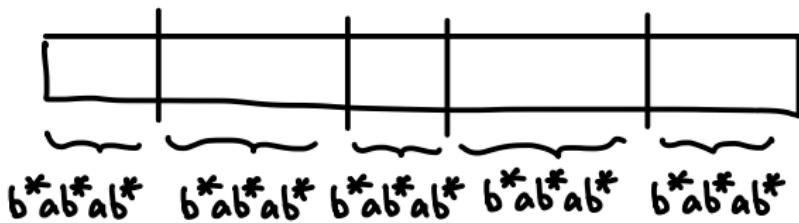
Sur $\Sigma = \{a, b\}$.

- $L((a + b)^* a (a + b)^*)$ est l'ensemble des mots qui contiennent au moins un a ,

62 Exemples

Sur $\Sigma = \{a, b\}$.

- ▶ $L((a + b)^* a (a + b)^*)$ est l'ensemble des mots qui contiennent au moins un a ,
- ▶ $L((b^* ab^* ab^*)^*) =$



62 Exemples

Sur $\Sigma = \{a, b\}$.

- ▶ $L((a + b)^* a (a + b)^*)$ est l'ensemble des mots qui contiennent au moins un a ,
- ▶ $L((b^* ab^* ab^*)^*) = \text{PAIR}$,

62 Exemples

Sur $\Sigma = \{a, b\}$.

- ▶ $L((a + b)^* a (a + b)^*)$ est l'ensemble des mots qui contiennent au moins un a ,
- ▶ $L((b^* ab^* ab^*)^*) = \text{PAIR}$,
- ▶ $L(aa^*)$ est

62 Exemples

Sur $\Sigma = \{a, b\}$.

- ▶ $L((a + b)^* a (a + b)^*)$ est l'ensemble des mots qui contiennent au moins un a ,
- ▶ $L((b^* ab^* ab^*)^*) = \text{PAIR}$,
- ▶ $L(aa^*)$ est l'ensemble des mots qui ne contiennent que des a , et au moins un

62 Exemples

Sur $\Sigma = \{a, b\}$.

- ▶ $L((a + b)^* a (a + b)^*)$ est l'ensemble des mots qui contiennent au moins un a ,
- ▶ $L((b^* ab^* ab^*)^*) = \text{PAIR}$,
- ▶ $L(aa^*)$ est l'ensemble des mots qui ne contiennent que des a , et au moins un
- ▶ $L(a^* b^*)$ est

62 Exemples

Sur $\Sigma = \{a, b\}$.

- ▶ $L((a + b)^* a (a + b)^*)$ est l'ensemble des mots qui contiennent au moins un a ,
- ▶ $L((b^* ab^* ab^*)^*) = \text{PAIR}$,
- ▶ $L(aa^*)$ est l'ensemble des mots qui ne contiennent que des a , et au moins un
- ▶ $L(a^* b^*)$ est l'ensemble des mots qui sont des séquences de a suivies des séquences de b .

62 Exemples

Sur $\Sigma = \{a, b\}$.

- ▶ $L((a + b)^* a (a + b)^*)$ est l'ensemble des mots qui contiennent au moins un a ,
- ▶ $L((b^* ab^* ab^*)^*) = \text{PAIR}$,
- ▶ $L(aa^*)$ est l'ensemble des mots qui ne contiennent que des a , et au moins un
- ▶ $L(a^* b^*)$ est l'ensemble des mots qui sont des séquences de a suivies des séquences de b .

63 Théorème dit de Kleene

Théorème

Tout langage est reconnaissable par un automate si et seulement si il est définissable par une expression rationnelle.

- ▶ Autrement dit, si $L \subseteq \Sigma^*$ est un langage, alors il existe un automate A tel que $L = L(A)$ ssi il existe une expression rationnelle E telle que $L = L(E)$.
- ▶ Ce théorème est effectif : il existe un algorithme qui transforme tout automate en expression rationnelle, et un algorithme qui transforme toute expression rationnelle en automate (ce qui donne d'ailleurs un algorithme pour étant donné un mot décide si ce dernier est dans le langage de l'expression rationnelle).

Preuve du théorème : expressions vers automates

La construction se fait par induction sur les expressions. Pour toute expression E , on va construire un AFN A_E tel que $L(E) = L(A_E)$ (il suffit ensuite de déterminiser A_E avec la construction des sous-ensembles pour terminer la preuve du théorème)

- si $E = \epsilon$, alors $A_E = (\{q_0\}, q_0, \{q_0\}, \Delta := \emptyset)$ (A_E accepte uniquement ϵ)



Preuve du théorème : expressions vers automates

La construction se fait par induction sur les expressions. Pour toute expression E , on va construire un AFN A_E tel que $L(E) = L(A_E)$ (il suffit ensuite de déterminiser A_E avec la construction des sous-ensembles pour terminer la preuve du théorème)

- ▶ si $E = \epsilon$, alors $A_E = (\{q_0\}, q_0, \{q_0\}, \Delta := \emptyset)$ (A_E accepte uniquement ϵ)
- ▶ si $E = a$ avec $a \in \Sigma$, alors $L(E) = \{a\}$
 $A_E = (\{q_0, q_1\}, q_0, \{q_1\}, \Delta := \{(q_0, a, q_1)\})$ (A_E accepte uniquement a)



