

# Modèles de la programmation et du calcul

**L3 Informatique**

**Université de Bordeaux**

Année 2023/24

# Équipe pédagogique

- ▶ Cours : Anca Muscholl (lundi 8h00-9h20)
- ▶ A1 : Pierre Bonnet jeudi 14h-16h50
- ▶ A2 : Mikhail Raskin lundi 14h-16h50
- ▶ A3 : Vincent Penelle lundi 14h-16h50
- ▶ A4 : Anca Muscholl vendredi 9h30-12h20

# Modalités du cours

- ▶ 12 cours, 12 TD.
- ▶ Contrôle continu (CC) : 2 tests durant les séances de TD et DS le 23 octobre 2023.
- ▶ Note finale session 1 et 2 :  
1/2 Examen + 1/2 CC.
- ▶ Supports de cours et TD :  
<https://amuschol.pages.emi.u-bordeaux.fr/mpc/>

# Objectifs (fiche UE)

1. Comprendre les fondaments des **modèles de calcul** utilisés en programmation et compilation.
2. Comprendre et savoir utiliser **4 notions** :
  - a. *automates finis*,
  - b. *expressions rationnelles*,
  - c. *grammaires algébriques*,
  - d. *automates à pile*.
3. Maîtriser l'**algorithmique** de ces objets et savoir **formaliser et justifier** leurs propriétés.

# Bibliographie

- ▶ [O. Carton.](#)  
Langages formels, Calculabilité et Complexité.  
[Vuibert, 2008.](#)
- ▶ [J.M. Autebert.](#)  
Théorie des langages et des automates.  
[Masson, 1997.](#)
- ▶ [J.E. Hopcroft, R. Motwani, J. D. Ullman.](#)  
Introduction to Automata Theory, Languages & Computation.  
[Addison-Wesley, 2005.](#)

# Comment réussir l'UE ?

- ▶ **Assiduité** en cours et TD.
- ▶ Travail personnel **1h à 2h** / semaine.
- ▶ **Participation** en cours et TD.

# Comment réussir l'UE ?

- ▶ **Assiduité** en cours et TD.
- ▶ Travail personnel **1h à 2h** / semaine.
- ▶ **Participation** en cours et TD.

Questions ?

# Plan du cours

1. Mots et langages
2. Expressions rationnelles
3. Automates finis
4. Grammaires algébriques
5. Automates à pile



# Plan du cours

1. Mots et langages
2. Expressions rationnelles
3. Automates finis
4. Grammaires algébriques
5. Automates à pile
6. Logique

# Qu'est-ce qu'un langage ?

**Objectif** : comprendre, définir, manipuler, transformer des langages.

Alphabet = ensemble fini de lettres.

Mot = séquence finie de lettres d'un alphabet donné.

Langage = ensemble de mots.

## Exemples

- ▶ Alphabet  $\{A, C, G, T\}$  génétique
- ▶ Alphabet  $\{A, C, \dots, Y\}$  acides aminés (décrire des protéines)
- ▶ Alphabet binaire  $\{0, 1\}$  nombres en base 2
- ▶ Alphabet décimal  $\{0, 1, \dots, 9\}$  nombres en base 10
- ▶ Alphabet hexadécimal  $\{0, 1, \dots, 9, A, \dots, F\}$  nombres en base 16
- ▶ Mots

$$2022 = (11111100110)_2 = (7E6)_{16} = MMXXII$$

# Mots

- ▶ On écrit les mots comme  $w = a_1 \dots a_n$ .  
Chaque  $a_i$  représente une lettre d'un alphabet donné  $A$ .

**Exemples :**

- ▶ La **longueur (ou taille)** du mot  $w = a_1 \dots a_n$ , notée  $|w|$ , est  $n$ .

**Exemples :**

- ▶ Mot **vide**, noté  $\epsilon$  : séquence vide (de longueur 0)

# Exemples de langages (1)

- ▶ Tous les mots **binaires** (= mots sur l'alphabet  $A = \{0, 1\}$ ).
- ▶ Le langage **vide**  $\emptyset$ .
- ▶ Le langage  $\{011\}$ .
- ▶ Les mots sur l'alphabet  $A = \{a, b\}$  de longueur paire.
- ▶ Les mots sur l'alphabet  $\{A, C, G, T\}$  qui contiennent le motif *TAC*.

# Exemples de langages (2)

- ▶ Tous les mots qui sont des noms de variables permis dans un langage de programmation donné (compilation : analyse lexicale)
  
- ▶ Tous les mots qui représentent des programmes dans un langage de programmation donné (compilation : parsing)
  
- ▶ Tous les mots qui représentent des algorithmes de tri en C.

# Quel intérêt de s'intéresser aux mots et langages ?

Toute information numérique peut être représentée par une séquence (mot) binaire.

- ▶ Nombreuses façons pour coder en binaire.
- ▶ Codages binaires standardisés pour des ensembles de caractères : ASCII, ISO-8859-1, Unicode, UTF, JIS, ...
- ▶ Codages d'objets « numériques »

**Exemple** : arbres binaires

# Mots : définitions et exemples (1)

- ▶ Alphabet  $A$  = ensemble fini de lettres (ou symboles)
  
- ▶ Mot = séquence finie sur alphabet  $A$
  
- ▶ Langage = ensemble (fini ou infini) de mots

## Mots : définitions et exemples (2)

- ▶ Longueur  $|w|$
- ▶ Mot vide  $\epsilon$
- ▶ Concaténation ou produit de mots :  $u \cdot v$  (ou  $uv$ ) est la juxtaposition de  $u$  et  $v$ .
- ▶ La concaténation est associative :  $u(vw) = (uv)w$
- ▶ L'ensemble des mots sur l'alphabet  $A$  est noté  $A^*$ .



# Mots : définitions et exemples (3)

- ▶ Si le mot  $w$  s'écrit comme produit  $w = uv$ , alors on dit que
  - ▶  $u$  est **préfixe** de  $w$ , et
  - ▶  $v$  est **suffixe** de  $w$ .
  
- ▶ Si le mot  $w$  s'écrit comme produit  $w = xuv$ , alors on dit que  $u$  est **facteur** de  $w$ .

## Question :

Quel est le préfixe le plus court de n'importe quel mot  $w$  ? Et le plus préfixe le plus long ?

# Qu'est-ce qu'un langage ?

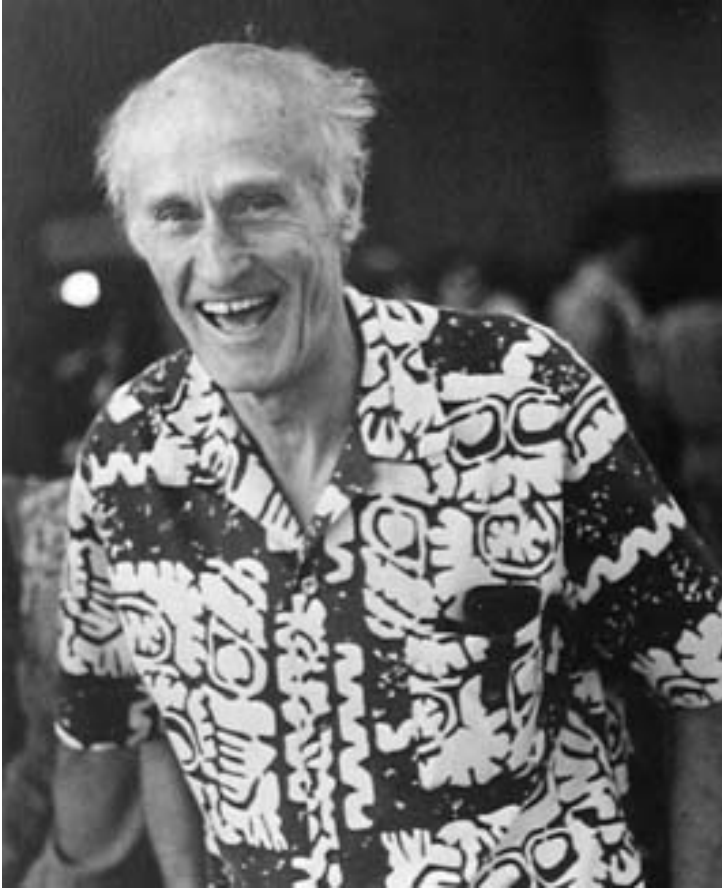
- ▶ Définition : Un langage est un **ensemble** (fini ou infini) de mots sur alphabet donné  $A$ .  
On écrit  $L \subseteq A^*$  pour désigner un langage  $L$  sur l'alphabet  $A$ .
- ▶ D'autres exemples :
  - ▶ Un nom de variable (identificateur) en Java est une suite de caractères (lettres minuscules/majuscules, chiffres, \$ ou \_) qui ne commence pas par un chiffre.
  - ▶ Une expression arithmétique est construite à partir des identificateurs en utilisant les opérateurs  $+$ ,  $-$ ,  $*$ ,  $/$  et les parenthèses  $(, )$ .

**Objectif** : définir de façon **simple** des langages **utiles**.

# Plan du cours

1. Mots et langages
2. Expressions rationnelles
3. Automates finis
4. Grammaires algébriques
5. Automates à pile
6. Logique

# Les expressions rationnelles (S. Kleene)



De nombreux langages utiles sont

construits avec 3 opérations simples.

- ▶ Union de 2 langages.
- ▶ Produit (ou concaténation) de 2 langages.
- ▶ Étoile (ou itération) d'un langage.

Contexte : proposées dans les années '50, utilisées dans les éditeurs de texte (Unix : grep, sed), le traitement automatique des langages, le développement logiciel, etc.

# Union de 2 langages

## ► Union

Un langage est un ensemble de mots.

L'**union** de 2 langages est l'union de ces deux ensembles de mots.

## Exemples

►  $L_1 = \{\epsilon, a, ab\}, L_2 = \{a, b, aab\}$

$$L_1 \cup L_2 = \{\epsilon, a, b, ab, aab\}$$

►  $L_1 = \{\epsilon, a, aa, aaa, \dots\}, L_2 = \{b, bb, bbb, \dots\}$

$L_1 \cup L_2 =$  ensemble des mots sur  $A = \{a, b\}$  constitués soit uniquement de  $a$ , ou uniquement de  $b$

# Produit (ou concaténation) de 2 langages

## ► Produit

Le **produit** des langages  $L_1, L_2$  est

$$L_1 \cdot L_2 = \{uv \mid u \in L_1, v \in L_2\}$$

## Exemples

►  $L_1 = \{\epsilon, a, ab\}, L_2 = \{a, b, aab\}$

$$L_1 \cdot L_2 = \{a, b, aab, aa, ab, aaab, aba, abb, abaab\}$$

►  $L_1 = \{\epsilon, a, aa, aaa, \dots\}, L_2 = \{\epsilon, b, bb, bbb, \dots\}$

$L_1 \cdot L_2 =$  ensemble des mots formés d'une suite de  $a$ , suivie par une suite de  $b$ .

## Questions :

Est-ce que le mot vide ( $\epsilon$ ) appartient à  $L_1 \cdot L_2$  ?

$$\emptyset \cdot L = ?$$

$$\{\epsilon\} \cdot L = ?$$

# Étoile d'un langage

## ► Étoile

On définit

$$L^0 = \{\epsilon\}, \quad L^{n+1} = L \cdot L^n$$

Question : Qui est  $L^1$  ?

L'**étoile (itération)** du langage  $L$  est définie par

$$L^* = \bigcup_{n \geq 0} L^n = L^0 \cup L^1 \cup L^2 \cup \dots$$

Question :  $\emptyset^* = ?$

De manière équivalente :

$$L^* = \{w_1 w_2 \cdots w_n \mid w_i \in L \text{ pour tout } i, n \geq 0\}$$

## Exemples

- Si  $A$  est un alphabet, alors  $A^*$  est l'ensemble des mots sur l'alphabet  $A$ .
- $L = \{\epsilon, a, ab\}$

$$L^* = \{\epsilon, a, aa, ab, aaa, aab, aba, \dots\}$$

# Les expressions rationnelles

Les **expressions atomiques** sont :  $a \in A$  (les éléments de  $A$ ),  $\varepsilon$  (le mot vide) et  $\emptyset$  (le langage vide).

Les **expressions rationnelles** sont obtenues à partir des expressions atomiques en utilisant **3 opérations** simples :

- ▶  $+$  qui représente l'union.

Si  $\mathbf{e}_1, \mathbf{e}_2$  sont des expressions, alors  $(\mathbf{e}_1 + \mathbf{e}_2)$  est une expression.

- ▶  $\cdot$  qui représente le produit.

Si  $\mathbf{e}_1, \mathbf{e}_2$  sont des expressions, alors  $(\mathbf{e}_1 \cdot \mathbf{e}_2)$  est une expression.

- ▶  $*$  qui représente l'étoile.

Si  $\mathbf{e}$  est une expression, alors  $(\mathbf{e}^*)$  est une expression.



# Expressions rationnelles - exemples

Comme pour les expressions arithmétiques, on peut omettre certaines parenthèses (ordre de priorité :  $*$   $>$   $\cdot$   $>$   $+$ ).

# Langages associées aux expressions rationnelles

A chaque expression rationnelle  $\mathbf{e}$  on associe un langage  $L(\mathbf{e})$  :

- ▶  $L(a) = \{a\}$ ,  $L(\epsilon) = \{\epsilon\}$ ,  $L(\emptyset) = \emptyset$
- ▶  $L(\mathbf{e}_1 + \mathbf{e}_2) = L(\mathbf{e}_1) \cup L(\mathbf{e}_2)$
- ▶  $L(\mathbf{e}_1 \cdot \mathbf{e}_2) = L(\mathbf{e}_1) \cdot L(\mathbf{e}_2)$
- ▶  $L(\mathbf{e}^*) = L(\mathbf{e})^*$

**Question** : pourquoi pas  
 $L(\emptyset) = \{\emptyset\}$  ?

**Remarque** : les expressions rationnelles sont de la syntaxe, et ici on définit leur sémantique.

# Expressions rationnelles : exemples (1)

- ▶ Ensemble des mots sur l'alphabet  $\{a, b\}$  qui **commencent** par **a**.
- ▶ Ensemble des mots sur l'alphabet  $\{a, b\}$  qui **contiennent** **aba**.
- ▶ Ensemble des mots sur l'alphabet  $\{a, b\}$  qui **ne contiennent pas** **ab**.

# Expressions rationnelles : exemples (2)

- ▶ Ensemble des [adresses mail valides](#).
  
- ▶ Représentations des entiers [divisibles par 2](#) (base 2).
  
- ▶ Représentations des entiers [divisibles par 5](#) (base 10).

## Expressions rationnelles : exemples (3)

- ▶ Langage des mots sur l'alphabet  $\{a, b\}$  de **longueur paire**.
  
  
  
  
  
  
  
  
  
  
- ▶ Langage des mots sur l'alphabet  $\{a, b\}$  qui ont un **nombre pair de a**.

## Expressions rationnelles : exemples (4)

- ▶ Mots sur l'alphabet  $\{0, 1\}$  qui ne contiennent ni  $00$ , ni  $11$ .
  
  
  
  
  
  
  
  
  
  
- ▶ Mots sur alphabet  $\{0, 1\}$  qui contiennent le facteur  $010$  mais pas  $101$ , et qui commencent et finissent par  $0$ .

# Expressions rationnelles : exemples (5)

- ▶ Représentations des nombres en base 10 **divisibles par 3** ?
- ▶ Nombres en base 10 qui ont **autant de 1 que de 2** ??
- ▶ Programmes C **syntactiquement corrects** ???.

