

Projet de Programmation C : Motifs et automates

INTRODUCTION: Recherche de motifs

Le problème - Première version

- ▶ **ENTRÉE** : Deux **chaînes de caractères** : `texte` et `motif`.
- ▶ **OBJECTIF** : Trouver toutes les occurrences de `motif` dans `texte`.

Le problème - Première version

- ▶ **ENTRÉE** : Deux **chaînes de caractères** : `texte` et `motif`.
- ▶ **OBJECTIF** : Trouver toutes les occurrences de `motif` dans `texte`.

Exemple

Chercher toutes les occurrences du `motif` « *ananas* » dans le texte suivant :

« Un jour, un ananas tomba du ciel. Le petit andré le ramassa. Il décida de bâtir un village : anana-ville. Là-bas, tout le monde célébrait chaque fête avec un énorme ananas au milieu de la place. »

Le problème - Première version

- ▶ **ENTRÉE** : Deux **chaînes de caractères** : `texte` et `motif`.
- ▶ **OBJECTIF** : Trouver toutes les occurrences de `motif` dans `texte`.

Exemple

Chercher toutes les occurrences du `motif` « *ananas* » dans le texte suivant :

« Un jour, un **ananas** tomba du ciel. Le petit andré le ramassa. Il décida de bâtir un village : anana-ville. Là-bas, tout le monde célébrait chaque fête avec un énorme **ananas** au milieu de la place. »

Le problème - Première version - Approche naïve

Objectif simplifié : Tester si `motif` apparaît dans `texte`.

Le problème - Première version - Approche naïve

Objectif simplifié : Tester si motif apparaît dans texte.

```
bool rech_pos(char* texte, char* motif, int i) { // O(|motif|)
    for (int j = 0; motif[j] != '\0'; j++) {
        if (texte[i+j] == '\0' || motif[j] != texte[i+j])
            return false;
    }
    return true;
}

bool rech(char* texte, char* motif){ // O(|motif| * |texte|)
    for (int i = 0; texte[i] != '\0'; i++)
        if (rech_pos(texte,motif,i))
            return true;
    return false;
}
```

Le problème - Première version - Approche efficace

Reprenons le **motif** « *ananas* ».

On le cherche dans le **texte** « *aasanaaananasn* ».

Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

On le cherche dans le texte « *aasanaaananasn* ».

Notre algorithme

Lire **texte** de gauche à droite en **mémorisant une information clé** :

Quel est le plus grand préfixe de *ananas* qu'on vient de lire ?

Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

On le cherche dans le texte « *aasanaaananasn* ».

Notre algorithme

Lire **texte** de gauche à droite en **mémorisant une information clé** :
Quel est le plus grand préfixe de *ananas* qu'on vient de lire ?



7 états mémoire suffisent

On change d'état à chaque nouvelle lettre lue.

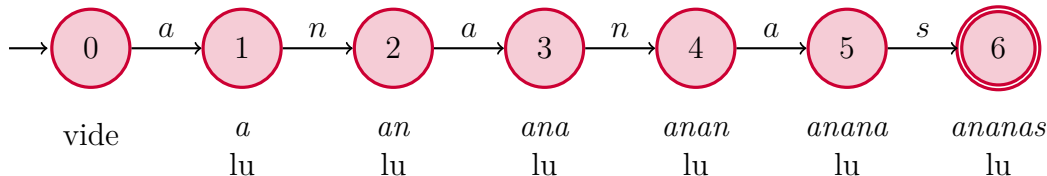
Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

On le cherche dans le texte « *aasanaaananasn* ».

Notre algorithme

Lire **texte** de gauche à droite en **mémorisant une information clé** :
Quel est le plus grand préfixe de *ananas* qu'on vient de lire ?



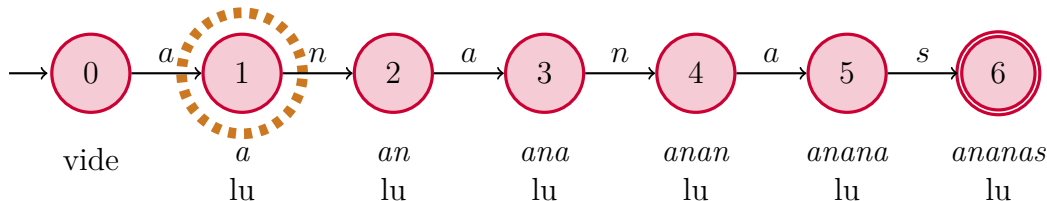
7 états mémoire suffisent

On change d'état à chaque nouvelle lettre lue.

Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

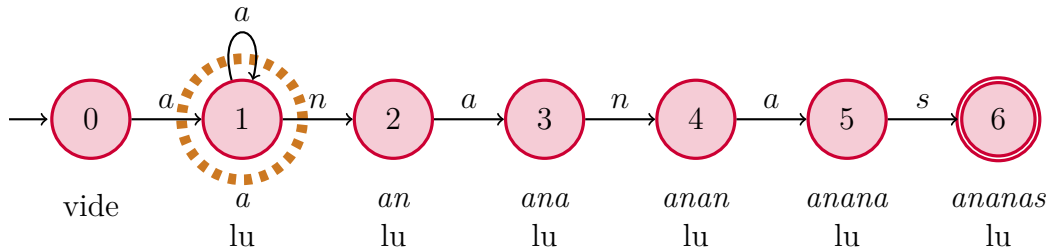
On le cherche dans le texte « *aasanaaanananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

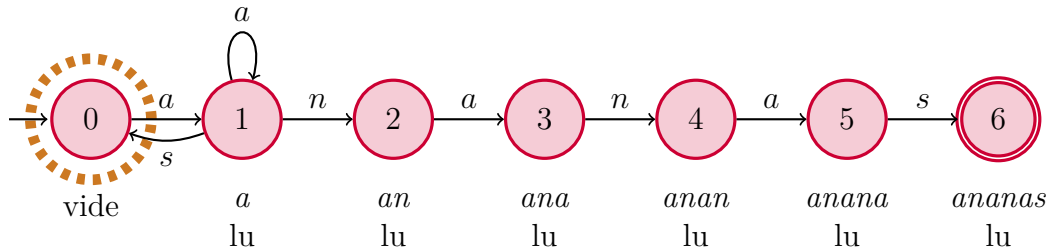
On le cherche dans le **texte** « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

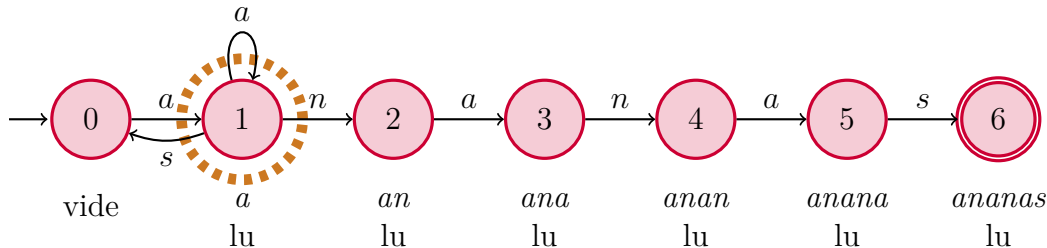
On le cherche dans le **texte** « *aasanaaanananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