Preuve du théorème : expressions vers automates

La construction se fait par induction sur les expressions. Pour toute expression E , on va construire un AFN A_E tel que $L(E) = L(A_E)$ (il suffit ensuite de déterminiser A_E avec la construction des sous-ensembles pour terminer la preuve du théorème)

- ▶ si $E = \epsilon$, alors $A_E = (\{q_0\}, q_0, \{q_0\}, \Delta := \emptyset)$ (A_E accepte uniquement ϵ)
- ▶ si $E = a$ avec $a \in \Sigma$, alors $A_E = (\{q_0, q_1\}, q_0, \{q_1\}, \Delta := \{(q_0, a, q_1)\})$ (A_E accepte uniquement a)
- ▶ si $E = \emptyset$, alors $A_E = (\{q_0\}, q_0, \emptyset, \emptyset)$

$$A_\phi \rightarrow O$$

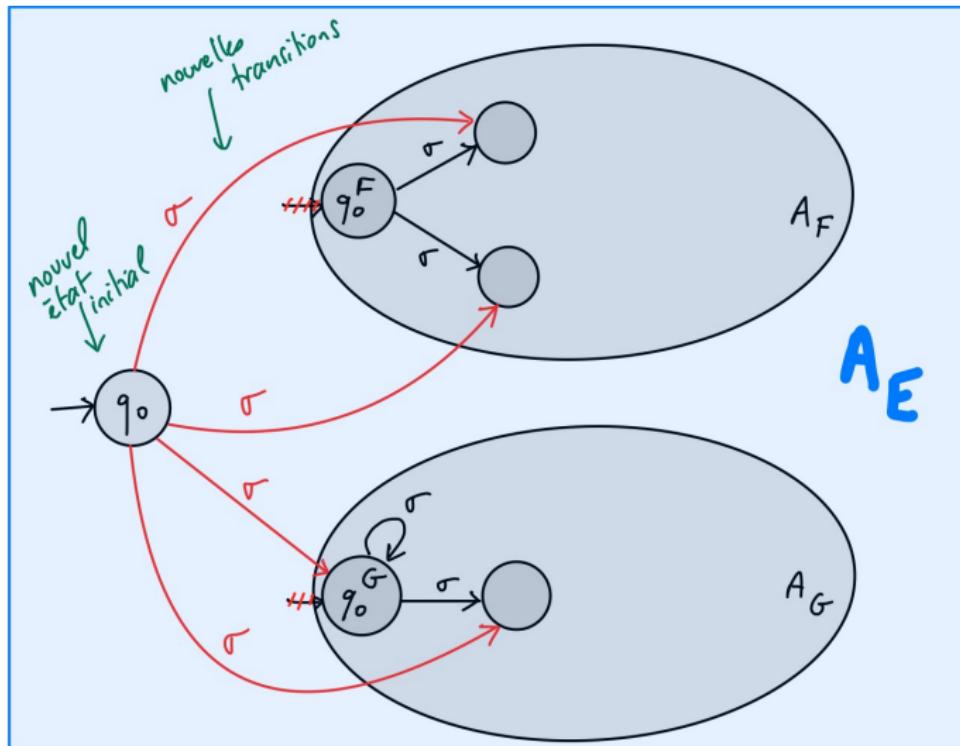
Preuve (suite) dans le cas $E = F + G$

On construit par induction A_F et A_G , à partir desquels on construit A_E tel que

$$L(A_E) = L(A_F) \cup L(A_G)$$

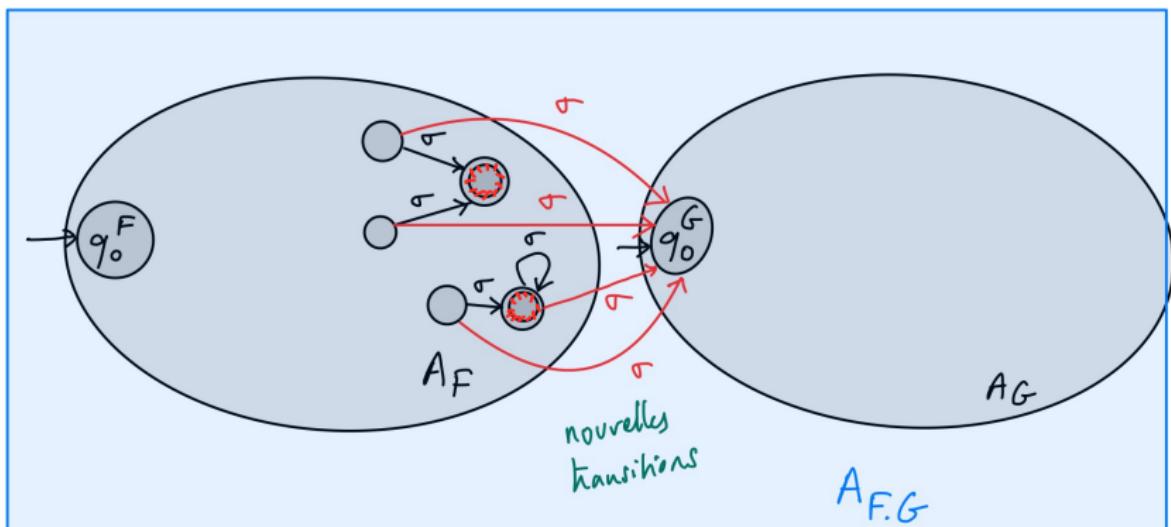
C'est la clôture par union, qu'on a déjà vu pour les automates déterministes. La même construction s'adapte aux automates non-déterministes, mais grâce au non-déterminisme on peut faire plus simple.