# Questions

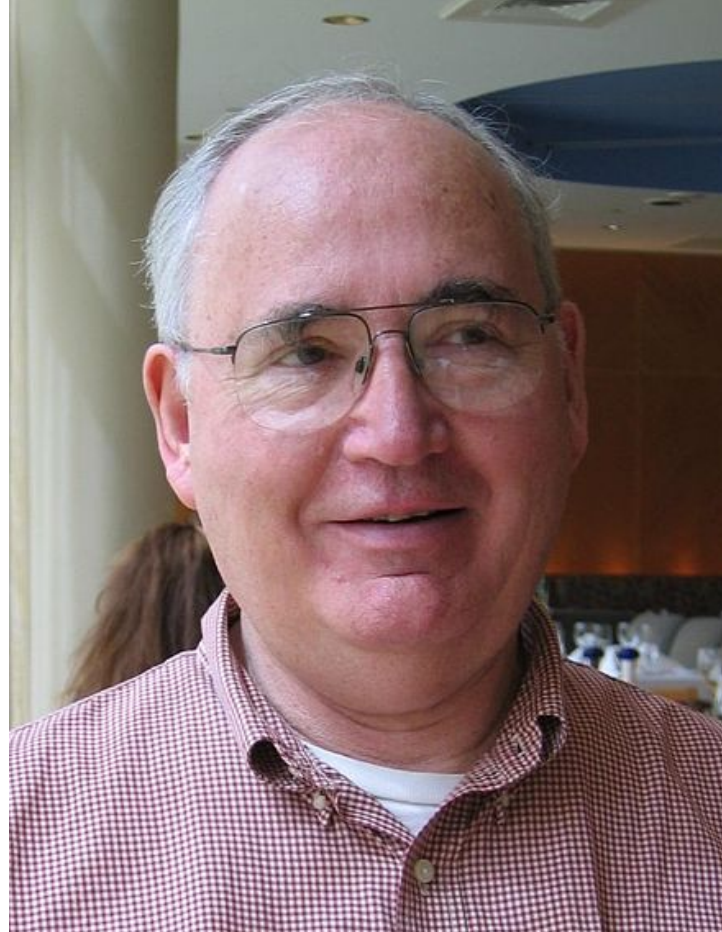
- ▶ A vous...
- ▶ **Tous** les langages sont-ils décrits par une expression rationnelle ?
- ▶ Sinon comment être sûr(e) qu'un langage n'a **aucune** expression ?
- ▶ Pourquoi est-ce important ?
- ▶ Utiliser le **complément** ou l'**intersection** permet-il d'exprimer plus ?



# Plan du cours

1. Mots et langages
2. Expressions rationnelles
3. Automates finis
4. Grammaires algébriques
5. Automates à pile
6. Logique

# Les automates finis



Michael O. Rabin,  
Dana Scott

# Les automates

## Qu'est-ce que c'est ?

Machine abstraite très simple qui permet aussi de définir des langages.

## Quelle différence par rapport aux expressions ?

- ▶ Une expression exprime **globalement** une propriété.
- ▶ Un automate exprime **localement** l'enchaînement des lettres.

On peut voir les expressions comme des propriétés (très simples) de programme, et les automates finis comme des programmes (très simples).

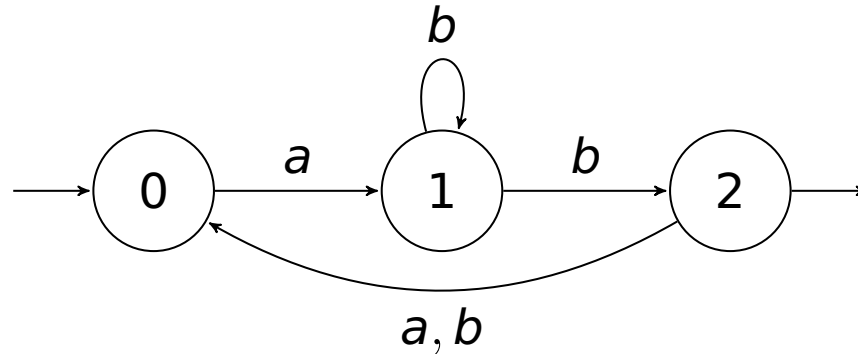
# Pourquoi les automates ?

## Quel intérêt par rapport aux expressions ?

- ▶ Ça **s'implémente** (contrairement à une expression).
- ▶ On peut passer d'une expression à un automate équivalent...
- ▶ ...et vice-versa.
- ▶ C'est un graphe : **algorithmes et logiciels disponibles.**
  - ▶ <http://www.jflap.org>
  - ▶ <http://www.cs.usfca.edu/~jbovet/vas.html>

# Automates : un exemple

- ▶ Lit un mot en entrée
- ▶ **Accepte** ou **rejette** ce mot.
- ▶ Capacités de calcul très limitées : chaque lettre lue ne peut qu'influencer une mémoire **finie** : les états.

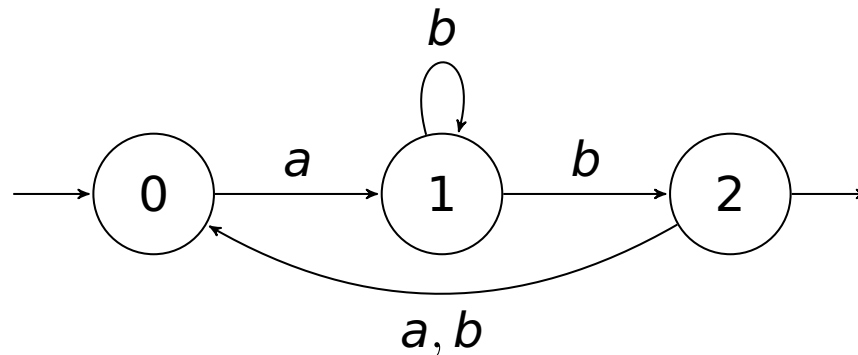


# Automates : définition

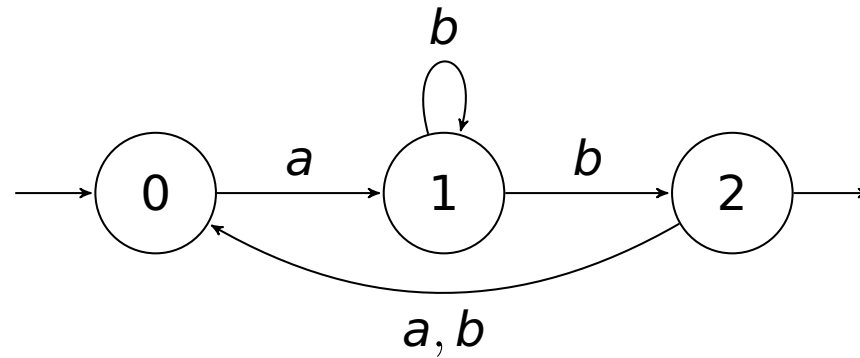
Un automate est donné par 5 ensembles :  $(A, Q, \delta, I, F)$

- ▶ Alphabet  $A$ .
- ▶ Ensemble **fini** d'états  $Q$ .
- ▶ Ensemble de transitions  $\delta$ , étiquetées sur l'alphabet  $A$ .
- ▶ États **initiaux**  $I \subseteq Q$ .
- ▶ États **finaux** (ou **acceptants**)  $F \subseteq Q$ .

flèches entrantes  
flèches sortantes

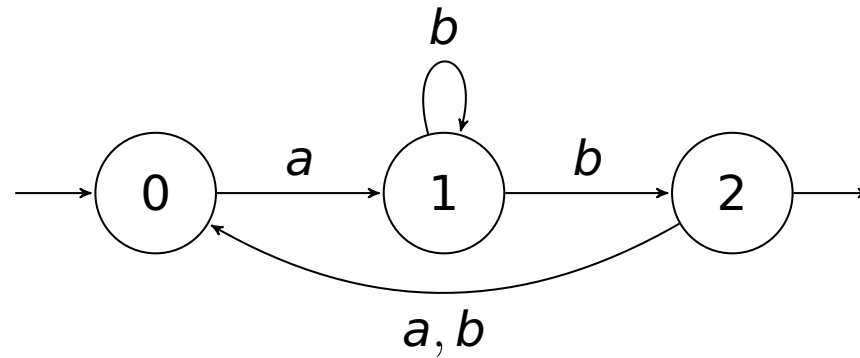


# Fonctionnement d'un automate



- ▶ Calcul sur un mot  $w$  : chemin étiqueté par  $w$  depuis un état initial.
- ▶ Mot  $w$  accepté si au moins un calcul sur  $w$  va à un état acceptant.
- ▶ Langage de l'automate : ensemble des mots acceptés.

# Fonctionnement d'un automate

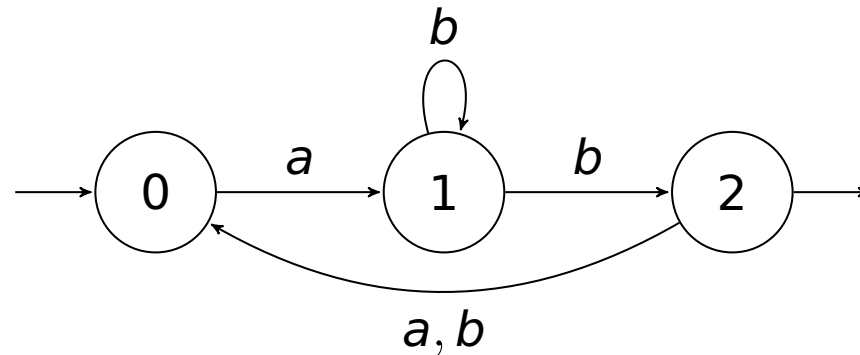


- ▶ Calcul sur un mot  $w$  : chemin étiqueté par  $w$  depuis un état initial.
- ▶ Mot  $w$  accepté si au moins un calcul sur  $w$  va à un état acceptant.
- ▶ Langage de l'automate : ensemble des mots acceptés.

Ici :  $(ab^*b(a + b))^*ab^*b$ .



# Fonctionnement d'un automate



**Remarque importante :** Il y a 2 façons de lire le mot  $abb$  dans cet automate.

- ▶ une façon mène à l'état 0, qui **n'est pas** un état final.
- ▶ une autre façon mène à l'état 2, qui est un état **final**.

Le mot est **accepté**, dès qu'il y a au moins une façon de le lire depuis un état initial jusqu'à un état final.

# Définition formelle d'un automate

Un automate est donné par  $\mathcal{A} = (A, Q, \delta, I, F)$  où

- ▶  $A$  est l'alphabet (parfois sous-entendu),
- ▶  $Q$  est l'ensemble fini des états,
- ▶  $I$  est l'ensemble des états initiaux.  
S'il n'y a qu'un état initial  $q_i$ , on peut écrire  $q_i$  au lieu de  $\{q_i\}$ .
- ▶  $F$  est l'ensemble des états finaux (ou acceptants).
- ▶  $\delta$  est l'ensemble des transitions :  $\delta \subseteq Q \times A \times Q$ .

Une transition de  $p$  à  $q$  par la lettre  $a$  peut se noter :  $(p, a, q)$  ou  $p \xrightarrow{a} q$ .

# Langage d'un automate

Un **calcul** sur un mot  $u$  est une suite de transitions **consécutives**

- ▶ partant d'un état initial,
- ▶ dont la suite des lettres des transitions est  $u = a_1 \cdots a_n$ .

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \cdots \xrightarrow{a_n} q_n$$

Ce calcul est **acceptant** si le dernier état,  $q_n$ , est **final**.

Q : quand est-ce que  $\epsilon$  est accepté ?

On dit que  $u$  est un **mot accepté par l'automate** s'il existe un calcul acceptant sur  $u$ .

Le **langage accepté** par un automate  $\mathcal{A}$  est l'ensemble des mots acceptés.

On note ce langage par  $L(\mathcal{A})$ .

# Automates : complet, déterministe

Un automate est **déterministe** si pour chaque état  $p$  et chaque lettre  $a \in A$  il a au plus un état  $q$  tel que  $p \xrightarrow{a} q$ , et  $|I| = 1$ .

Un automate est **complet** si pour chaque état  $p$  et chaque lettre  $a \in A$  il a au moins un état  $q$  tel que  $p \xrightarrow{a} q$ , et  $I \neq \emptyset$ .

Remarque : notre automate exemple n'est ni complet, ni déterministe

**Question** : pourquoi ?

# Automates : complet, déterministe

- ▶ Si un automate est déterministe, alors pour chaque mot il a **au plus** un calcul sur ce mot.
- ▶ Si un automate est complet, alors pour chaque mot il a **au moins** un calcul sur ce mot.

S'il n'est pas complet, certains mots ne peuvent pas être lus par l'automate (en particulier, ils **ne sont pas acceptés**).

On **complète** un automate  $\mathcal{A} = (A, Q, I, F, \delta)$  en rajoutant un état  $d \notin Q$  (appelé état puits), et des transitions  $(q, a, d)$  si  $\delta$  ne contient aucun triplet  $(q, a, *)$ ; on rajoute aussi  $(d, a, d)$ , pour tout  $a \in A$ .

Le langage de l'automate complété reste le même.

# Automates : exemples (1)

- ▶ Ensemble des mots sur  $\{a, b\}$  qui commencent par un  $a$ .
- ▶ Ensemble des mots sur  $\{a, b\}$  qui contiennent  $aba$ .
- ▶ Ensemble des mots sur  $\{a, b\}$  qui ne contiennent pas  $ab$ .

# Automates : exemples (2)

- ▶ Ensemble des **identificateurs** en C.
  
- ▶ Représentations des entiers **divisibles par 2** (base 2).
  
- ▶ Représentations des entiers **divisibles par 5** (base 10).

# Automates : exemples (3)

- ▶ Ensemble des mots sur  $\{a, b\}$  qui **contiennent** un nombre pair de **a**.
- ▶ Sur alphabet  $\{0, 1\}$ , mots qui contiennent le motif (facteur) **010** mais pas **101**.
- ▶ Mots qui **ne contiennent** ni **00**, ni **11**.

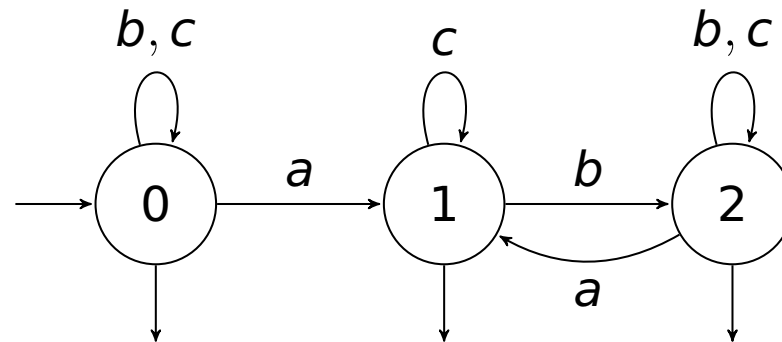


# Automates : exemples (4)

- ▶ Représentations des entiers **divisibles par 3** (base 2) ??
- ▶ Nombres en base 10 qui ont **autant de 1 que de 2** ???
- ▶ Programmes C **syntactiquement corrects** ????.

# Automates : exemples

$\mathcal{A}$  :



On écrit  $p \xrightarrow{w} q$  s'il existe au moins un calcul sur le mot  $w$  qui part de l'état  $p$  et qui finit dans l'état  $q$ .