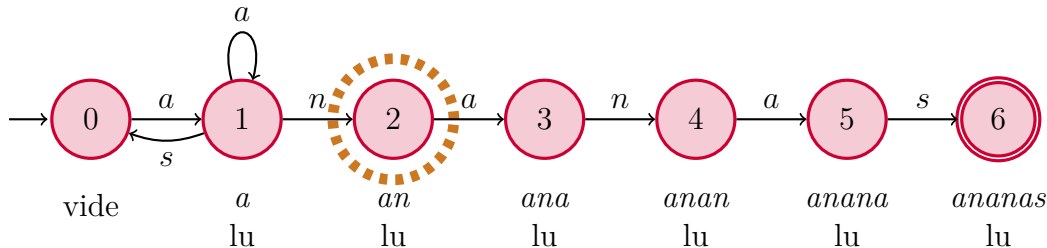
On le cherche dans le **texte** « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

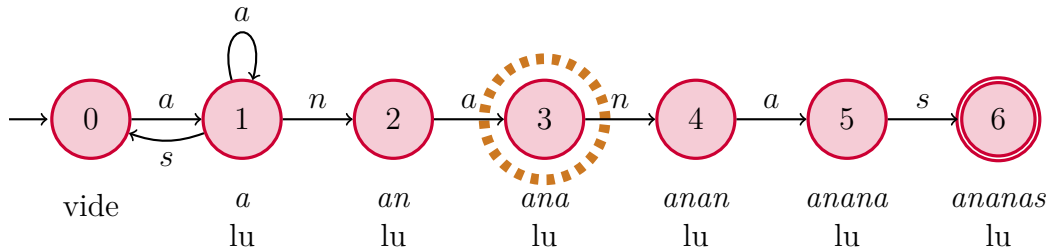
On le cherche dans le **texte** « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

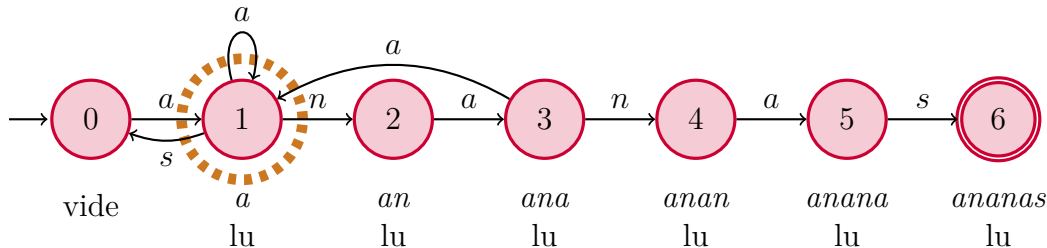
On le cherche dans le **texte** « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

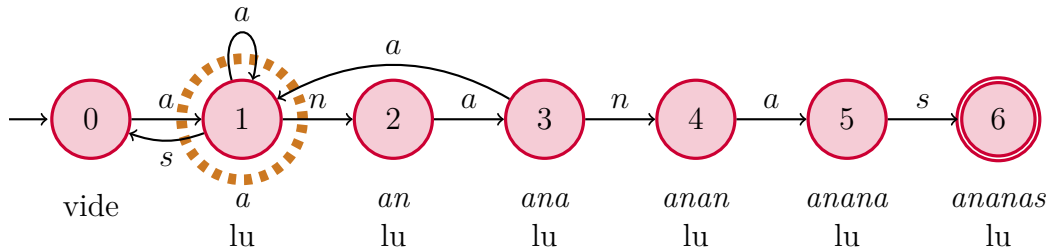
On le cherche dans le texte « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

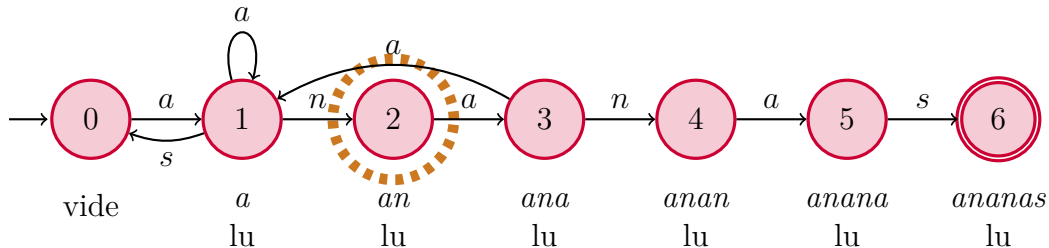
On le cherche dans le texte « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

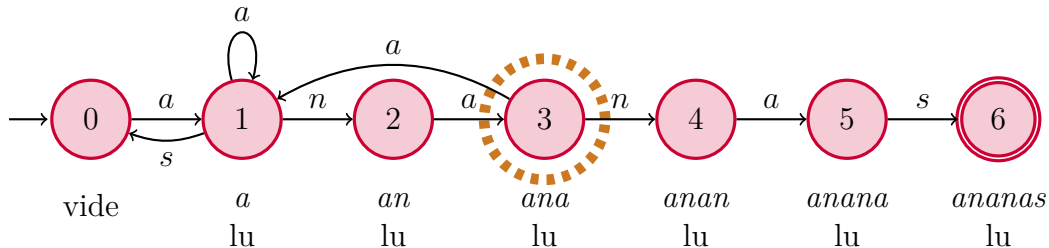
On le cherche dans le texte « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

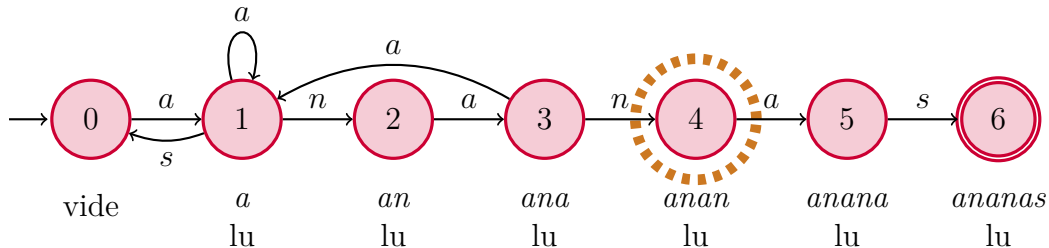
On le cherche dans le texte « *aasanaaanananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

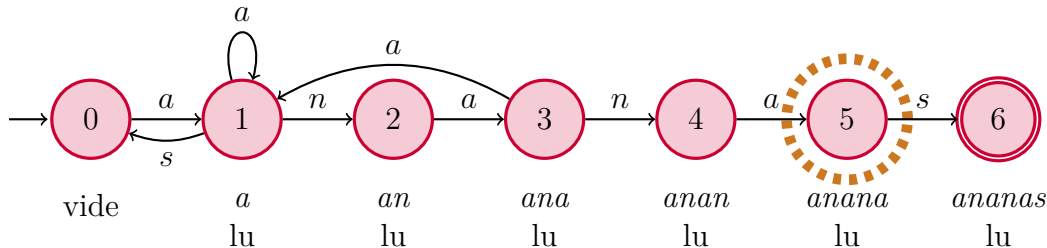
On le cherche dans le texte « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

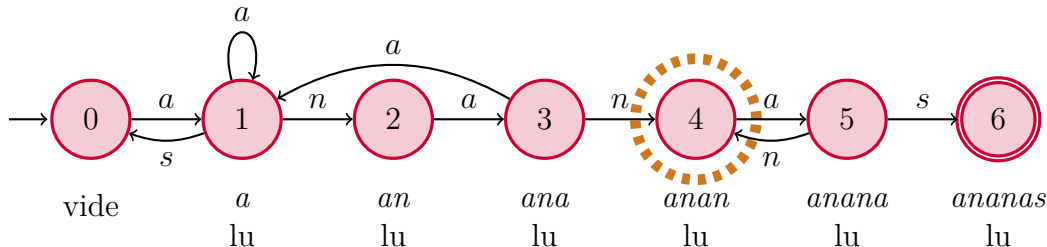
On le cherche dans le texte « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