- ▶ en lisant la première lettre σ , on va soit dans le premier automate soit dans le deuxième, en utilisant le non-déterminisme
- ▶ précisément, A_E a un état initial q_0 . On note q_0^F l'état initial de A_F et q_0^G celui de A_G . Pour toute lettre σ , pour toute transition $(q_0^F, \sigma, q) \in \Delta_F$, on ajoute la transition (q_0, σ, q) (on fait la même chose pour Δ_G)
- ▶ on rend le nouvel état initial acceptant si q_0^F ou q_0^G étaient acceptants (pour accepter le mot vide)

Preuve (suite) dans le cas $E = F + G$ 

Preuve (suite) dans le cas $E = F \cdot G$

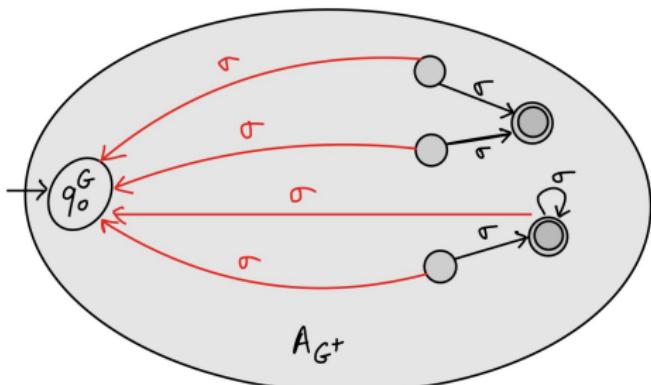
On construit A_F et A_G par induction, puis pour toute lettre σ , pour tout état de A_F qui allait vers un état final de A_F en lisant σ , on ajoute une transition vers l'état initial de A_G . **Important** : les états acceptants de A_F deviennent non-acceptants, et l'état initial de A_E est l'état initial de A_F .



Preuve (suite) dans le cas $E = G^*$

Il faut faire attention au fait que $\epsilon \in L(E)$. On réécrit d'abord E comme $E = \epsilon + G^+$ où G^+ est comme l'itération G^* mais au moins une fois. En particulier, $L(G^+)$ est l'ensemble des mots de la forme $u_1 u_2 \dots u_k$ avec $k \geq 1$ et $u_i \in L(G)$ pour tout i .

Ensuite, on construit A_ϵ , et A_G par induction. On montre maintenant comment obtenir A_{G^+} , et enfin A_E sera obtenu en utilisant la clôture par union de A_ϵ et A_{G^+} . Pour construire A_{G^+} , c'est similaire à la concaténation sauf qu'on ajoute des transitions vers l'état initial de G :

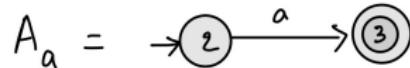
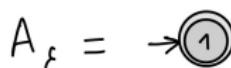


Exercice

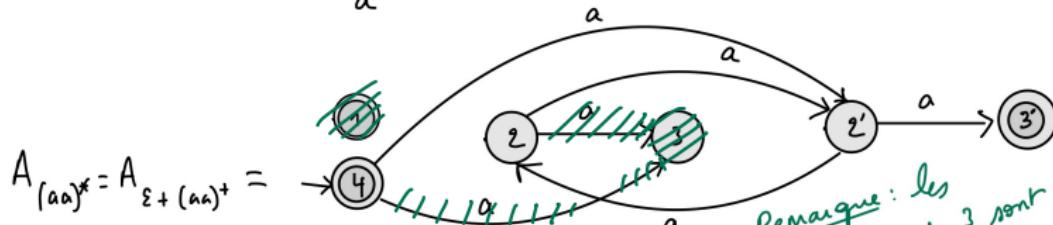
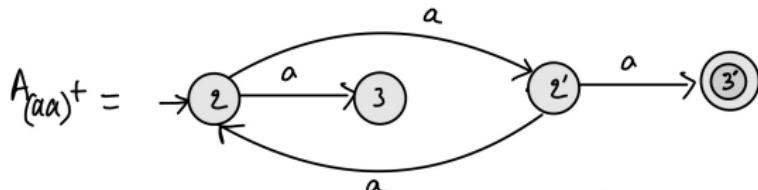
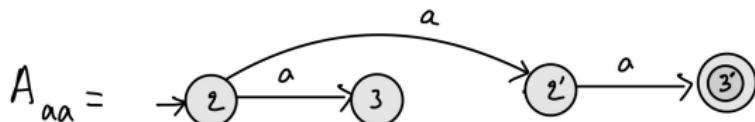
Construire un AFN pour l'expression $(aa)^*$ sur l'alphabet $\{a\}$.

