Théorie des langages et Compilation : Analyse Lexicale

Didier Lime

École Centrale de Nantes – LS2N

Année 2017 – 2018

Plan

Plan

Analyse lexicale

- Le but de l'analyse lexicale est de transformer une suite de caractères en mots (lexèmes ou *tokens*).
- L'analyse syntaxique vient ensuite examiner l'agencement des mots
- L'analyseur syntaxique est strictement plus puissant que l'analyseur lexical
- La passe « analyse lexicale » est faite par commodité

Première idée

Exemple

```
void readIdentifier(char* token, char * s) {
    int i = 0;
    char c;
    while ((c = getc()) != EOF && c >= 'a' && c <= 'z') {
        s[i] = c;
        i++;
    }
    if (!contains(keyword_table, s))
       strcpy(token, IDENTIFIER);
    else
       strcpy(token, KEYWORD);
}
void readInteger(char* token, char * id) { ... }
```

(Inspiré de l'exemple du cours de Compilation de J. Ferber, Montpellier)

Première idée

- Chaque fonction reconnaît assez bien les mots composés par la répétition C^* de caractères d'une même classe C
- Pour des constructions plus compliquées les fonctions deviennent lourdes
- Quelle fonction appeler ?
- Nécessité d'une approche **globale** et donc **générique**

Plan

Langages Formels

Lettres et mots :

- Soit un ensemble fini Σ. On l'appelle **alphabet** ;
- Les éléments de Σ sont appelés **lettres** ;
- Les **mots** sont des séquences de lettres $a_1 \dots a_N$;
- On note Σ^* l'ensemble des mots sur Σ
- On note uv le mot obtenu par **concaténation** de deux mots u et v ;
  associative
- La concaténation est associative ;
- On note son élément neutre ϵ, appelé **mot vide** ;

Langages Formels

Lettres et mots :

- Soit un ensemble fini Σ. On l'appelle **alphabet** ;
- Les éléments de Σ sont appelés **lettres** ;
- Les **mots** sont des séquences de lettres $a_1 \dots a_N$;
- On note Σ^* l'ensemble des mots sur Σ
- On note uv le mot obtenu par **concaténation** de deux mots u et v ;
- La concaténation est associative ;
- On note son élément neutre ϵ, appelé **mot vide** ;

Langages :

- Un **langage** sur Σ est un sous-ensemble de Σ^* ;
- Soient U et V deux langages sur Σ, $UV = \{uv | u \in U, v \in V\}$;
- $U^* = \{u_0 \dots u_n | n \geq 0, \forall i, u_i \in U\}$;

Langages Réguliers

- On veut **reconnaître** des langages ;
- Le problème est a priori **indécidable** ;
- On doit donc se contenter de langages particuliers ;
- On veut en plus que l'analyse soit **efficace** : on choisit les **langages réguliers**.

Langages Réguliers

- On veut **reconnaître** des langages ;
- Le problème est a priori **indécidable** ;
- On doit donc se contenter de langages particuliers ;
- On veut en plus que l'analyse soit **efficace** : on choisit les **langages réguliers**.

Définition (Langages réguliers)

Un langage est dit **régulier** s'il est **reconnu** par un **automate fini**.

Plan

Automates Finis Déterministes (DFA)

Définition (DFA)

Un automate fini déterministe (DFA) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'**états** ;
- Σ est un **alphabet** fini ;
- $\delta : Q \times \Sigma \to Q$ est une fonction partielle appelée **fonction de transition** ;
- $q_0 \in Q$ est l'**état initial** ;
- $F \subseteq Q$ est l'ensemble des **états accepteurs** (ou terminaux).

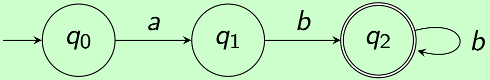
Automates Finis Déterministes (DFA)

Définition (DFA)

Un automate fini déterministe (DFA) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'**états** ;
- Σ est un **alphabet** fini ;
- $\delta : Q \times \Sigma \to Q$ est une fonction partielle appelée **fonction de transition** ;
- $q_0 \in Q$ est l'**état initial** ;
- $F \subseteq Q$ est l'ensemble des **états accepteurs** (ou terminaux).