**Exemple :**  $L(\mathcal{A}) = \{w \in A^* \mid p \xrightarrow{w} q \text{ t.q. } p \in I, q \in F\}$ .

# **Transformer un automate en expression rationnelle**

**... et inversement**

# Transformer un automate en expression

## Intérêt ?

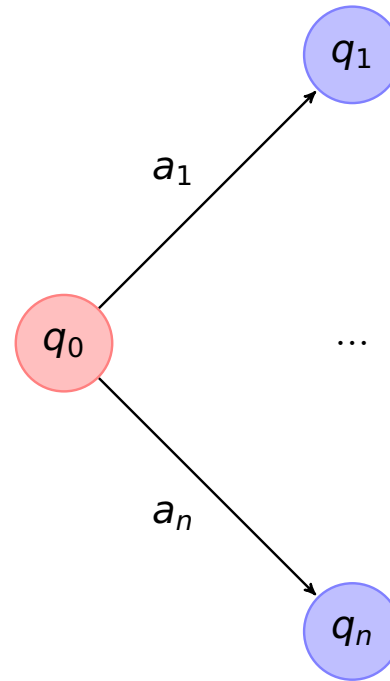
- ▶ Permettra de prouver que les automates et les expression rationnelles définissent **les mêmes langages**.

Plusieurs **algorithmes** classiques :

- ▶ McNaughton-Yamada      Similaire à **Floyd-Warshall**.
- ▶ Brzozowski-McCluskey.
- ▶ **Équations**, méthode basée sur le **Lemme d'Arden**.

# D'un automate à une expression équivalente

- ▶  $X_k$  = langage des mots acceptés en prenant  $q_k$  comme état initial.
- ▶ Si depuis  $q_0$ , on a les transitions



$$\text{alors } X_0 = \begin{cases} a_1 X_1 + \cdots + a_n X_n + \epsilon & \text{si } q_0 \text{ est } \mathbf{final} \\ a_1 X_1 + \cdots + a_n X_n & \text{sinon} \end{cases}$$

# Lemme d'Arden

A partir d'un automate, on obtient donc des “équations du 1er degré”.

**Lemme d'Arden.** Si  $U$ ,  $V$  et  $X$  sont des langages tels que :

$$\begin{aligned}\varepsilon &\notin U \\ X &= UX + V\end{aligned}$$

alors, le seul langage  $X$  solution est

$$X = U^*V$$

# Lemme d'Arden

On montre que

- ▶ Toute solution  $X$  de  $X = UX + V$  contient le langage  $U^*V$ .
- ▶ Si  $\epsilon \notin U$  alors  $U^*V$  est la plus grande solution de  $X = UX + V$ .

Rq :  $UX \subseteq X$  et  $V \subseteq X$

Rq : plus grande par rapport à  $\subseteq$

# Lemme d'Arden

On montre que

- ▶ Toute solution  $X$  de  $X = UX + V$  contient le langage  $U^*V$ .
- ▶ Si  $\epsilon \notin U$  alors  $U^*V$  est la plus grande solution de  $X = UX + V$ .

Rq :  $UX \subseteq X$  et  $V \subseteq X$

Rq : plus grande par rapport à  $\subseteq$

Toute solution  $X$  contient  $U^nV$ , pour tout  $n \geq 0$  : récurrence sur  $n$

- ▶  $n = 0$  :  $V \subseteq X$     ok
- ▶ Si  $U^nV \subseteq X$  alors  $U^{n+1}V = UU^nV \subseteq UX \subseteq X$     ok



# Lemme d'Arden

On montre que

- ▶ Toute solution  $X$  de  $X = UX + V$  contient le langage  $U^*V$ .
- ▶ Si  $\epsilon \notin U$  alors  $U^*V$  est la **plus grande** solution de  $X = UX + V$ .

**Rq :** plus grande par rapport à  $\subseteq$

# Lemme d'Arden

On montre que

▶ Toute solution  $X$  de  $X = UX + V$  contient le langage  $U^*V$ .

▶ Si  $\epsilon \notin U$  alors  $U^*V$  est la **plus grande** solution de  $X = UX + V$ .

**Rq :** plus grande par rapport à  $\subseteq$

Soit  $X$  une solution de  $X = UX + V$  et  $w \in X$ . Deux cas possibles :

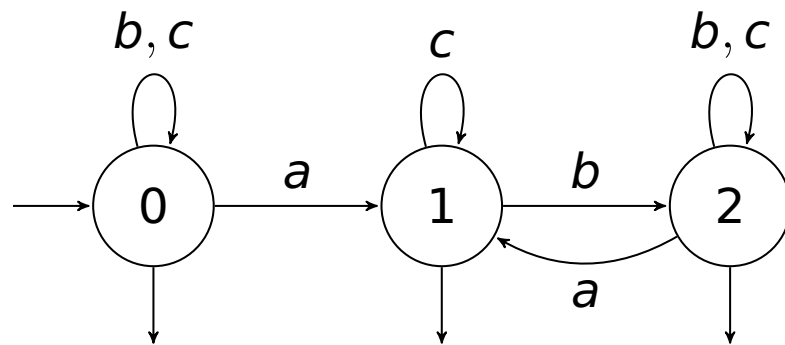
1. Soit  $w \in V$ , donc  $w \in U^*V$ ,
2. Ou on écrit  $w = uv$ , avec  $u \in U, v \in X$ .

Comme  $\epsilon \notin U$  on a  $|v| < |w|$ .

En appliquant une récurrence sur  $|w|$  on déduit que  $v \in U^*V$ .

Donc  $w = uv \in U^*V$ .

# D'un automate à une expression : exemple



$$X_0 = (b + c)X_0 + aX_1 + \epsilon$$

$$X_1 = cX_1 + bX_2 + \epsilon$$

$$X_2 = (b + c)X_2 + aX_1 + \epsilon$$

On commence par appliquer Arden à la dernière équation :

$$X_2 = (b + c)^*(aX_1 + \epsilon)$$

On substitue cette dernière expression dans la deuxième équation et on obtient :

$$\begin{aligned} X_1 &= cX_1 + b(b + c)^*(aX_1 + \epsilon) + \epsilon \\ &= (c + b(b + c)^*a)X_1 + b(b + c)^* + \epsilon \end{aligned}$$

On applique Arden sur la dernière équation et obtient :

$$X_1 = (c + b(b + c)^*a)^*(b(b + c)^* + \epsilon)$$

On substitue la dernière expression dans la première équation et on obtient :

$$X_0 = (b + c)X_0 + a(c + b(b + c)^*a)^*(b(b + c)^* + \epsilon) + \epsilon$$

Une dernière application de Arden donne :

$$L(\mathcal{A}) = X_0 = (b + c)^*(a(c + b(b + c)^*a)^*(b(b + c)^* + \epsilon) + \epsilon)$$

# Transformer une expression en automate

**Intérêt ?**

# Transformer une expression en automate

## Intérêt ?

1. Analyse lexicale !
2. On peut manipuler les automates par des algorithmes.

# Transformer une expression en automate

## Intérêt ?

1. Analyse lexicale !
2. On peut manipuler les automates par des **algorithmes**.

**Exemple** on verra comment calculer un automate qui reconnaît

- ▶ le **complémentaire** d'un langage reconnu par un automate.
- ▶ l'**intersection** de 2 tels langages.

Ces opérations ne sont **pas évidentes** sur les expressions.

## Inversement

- ▶ on peut transformer les automates en expressions équivalentes.

# Automates : union et intersection

Idée : on « synchronise » deux calculs

Produit de deux automates  $\mathcal{A}_1 = (A, Q_1, I_1, F_1, \delta_1)$ ,  $\mathcal{A}_2 = (A, Q_2, I_2, F_2, \delta_2)$  :

automate  $\mathcal{A}_1 \times \mathcal{A}_2 = (A, Q_1 \times Q_2, I_1 \times I_2, F, \delta)$

- ▶  $(q_1, q_2) \xrightarrow{a} (q'_1, q'_2)$  si  
 $q_1 \xrightarrow{a} q'_1$  dans  $\mathcal{A}_1$ , et  $q_2 \xrightarrow{a} q'_2$  dans  $\mathcal{A}_2$

Le langage de l'automate produit  $\mathcal{A}_1 \times \mathcal{A}_2$  est

- ▶  $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$  avec  $F = F_1 \times F_2$
- ▶  $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$  avec  $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$  si  $\mathcal{A}_1$  et  $\mathcal{A}_2$  sont complets

# Automates : complémentation

Le **complémentaire** d'un langage  $L \subseteq A^*$  est le langage  $L^{co} = A^* \setminus L$ .

**Exemples :**  $\emptyset^{co} = A^*$        $((ab)^*)^{co} = b(a+b)^* + (a+b)^*a + (a+b)^*(aa+bb)(a+b)^*$

Si un automate  $\mathcal{A} = (A, Q, I, F, \delta)$  est **déterministe et complet**, alors on obtient un automate qui accepte  $L(\mathcal{A})^{co}$  en **remplaçant**  $F$  par  $Q \setminus F$  :

$$\begin{aligned}\mathcal{A}' &= (A, Q, I, Q \setminus F, \delta) \\ L(\mathcal{A}') &= (L(\mathcal{A}))^{co}\end{aligned}$$

**Question :** est-il nécessaire de demander « déterministe » ? et « complet » ?



# Transformer une expression en automate

Premier algorithme : Algorithme de Thomson.

## Principe

- ▶ Induction sur la structure des expressions.
- ▶ On construit des automates pour  $a, b, \dots, \varepsilon$ .
- ▶ Supposant qu'on sait faire pour  $e_1$  et  $e_2$ , on le fait pour  $e_1 \cdot e_2$ ,  $e_1 + e_2$ ,  $e_1^*$ .
- ▶ On utilise des transitions  $\varepsilon$ .

# Automates avec transitions $\epsilon$

Automate avec transitions  $\epsilon$  :

$$\mathcal{A} = (A, Q, I, F, \delta) \quad \delta \subseteq Q \times (A \cup \{\epsilon\}) \times Q$$

**Calcul** sur un mot  $w$  : chemin étiqueté par  $w$  depuis un état initial.

Différence : certaines transitions sont étiquetées par  $\epsilon$ . Mais si on concatène toutes les étiquettes du chemin, on obtient  $w$ .

# Automates avec transitions $\epsilon$

Automate avec transitions  $\epsilon$  :

$$\mathcal{A} = (A, Q, I, F, \delta) \quad \delta \subseteq Q \times (A \cup \{\epsilon\}) \times Q$$

Calcul sur un mot  $w$  : chemin étiqueté par  $w$  depuis un état initial.

Différence : certaines transitions sont étiquetées par  $\epsilon$ . Mais si on concatène toutes les étiquettes du chemin, on obtient  $w$ .

**Elimination des transitions  $\epsilon$ .** Tout automate  $\mathcal{A}$  avec transitions  $\epsilon$  peut être transformé en un automate équivalent  $\mathcal{A}'$  sans transitions  $\epsilon$ .

- ▶ On rajoute une transition  $p \xrightarrow{a} q$  chaque fois qu'on a un chemin

$$p \xrightarrow{\epsilon} p_1 \xrightarrow{\epsilon} p_2 \cdots \xrightarrow{\epsilon} p_k \xrightarrow{a} q$$

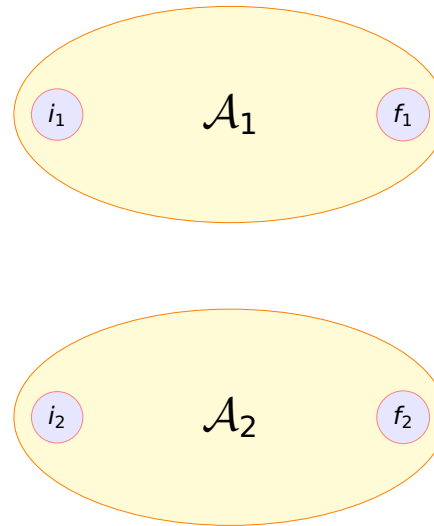
- ▶ On rend un état  $p$  acceptant chaque fois qu'on a un état acceptant  $q$  et un chemin

$$p \xrightarrow{\epsilon} p_1 \cdots \xrightarrow{\epsilon} p_k = q$$

- ▶ On supprime toutes les transitions  $\epsilon$ .

# Algorithme de Thomson

► Union.

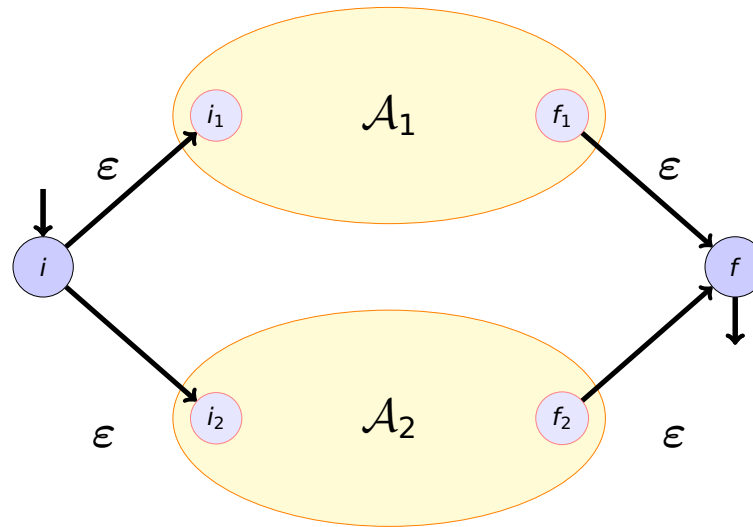


► Produit

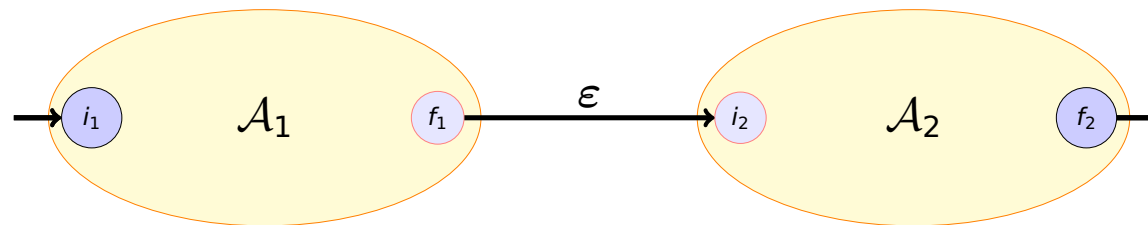


# Algorithme de Thomson

## ► Union.

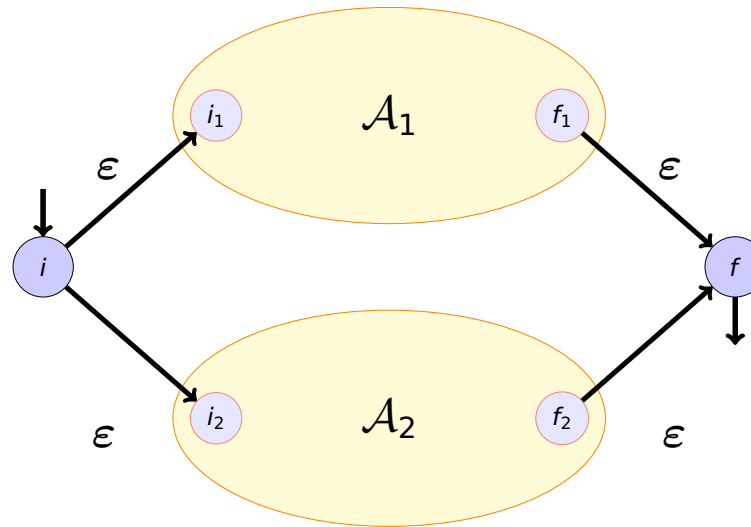


## ► Produit

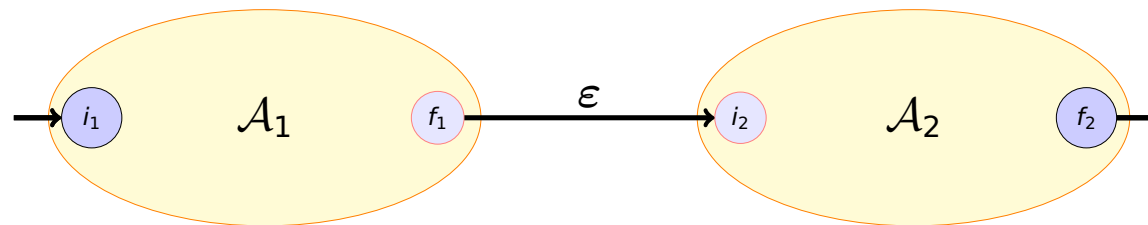


# Algorithme de Thomson

## ► Union.



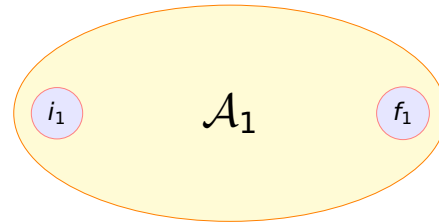
## ► Produit



Question : est-ce que c'est correct de juste identifier les états finaux du premier automate avec les états initiaux du deuxième ? pourquoi pas ?