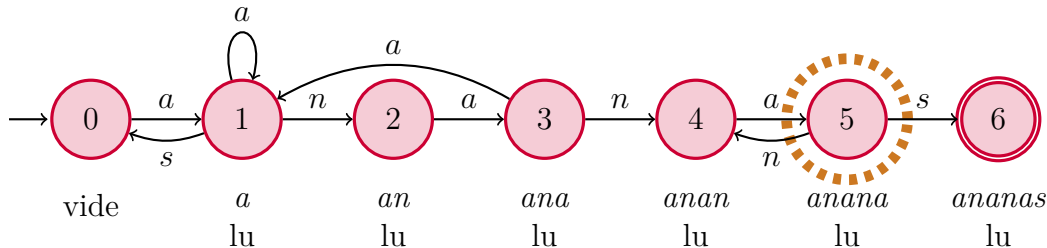
On le cherche dans le texte « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

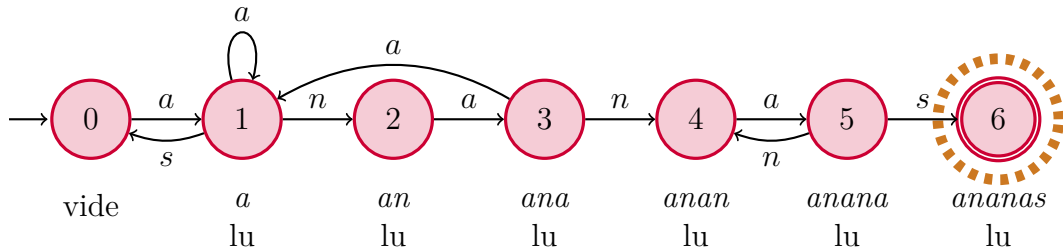
On le cherche dans le texte « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

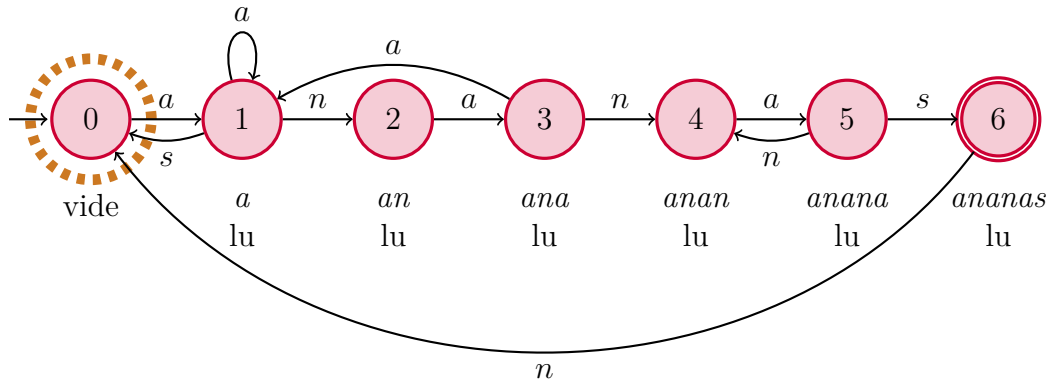
On le cherche dans le texte « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

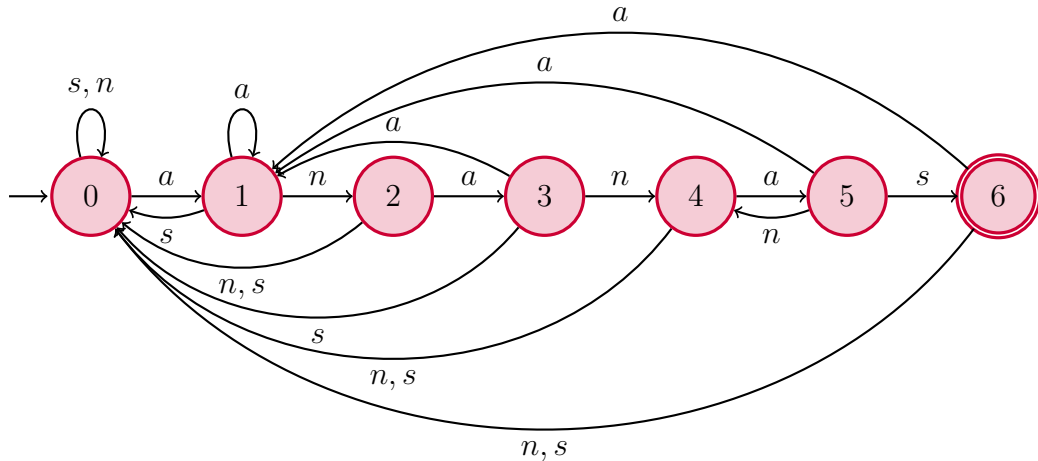
On le cherche dans le texte « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

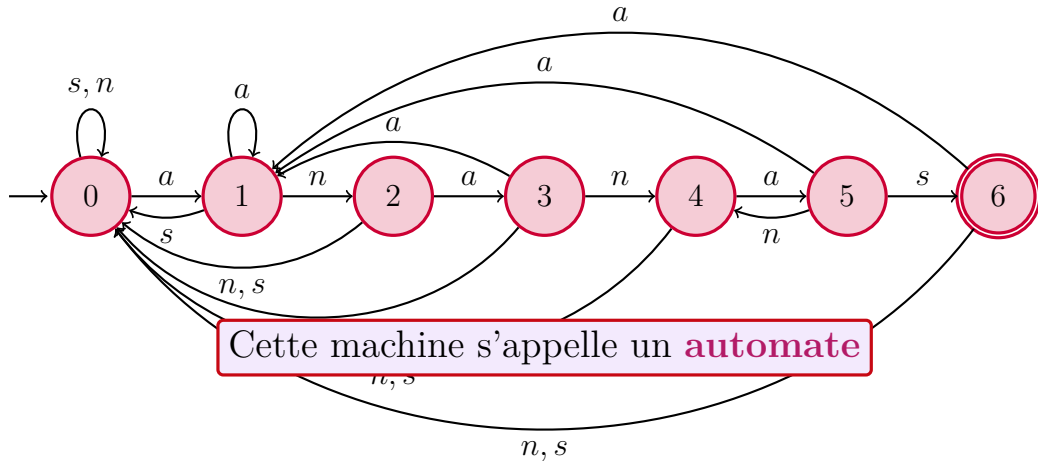
On le cherche dans le texte « *aasanaaananasn* ».



Le problème - Première version - Approche efficace

Reprenons le motif « *ananas* ».

On le cherche dans le texte « *aasanaaananasn* ».



Deux questions

1. Les automates c'est super. Je dois **tout savoir sur eux**!

Deux questions

1. Les automates c'est super. Je dois **tout savoir sur eux** !
2. Les automates c'est génial. **Comment puis-je les construire** ?