Exemple



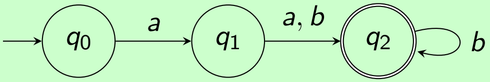
Automates Finis Déterministes (DFA)

Définition (DFA)

Un automate fini déterministe (DFA) est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'**états** ;
- Σ est un **alphabet** fini ;
- $\delta : Q \times \Sigma \to Q$ est une fonction partielle appelée **fonction de transition** ;
- $q_0 \in Q$ est l'**état initial** ;
- $F \subseteq Q$ est l'ensemble des **états accepteurs** (ou terminaux).

Exemple



Langage reconnu par un DFA

Soit un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$

- δ peut être étendu sur $Q \times \Sigma^*$:
  $\forall w \in \Sigma^*$ t.q. $|w| \geq 2, \exists a \in \Sigma, w' \in \Sigma^*$ t.q. $w = aw'$.
  Alors, $\delta(q, w) = \delta(\delta(q, a), w')$;

Langage reconnu par un DFA

Soit un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$

- δ peut être étendu sur $Q \times \Sigma^*$:
  $\forall w \in \Sigma^*$ t.q. $|w| \geq 2, \exists a \in \Sigma, w' \in \Sigma^*$ t.q. $w = aw'$.
  Alors, $\delta(q, w) = \delta(\delta(q, a), w')$;
- Un mot $w \in \Sigma^*$ est **accepté** (ou reconnu) par $\mathcal{A}$ si $\delta(q_0, w) \in F$.

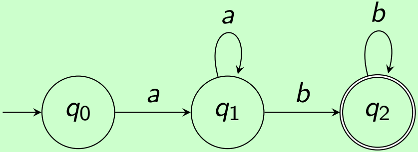
Langage reconnu par un DFA

Soit un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$

- δ peut être étendu sur $Q \times \Sigma^*$:
  $\forall w \in \Sigma^*$ t.q. $|w| \geq 2, \exists a \in \Sigma, w' \in \Sigma^*$ t.q. $w = aw'$.
  Alors, $\delta(q, w) = \delta(\delta(q, a), w')$;
- Un mot $w \in \Sigma^*$ est **accepté** (ou reconnu) par $\mathcal{A}$ si $\delta(q_0, w) \in F$.
- Le **langage** $\mathcal{L}(\mathcal{A})$ de $\mathcal{A}$ est l'ensemble des mots acceptés par $\mathcal{A}$.

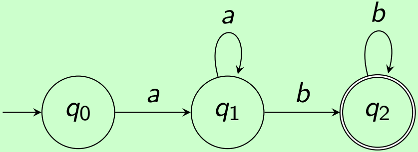
Langage reconnu par un DFA

Exemple



Langage reconnu par un DFA

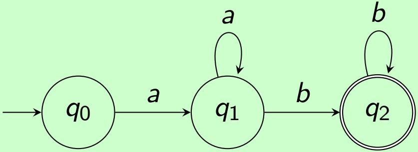
Exemple



$$\mathcal{L}(\mathcal{A}) = \{a^m b^n | m, n \in \mathbb{N} \setminus \{0\}\}$$

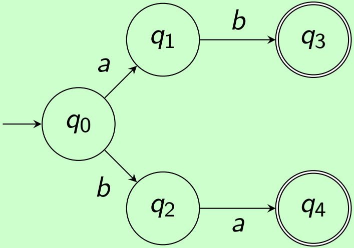
Langage reconnu par un DFA

Exemple



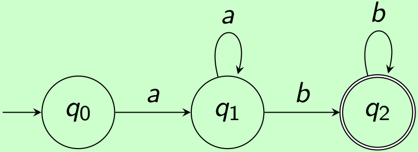
$$\mathcal{L}(\mathcal{A}) = \{a^m b^n | m, n \in \mathbb{N} \setminus \{0\}\}$$

Exemple



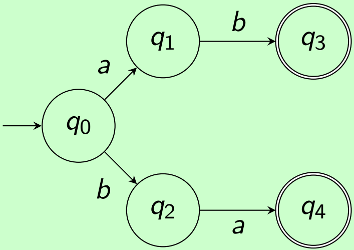
Langage reconnu par un DFA

Exemple



$$\mathcal{L}(\mathcal{A}) = \{a^m b^n | m, n \in \mathbb{N} \setminus \{0\}\}$$

Exemple



$$\mathcal{L}(\mathcal{A}) = \{ab\} \cup \{ba\} = \{ab, ba\}$$

Expressions Régulières

Pour décrire les langages, on utilise le formalisme des expressions régulières (aussi appelées rationnelles).