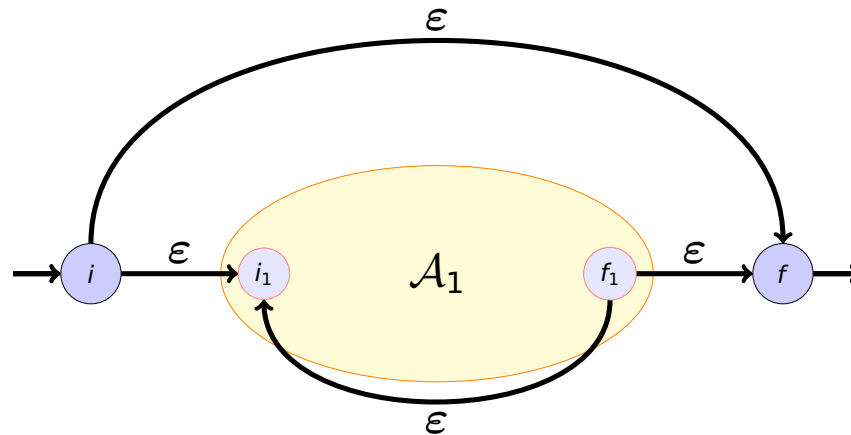
# Algorithme de Thomson

► Étoile.



# Algorithme de Thomson

- ▶ Étoile.



- ▶ On part d'automates à 1 ou 2 états pour expressions atomiques  $a$ ,  $\epsilon$ ,  $\emptyset$ .
- ▶ La construction utilise des transitions  $\epsilon$  (qu'on peut supprimer ensuite).
- ▶ On assure d'avoir un unique état initial et un unique état final.



# Algorithme de Glushkov



La construction de Thomson produit (beaucoup) de transitions  $\epsilon$ .

## Idée de Glushkov

L'automate mémorise la **position de l'expression** à laquelle on peut être.

On commence par **renommer** les lettres pour avoir des noms uniques.

**Nom = position**. Par exemple :

$$a(ab + b)^*b \quad \hookrightarrow \quad a_1(a_2b_1 + b_2)^*b_3.$$

# Algorithme de Glushkov



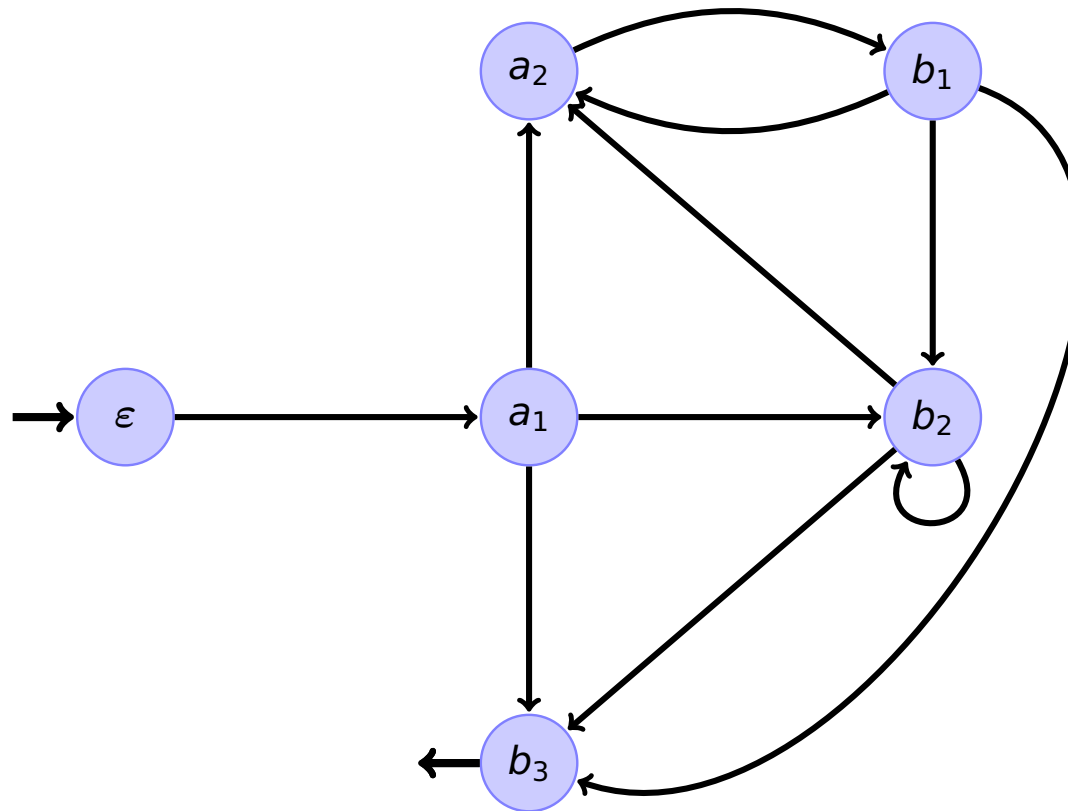
Chaque état correspond à une des (nouvelles lettres)

On a un état supplémentaire :  $\epsilon$ .

L'automate calcule de façon incrémentale l'état suivant.

# Algorithme de Glushkov : Exemple

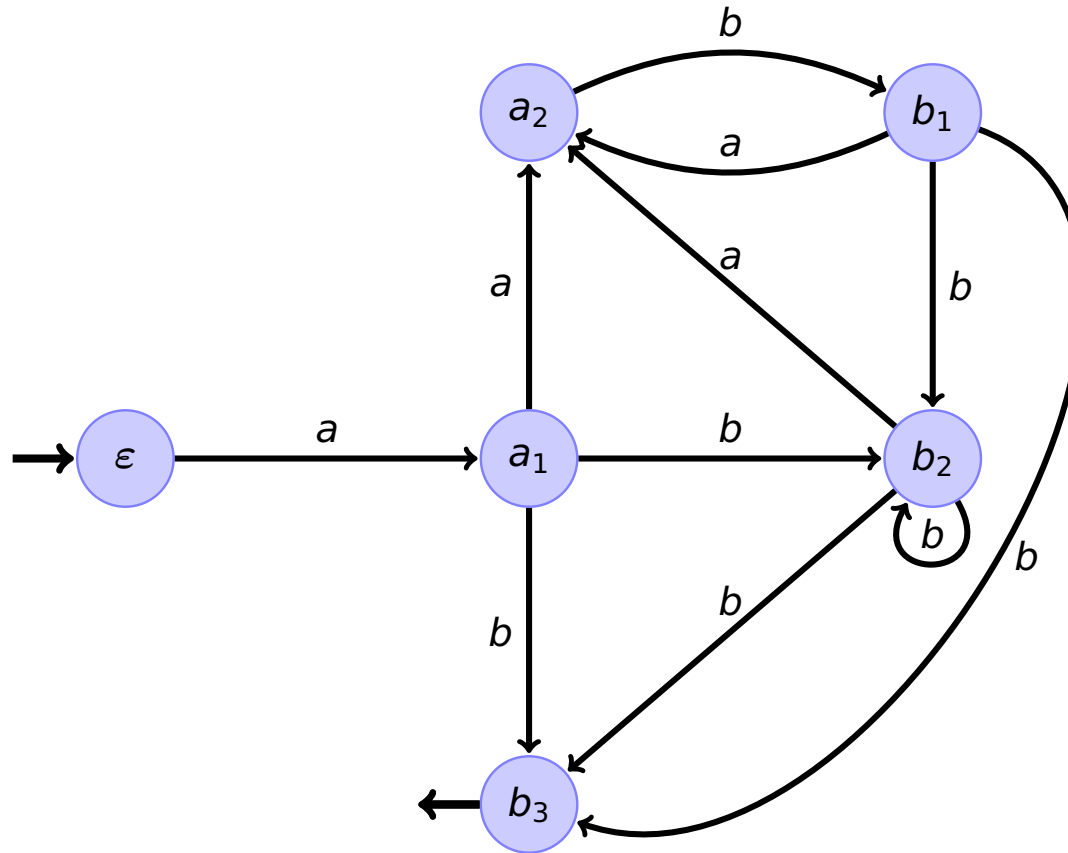
$$a(ab + b)^*b \quad \hookrightarrow \quad a_1(a_2b_1 + b_2)^*b_3.$$



- Les transitions allant à un état sont étiquetées par la lettre de l'état.

# Algorithme de Glushkov : Exemple

$$a(ab + b)^*b \quad \hookrightarrow \quad a_1(a_2b_1 + b_2)^*b_3.$$



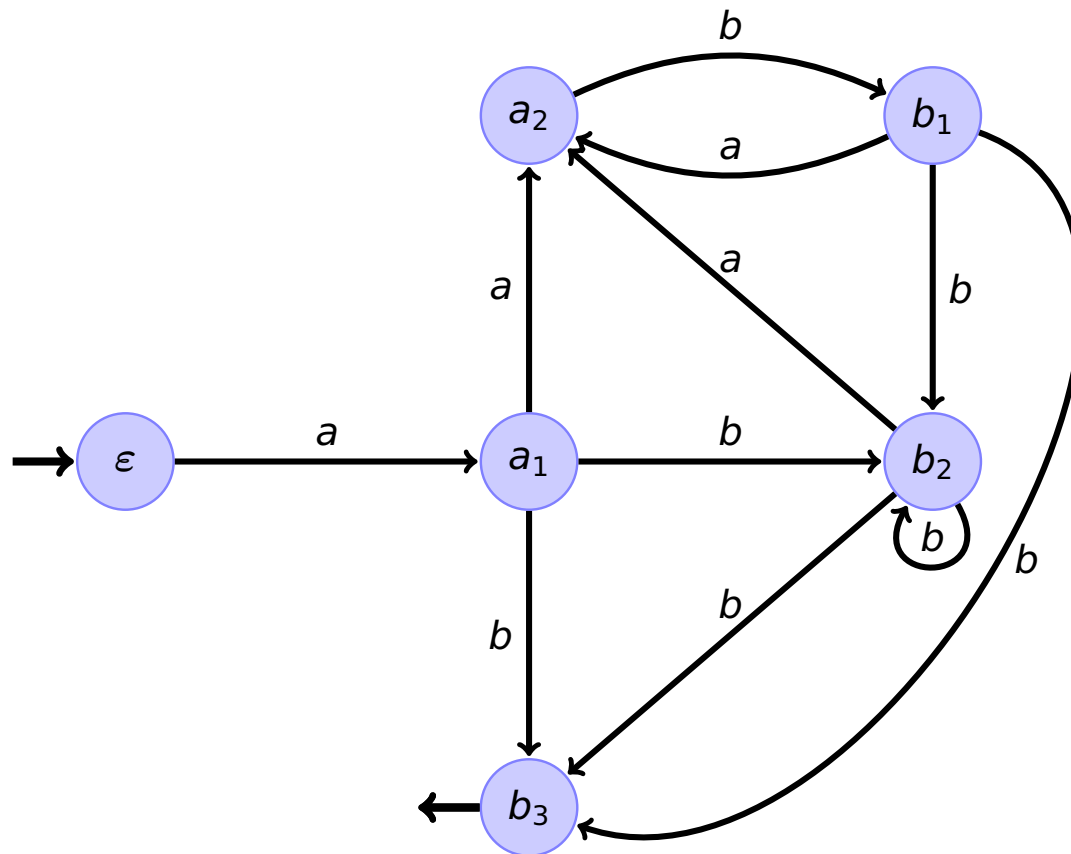
- Les transitions allant à un état sont étiquetées par la lettre de l'état.

# Algorithme de Glushkov : idée

- ▶ L'état  $\varepsilon$  est le seul initial.
- ▶ Les transitions depuis  $\varepsilon$  vont vers les états dont les lettres **peuvent commencer** un mot du langage.
- ▶ Les états finaux sont ceux étiquetés par les lettres qui **peuvent terminer** un mot du langage, ainsi que  $\varepsilon$  s'il est dans le langage.
- ▶ Il y a une transition de l'état  $a_i$  vers l'état  $b_k$  si un mot du langage **contient le facteur**  $a_i b_k$ .