PARTIE I:

Automates déterministes

Définition - Un automate est un **programme**

Chaque programme est composé de :

Définition - Un automate est un **programme**

Chaque programme est composé de :

1. **Alphabet fini A.**

$$A = \{a, b\}$$

ENTRÉE : chaîne sur les caractères de A .
(on parle d'un **mot sur l'alphabet A**)

Définition - Un automate est un **programme**

Chaque programme est composé de :

1. **Alphabet fini A.**
2. **Ensemble fini d'états Q.**

$$A = \{a, b\} \quad Q = \{q_0, q_1, q_2\}$$



ENTRÉE : chaîne sur les caractères de A .
(on parle d'un **mot sur l'alphabet A**)

Définition - Un automate est un **programme**

Chaque programme est composé de :

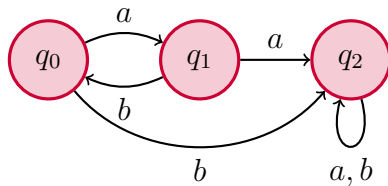
1. **Alphabet fini A .**
2. **Ensemble fini d'états Q .**
3. **Fonction de transition $\delta : Q \times A \rightarrow Q$.**

$$A = \{a, b\} \quad Q = \{q_0, q_1, q_2\}$$

$$\delta(q_0, a) = q_1 \quad \delta(q_0, b) = q_2$$

$$\delta(q_1, a) = q_2 \quad \delta(q_1, b) = q_0$$

$$\delta(q_2, a) = q_2 \quad \delta(q_2, b) = q_2$$



ENTRÉE : chaîne sur les caractères de A .
(on parle d'un **mot sur l'alphabet A**)

Définition - Un automate est un **programme**

Chaque programme est composé de :

1. **Alphabet fini A .**
2. **Ensemble fini d'états Q .**
3. **Fonction de transition $\delta : Q \times A \rightarrow Q$.**
4. **État initial $q_{\text{init}} \in Q$.**

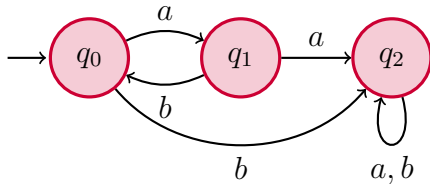
$$A = \{a, b\} \quad Q = \{q_0, q_1, q_2\}$$

$$\delta(q_0, a) = q_1 \quad \delta(q_0, b) = q_2$$

$$\delta(q_1, a) = q_2 \quad \delta(q_1, b) = q_0$$

$$\delta(q_2, a) = q_2 \quad \delta(q_2, b) = q_2$$

$$q_{\text{init}} = q_0$$



ENTRÉE : chaîne sur les caractères de A .
(on parle d'un **mot sur l'alphabet A**)

Définition - Un automate est un **programme**

Chaque programme est composé de :

1. **Alphabet fini A .**
2. **Ensemble fini d'états Q .**
3. **Fonction de transition $\delta : Q \times A \rightarrow Q$.**
4. **État initial $q_{init} \in Q$.**
5. **Ensemble d'états finaux $F \subseteq Q$.**

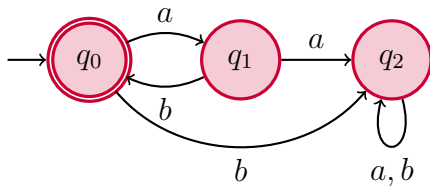
$$A = \{a, b\} \quad Q = \{q_0, q_1, q_2\}$$

$$\delta(q_0, a) = q_1 \quad \delta(q_0, b) = q_2$$

$$\delta(q_1, a) = q_2 \quad \delta(q_1, b) = q_0$$

$$\delta(q_2, a) = q_2 \quad \delta(q_2, b) = q_2$$

$$q_{init} = q_0 \quad F = \{q_0\}$$



ENTRÉE : chaîne sur les caractères de A .
(on parle d'un **mot sur l'alphabet A**)

Définition - Un automate est un **programme**

Chaque programme est composé de :

1. **Alphabet fini A .**
2. **Ensemble fini d'états Q .**
3. **Fonction de transition $\delta : Q \times A \rightarrow Q$.**
4. **État initial $q_{init} \in Q$.**
5. **Ensemble d'états finaux $F \subseteq Q$.**

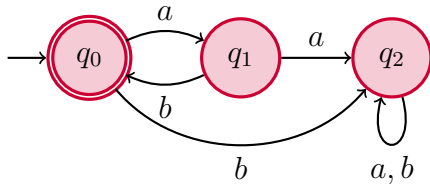
$$A = \{a, b\} \quad Q = \{q_0, q_1, q_2\}$$

$$\delta(q_0, a) = q_1 \quad \delta(q_0, b) = q_2$$

$$\delta(q_1, a) = q_2 \quad \delta(q_1, b) = q_0$$

$$\delta(q_2, a) = q_2 \quad \delta(q_2, b) = q_2$$

$$q_{init} = q_0 \quad F = \{q_0\}$$



ENTRÉE : chaîne sur les caractères de A .
(on parle d'un **mot sur l'alphabet A**)

Un mot est **accepté** ou **rejeté** par l'automate.

Définition - Un automate est un **programme**

Chaque programme est composé de :

1. **Alphabet fini A .**
2. **Ensemble fini d'états Q .**
3. **Fonction de transition $\delta : Q \times A \rightarrow Q$.**
4. **État initial $q_{init} \in Q$.**
5. **Ensemble d'états finaux $F \subseteq Q$.**

ENTRÉE : chaîne sur les caractères de A .
(on parle d'un **mot sur l'alphabet A**)

Un mot est **accepté** ou **rejeté** par l'automate.

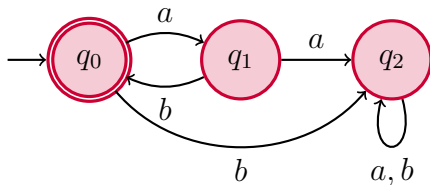
$$A = \{a, b\} \quad Q = \{q_0, q_1, q_2\}$$

$$\delta(q_0, a) = q_1 \quad \delta(q_0, b) = q_2$$

$$\delta(q_1, a) = q_2 \quad \delta(q_1, b) = q_0$$

$$\delta(q_2, a) = q_2 \quad \delta(q_2, b) = q_2$$

$$q_{init} = q_0 \quad F = \{q_0\}$$



Accepte les mots de la forme :
« $ababab \dots ab$ »

Définition - Un automate est un **programme**

Chaque programme est composé de :

1. **Alphabet fini A .**
2. **Ensemble fini d'états Q .**
3. **Fonction de transition $\delta : Q \times A \rightarrow Q$.**
4. **État initial $q_{init} \in Q$.**
5. **Ensemble d'états finaux $F \subseteq Q$.**

ENTRÉE : chaîne sur les caractères de A .
(on parle d'un **mot sur l'alphabet A**)

Un mot est **accepté** ou **rejeté** par l'automate.

Ensemble des mots acceptés :

langage reconnu par l'automate.

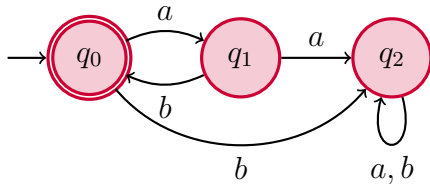
$$A = \{a, b\} \quad Q = \{q_0, q_1, q_2\}$$

$$\delta(q_0, a) = q_1 \quad \delta(q_0, b) = q_2$$

$$\delta(q_1, a) = q_2 \quad \delta(q_1, b) = q_0$$

$$\delta(q_2, a) = q_2 \quad \delta(q_2, b) = q_2$$

$$q_{init} = q_0 \quad F = \{q_0\}$$



Accepte les mots de la forme :

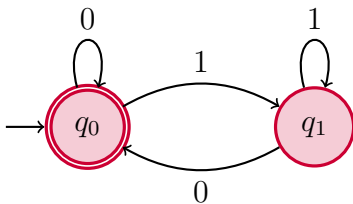
« *ababab...ab* »

Nombres pairs

- ▶ Alphabet $A = \{0, 1\}$. On code des entiers en binaire.
- ▶ Écrire un automate qui reconnaît le langage des mots codant un entier pair.