Définition (Expressions Régulières (RE))

Les **expressions régulières** sont définies inductivement par :

- $\emptyset$ est une expression régulière
- ϵ est une expression régulière
- $\forall a \in \Sigma, a$ est un expression régulière
- Pour toutes expressions régulières r et s, rs et $r|s$ sont des expressions régulières
- Pour toute expression régulière r, r^* est une expression régulière

Expressions Régulières

À chaque RE, on associe un langage :

Définition (Langage associé aux RE)

Le langage $\mathcal{L}(r)$ d'une RE r est défini par :

- $\mathcal{L}(\emptyset) = \emptyset$;
- $\mathcal{L}(\epsilon) = \{\epsilon\}$;
- $\forall a \in \Sigma, \mathcal{L}(a) = \{a\}$;
- Pour toutes RE r et s, $\mathcal{L}(rs) = \mathcal{L}(r)\mathcal{L}(s)$ et $\mathcal{L}(r|s) = \mathcal{L}(r) \cup \mathcal{L}(s)$;
- Pour toute RE r, $\mathcal{L}(r^*) = \mathcal{L}(r)^*$.

Expressions Régulières

À chaque RE, on associe un langage :

Définition (Langage associé aux RE)

Le langage $\mathcal{L}(r)$ d'une RE r est défini par :

- $\mathcal{L}(\emptyset) = \emptyset$;
- $\mathcal{L}(\epsilon) = \{\epsilon\}$;
- $\forall a \in \Sigma, \mathcal{L}(a) = \{a\}$;
- Pour toutes RE r et s, $\mathcal{L}(rs) = \mathcal{L}(r)\mathcal{L}(s)$ et $\mathcal{L}(r|s) = \mathcal{L}(r) \cup \mathcal{L}(s)$;
- Pour toute RE r, $\mathcal{L}(r^*) = \mathcal{L}(r)^*$.

L'**union**, la **concaténation** et l'**étoile** sont les opérations dites **régulières** sur les langages.

Expressions Régulières et Langages Réguliers

Théorème (Théorème de Kleene 1)

Pour toute expression régulière r, son langage $\mathcal{L}(r)$ est régulier.

Expressions Régulières et Langages Réguliers

Théorème (Théorème de Kleene 1)

Pour toute expression régulière r, son langage $\mathcal{L}(r)$ est régulier.

Théorème (Théorème de Kleene 2)

Pour tout langage régulier $\mathcal{L}$, il existe une expression régulière r telle que $\mathcal{L}(r) = \mathcal{L}$.

Expressions Régulières Étendues

À partir des expressions précédentes, on peut définir les **raccourcis** suivants :

- $[abc] = a|b|c$;
- Si Σ est ordonné, $[a_1 - a_2] = \{b \in \Sigma | a_1 \leq b \leq a_2\}$;
- $r? = r|\epsilon$;
- $r^+ = rr^*$;
- $[\hat{}abc] = \Sigma \setminus \{a, b, c\}$;
- $[\hat{}a_1 - a_2] = \Sigma \setminus [a_1 - a_2]$;
- $. = \Sigma$ et $* = \Sigma^*$.

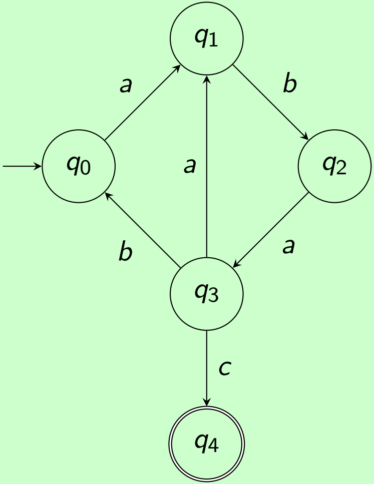
Calcul du Langage d'un DFA

Soit un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$.