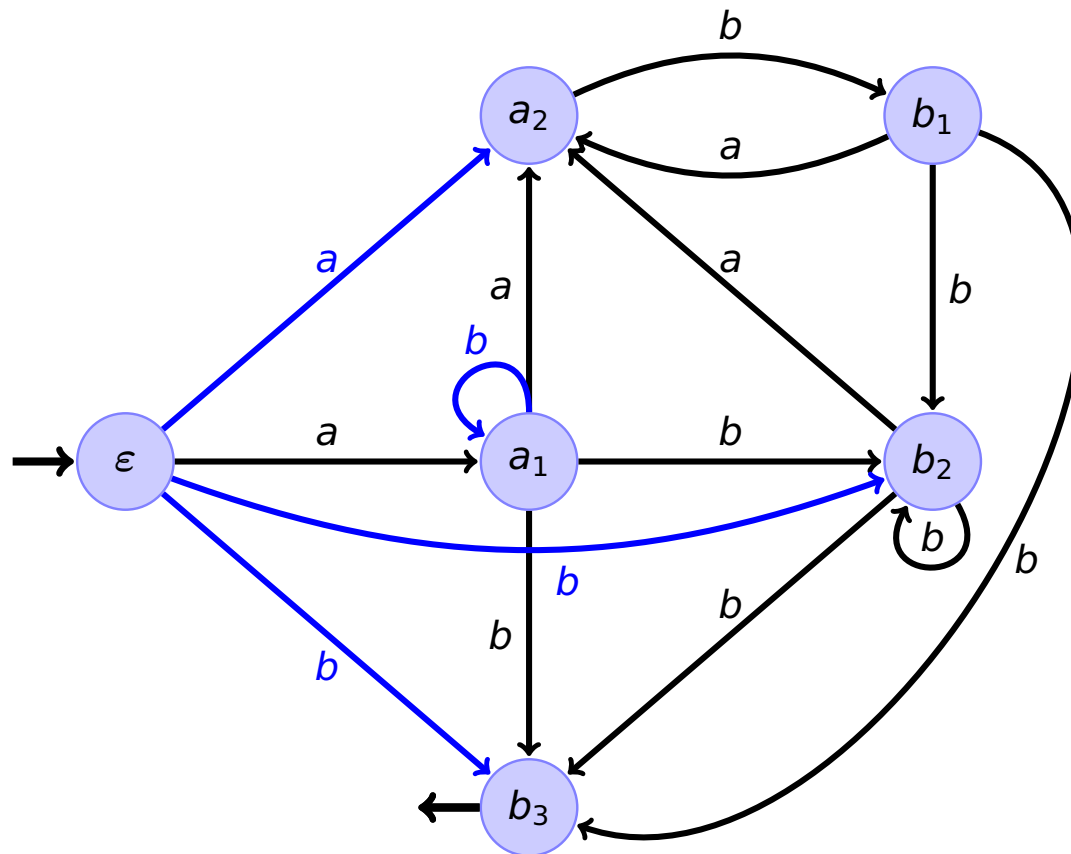
# Algorithme de Glushkov : **Exemple**

$$a(ab + b)^*b \quad \hookrightarrow \quad a_1(a_2b_1 + b_2)^*b_3.$$



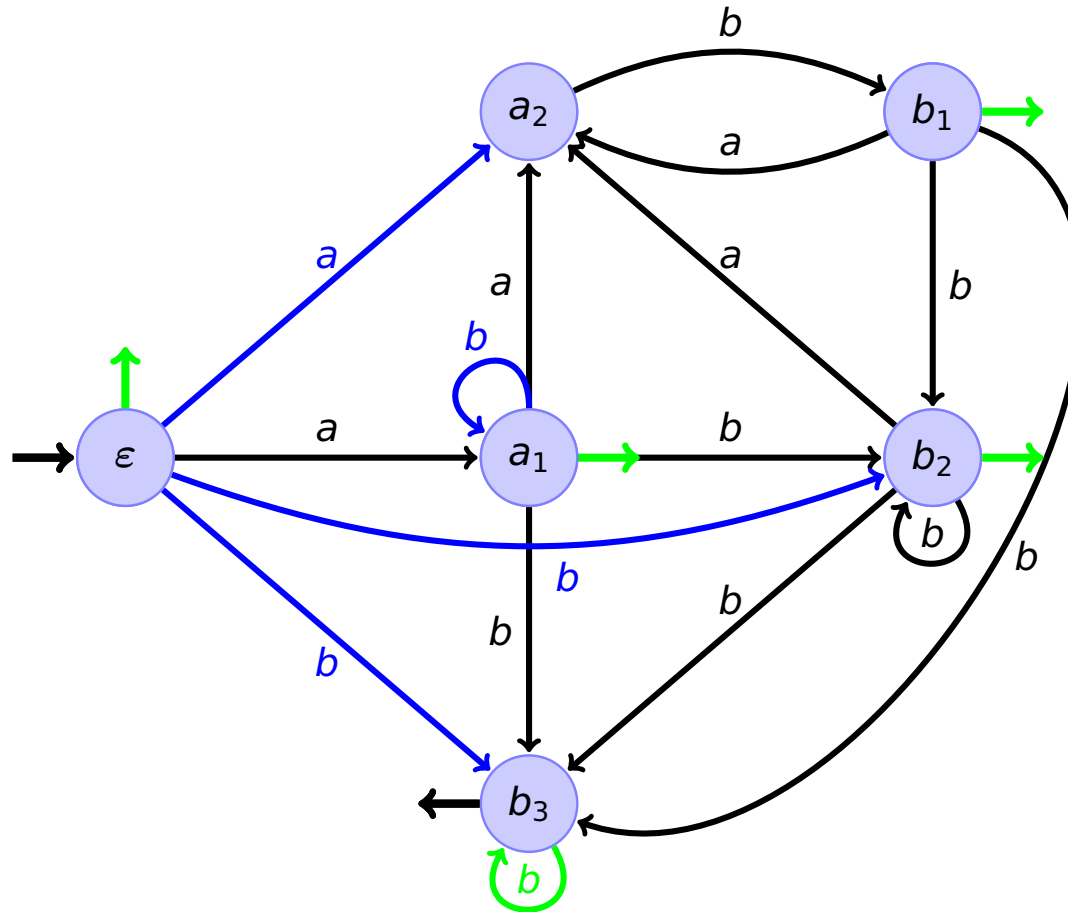
# Algorithme de Glushkov : Exemple

$$a^*(ab + b)^*b \quad \hookrightarrow \quad a_1^*(a_2b_1 + b_2)^*b_3.$$



# Algorithme de Glushkov : Exemple

$$a^*(ab + b)^*b^* \quad \hookrightarrow \quad a_1^*(a_2b_1 + b_2)^*b_3^*.$$





# Les sous-expressions effaçables

Algorithme récursif pour déterminer les expressions qui “contiennent”  $\varepsilon$ .  
On note  $\text{Effaçable}(E)$  si l'expression  $E$  génère  $\varepsilon$ .

$$\text{Effaçable}(\varepsilon) = \text{True}$$

$$\text{Effaçable}(a) = \text{False}$$

$$\text{Effaçable}(E_1 + E_2) = \text{Effaçable}(E_1) \vee \text{Effaçable}(E_2)$$

$$\text{Effaçable}(E_1 \cdot E_2) = \text{Effaçable}(E_1) \wedge \text{Effaçable}(E_2)$$

$$\text{Effaçable}(E^*) = \text{True}.$$

# Les premières lettres

Algorithme récursif pour déterminer les **premières** lettres possibles.

$$\text{Premier}(E) \stackrel{\text{def}}{=} \{a \in A \mid \exists u, au \in \mathcal{L}(E)\}$$

**Exemple**  $\text{Premier}(a^*b^*cd^*) = \{a, b, c\}$ .

$$\text{Premier}(\varepsilon) = \emptyset$$

$$\text{Premier}(a) = \{a\}$$

$$\text{Premier}(E_1 + E_2) = \text{Premier}(E_1) \cup \text{Premier}(E_2)$$

$$\text{Premier}(E_1E_2) = \begin{cases} \text{Premier}(E_1) \cup \text{Premier}(E_2) & \text{si Effaçable}(E_1) \\ \text{Premier}(E_1) & \text{sinon} \end{cases}$$

$$\text{Premier}(E^*) = \text{Premier}(E)$$

# Les transitions

Algorithme récursif pour déterminer les **dernières** lettres : idem.

Algorithme récursif pour déterminer les **transitions** :

$$\text{Trans}(E) \stackrel{\text{def}}{=} \{ab \mid ab \text{ est facteur d'un mot de } \mathcal{L}(E)\}$$

**Exemple**  $\text{Trans}((a^+b)^*) = \{aa, ab, ba\}$ .

$$\text{Trans}(\varepsilon) = \text{Trans}(a) = \emptyset$$

$$\text{Trans}(E_1 + E_2) = \text{Trans}(E_1) \cup \text{Trans}(E_2)$$

$$\text{Trans}(E_1E_2) = \text{Trans}(E_1) \cup \text{Trans}(E_2) \cup \text{Dernier}(E_1) \cdot \text{Premier}(E_2)$$

$$\text{Trans}(E^*) = \text{Trans}(E) \cup \text{Dernier}(E) \cdot \text{Premier}(E)$$

# Automates et expressions : langages réguliers

**Résumé** : 2 algorithmes “inverses” l’un de l’autre :

- ▶ Celui basé sur les équations et le lemme d’**Arden** : automate  $\rightarrow$  expression.
- ▶ **Glushkov** : expression  $\rightarrow$  automate (**non déterministe**).

**Conclusion** : les automates et les expressions rationnelles permettent d’exprimer les mêmes langages : **les langages réguliers** (appelés aussi **rationnels**).

Un langage  $L \subseteq A^*$  est **régulier** s’il existe un automate fini  $\mathcal{A}$  qui l’accepte ( $L = L(\mathcal{A})$ ), ou, de manière équivalente, s’il existe une expression régulière  $E$  qui le décrit ( $L = L(E)$ ).

# Opérations booléennes sur les automates : Complémentaire

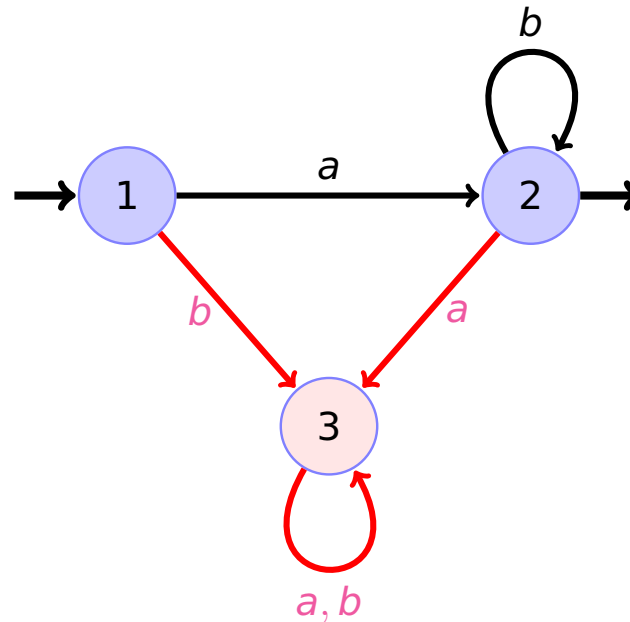
# Le complémentaire

Problème pour compléter en échangeant final  $\leftrightarrow$  non-final :

- ▶ certains mots ne peuvent pas être lus.
- ▶ L'automate n'est **pas complet**.

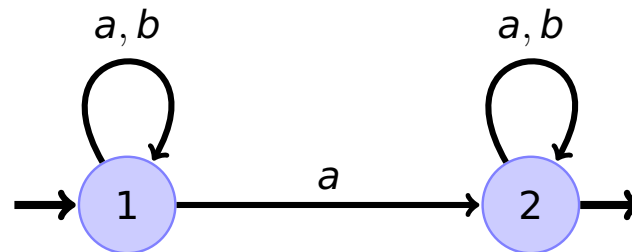
Facile à corriger : ajouter

- ▶ un nouvel état **non final**.
- ▶ les transitions manquantes vers cet état.



# Le complémentaire

- ▶ **Note** : Répéter la complétion ne change plus l'automate.
- ▶ **Question** : Avoir un automate complet suffit-il pour que l'échange final  $\leftrightarrow$  non-final fonctionne pour la complémentation ?



# La déterminisation

## La construction par sous-ensemble

- ▶ permet de construire un automate déterministe équivalent.
- ▶ Partant d'un automate à  $n$  états, le nombre d'état de l'automate déterminisé est au pire  $2^n$ .
- ▶ Cette borne peut être atteinte.



# Déterminisation

A partir d'un automate (non-déterministe)  $\mathcal{A} = (A, Q, I, F, \delta)$  on construit l'automate « des sous-parties »  $\mathcal{B} = (A, \mathcal{P}(Q), \{I\}, \mathcal{F}, \Delta)$  :

▶ les états sont les sous-ensembles des états de  $\mathcal{A}$ .

$$\mathcal{P}(Q) = \{X \mid X \subseteq Q\}$$

▶ l'état initial est l'ensemble  $I$

▶  $\mathcal{F} = \{X \subseteq Q \mid X \cap F \neq \emptyset\}$

▶  $X \xrightarrow{a} Y$  si

$$Y = \{q \in Q \mid p \xrightarrow{a} q \text{ pour un } p \in X\}$$

Dans l'automate « des sous-parties »  $\mathcal{B}$  on a :

On calcule l'ensemble des états atteignables en lisant un mot  $w$

$X \xrightarrow{w} Y$  si et seulement si  $Y = \{q \in Q \mid p \xrightarrow{w} q \text{ pour un } p \in X\}$ .

$$L(\mathcal{B}) = \{w \in A^* \mid I \xrightarrow{w} X \text{ pour un } X \in \mathcal{F}\} = L(\mathcal{A})$$

Pourquoi  $L(\mathcal{B}) = L(\mathcal{A})$  ?

# Déterminisation : exemple

## Question

Quand est-ce qu'un langage est régulier ?

- ▶ Montrer qu'un langage **est** régulier : **facile**.  
On donne un automate.
- ▶ Montrer qu'un langage **n'est pas** régulier : **pas évident**.  
On ne peut pas passer en revue tous les automates.

# Les résiduels

## Question

Quand est-ce qu'un langage est régulier ?

- ▶ Montrer qu'un langage **est** régulier : **facile**.  
On donne un automate.
- ▶ Montrer qu'un langage **n'est pas** régulier : **pas évident**.  
On ne peut pas passer en revue tous les automates.

**Solution (I) : les résiduels**

# Résiduels

$L \subseteq A^*$  langage,  $w \in A^*$  mot.

$$w^{-1}L = \{v \in A^* \mid wv \in L\}$$

$w^{-1}L$  s'appelle « résiduel de  $L$  par  $w$  »

Exemple :

$$a^{-1}(a^*b^*) =$$

$$b^{-1}(a^*b^*) =$$

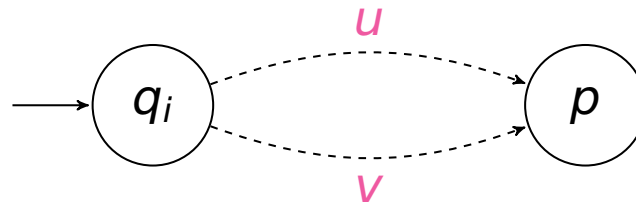
$$c^{-1}(a^*b^*) =$$

Question :  $\epsilon^{-1}L = ?$

Rq : un résiduel est un langage

# Régularité et résiduels

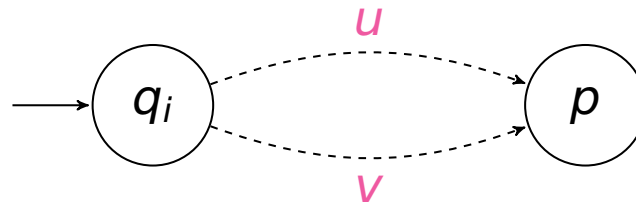
Soit  $\mathcal{A} = (A, Q, \{q_i\}, F, \delta)$  un automate **déterministe**, qui accepte le langage  $L$ .



**Question :** Que sait-on sur les résiduels  $u^{-1}L$  et  $v^{-1}L$  ?

# Régularité et résiduels

Soit  $\mathcal{A} = (A, Q, \{q_i\}, F, \delta)$  un automate **déterministe**, qui accepte le langage  $L$ .



**Question :** Que sait-on sur les résiduels  $u^{-1}L$  et  $v^{-1}L$  ?

$$u^{-1}L = v^{-1}L$$

Tout langage régulier a un nombre fini de résiduels.

**Q :** combien ?

# L'automate minimal

## Question

2 automates/expressions représentent-ils **le même langage** ?

## Solution

Objet “canonique” : un automate qui **ne dépend que du langage**.



# Automate minimal : principe

On se donne un automate  $\mathcal{A} = (A, Q, q_I, F, \delta)$  **déterministe**.

- ▶ On note  $L_q$  le langage des mots acceptés à partir de l'état  $q$ .
- ▶ Si  $L_p = L_q$ , on note  $p \sim q$  ( $p$  **équivalent** à  $q$ ).
- ▶ Automate **minimal** : obtenu en **identifiant** les états équivalents.

# Automate minimal : principe

On se donne un automate  $\mathcal{A} = (A, Q, q_I, F, \delta)$  **déterministe**.