Nombres pairs

- ▶ Alphabet $A = \{0, 1\}$. On code des entiers en binaire.
- ▶ Écrire un automate qui reconnaît le langage des mots codant un entier pair.

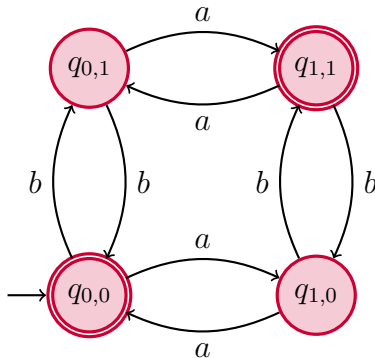


Parité

- ▶ Alphabet $A = \{a, b\}$.
- ▶ Écrire un automate qui reconnaît le langage des mots qui contiennent :
 - ▶ soit un nombre pair de a et un nombre pair de b ,
 - ▶ soit un nombre impair de a et un nombre impair de b .

Parité

- ▶ Alphabet $A = \{a, b\}$.
- ▶ Écrire un automate qui reconnaît le langage des mots qui contiennent :
 - ▶ soit un nombre pair de a et un nombre pair de b ,
 - ▶ soit un nombre impair de a et un nombre impair de b .

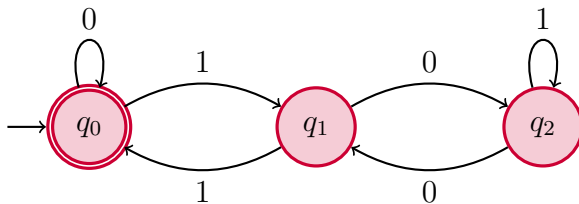


Multiples de 3

- ▶ Alphabet $A = \{0, 1\}$. On code des entiers en binaire.
- ▶ Écrire un automate qui reconnaît le langage des mots codant un multiple de 3.

Multiples de 3

- ▶ Alphabet $A = \{0, 1\}$. On code des entiers en binaire.
- ▶ Écrire un automate qui reconnait le langage des mots codant un multiple de 3.



Langages reconnaissables

Pour un alphabet A , on appelle **langage** un ensemble de mots écrits avec A .

- ▶ On note A^* le langage de **tous les mots** sur l'alphabet A .
- ▶ Un langage peut être infini (par exemple, A^*).

Langages reconnaissables

Pour un alphabet A , on appelle **langage** un ensemble de mots écrits avec A .

- ▶ On note A^* le langage de **tous les mots** sur l'alphabet A .
- ▶ Un langage peut être infini (par exemple, A^*).

Les langages reconnaissables (REC)

Un langage est **reconnaissable** si il existe un automate qui le reconnaît.

Langages reconnaissables

Pour un alphabet A , on appelle **langage** un ensemble de mots écrits avec A .

- ▶ On note A^* le langage de **tous les mots** sur l'alphabet A .
- ▶ Un langage peut être infini (par exemple, A^*).

Les langages reconnaissables (REC)

Un langage est **reconnaissable** si il existe un automate qui le reconnaît.

Attention : Les langages ne sont pas tous reconnaissables

Prenons $A = \{a, b\}$:

Le langage des mots qui contiennent autant de a que de b n'est **pas** reconnaissable.

Récapitulatif

Nous en savons maintenant un peu plus sur les automates.

Récapitulatif

Nous en savons maintenant un peu plus sur les automates.

Notre Objectif : Nous en servir pour de la reconnaissance de motifs.

- ▶ On a commencé avec des motifs simples (souvenons-nous du mot « ananas »).
- ▶ Les automates permettent de rechercher des **motifs plus complexes**.

Récapitulatif

Nous en savons maintenant un peu plus sur les automates.

Notre Objectif : Nous en servir pour de la reconnaissance de motifs.

- ▶ On a commencé avec des motifs simples (souvenons-nous du mot « ananas »).
- ▶ Les automates permettent de rechercher des **motifs plus complexes**.

Exemple : recherche de fichiers dans un dossier

Recherche de tous les fichiers « .c » qui commencent par le préfixe « algo ».

Récapitulatif

Nous en savons maintenant un peu plus sur les automates.

Notre Objectif : Nous en servir pour de la reconnaissance de motifs.

- ▶ On a commencé avec des motifs simples (souvenons-nous du mot « ananas »).
- ▶ Les automates permettent de rechercher des **motifs plus complexes**.

Exemple : recherche de fichiers dans un dossier

Recherche de tous les fichiers « .c » qui commencent par le préfixe « algo ».

Problème

Les automates sont **bien pour l'algorithmique, pas pour décrire les motifs**.
⇒ On a besoin d'une syntaxe pratique pour décrire nos motifs.

PARTIE II:

Expressions régulières

Expressions régulières : une syntaxe pour décrire des langages

Chaque expression régulière représente un **langage**.

Expressions régulières : une syntaxe pour décrire des langages

Chaque expression régulière représente un **langage**.

Quelques exemples pour commencer

- ▶ Expression « *ananas + banane* » :
langage (contenant deux mots) $\{ananas, banane\}$.
- ▶ Expression « $(ab)^*$ » :
langage (infini) des mots de la forme $ababab \cdots ab$.
- ▶ Expression « $(b + ab)^*$ » :
langage (infini) des mots où tout « a » est suivi d'un « b ».

Expressions régulières : une syntaxe pour décrire des langages

On note A l'alphabet.

Expressions régulières basiques

► Expression « $\emptyset$ ».

Définit le langage vide : $L(\emptyset) = \emptyset$.

Expressions régulières : une syntaxe pour décrire des langages

On note A l'alphabet.

Expressions régulières basiques

- ▶ Expression « $\emptyset$ ».
Définit le langage vide : $L(\emptyset) = \emptyset$.
- ▶ Expression « a » (avec $a \in A$ une lettre de l'alphabet).
Définit le langage $L(a) = \{a\}$.

Expressions régulières : une syntaxe pour décrire des langages

Expressions régulières : règles récursives

On peut combiner des expressions pour en former des nouvelles :

Le mot « ε » est le **mot vide** (la chaîne de caractère de longueur 0).

Expressions régulières : une syntaxe pour décrire des langages

Expressions régulières : règles récursives

On peut combiner des expressions pour en former des nouvelles :

- ▶ Si E_1 et E_2 sont des expressions, alors $E_1 + E_2$ est une expression.
Définit le langage $L(E_1 + E_2) = L(E_1) \cup L(E_2)$.
Exemple : $L(a + b + e) = \{a, b, e\}$ (3 mots).

Le mot « ε » est le **mot vide** (la chaîne de caractère de longueur 0).

Expressions régulières : une syntaxe pour décrire des langages

Expressions régulières : règles récursives

On peut combiner des expressions pour en former des nouvelles :

- ▶ Si E_1 et E_2 sont des expressions, alors $E_1 + E_2$ est une expression.
Définit le langage $L(E_1 + E_2) = L(E_1) \cup L(E_2)$.
Exemple : $L(a + b + e) = \{a, b, e\}$ (3 mots).
- ▶ Si E_1 et E_2 sont des régulières, alors $E_1 E_2$ est une expression.
Langage $L(E_1 E_2) = \{w_1 w_2 \mid w_1 \in L(E_1) \text{ et } w_2 \in L_2\}$.
Exemple : $L((a + b)(aa + ba + b)) = \{aaaa, aba, ab, baa, bba, bb\}$ (6 mots).