- On note $\mathcal{L}_{ij}$ le langage de $\mathcal{A}$ obtenu entre l'état q_i et l'état q_j ;
- Si $F = \{q_N, \ldots, q_{N+k}\}$ alors $\mathcal{L}(\mathcal{A}) = \mathcal{L}_{0N}|\cdots|\mathcal{L}_{0N+k}$;
- Si $q_{i+1}, \ldots, q_{i+k}$ sont les sucesseurs de q_i alors $\mathcal{L}_{ij} = \mathcal{L}_{ii+1}\mathcal{L}_{i+1j}|\cdots|\mathcal{L}_{ii+k}\mathcal{L}_{i+kj}$;
- S'il existe $a_1, \ldots, a_k$ tels que $\delta(q_i, a_1) = \cdots = \delta(q_i, a_k) = q_j$ alors $\mathcal{L}_{ij} = a_1|\cdots|a_k$;
- **Si $\mathcal{L}_{ij} = r\mathcal{L}_{ij}|s$ alors $\mathcal{L}_{ij} = r^*s$** ;
- Si q_i n'a pas de boucle ($a \in \Sigma$ t.q. $\delta(q_i, a) = q_i$) alors $\mathcal{L}_{ii} = \epsilon$.

Calcul du Langage d'un DFA

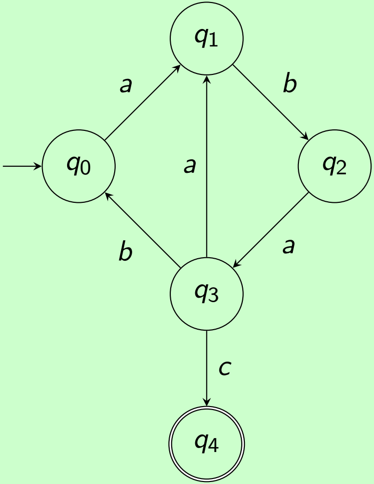
Exemple



$$\mathcal{L}(\mathcal{A}) = \mathcal{L}_{04}$$

Calcul du Langage d'un DFA

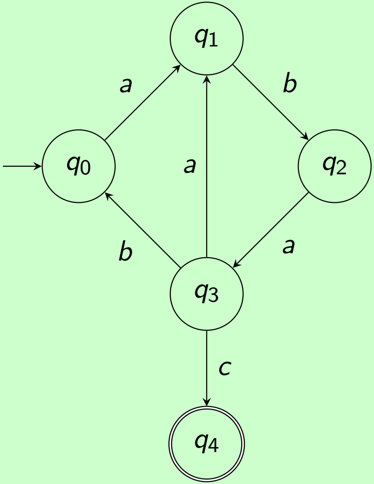
Exemple



$$\mathcal{L}(\mathcal{A}) = \mathcal{L}_{04}$$
$$\mathcal{L}(\mathcal{A}) = a\mathcal{L}_{14}$$

Calcul du Langage d'un DFA

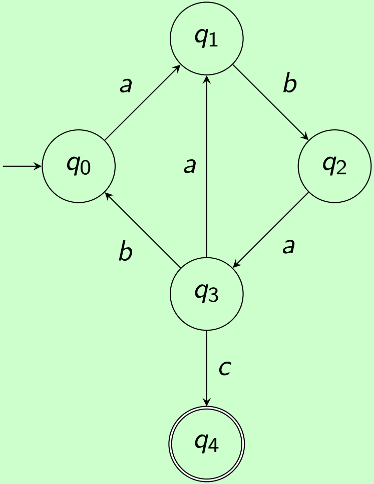
Exemple



$$\mathcal{L}(\mathcal{A}) = \mathcal{L}_{04}$$
$$\mathcal{L}(\mathcal{A}) = a\mathcal{L}_{14}$$
$$\mathcal{L}(\mathcal{A}) = ab\mathcal{L}_{24}$$