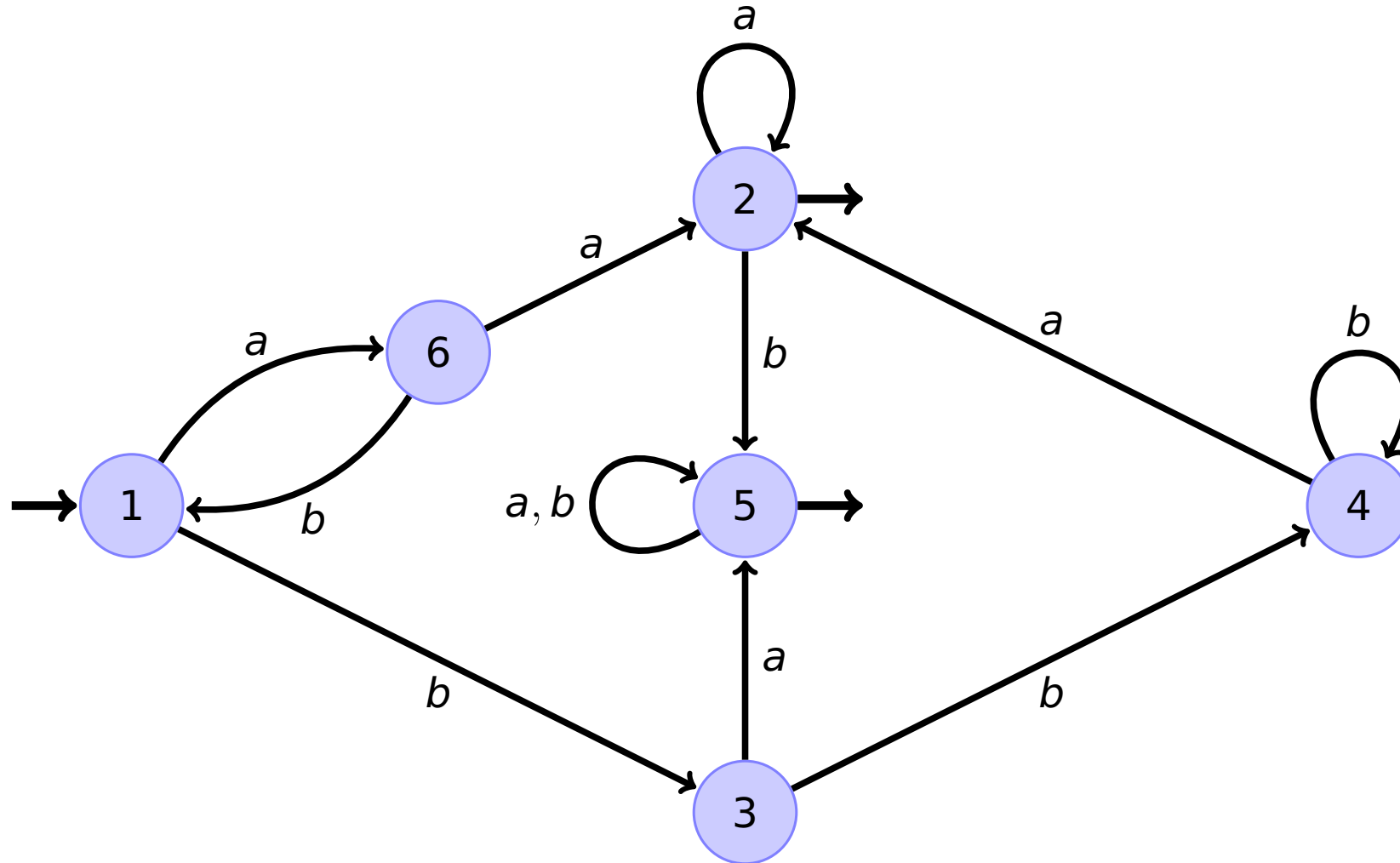
- ▶ On note  $L_q$  le langage des mots acceptés à partir de l'état  $q$ .
- ▶ Si  $L_p = L_q$ , on note  $p \sim q$  ( $p$  **équivalent** à  $q$ ).
- ▶ Automate **minimal** : obtenu en **identifiant** les états équivalents.

Si on identifie  $p \sim q$ , on conserve un automate déterministe car :

$$p \sim q \implies \delta(p, a) \sim \delta(q, a) \text{ pour toute lettre } a$$

Pourquoi ?

# Automate minimal : exemple



# Automate minimal : algorithme

- ▶ On calcule une suite de relations d'équivalence sur l'ensemble  $Q$  des états :

$$\sim_0, \sim_1, \sim_2, \dots$$

$$p \sim_k q \quad \text{si} \quad L_p \cap A^{\leq k} = L_q \cap A^{\leq k}$$

$$A^{\leq k} = A^0 \cup A^1 \cup \dots \cup A^k$$

$p \sim_k q$  si l'automate accepte à partir de  $p$  les mêmes mots jusqu'à la longueur  $k$  qu'à partir de  $q$ .

- ▶ La relation  $\sim_0$  a 2 classes d'équivalence :

# Automate minimal : algorithme

- ▶ On calcule une suite de relations d'équivalence sur l'ensemble  $Q$  des états :

$$\sim_0, \sim_1, \sim_2, \dots$$

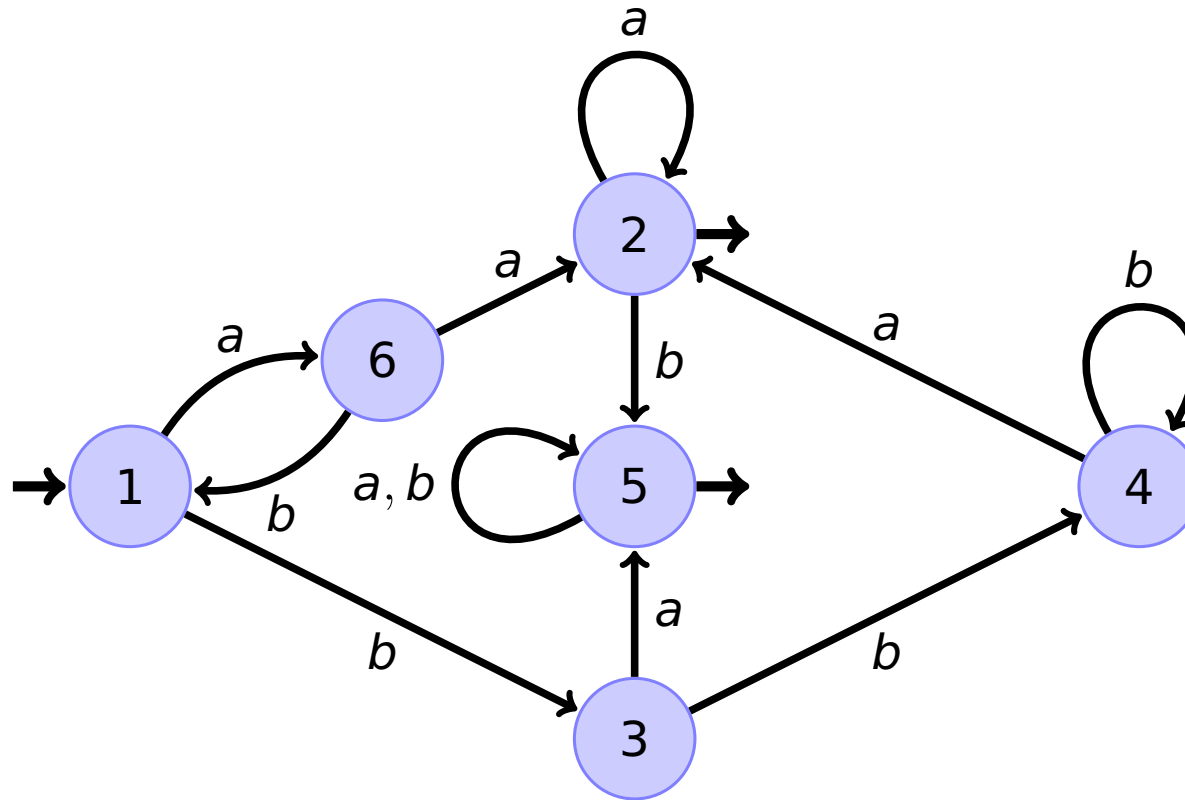
$$p \sim_k q \quad \text{si} \quad L_p \cap A^{\leq k} = L_q \cap A^{\leq k}$$

$$A^{\leq k} = A^0 \cup A^1 \cup \dots \cup A^k$$

$p \sim_k q$  si l'automate accepte à partir de  $p$  les mêmes mots jusqu'à la longueur  $k$  qu'à partir de  $q$ .

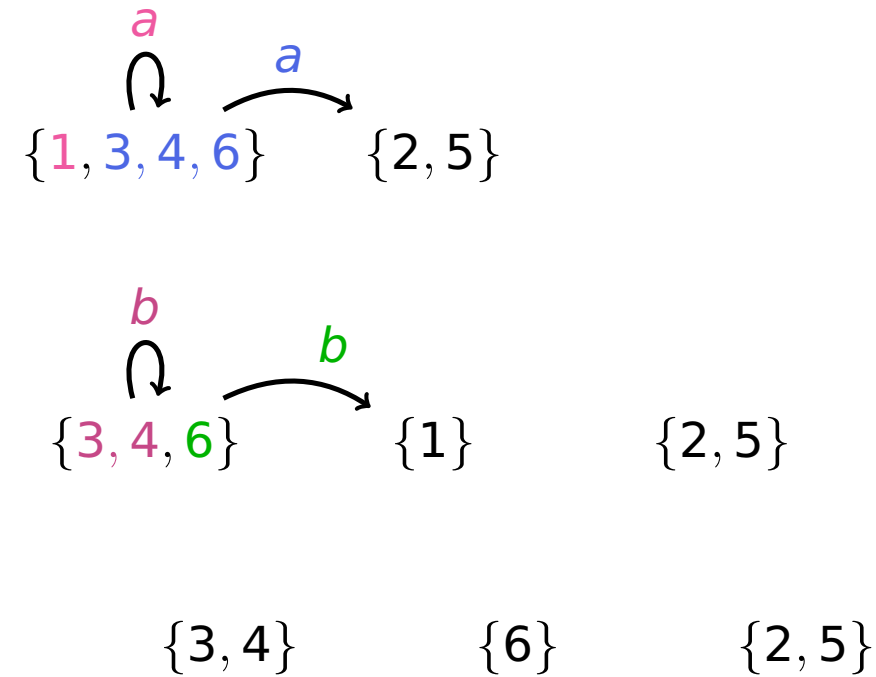
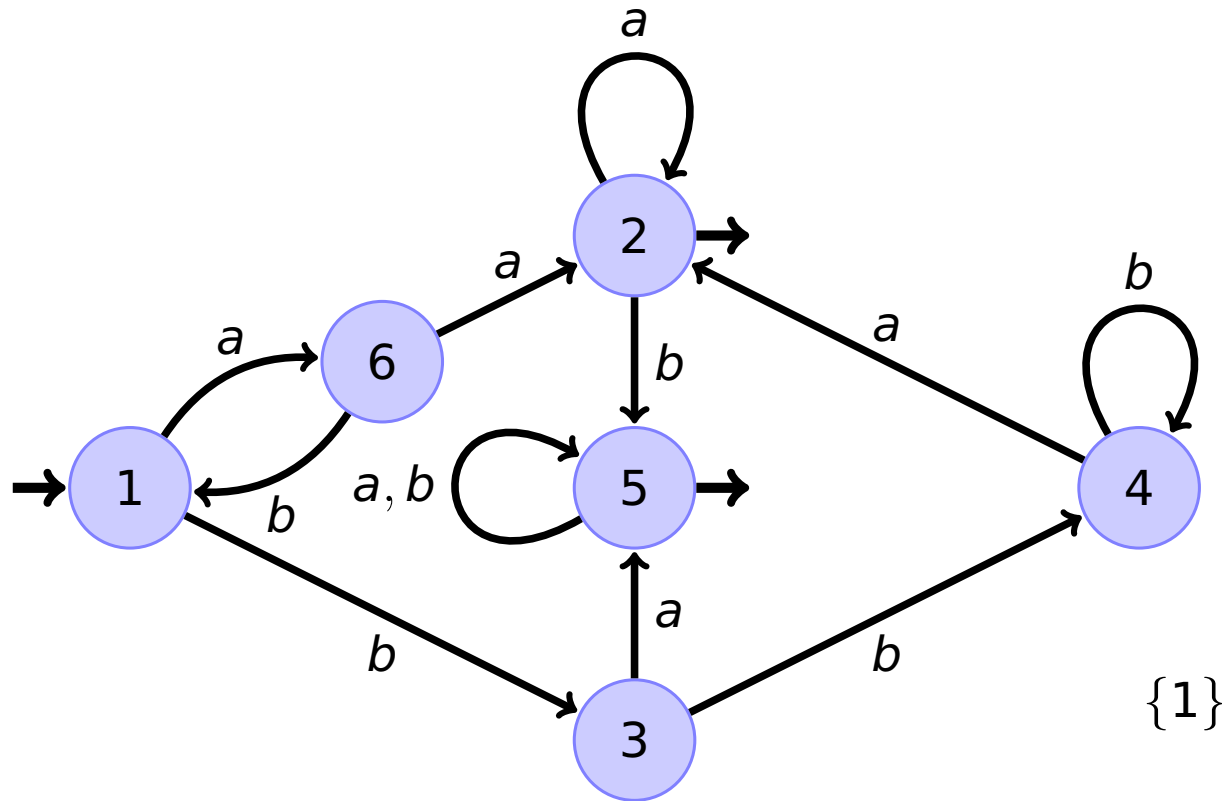
- ▶ La relation  $\sim_0$  a 2 classes d'équivalence :  $F$  et  $Q \setminus F$

# Exemple (bis)



$\sim_k$	classes de $\sim_k$
$k = 0$	$\{1, 3, 4, 6\}$ et $\{2, 5\}$
$k = 1$	$\{1\}$ , $\{3, 4, 6\}$ , et $\{2, 5\}$
$k = 2$	$\{1\}$ , $\{6\}$ , $\{3, 4\}$ , et $\{2, 5\}$
$k = 3$	$\{1\}$ , $\{6\}$ , $\{3, 4\}$ , et $\{2, 5\}$

# Exemple (bis) : minimisation



$\sim_k$	classes de $\sim_k$
$k = 0$	$\{1, 3, 4, 6\}$ et $\{2, 5\}$
$k = 1$	$\{1\}$ , $\{3, 4, 6\}$ , et $\{2, 5\}$
$k = 2$	$\{1\}$ , $\{6\}$ , $\{3, 4\}$ , et $\{2, 5\}$
$k = 3$	$\{1\}$ , $\{6\}$ , $\{3, 4\}$ , et $\{2, 5\}$

# Automate minimal : algorithme

- ▶ On calcule une suite de relations d'équivalence sur l'ensemble  $Q$  des états :

$$\sim_0, \sim_1, \sim_2, \dots$$

$$p \sim_k q \quad \text{si} \quad L_p \cap A^{\leq k} = L_q \cap A^{\leq k}$$

$$A^{\leq k} = A^0 \cup A^1 \cup \dots \cup A^k$$

$p \sim_k q$  si l'automate accepte à partir de  $p$  les mêmes mots jusqu'à la longueur  $k$  qu'à partir de  $q$ .

$$p \sim_{k+1} q \text{ implique } p \sim_k q$$

▶  $\sim_{k+1}$  **raffine**  $\sim_k$

Chaque classe d'équivalence de  $\sim_k$  est une union de classes d'équivalence de  $\sim_{k+1}$ .



# Automate minimal : algorithme

- ▶ On calcule une suite de relations d'équivalence sur l'ensemble  $Q$  des états :

$$\sim_0, \sim_1, \sim_2, \dots$$

$$p \sim_k q \quad \text{si} \quad L_p \cap A^{\leq k} = L_q \cap A^{\leq k}$$

$$A^{\leq k} = A^0 \cup A^1 \cup \dots \cup A^k$$

$p \sim_k q$  si l'automate accepte à partir de  $p$  les mêmes mots jusqu'à la longueur  $k$  qu'à partir de  $q$ .

$$p \sim_{k+1} q \text{ implique } p \sim_k q$$

▶  $\sim_{k+1}$  **raffine**  $\sim_k$

Chaque classe d'équivalence de  $\sim_k$  est une union de classes d'équivalence de  $\sim_{k+1}$ .

**Question** : est-ce que la suite  $\sim_0, \sim_1, \sim_2, \dots$  se stabilise ? Quand ?

# Automate minimal : algorithme très naïf

- ▶ Chaque classe d'équivalence de  $\sim_k$  est une **union** de classes d'équivalence de  $\sim_{k+1}$ .
- ▶ On a  $\sim_k = \sim_{k+1}$  dès que  $k \geq |Q|$

# Automate minimal : algorithme très naïf

- ▶ Chaque classe d'équivalence de  $\sim_k$  est une **union** de classes d'équivalence de  $\sim_{k+1}$ .
- ▶ On a  $\sim_k = \sim_{k+1}$  dès que  $k \geq |Q|$

Comment calculer la relation  $\sim_k$ ? Directement?