Exercice

Construire un AFN pour l'expression $(aa)^*$ sur l'alphabet $\{a\}$.



$$(aa)^* = \epsilon + (aa)^+$$



Remarque : les états 1 et 3 sont inutiles

Python : expressions régulières

Pascal Vanier

(Meru)

Python — M1 Informatique, FST, UPEC, 2015/2016

(PAS À L'EXAMEN)

Expressions régulières

Le module `re` fait partie des modules de base de python.

Imaginons que l'on veuille trouver un mot ne commençant pas par Z dans un texte :

```
1 import re
2
3 z=re.compile(r"[bB] [a-zA-Z]*") # chaines commençant par b ou B
4 # suivies de n'importe quelles
5 # lettres de a-z ou A-Z
6 w=re.compile(r"\w+")
7 # mots d'au moins une lettre de \w toutes
8 # ie. toutes les lettres (y compris accents)
9
10 phrase = "Zorro zozotte contrairement à Bernardo"
11
12 w.findall(phrase) # on cherche tous les w dans phrase
13 # et renvoie une liste de ceux-ci
14
15 z.search(phrase) # on cherche la premiere occurence de z
# dans phrase
```

Expressions régulières

Les [] déterminent un sous ensemble de lettres :

```
[ski] # les lettres s k et i  
[a-z] # les lettres de a à z = [abcdefghijklmnopqrstuvwxyz]  
[^a-z] # tout sauf [a-z]
```

On a des groupes de caractères :

```
. # tous les caractères sauf \n  
\w # les lettres (unicode) qui peuvent être dans un mot,  
# y compris les chiffres et _  
\W # les caractères qui ne sont pas dans \w  
\s # les caractères d'espacement (espace, tabulation, retour  
# à la ligne...)  
\S # les caractères qui ne sont pas dans \s  
\d # les chiffres (unicode)  
\D # les caractères qui ne sont pas dans \d  
\b # caractère vide mais uniquement au début ou à la fin  
# d'un mot  
\B # caractère vide mais pas au début et à la fin d'un mot.  
\A # début de la chaîne de caractères = ^  
\Z # fin de la chaîne de caractères = $
```

On

peut composer :

```
[\d\s] # les chiffres et les espaces
```

Expressions régulières : construction

- `expr1|expr2` on matche soit `expr1` soit `expr2`
- `expr*` on matche une répétition de `expr` (potentiellement 0 fois)
- `expr+` on matche une répétition de `expr` (au moins 1 fois)
- `expr?` on matche `expr` au plus une fois
- `+, ? et *` font référence au caractère précédent : on peut utiliser des `(?:expr)` pour englober une expression.
- `^` permet de matcher le début de ligne et
- `$` la fin de ligne.

Par défaut, `search` ne cherche que sur la première ligne, il faut spécifier l'option `re.MULTILINE` (= `re.M`) pour qu'il cherche sur toutes les lignes.

Exemples

- exemples du début
- extraire les adresses emails d'une page web

[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

- code Python

```
import re

file = open("faculte.html")
line = file.read().replace("\n", " ")
file.close()

email_re = re.compile(r"[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+")
email_re.findall(line)

email_re_a = re.compile(r"\ba[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+\b")
email_re_a.findall(line)
```

2 Minimisation

- ▶ étant donné un automate A , l'objectif de la minimisation est de construire un automate complet M tel que M a un nombre d'états minimal, et M est équivalent à A .

2 Minimisation

- étant donné un automate A , l'objectif de la minimisation est de construire un automate complet M tel que M a un nombre d'états minimal, et M est équivalent à A .

Question ?

1. est-ce qu'il existe un automate minimal complet unique ?
2. si oui, comment le calculer à partir de A ?

2 Minimisation

- ▶ étant donné un automate A , l'objectif de la minimisation est de construire un automate complet M tel que M a un nombre d'états minimal, et M est équivalent à A .

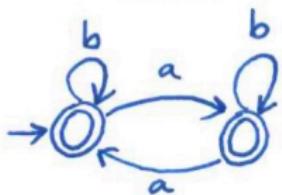
Question ?

1. est-ce qu'il existe un automate minimal complet unique ?
2. si oui, comment le calculer à partir de A ?

Nous allons d'abord répondre à la première question, puis à la deuxième.

3 Exemples

Minimal ?

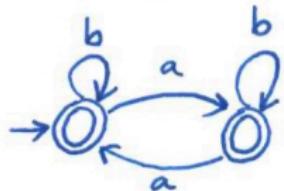


Non



3 Exemples

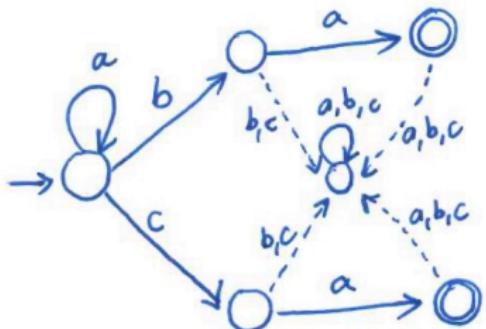
Minimal ?