Calcul du Langage d'un DFA

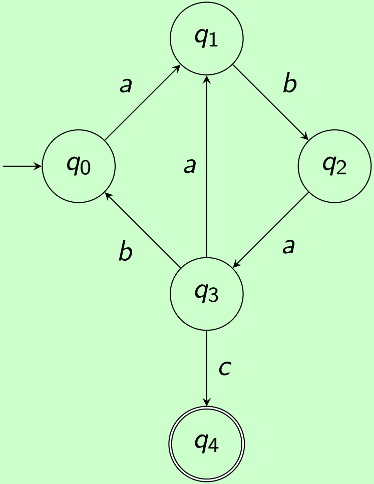
Exemple



$$\mathcal{L}(\mathcal{A}) = \mathcal{L}_{04}$$
$$\mathcal{L}(\mathcal{A}) = a\mathcal{L}_{14}$$
$$\mathcal{L}(\mathcal{A}) = ab\mathcal{L}_{24}$$
$$\mathcal{L}(\mathcal{A}) = aba\mathcal{L}_{34}$$

Calcul du Langage d'un DFA

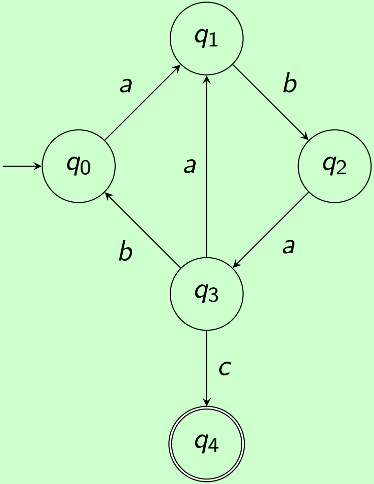
Exemple



$\mathcal{L}(\mathcal{A}) = \mathcal{L}_{04}$
$\mathcal{L}(\mathcal{A}) = a\mathcal{L}_{14}$
$\mathcal{L}(\mathcal{A}) = ab\mathcal{L}_{24}$
$\mathcal{L}(\mathcal{A}) = aba\mathcal{L}_{34}$
$\mathcal{L}_{34} = c\mathcal{L}_{44} | a\mathcal{L}_{14} | b\mathcal{L}_{04}$

Calcul du Langage d'un DFA

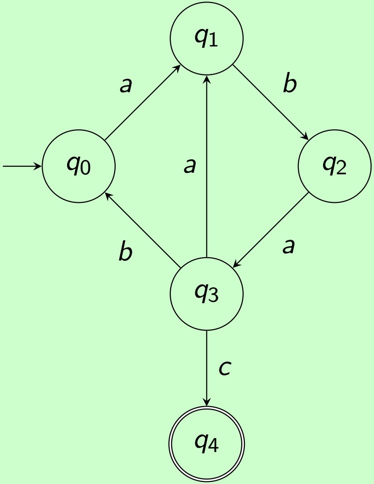
Exemple



$$\mathcal{L}(\mathcal{A}) = \mathcal{L}_{04}$$
$$\mathcal{L}(\mathcal{A}) = a\mathcal{L}_{14}$$
$$\mathcal{L}(\mathcal{A}) = ab\mathcal{L}_{24}$$
$$\mathcal{L}(\mathcal{A}) = aba\mathcal{L}_{34}$$
$$\mathcal{L}_{34} = c\mathcal{L}_{44}|a\mathcal{L}_{14}|b\mathcal{L}_{04}$$
$$\mathcal{L}_{34} = c\epsilon|a\mathcal{L}_{14}|ba\mathcal{L}_{14}$$