Le mot « ε » est le **mot vide** (la chaîne de caractère de longueur 0).

Expressions régulières : une syntaxe pour décrire des langages

Expressions régulières : règles récursives

On peut combiner des expressions pour en former des nouvelles :

- ▶ Si E_1 et E_2 sont des expressions, alors $E_1 + E_2$ est une expression.
Définit le langage $L(E_1 + E_2) = L(E_1) \cup L(E_2)$.
Exemple : $L(a + b + e) = \{a, b, e\}$ (3 mots).
- ▶ Si E_1 et E_2 sont des régulières, alors $E_1 E_2$ est une expression.
Langage $L(E_1 E_2) = \{w_1 w_2 \mid w_1 \in L(E_1) \text{ et } w_2 \in L_2\}$.
Exemple : $L((a + b)(aa + ba + b)) = \{aaaa, aba, ab, baa, bba, bb\}$ (6 mots).
- ▶ si E est une expression, alors E^* est une expression régulière.
Langage $L(E^*) = \{w_1 \cdots w_n \mid n \in \mathbb{N} \text{ et } w_1, \dots, w_n \in L(E)\}$.
Exemple : $L((ab)^*) = \{\varepsilon, ab, abab, ababab, abababab, \dots\}$ (infini).

Le mot « ε » est le **mot vide** (la chaîne de caractère de longueur 0).

Quelques exemples

Décrire les langages suivants (sur l'alphabet $A = \{a, b\}$) en français :

- Expression $(a + b)^*a(a + b)^*a(a + b)^*b(a + b)^*$.

Quelques exemples

Décrire les langages suivants (sur l'alphabet $A = \{a, b\}$) en français :

- Expression $(a + b)^*a(a + b)^*a(a + b)^*b(a + b)^*$.
Langage des mots qui contiennent le sous-mot aab .

Quelques exemples

Décrire les langages suivants (sur l'alphabet $A = \{a, b\}$) en français :

- ▶ Expression $(a + b)^*a(a + b)^*a(a + b)^*b(a + b)^*$.
Langage des mots qui contiennent le sous-mot aab .
- ▶ Expression $((a + b)(a + b))^*$:

Quelques exemples

Décrire les langages suivants (sur l'alphabet $A = \{a, b\}$) en français :

- ▶ Expression $(a + b)^*a(a + b)^*a(a + b)^*b(a + b)^*$.
Langage des mots qui contiennent le sous-mot aab .
- ▶ Expression $((a + b)(a + b))^*$:
Langage des mots de longueur paire.

Quelques exemples

Décrire les langages suivants (sur l'alphabet $A = \{a, b\}$) en français :

- ▶ Expression $(a + b)^*a(a + b)^*a(a + b)^*b(a + b)^*$.
Langage des mots qui contiennent le sous-mot aab .
- ▶ Expression $((a + b)(a + b))^*$:
Langage des mots de longueur paire.
- ▶ Expression $a^*(b(aa)^*)^*a^*$:

Quelques exemples

Décrire les langages suivants (sur l'alphabet $A = \{a, b\}$) en français :

- ▶ Expression $(a + b)^*a(a + b)^*a(a + b)^*b(a + b)^*$.
Langage des mots qui contiennent le sous-mot aab .
- ▶ Expression $((a + b)(a + b))^*$:
Langage des mots de longueur paire.
- ▶ Expression $a^*(b(aa)^*)^*a^*$:
Langage des mots tels que le nombre de a entre deux b est toujours pair.

Quelques exemples

Décrire les langages suivants (sur l'alphabet $A = \{a, b\}$) en français :

- ▶ Expression $(a + b)^*a(a + b)^*a(a + b)^*b(a + b)^*$.
Langage des mots qui contiennent le sous-mot aab .
- ▶ Expression $((a + b)(a + b))^*$:
Langage des mots de longueur paire.
- ▶ Expression $a^*(b(aa)^*)^*a^*$:
Langage des mots tels que le nombre de a entre deux b est toujours pair.
- ▶ Expression $(b(a + b)^* + (a + b)^*aa(a + b)^* + (a + b)^*bb(a + b)^* + (a + b)^*a)$:

Quelques exemples

Décrire les langages suivants (sur l'alphabet $A = \{a, b\}$) en français :

- ▶ Expression $(a + b)^*a(a + b)^*a(a + b)^*b(a + b)^*$.
Langage des mots qui contiennent le sous-mot aab .
- ▶ Expression $((a + b)(a + b))^*$:
Langage des mots de longueur paire.
- ▶ Expression $a^*(b(aa)^*)^*a^*$:
Langage des mots tels que le nombre de a entre deux b est toujours pair.
- ▶ Expression $(b(a + b)^* + (a + b)^*aa(a + b)^* + (a + b)^*bb(a + b)^* + (a + b)^*a)$:
Mots qui ne sont **pas** de la forme $abab \cdots ab$.

Théorème de Kleene (1956)

Les langages réguliers (REG)

Un langage est **régulier** si il existe une expression régulière qui le définit.

Théorème de Kleene (1956)

Les langages réguliers (REG)

Un langage est **régulier** si il existe une expression régulière qui le définit.

Théorème de Kleene : $\text{REG} = \text{REC}$

Pour tout langage L les deux propriétés suivantes sont équivalentes :

1. L est régulier (défini par une expression régulière).
2. L est reconnaissable (reconnu par un automate).

Théorème de Kleene (1956)

Les langages réguliers (REG)

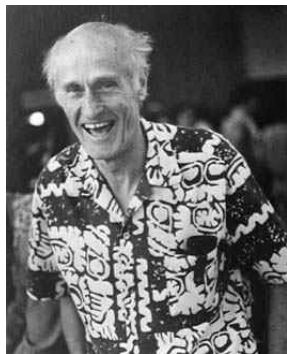
Un langage est **régulier** si il existe une expression régulière qui le définit.

Théorème de Kleene : $REG = REC$

Pour tout langage L les deux propriétés suivantes sont équivalentes :

1. L est régulier (défini par une expression régulière).
2. L est reconnaissable (reconnu par un automate).

On connaît de plusieurs **algorithmes de**
« **traduction** » dans les deux sens.



Des expressions aux automates : algorithme de Glushkov (1961)



Des expressions aux automates : algorithme de Glushkov (1961)



On part de l'expression $E_{\text{orig}} = a(ab + b)^*a^*$.

Des expressions aux automates : algorithme de Glushkov (1961)



On part de l'expression $E_{\text{orig}} = a(ab + b)^*a^*$.

Étape 1 : On renomme les lettres $E_{\text{renamed}} = a_1(a_2b_1 + b_2)^*a_3^*$.

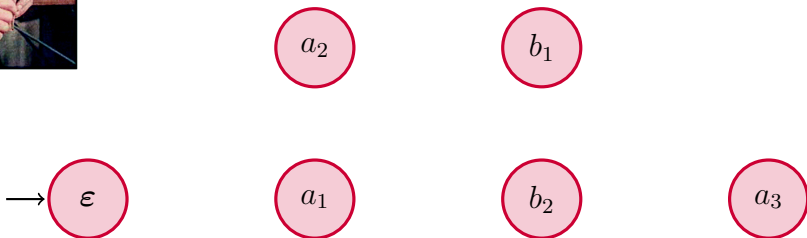
Des expressions aux automates : algorithme de Glushkov (1961)



On part de l'expression $E_{\text{orig}} = a(ab + b)^*a^*$.