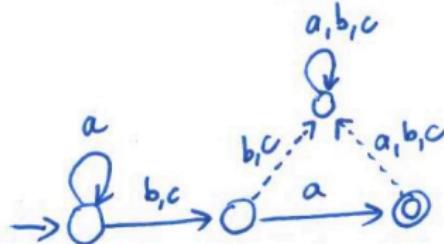
Non



Minimal ?



Non



4 Calcul de l'automate minimal à partir d'un automate

- ▶ nous allons définir une procédure qui, étant donné un automate complet A , construit l'automate minimal qui accepte $L(A)$ (on peut en effet montrer qu'il est unique)

4 Calcul de l'automate minimal à partir d'un automate

- ▶ nous allons définir une procédure qui, étant donné un automate complet A , construit l'automate minimal qui accepte $L(A)$ (on peut en effet montrer qu'il est unique)

Définition

Soit $A = (Q, q_0, F, \delta)$ un automate complet sur un alphabet Σ . Pour tout mot $u \in \Sigma^*$, pour tout état $q \in Q$, on note :

- ▶ $q \cdot u$ l'état atteint par A après lecture de u à partir de q (il existe car A est complet)
- ▶ L_q le langage formé des mots u tels que $q \cdot u \in F$, i.e.
 $L_q = \{u \in \Sigma^* \mid q \cdot u \in F\}$ est le langage des mots acceptés à partir de q .
- ▶ pour tout $p, q \in Q$, $p \equiv_A q$ si $L_p = L_q$.

4 Calcul de l'automate minimal à partir d'un automate

- ▶ nous allons définir une procédure qui, étant donné un automate complet A , construit l'automate minimal qui accepte $L(A)$ (on peut en effet montrer qu'il est unique)

Définition

Soit $A = (Q, q_0, F, \delta)$ un automate complet sur un alphabet Σ . Pour tout mot $u \in \Sigma^*$, pour tout état $q \in Q$, on note :

- ▶ $q \cdot u$ l'état atteint par A après lecture de u à partir de q (il existe car A est complet)
- ▶ L_q le langage formé des mots u tels que $q \cdot u \in F$, i.e.
 $L_q = \{u \in \Sigma^* \mid q \cdot u \in F\}$ est le langage des mots acceptés à partir de q .
- ▶ pour tout $p, q \in Q$, $p \equiv_A q$ si $L_p = L_q$.

Remarques : $L_{q_0} =$

4 Calcul de l'automate minimal à partir d'un automate

- ▶ nous allons définir une procédure qui, étant donné un automate complet A , construit l'automate minimal qui accepte $L(A)$ (on peut en effet montrer qu'il est unique)

Définition

Soit $A = (Q, q_0, F, \delta)$ un automate complet sur un alphabet Σ . Pour tout mot $u \in \Sigma^*$, pour tout état $q \in Q$, on note :

- ▶ $q \cdot u$ l'état atteint par A après lecture de u à partir de q (il existe car A est complet)
- ▶ L_q le langage formé des mots u tels que $q \cdot u \in F$, i.e.
 $L_q = \{u \in \Sigma^* \mid q \cdot u \in F\}$ est le langage des mots acceptés à partir de q .
- ▶ pour tout $p, q \in Q$, $p \equiv_A q$ si $L_p = L_q$.

Remarques : $L_{q_0} = L(A)$

4 Calcul de l'automate minimal à partir d'un automate

- ▶ nous allons définir une procédure qui, étant donné un automate complet A , construit l'automate minimal qui accepte $L(A)$ (on peut en effet montrer qu'il est unique)

Définition

Soit $A = (Q, q_0, F, \delta)$ un automate complet sur un alphabet Σ . Pour tout mot $u \in \Sigma^*$, pour tout état $q \in Q$, on note :

- ▶ $q \cdot u$ l'état atteint par A après lecture de u à partir de q (il existe car A est complet)
- ▶ L_q le langage formé des mots u tels que $q \cdot u \in F$, i.e.
 $L_q = \{u \in \Sigma^* \mid q \cdot u \in F\}$ est le langage des mots acceptés à partir de q .
- ▶ pour tout $p, q \in Q$, $p \equiv_A q$ si $L_p = L_q$.

Remarques : $L_{q_0} = L$, $\epsilon \in L_q$ pour tout $q \in F$.