Calcul du Langage d'un DFA

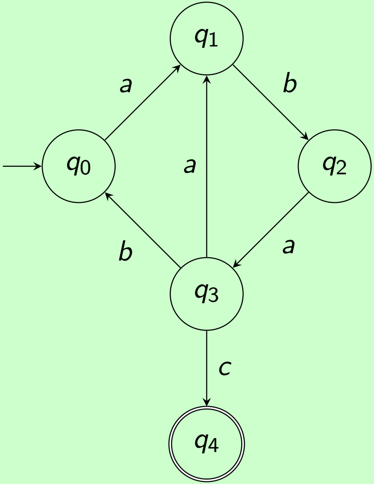
Exemple



$$\mathcal{L}(\mathcal{A}) = \mathcal{L}_{04}$$
$$\mathcal{L}(\mathcal{A}) = a\mathcal{L}_{14}$$
$$\mathcal{L}(\mathcal{A}) = ab\mathcal{L}_{24}$$
$$\mathcal{L}(\mathcal{A}) = aba\mathcal{L}_{34}$$
$$\mathcal{L}_{34} = c\mathcal{L}_{44}|a\mathcal{L}_{14}|b\mathcal{L}_{04}$$
$$\mathcal{L}_{34} = c\epsilon|a\mathcal{L}_{14}|ba\mathcal{L}_{14}$$
$$\mathcal{L}_{34} = c|(\epsilon|b)a\mathcal{L}_{14}$$

Calcul du Langage d'un DFA

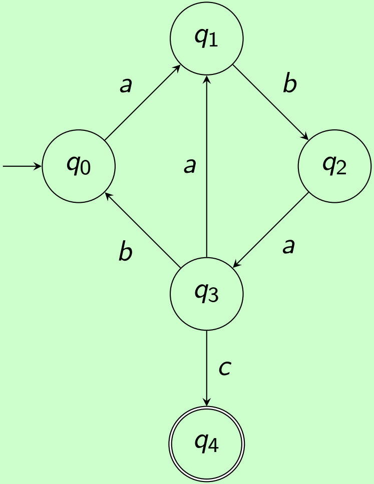
Exemple



$$\mathcal{L}(\mathcal{A}) = \mathcal{L}_{04}$$
$$\mathcal{L}(\mathcal{A}) = a\mathcal{L}_{14}$$
$$\mathcal{L}(\mathcal{A}) = ab\mathcal{L}_{24}$$
$$\mathcal{L}(\mathcal{A}) = aba\mathcal{L}_{34}$$
$$\mathcal{L}_{34} = c\mathcal{L}_{44}|a\mathcal{L}_{14}|b\mathcal{L}_{04}$$
$$\mathcal{L}_{34} = c\epsilon|a\mathcal{L}_{14}|ba\mathcal{L}_{14}$$
$$\mathcal{L}_{34} = c|(\epsilon|b)a\mathcal{L}_{14}$$
$$\mathcal{L}_{34} = c|(\epsilon|b)aba\mathcal{L}_{34}$$

Calcul du Langage d'un DFA

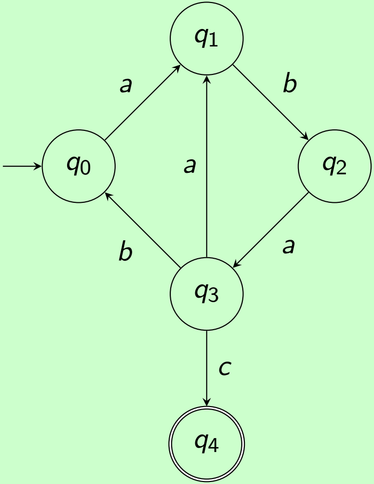
Exemple



$\mathcal{L}(\mathcal{A}) = \mathcal{L}_{04}$
$\mathcal{L}(\mathcal{A}) = a\mathcal{L}_{14}$
$\mathcal{L}(\mathcal{A}) = ab\mathcal{L}_{24}$
$\mathcal{L}(\mathcal{A}) = aba\mathcal{L}_{34}$
$\mathcal{L}_{34} = c\mathcal{L}_{44}|a\mathcal{L}_{14}|b\mathcal{L}_{04}$
$\mathcal{L}_{34} = c\epsilon|a\mathcal{L}_{14}|ba\mathcal{L}_{14}$
$\mathcal{L}_{34} = c|(\epsilon|b)a\mathcal{L}_{14}$
$\mathcal{L}_{34} = c|(\epsilon|b)aba\mathcal{L}_{34}$
$\mathcal{L}_{34} = ((\epsilon|b)aba)^*c$