- ▶ pour chaque  $p, q$ , tester si  $p, q$  acceptent les mêmes mots de longueur  $\leq k$ .

Coût? Prenons  $|A| = m, |Q| = n$ .

# Automate minimal : algorithme très naïf

- ▶ Chaque classe d'équivalence de  $\sim_k$  est une **union** de classes d'équivalence de  $\sim_{k+1}$ .
- ▶ On a  $\sim_k = \sim_{k+1}$  dès que  $k \geq |Q|$

Comment calculer la relation  $\sim_k$ ? Directement?

- ▶ pour chaque  $p, q$ , tester si  $p, q$  acceptent les mêmes mots de longueur  $\leq k$ .

Coût? Prenons  $|A| = m, |Q| = n$ .

- ▶ Quel est le coût d'une étape?
- ▶ Combien d'étapes?

Q : combien il y a des mots de longueur  $k$ ?

$n + 1$

# Automate minimal : algorithme très naïf

- ▶ Chaque classe d'équivalence de  $\sim_k$  est une **union** de classes d'équivalence de  $\sim_{k+1}$ .
- ▶ On a  $\sim_k = \sim_{k+1}$  dès que  $k \geq |Q|$

Comment calculer la relation  $\sim_k$ ? Directement?

- ▶ pour chaque  $p, q$ , tester si  $p, q$  acceptent les mêmes mots de longueur  $\leq k$ .

Coût? Prenons  $|A| = m, |Q| = n$ .

- ▶ Quel est le coût d'une étape?
- ▶ Combien d'étapes?

$$O(m^{n+1} \cdot n^2 \cdot n)$$

Q : combien il y a des mots de longueur  $k$ ?

$$n + 1$$

mauvais

# Automate minimal : algorithme moins naïf

Idée : exploiter le calcul de  $\sim_k$  pour calculer  $\sim_{k+1}$  :

$$p \sim_{k+1} q \iff p \sim_k q \text{ et } (\delta(p, a) \sim_k \delta(q, a) \text{ pour tout } a).$$

Coût?  $|A| = m, |Q| = n.$

# Automate minimal : algorithme moins naïf

Idée : exploiter le calcul de  $\sim_k$  pour calculer  $\sim_{k+1}$  :

$$p \sim_{k+1} q \iff p \sim_k q \text{ et } (\delta(p, a) \sim_k \delta(q, a) \text{ pour tout } a).$$

Coût?  $|A| = m, |Q| = n.$

$O(mn^3)$

mieux !

# Automate minimal : algorithme moins naïf

Idée : exploiter le calcul de  $\sim_k$  pour calculer  $\sim_{k+1}$  :

$$p \sim_{k+1} q \iff p \sim_k q \text{ et } (\delta(p, a) \sim_k \delta(q, a) \text{ pour tout } a).$$

Coût?  $|A| = m, |Q| = n.$

$O(mn^3)$

mieux !

Question : peut-on faire encore mieux ?



# Automate minimal : algorithme de Moore

On utilise le tri lexicographique pour passer de

$$O(mn^3)$$

à

$$O(mn^2).$$

À chaque étape, on veut identifier les états qui ont le même mot

$$x_p = C(p) C(\delta(p, a_1)) \dots C(\delta(p, a_m)).$$

où  $C(p)$  est la classe de l'état  $p$ .

Les mots  $x_p$  sont des mots de longueur  $|A| + 1 = m + 1$  sur un alphabet de taille  $n = |Q|$  au plus.

# Tri lexicographique

Trier  $n$  mots de longueur  $k$  en temps  $O(kn)$ .

# Myhill-Nerode

Résiduels d'un langage  $L \subseteq A^*$  : langages  $u^{-1}L = \{v \in A^* \mid uv \in L\}$

Théorème de Myhill-Nerode :

Un langage  $L \subseteq A^*$  est régulier si et seulement si il a un nombre fini de résiduels.

Exemple :  $L_p = \{w \in \{a, b\}^* \mid |w| \text{ pair}\}$

$$\epsilon^{-1}L_p = L_p$$

$$a^{-1}L_p = \{v \in \{a, b\}^* \mid |v| \text{ est impair}\}$$

$$b^{-1}L_p = \{v \in \{a, b\}^* \mid |v| \text{ est impair}\}$$

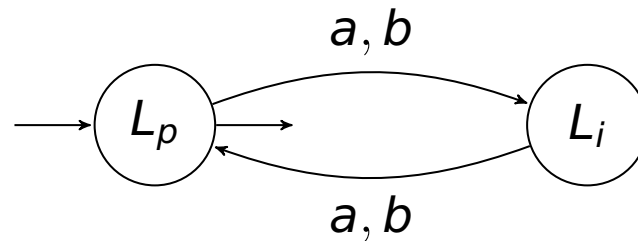
$$(aa)^{-1}L = L_p$$

# Automate des résiduels

Exemple :  $L_p = \{w \in \{a, b\}^* \mid |w| \text{ pair}\}$

$L_p$  a deux résiduels :  $L_p$  et  $L_i = \{w \in \{a, b\}^* \mid |w| \text{ est impair}\}$

L'automate pour  $L_p$  a deux états :

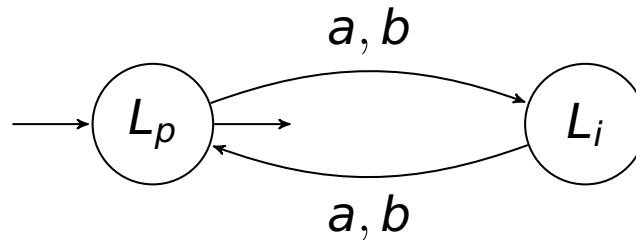


# Automate des résiduels

Exemple :  $L_p = \{w \in \{a, b\}^* \mid |w| \text{ pair}\}$

$L_p$  a deux résiduels :  $L_p$  et  $L_i = \{w \in \{a, b\}^* \mid |w| \text{ est impair}\}$

L'automate pour  $L_p$  a deux états :



## Automate des résiduels de $L$ :

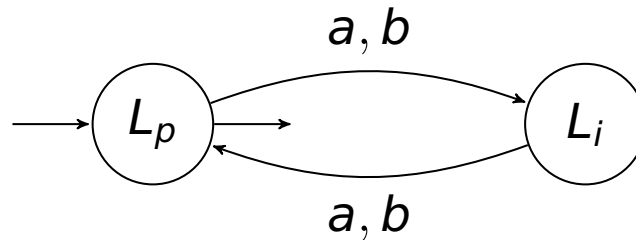
- ▶ pour chaque résiduel  $u^{-1}L$  : un état  $q_u$
- ▶ état initial :  $q_\epsilon$
- ▶ transitions :  $q_{au} \xrightarrow{a} q_u$
- ▶ états finaux :  $F = \{q_u \mid u \in L\}$

# Automate des résiduels

Exemple :  $L_p = \{w \in \{a, b\}^* \mid |w| \text{ pair}\}$

$L_p$  a deux résiduels :  $L_p$  et  $L_i = \{w \in \{a, b\}^* \mid |w| \text{ est impair}\}$

L'automate pour  $L_p$  a deux états :



## Automate des résiduels de $L$ :

- ▶ pour chaque résiduel  $u^{-1}L$  : un état  $q_u$
- ▶ état initial :  $q_\epsilon$
- ▶ transitions :  $q_{au} \xrightarrow{a} q_u$
- ▶ états finaux :  $F = \{q_u \mid u \in L\}$

Pour tout langage régulier  $L$  :  
l'automate des résiduels de  $L$  est  
l'automate minimal qui accepte  $L$ .

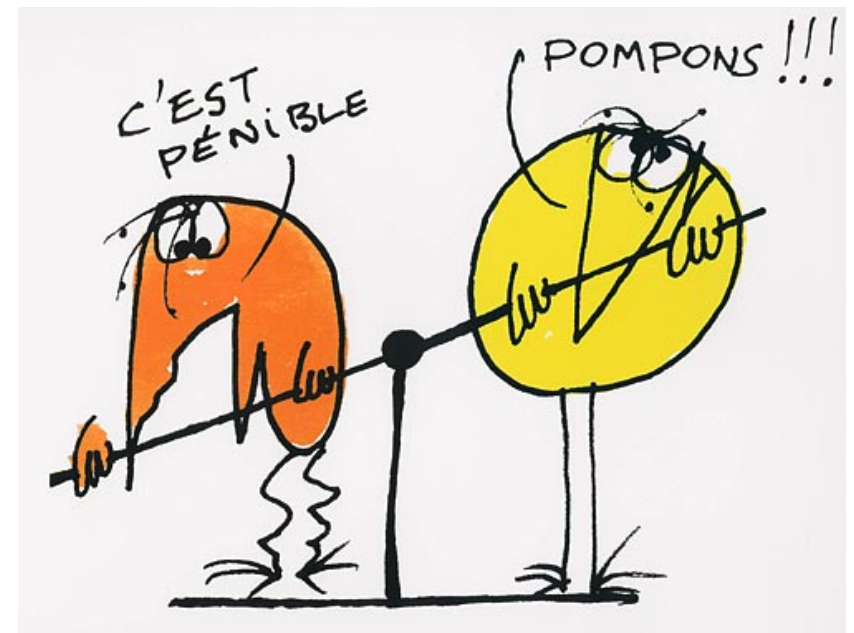
# Lemme de pompage pour les langages réguliers

Soit  $L \subseteq A^*$  un langage régulier.

Il existe un entier  $N > 0$  tel que tout mot  $x \in L$  de longueur au moins  $N$  peut être décomposé en  $x = uvw$  tel que les conditions suivantes sont satisfaites :

- ▶  $v \neq \epsilon$
- ▶  $|v| < N$
- ▶  $uv^k w \in L$  pour tout  $k \geq 0$

variante :  $|uv| < N$



# Lemme de pompage pour les langages réguliers

Soit  $L \subseteq A^*$  un langage régulier.

Il existe un entier  $N > 0$  tel que tout mot  $x \in L$  de longueur au moins  $N$  peut être décomposé en  $x = uvw$  tel que les conditions suivantes sont satisfaites :

- ▶  $v \neq \epsilon$
- ▶  $|v| < N$
- ▶  $uv^k w \in L$  pour tout  $k \geq 0$

variante :  $|uv| < N$

Preuve :

- ▶  $N$  est le nombre d'états d'un automate  $\mathcal{A}$  qui accepte  $L$
- ▶ tout calcul acceptant de  $\mathcal{A}$  sur un mot  $x \in L$  de longueur au moins  $N$  doit contenir une boucle :  $x = uvw$ , avec  $v$  boucle
- ▶ répéter la boucle  $k \geq 0$  fois ne change pas l'acceptation

$v =$  première boucle



# Comment utiliser le lemme de pompage

Pour montrer qu'un langage **n'est pas régulier** on peut appliquer la contreposée du lemme :  $L$  n'est pas régulier si

- ▶ Pour tout  $N > 0$ ...
- ▶ il existe un mot  $x \in L$  de longueur  $\geq N$ ...
- ▶ tel que pour toute décomposition  $x = uvw$  qui satisfait  $|uv| < N$ ,  $v \neq \epsilon$ ...
- ▶ il existe  $k \geq 0$  tel que  $uv^k w \notin L$ .

Exemple :  $\{a^n b^n \mid n \geq 0\}$

# Langages non-réguliers

Pour montrer qu'un langage ...

- ▶ ... **est régulier** : on donne une expression rationnelle, ou un automate fini qui reconnaît le langage
- ▶ ... **n'est pas régulier** : on applique le lemme de pompage, ou on montre que le nombre de résiduels du langage est infini.

# Langages non-réguliers

Pour montrer qu'un langage ...

- ▶ ... **est régulier** : on donne une expression rationnelle, ou un automate fini qui reconnaît le langage
- ▶ ... **n'est pas régulier** : on applique le lemme de pompage, ou on montre que le nombre de résiduels du langage est infini.

**Remarque** : pour montrer que  $L = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$  n'est pas régulier, on peut raisonner plus simplement.

- ▶  $L \cap a^*b^* = \{a^n b^n \mid n \geq 0\}$ .
- ▶ Si  $L$  était régulier, alors  $\{a^n b^n \mid n \geq 0\}$  le serait aussi. Contradiction.

# Limitations des langages réguliers

Les langages réguliers sont

- ▶ utiles pour décrire des ensembles de mots,
- ▶ ont une algorithmique simple,

mais ils sont limités :

- ▶ Des constructions très fréquentes en informatique ne sont pas rationnelles : mots bien parenthésés.
- ▶ De manière générale, avoir une quantité finie de mémoire est très limitant.

# Grammaires hors-contexte et automates à pile

Autre moyen de décrire des langages :

- ▶ Grammaires hors-contexte.
- ▶ “Machines” associées : automates à pile.

# Grammaires hors-contexte (algébriques)

Cf. cours analyse syntaxique.

$G = (V, A, R, S)$  où

▶  $A$  : alphabet (symboles terminaux)

▶  $V$  : variables (symboles non-terminaux)

▶  $R$  : règles de la forme

$$X \rightarrow \alpha \quad \text{avec } X \in V \text{ et } \alpha \in (A \cup V)^*$$

▶  $S \in V$  : symbole de départ.

# Utilisation d'une grammaire

- ▶ Etape de dérivation : si

$$X \rightarrow \alpha$$

est une règle, alors

$$u X v \rightarrow u \alpha v \quad u, v \in (A \cup V)^*$$

est une étape de dérivation.

- ▶  $\alpha \xrightarrow{k} \beta$  si on passe de  $\alpha$  à  $\beta$  en  $k$  étapes de dérivation.
- ▶  $\alpha \xrightarrow{*} \beta$  si on passe de  $\alpha$  à  $\beta$  en 0 ou plus étapes de dérivation.
- ▶ On peut représenter une dérivation par un **arbre de dérivation**.  
(on perd l'ordre des étapes de dérivation).

# Langages hors-contexte (algébriques)

- ▶ Langage généré par  $G = (V, A, R, S)$  = mots sur l'alphabet  $A$  que l'on peut dériver à partir de  $S$  :

$$L(G) = \{u \in A^* \mid S \xrightarrow{*} u\}$$

- ▶ Un langage  $L \subseteq A^*$  est **hors-contexte** (ou algébrique) s'il est généré par une grammaire hors-contexte  $G$ .





# Grammaires réduites

Une grammaire  $G = (V, A, R, S)$  est réduite si toute variable est **utile**. Formellement :

1. Pour tout  $X \in V$ , il existe  $u \in A^*$  tel que :

$$X \xrightarrow{*} u \in A^*$$

(la variable  $X$  est **productive**)

2. Pour tout  $X \in V$ , il existe  $\alpha, \beta \in (V \cup A)^*$  tels que :

$$S \xrightarrow{*} \alpha X \beta$$

(la variable  $X$  est **accessible**).

Une variable  $X \in V$  est donc **utile** si et seulement si elle apparaît dans la dérivation d'un mot de  $L(G)$ .

# Réduction des grammaires

Rendre une grammaire réduite sans changer le langage généré :

- ▶ Supprimer **d'abord** les variables qui ne génèrent aucun mot :

$$\mathcal{E}_0 = A$$

$$\mathcal{E}_{k+1} = \mathcal{E}_k \cup \{X \in V \mid X \rightarrow \alpha \text{ et } \alpha \in (\mathcal{E}_k)^*\}$$

- ▶  $\mathcal{E}_0 \subsetneq \mathcal{E}_1 \subsetneq \dots \subsetneq \mathcal{E}_p = \mathcal{E}_{p+1}$ .
- ▶  $\mathcal{E}_p =$  variables pouvant produire un mot de  $A^*$  (**variables productibles**).
- ▶ On peut donc **supprimer** les variables qui **ne sont pas** dans  $\mathcal{E}_p$ .