4 Calcul de l'automate minimal à partir d'un automate

- ▶ nous allons définir une procédure qui, étant donné un automate complet A , construit l'automate minimal qui accepte $L(A)$ (on peut en effet montrer qu'il est unique)

Définition

Soit $A = (Q, q_0, F, \delta)$ un automate complet sur un alphabet Σ . Pour tout mot $u \in \Sigma^*$, pour tout état $q \in Q$, on note :

- ▶ $q \cdot u$ l'état atteint par A après lecture de u à partir de q (il existe car A est complet)
- ▶ L_q le langage formé des mots u tels que $q \cdot u \in F$, i.e.
 $L_q = \{u \in \Sigma^* \mid q \cdot u \in F\}$ est le langage des mots acceptés à partir de q .
- ▶ pour tout $p, q \in Q$, $p \equiv_A q$ si $L_p = L_q$.

Remarques : $L_{q_0} = L$, $\epsilon \in L_q$ pour tout $q \in F$. Pensez-vous que \equiv_A est une relation d'équivalence ?

4 Calcul de l'automate minimal à partir d'un automate

- ▶ nous allons définir une procédure qui, étant donné un automate complet A , construit l'automate minimal qui accepte $L(A)$ (on peut en effet montrer qu'il est unique)

Définition

Soit $A = (Q, q_0, F, \delta)$ un automate complet sur un alphabet Σ . Pour tout mot $u \in \Sigma^*$, pour tout état $q \in Q$, on note :

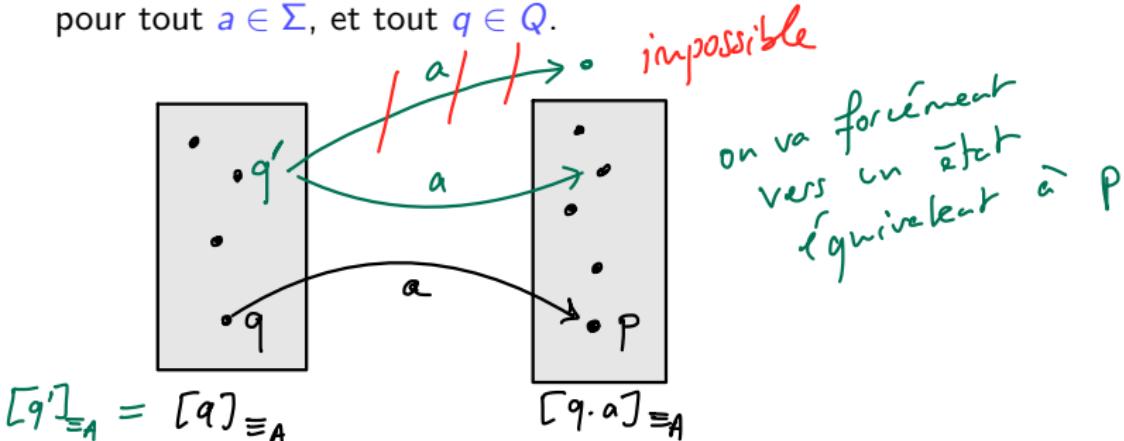
- ▶ $q \cdot u$ l'état atteint par A après lecture de u à partir de q (il existe car A est complet)
- ▶ L_q le langage formé des mots u tels que $q \cdot u \in F$, i.e.
 $L_q = \{u \in \Sigma^* \mid q \cdot u \in F\}$ est le langage des mots acceptés à partir de q .
- ▶ pour tout $p, q \in Q$, $p \equiv_A q$ si $L_p = L_q$.

Remarques : $L_{q_0} = L$, $\epsilon \in L_q$ pour tout $q \in F$. Pensez-vous que \equiv_A est une relation d'équivalence ? oui. On note $[p]_{\equiv_A}$ la classe de tout état p .

5 Calcul de l'automate minimal à partir d'un automate

A partir d'un automate A , on calcule un automate M_A comme suit :

- ▶ il suffit de calculer les classes d'équivalences de Q pour \equiv_A , ce qui donnera les états
- ▶ la classe de q_0 est l'état initial
- ▶ toute classe qui contient un état final est finale (en fait, si une classe contient un état final, alors tous ces états sont finaux)
- ▶ on met une transition de l'état $[q]_{\equiv_A}$ à l'état $[q \cdot a]_{\equiv_A}$ en lisant a , pour tout $a \in \Sigma$, et tout $q \in Q$.



5 Calcul de l'automate minimal à partir d'un automate

A partir d'un automate A , on calcule un automate M_A comme suit :

- ▶ il suffit de calculer les classes d'équivalences de Q pour \equiv_A , ce qui donnera les états
- ▶ la classe de q_0 est l'état initial
- ▶ toute classe qui contient un état final est finale (en fait, si une classe contient un état final, alors tous ces états sont finaux)
- ▶ on met une transition de l'état $[q]_{\equiv_A}$ à l'état $[q \cdot a]_{\equiv_A}$ en lisant a , pour tout $a \in \Sigma$, et tout $q \in Q$.

La fonction de transition est-elle bien définie ? Autrement dit, est-elle indépendante du choix du représentant dans la classe ?

5 Calcul de l'automate minimal à partir d'un automate

A partir d'un automate A , on calcule un automate M_A comme suit :

- ▶ il suffit de calculer les classes d'équivalences de Q pour \equiv_A , ce qui donnera les états
- ▶ la classe de q_0 est l'état initial
- ▶ toute classe qui contient un état final est finale (en fait, si une classe contient un état final, alors tous ces états sont finaux)
- ▶ on met une transition de l'état $[q]_{\equiv_A}$ à l'état $[q \cdot a]_{\equiv_A}$ en lisant a , pour tout $a \in \Sigma$, et tout $q \in Q$.