Étape 1 : On renomme les lettres $E_{\text{renamed}} = a_1(a_2b_1 + b_2)^*a_3^*$.

Étape 2 : On crée un état par lettre plus un état initial ε .



Des expressions aux automates : algorithme de Glushkov (1961)

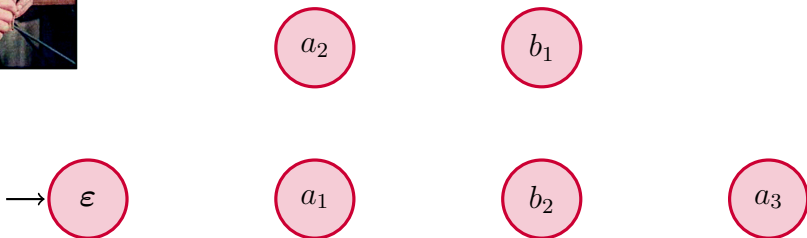


On part de l'expression $E_{\text{orig}} = a(ab + b)^*a^*$.

Étape 1 : On renomme les lettres $E_{\text{renamed}} = a_1(a_2b_1 + b_2)^*a_3^*$.

Étape 2 : On crée un état par lettre plus un état initial ε .

Étape 3 : Calcul des transitions et des états finaux.



Des expressions aux automates : algorithme de Glushkov (1961)

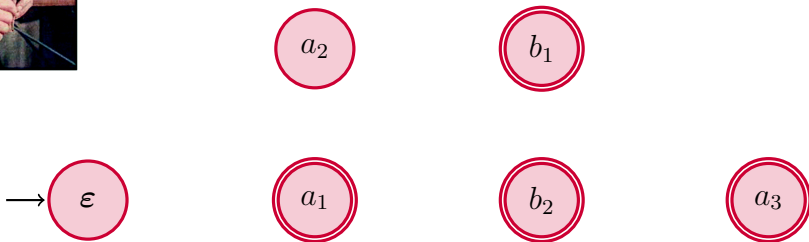


On part de l'expression $E_{\text{orig}} = a(ab + b)^*a^*$.

Étape 1 : On renomme les lettres $E_{\text{renamed}} = a_1(a_2b_1 + b_2)^*a_3^*$.

Étape 2 : On crée un état par lettre plus un état initial ε .

Étape 3 : Calcul des transitions et des états finaux.



Des expressions aux automates : algorithme de Glushkov (1961)

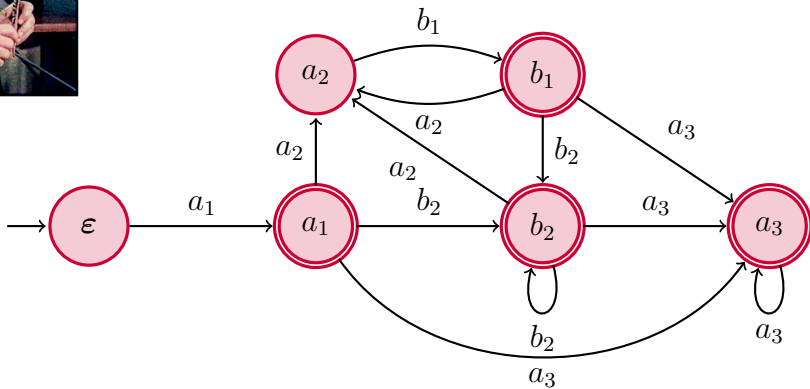


On part de l'expression $E_{\text{orig}} = a(ab + b)^*a^*$.

Étape 1 : On renomme les lettres $E_{\text{renamed}} = a_1(a_2b_1 + b_2)^*a_3^*$.

Étape 2 : On crée un état par lettre plus un état initial ϵ .

Étape 3 : Calcul des transitions et des états finaux.



Des expressions aux automates : algorithme de Glushkov (1961)

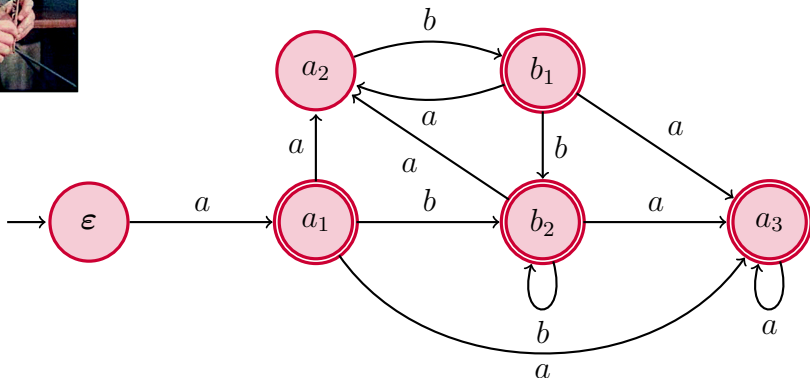


On part de l'expression $E_{\text{orig}} = a(ab + b)^*a^*$.

Étape 1 : On renomme les lettres $E_{\text{renamed}} = a_1(a_2b_1 + b_2)^*a_3^*$.

Étape 2 : On crée un état par lettre plus un état initial ε .

Étape 3 : Calcul des transitions et des états finaux.



Des expressions aux automates : algorithme de Glushkov (1961)

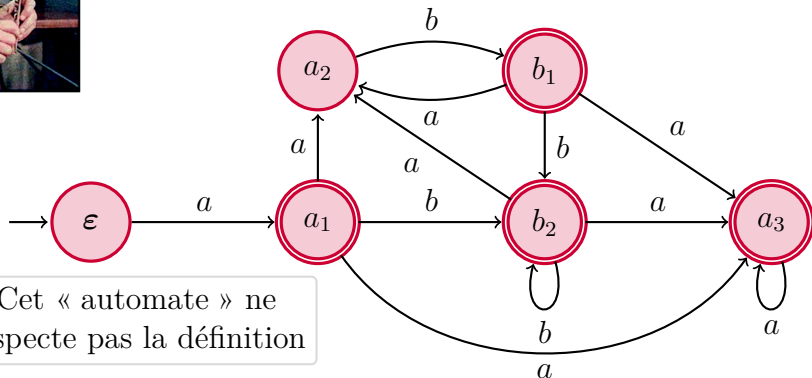


On part de l'expression $E_{\text{orig}} = a(ab + b)^*a^*$.

Étape 1 : On renomme les lettres $E_{\text{renamed}} = a_1(a_2b_1 + b_2)^*a_3^*$.

Étape 2 : On crée un état par lettre plus un état initial ϵ .

Étape 3 : Calcul des transitions et des états finaux.