Cet algorithme permet de savoir si  $L(G) \neq \emptyset$ .

# Réduction des grammaires

Rendre une grammaire réduite sans changer le langage généré :

- ▶ Supprimer **ensuite** les variables inaccessibles depuis  $S$ .

$$\mathcal{F}_0 = \{S\}$$

$$\mathcal{F}_{k+1} = \mathcal{F}_k \cup \{X \in V \mid Y \rightarrow \alpha X \beta \text{ et } Y \in \mathcal{F}_k, \alpha, \beta \in (V \cup A)^*\}$$

$$\mathcal{F}_0 \subsetneq \mathcal{F}_1 \subsetneq \dots \subsetneq \mathcal{F}_p = \mathcal{F}_{p+1}$$

$\mathcal{F}_p =$  variables accessibles à partir de  $S$

On peut réduire une grammaire hors-contexte avec  $n$  variables et ensemble de règles  $R$  en temps  $O(n \cdot \text{taille}(R))$ .

Comment ?

# Grammaires propres

Une grammaire est **propre** si elle n'a aucune règle de la forme :

- ▶  $X \rightarrow \varepsilon$  (sauf éventuellement  $S \rightarrow \varepsilon$ , si  $\varepsilon$  est dans le langage)
- ▶  $X \rightarrow Y$  où  $Y$  est une variable.

Intérêt : permet de résoudre le problème d'analyse syntaxique.

On peut rendre une grammaire propre en préservant le langage généré. Si la grammaire a  $n$  variables et ensemble de règles  $R$ , ça se fait en temps  $O(n \cdot \text{taille}(R))$ .

Comment ?

# Forme normale quadratique

Une grammaire est **en forme normale quadratique** si toutes les règles ont la forme :

- ▶  $S \rightarrow \varepsilon$  si  $\varepsilon$  est dans le langage.
- ▶  $X \rightarrow a$  où  $a$  est symbole terminal.
- ▶  $X \rightarrow YZ$  où  $Y, Z$  sont des variables.

On peut mettre une grammaire en forme normale quadratique en préservant le langage généré. Si la grammaire a  $n$  variables et ensemble de règles  $R$ , ça se fait en temps  $O(n \cdot \text{taille}(R))$ .

Comment ?

$X \rightarrow Y_1 \cdots Y_k$  est remplacé par  $X \rightarrow Y_1 Z_1, Z_1 \rightarrow Y_2 Z_2, \dots, Z_{k-2} \rightarrow Y_{k-1} Y_k$  (les  $Z_i$  sont des nouvelles variables).

# Algorithme de Cocke-Younger-Kasami (CYK)

Permet de répondre à la question  $w \in L(G)$  ?

**Idée :** soit le mot  $w = a_1 \dots a_n$  ( $n \geq 0$  et  $a_i \in A$ ,  $1 \leq i \leq n$ ).

On note  $w[i, j]$  le facteur  $a_i \dots a_j$  ( $1 \leq i \leq j \leq n$ ).

On va calculer les ensembles (de variables)

$$\mathcal{T}[i, j] = \{X \in V \mid X \xrightarrow{*} w[i, j]\}$$

A la fin on aura le résultat suivant pour  $w \neq \epsilon$  :

$$w \in L(G) \quad \text{si et seulement si} \quad S \in \mathcal{T}[1, n]$$

# Cocke-Younger-Kasami (CYK)

$$\mathcal{T}[i, j] = \{X \in V \mid X \xrightarrow{*} w[i, j]\}$$

**Input:** Mot  $w = a_1 \dots a_n$  ( $n > 0$ )

**Output:**  $w \in L(G)$ ?

**for**  $i = 1, \dots, n$  **do**

|  $\mathcal{T}[i, i] := \{X \in V \mid X \longrightarrow a_i\};$

**end**

**for**  $d = 1, \dots, n - 1$  **do**

| **for**  $i = 1, \dots, n - d$  **do**

| |  $j = i + d;$

| |  $\mathcal{T}[i, j] := \emptyset;$

| | **for**  $k = i, \dots, j - 1$  **do**

| | | **forall**  $Y \in \mathcal{T}[i, k], Z \in \mathcal{T}[k + 1, j]$  et  $X \longrightarrow YZ$  **do**

| | | | rajouter  $X$  à  $\mathcal{T}[i, j]$

| | | **end**

| | **end**

| **end**

**end**

**return** oui si  $S \in \mathcal{T}[1, n]$ , non sinon;



# Lemme de pompage pour les langages hors-contexte

Soit  $L \subseteq A^*$  un langage hors-contexte.

Il existe un entier  $N > 0$  tel que tout mot  $z \in L$  de longueur  $> N$  peut être décomposé en  $z = uvwxy$  tel que les conditions suivantes sont satisfaites :

- ▶  $vx \neq \epsilon$
- ▶  $|vwx| < N$
- ▶  $uv^kwx^ky \in L$  pour tout  $k \geq 0$

# Lemme de pompage pour les langages hors-contexte

Soit  $L \subseteq A^*$  un langage hors-contexte.

Il existe un entier  $N > 0$  tel que tout mot  $z \in L$  de longueur  $> N$  peut être décomposé en  $z = uvwxy$  tel que les conditions suivantes sont satisfaites :

- ▶  $vx \neq \epsilon$
- ▶  $|vwx| < N$
- ▶  $uv^kwx^ky \in L$  pour tout  $k \geq 0$

Preuve :

- ▶  $L = L(G)$ ,  $G$  en forme normale quadratique avec  $M$  variables
- ▶  $N = 2^M$
- ▶ tout arbre de dérivation pour un mot  $z \in L$  de longueur  $> N$  est de profondeur  $> M$ , donc il contient un chemin sur lequel une variable se répète
- ▶ on considère un tel chemin et la première variable répétée (des feuilles vers la racine) :

$$S \xrightarrow{*} u\underline{X}y \xrightarrow{*} uv\underline{X}xy \xrightarrow{*} uv\underline{w}xy$$

# Comment utiliser le lemme de pompage

Pour montrer qu'un langage **n'est pas hors-contexte** on peut appliquer la contreposée du lemme :  $L$  n'est pas hors-contexte si

- ▶ Pour tout  $N > 0$ ...
- ▶ il existe un mot  $z \in L$  de longueur  $> N$ ...
- ▶ tel que pour toute décomposition  $z = uvwxy$  qui satisfait  $vx \neq \epsilon$ ,  $|vwx| \leq N$ ...
- ▶ il existe  $k \geq 0$  tel que  $uv^kwx^ky \notin L$ .

Exemples :  $\{a^n b^n c^n \mid n \geq 0\}$ ,  $\{ww \mid w \in \{a, b\}^*\}$

# Automates à pile

Un automate à pile ("pushdown automaton") est un automate fini auquel on rajoute une mémoire sous forme de pile ("last-in-first-out").

Un automate à pile est donné par 5 ensembles :  $(A, B, Q, \delta, q_0, F, Z)$

- ▶ Alphabets  $A, B$  : les mots lus par l'automate sont sur l'alphabet  $A$  ; la pile est un mot sur l'alphabet  $B$ .
- ▶ Ensemble **fini** d'états (de contrôle)  $Q$ .
- ▶ Ensemble de transitions  $\delta \subseteq Q \times (A \cup \{\epsilon\}) \times B \times Q \times B^*$ .
- ▶ **État initial**  $q_0 \in Q$ .
- ▶ **États finaux** (ou **acceptants**)  $F \subseteq Q$ .
- ▶ Symbole initial de pile  $Z \in B$ .

Une transition  $(p, a, X, q, v)$  peut être effectuée si l'état est  $p$ , le symbole actuel du mot d'entrée est  $a$  (pas de contrainte si  $a = \epsilon$ ) et le sommet de la pile est  $X$ . L'effet de la transition est de changer l'état en  $q$ , remplacer le sommet de la pile  $X$  par le mot  $v \in B^*$  et passer au symbole suivant de l'entrée si  $a \neq \epsilon$ .

# Automates à pile : définition

$$\mathcal{A} = (A, B, Q, \delta, q_0, F, Z)$$

- ▶ Une **configuration** de  $\mathcal{A}$  est une paire  $(p, v) \in Q \times B^*$ , constituée de l'état  $p$  et le contenu  $v$  de la pile (avec le sommet de pile à gauche).
- ▶ **Transition**  $(p, Xw) \xrightarrow{a} (q, vw)$  si  $(p, a, X, q, v) \in \delta$ .
- ▶ On écrit  $(p, w) \xrightarrow{u} (p', w')$  s'il existe une suite de transitions  $(p, w) \xrightarrow{a_0} (p_1, w_1) \xrightarrow{a_1} \dots (p_n, w_n) \xrightarrow{a_n} (p', w')$  telle que  $u = a_0 \dots a_n$ .
- ▶ Le **langage accepté** par  $\mathcal{A}$  est

$$L(\mathcal{A}) = \{u \in A^* \mid (q_0, Z) \xrightarrow{u} (p, w) \in F \times B^*\}.$$

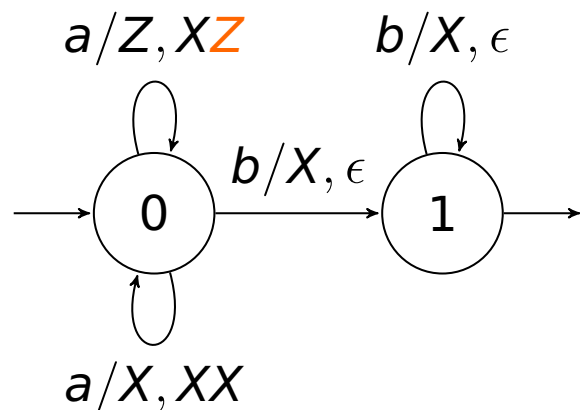
**Remarque** On peut aussi définir le langage accepté par **pile vide** :

$$L(\mathcal{A}) = \{u \in A^* \mid (q_0, Z) \xrightarrow{u} (p, \epsilon), p \in Q\}$$

Ces deux variantes d'acceptation sont équivalentes, sauf pour les automates déterministes.

# Exemples

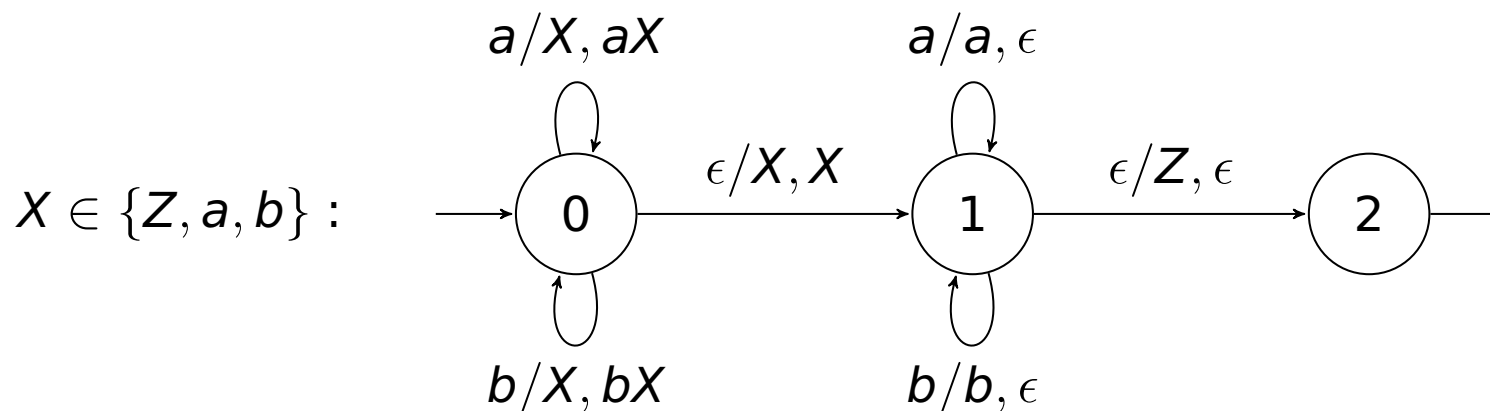
L'automate suivant accepte le langage  $\{a^m b^n \mid m \geq n > 0\}$  :



Comment modifier pour  $\{a^n b^n \mid n \geq 1\}$  ?

on rajoute  $1 \xrightarrow{\epsilon/Z, \epsilon} 2$  et seulement 2 est état final

L'automate suivant accepte les palindromes de longueur paire. L'alphabet de pile est  $B = \{Z, a, b\}$  :



# Automate à pile et grammaires

Pour tout langage hors-contexte  $L$  il existe un automate à pile (à un seul état) qui accepte  $L$  avec pile vide. Réciproquement, les langages acceptés par les automates à pile sont des langages hors-contexte.

Les automates à pile **déterministes** sont strictement moins expressifs (par exemple, le langage des palindromes ne peut pas être accepté par un automate à pile déterministe).

# Des grammaires vers les automates à pile

Soit  $G = (V, A, R, S)$  une grammaire en forme normale quadratique.

$$X \longrightarrow YZ \text{ ou } X \longrightarrow a.$$

On définit un automate à pile  $\mathcal{A} = (A, A \cup V, \{q\}, \delta, q, -, S)$  qui accepte par pile vide :

$$\delta = \{(q, \epsilon, X, q, YZ) \mid X \longrightarrow YZ \text{ dans } R\} \cup \{(q, a, X, q, \epsilon) \mid X \longrightarrow a \text{ dans } R\}$$

$$\text{On a : } L(G) = L(\mathcal{A})$$

Cet automate simule une **dérivation de gauche** (on remplace toujours la variable le plus à gauche).

**Exemple :**  $S \longrightarrow aSb \mid SS \mid \epsilon$

$$S \longrightarrow \underline{S}S \longrightarrow a\underline{S}bS \longrightarrow a a\underline{S}b bS \longrightarrow aabb \underline{S} \longrightarrow aabb$$



# Des automates à pile vers les grammaires

Soit  $\mathcal{A} = (A, B, Q, \delta, q_0, -, Z)$  un automate qui accepte avec pile vide.

On construit une grammaire  $G$  avec variables :

$$V = \{ \langle p, X, q \rangle \mid p, q, \in Q, X \in B \}$$

**Principe :**  $\langle p, X, q \rangle \xrightarrow{*} w$  dans  $G$  si et seulement si  $(p, X) \xrightarrow{w} (q, \epsilon)$  dans  $\mathcal{A}$ .

Pour toute transition  $(p, a, X, q, Y_1 \dots Y_k) \in \delta$  de  $\mathcal{A}$  on rajoute des règles

$$\langle p, X, r \rangle \longrightarrow a \langle q, Y_1, r_1 \rangle \langle r_1, Y_2, r_2 \rangle \dots \langle r_{k-1}, Y_k, r \rangle$$

pour tous les états  $r, r_1, \dots, r_{k-1} \in Q$  possibles.

# Propriétés algorithmiques

Les langages réguliers ont beaucoup de bonnes propriétés algorithmiques. Les langages hors-contexte en ont moins.

- ▶ Il existe des algorithmes pour savoir si le langage d'un automate fini, ou d'une grammaire (ou automate à pile) est **non-vide**.
- ▶ Le **complémentaire** d'un langage régulier est aussi régulier. Mais il existe des langages hors-contexte dont le complémentaire n'est pas hors-contexte.
- ▶ L'**intersection** de deux langages réguliers est un langage régulier. Mais il existe des langages hors-contexte dont l'intersection n'est pas hors-contexte.
- ▶ Il existe un algorithme pour savoir si l'**intersection** de deux langages réguliers **est non-vide**. Mais **il n'existe pas d'algorithme** qui permet de savoir si l'intersection de deux langages hors-contexte est non-vide.