La fonction de transition est-elle bien définie ? Autrement dit, est-elle indépendante du choix du représentant dans la classe ? oui

Proposition

Si $p \equiv_A q$, alors $p \cdot a \equiv_A q \cdot a$ pour tout $a \in \Sigma$.

Preuve : Si $p \equiv_A q$, alors $L_p = L_q$ par définition. Soit $a \in \Sigma$. Soit $p' = p \cdot a$ et $q' = q \cdot a$. Soit $w \in L_{p'}$, alors $aw \in L_p$, donc $aw \in L_q$, donc $w \in L_{q'}$. La réciproque est symétrique.

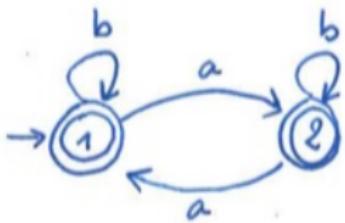
Théorème (non démontré)

Théorème

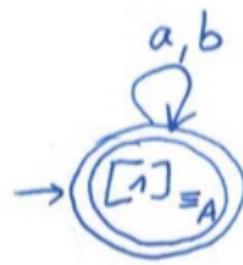
L'automate M_A est minimal en nombre d'états, et il est l'unique automate minimal qui accepte $L(A)$.

En particulier, si $L(A) = L(B)$, alors $M_A = M_B$: la définition de M_A ne dépend pas de A mais uniquement du langage accepté $L(A)$.

7 Exemples de calcul de l'automate minimal à partir d'un automate

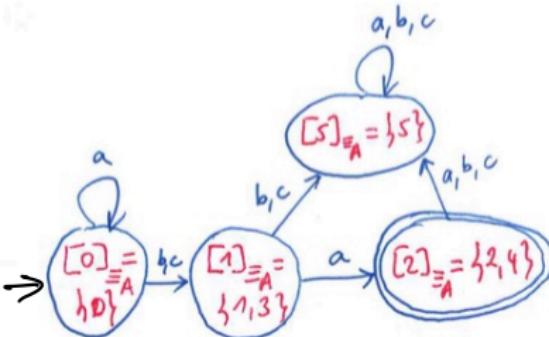
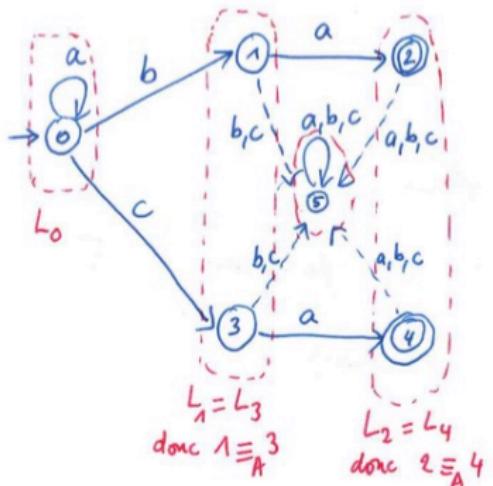


$$L_1 = L_2 = \{a, b\}^*$$

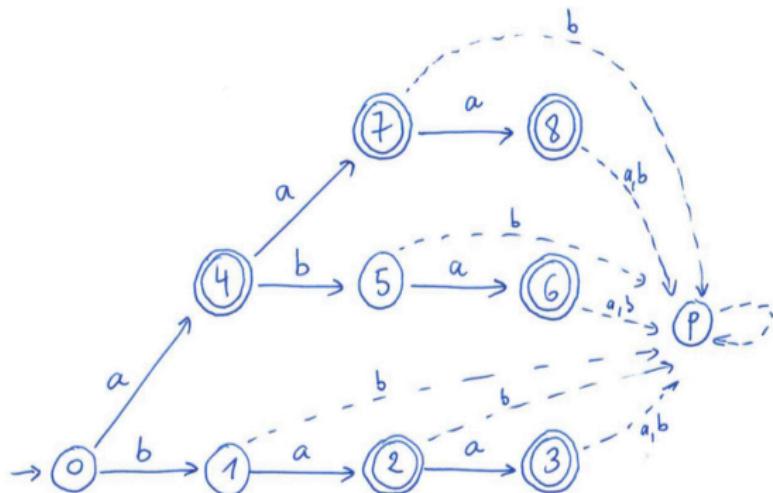


$$\begin{aligned}
 [1]_{\equiv_A} &\xrightarrow{a} [1.a]_{\equiv_A} = [2]_{\equiv_A} \\
 &= [1]_{\equiv_A}
 \end{aligned}$$