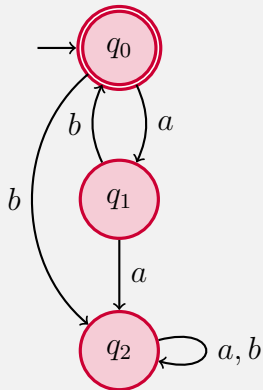
Cet « automate » ne respecte pas la définition

Déterminisme et non-déterminisme

Modèle qu'on a vu :

Automate déterministe (**DFA**) complet

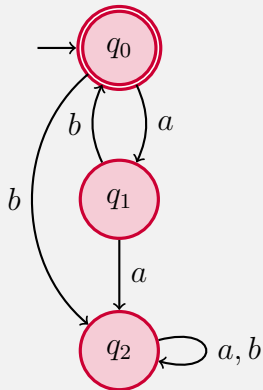
Fonction de transtion



Déterminisme et non-déterminisme

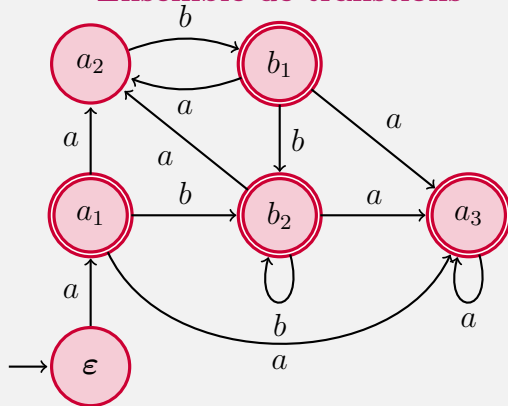
Modèle qu'on a vu :

Automate déterministe (**DFA**) complet
Fonction de transtion



Modèle utilisé par Glushkov :

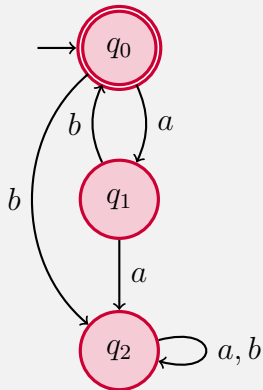
Automate non-déterministe (**NFA**)
Ensemble de transtions



Déterminisme et non-déterminisme

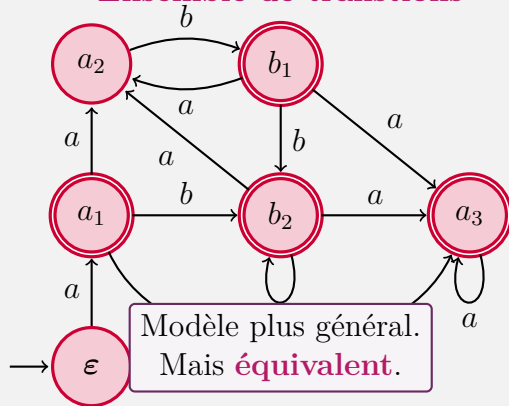
Modèle qu'on a vu :

Automate déterministe (**DFA**) complet
Fonction de transtion



Modèle utilisé par Glushkov :

Automate non-déterministe (**NFA**)
Ensemble de transtions



Théorème (Rabin-Scott 1959)

Pour tout **NFA**, on peut construire un **DFA** complet qui reconnaît le même langage.

Algorithme : **Construction des sous-ensembles**.



Théorème (Rabin-Scott 1959)



Pour tout **NFA**, on peut construire un **DFA** complet qui reconnaît le même langage.

Algorithme : **Construction des sous-ensembles.**

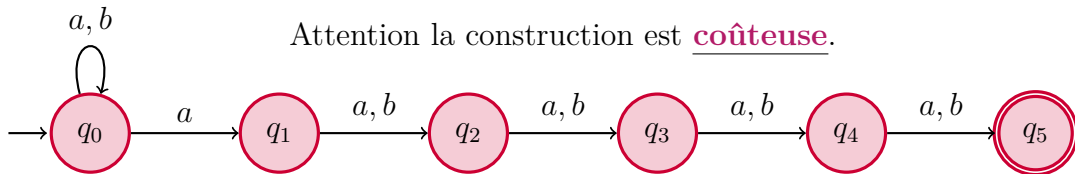
Attention la construction est coûteuse.

Déterminisation

Théorème (Rabin-Scott 1959)

Pour tout **NFA**, on peut construire un **DFA** complet qui reconnaît le même langage.

Algorithme : **Construction des sous-ensembles.**



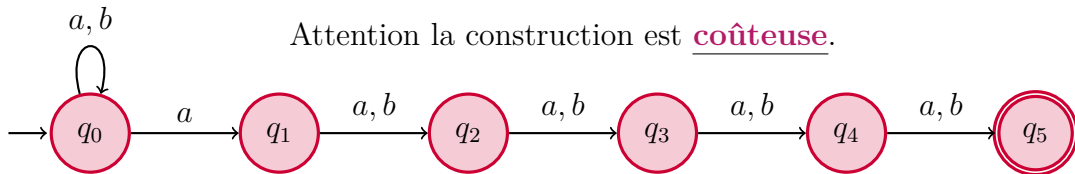
Déterminisation

Théorème (Rabin-Scott 1959)



Pour tout **NFA**, on peut construire un **DFA** complet qui reconnaît le même langage.

Algorithme : **Construction des sous-ensembles.**



Mots dont la 5^{ème} lettre avant la fin est un « **a** »

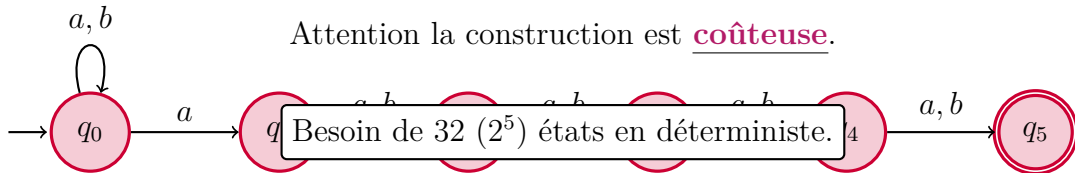
Déterminisation

Théorème (Rabin-Scott 1959)



Pour tout **NFA**, on peut construire un **DFA** complet qui reconnaît le même langage.

Algorithme : **Construction des sous-ensembles.**



Mots dont la 5ème lettre avant la fin est un « **a** »

Le projet

Que va-t'on faire ?

Le projet

Objectif principal

Recherche de motifs spécifiés avec une **expression régulière**.

Le projet

Objectif principal

Recherche de motifs spécifiés avec une **expression régulière**.

Partie 1 : Implémentation des automates (le plus gros du travail) :

- ▶ Fonctions de base (expressions).
- ▶ Fonctions de base (automates).
- ▶ Traduction expressions $\Rightarrow$ automates.
- ▶ Traduction automates $\Rightarrow$ expressions.
- ▶ Lecture d'un mot dans un automate.
- ▶ Opérations sur les automates.
- ▶ Déterminisation.

Le projet

Objectif principal

Recherche de motifs spécifiés avec une **expression régulière**.

Partie 1 : Implémentation des automates (le plus gros du travail) :

- ▶ Fonctions de base (expressions).
- ▶ Fonctions de base (automates).
- ▶ Traduction expressions $\Rightarrow$ automates.
- ▶ Traduction automates $\Rightarrow$ expressions.
- ▶ Lecture d'un mot dans un automate.
- ▶ Opérations sur les automates.
- ▶ Déterminisation.

Partie 2 : Implémentation des algorithmes de recherche eux-mêmes.
(**basés sur la partie I**).

Le projet

Objectif principal

Recherche de motifs spécifiés avec une **expression régulière**.

Partie 1 : Implémentation des automates (le plus gros du travail) :

- ▶ Fonctions de base (expressions).
- ▶ Fonctions de base (automates).
- ▶ Traduction expressions $\Rightarrow$ automates.
- ▶ Traduction automates $\Rightarrow$ expressions.
- ▶ Lecture d'un mot dans un automate.
- ▶ Opérations sur les automates.
- ▶ Déterminisation.

Partie 2 : Implémentation des algorithmes de recherche eux-mêmes.
(**basés sur la partie I**).

La partie « interface utilisateur » vous est donnée.