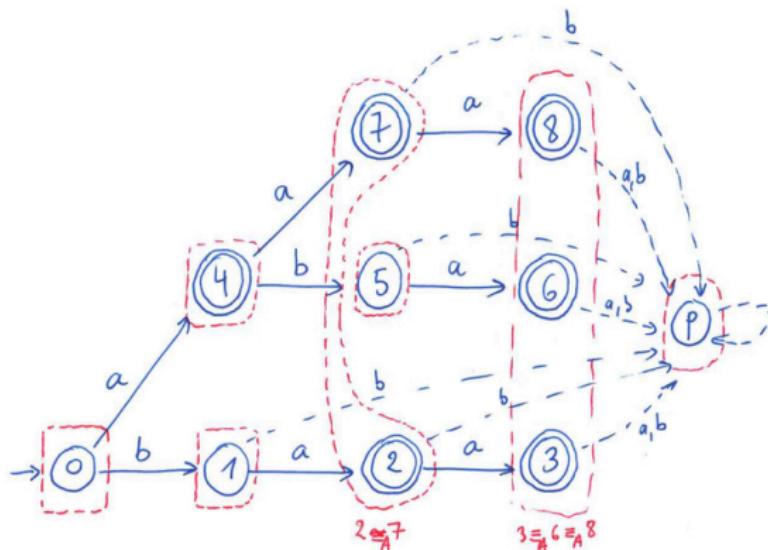
8 Exemples de calcul de l'automate minimal à partir d'un automate



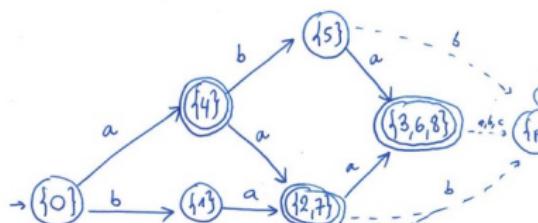
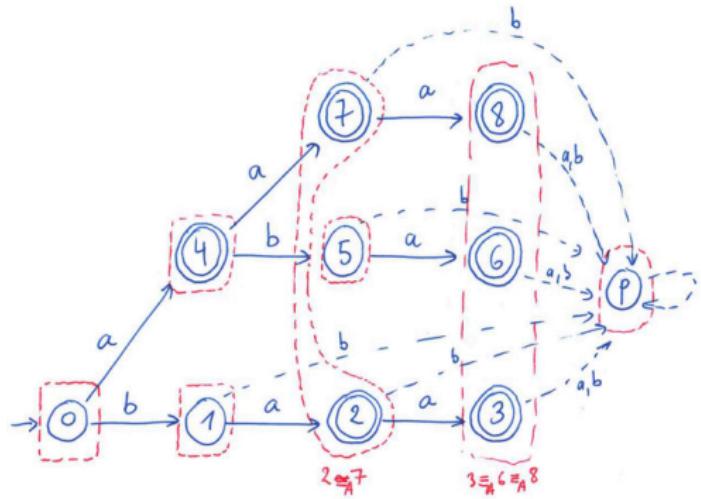
9 Exemples de calcul de l'automate minimal à partir d'un automate



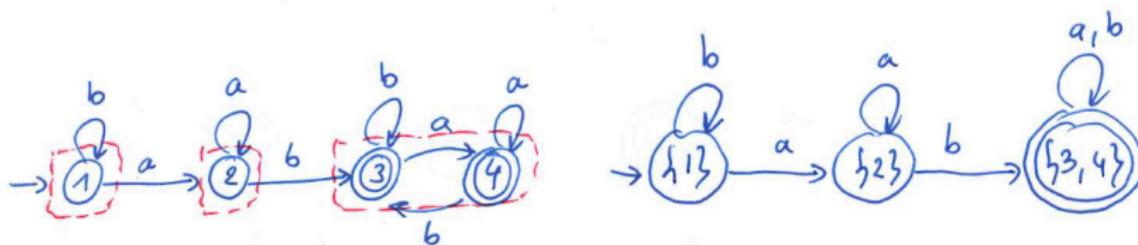
10 Exemples de calcul de l'automate minimal à partir d'un automate



11 Exemples de calcul de l'automate minimal à partir d'un automate



12 Exemples de calcul de l'automate minimal à partir d'un automate



13 En pratique ...

- ▶ pour décider si $p \equiv_A q$, on peut tester l'équivalence de l'automate A où on remplace l'état initial par p , et de l'automate A où on remplace l'état initial par q ;
- ▶ en pratique ce n'est pas très efficace, et on peut faire mieux, sans passer par le test d'équivalence, et par raffinement successif de relations d'équivalence qui convergent vers \equiv_A . Cela est même possible en $O(n \cdot \log_2 n)$ où n est le nombre d'états de l'automate.
- ▶ cette dernière méthode, non abordée dans le cours, donne aussi une autre façon efficace de tester l'équivalence de deux automates A et B : construire l'automate minimal pour $L(A)$, l'automate minimal pour $L(B)$, et tester qu'ils sont égaux (à renommage des états près).
- ▶ pour les AFNs, le problème de minimisation est plus compliqué (décider si un AFN est équivalent à un AFN avec au plus k états, où k est donné, est PSPACE-dur¹)

1. aussi "difficile" que n'importe quel problème pouvant être résolu en espace polynomial