Calcul du Langage d'un DFA

Exemple

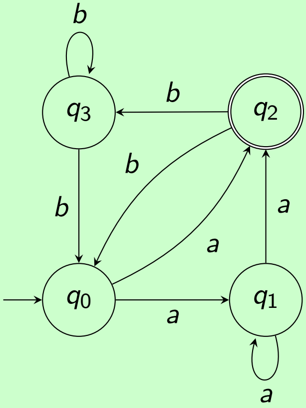


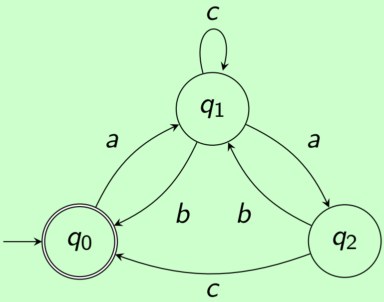
$$\mathcal{L}(\mathcal{A}) = \mathcal{L}_{04}$$
$$\mathcal{L}(\mathcal{A}) = a\mathcal{L}_{14}$$
$$\mathcal{L}(\mathcal{A}) = ab\mathcal{L}_{24}$$
$$\mathcal{L}(\mathcal{A}) = aba\mathcal{L}_{34}$$
$$\mathcal{L}_{34} = c\mathcal{L}_{44}|a\mathcal{L}_{14}|b\mathcal{L}_{04}$$
$$\mathcal{L}_{34} = c\epsilon|a\mathcal{L}_{14}|ba\mathcal{L}_{14}$$
$$\mathcal{L}_{34} = c|(\epsilon|b)a\mathcal{L}_{14}$$
$$\mathcal{L}_{34} = c|(\epsilon|b)aba\mathcal{L}_{34}$$
$$\mathcal{L}_{34} = ((\epsilon|b)aba)^*c$$
$$\mathcal{L}(\mathcal{A}) = aba((\epsilon|b)aba)^*c$$

Calcul du Langage : Exercice

Exercice

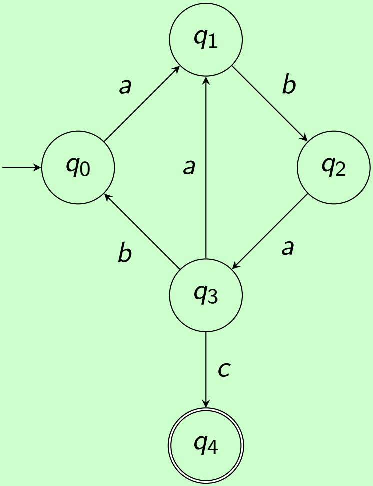
Calculez des expressions rationnelles représentant les langages des automates (non déterministe pour celui de gauche) suivants :





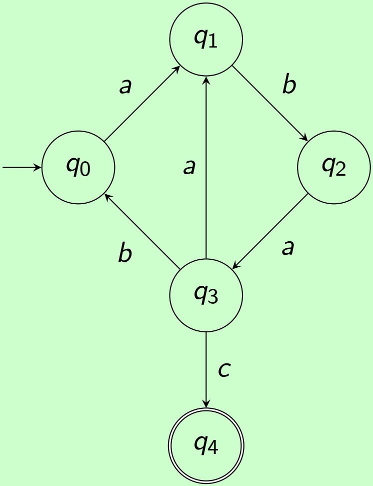
Implémentation d'un DFA

Exemple



Implémentation d'un DFA

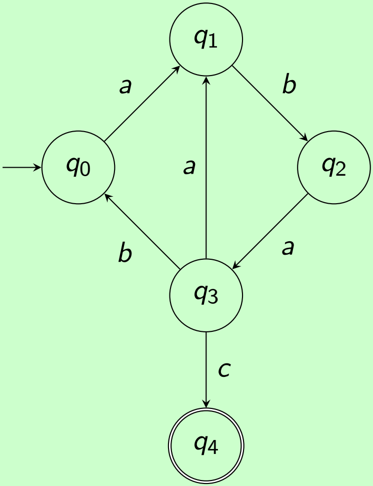
Exemple




```
int etat = 0; char t;
scanf("%c", &t);
while (t<='z' && t >='a') {
    switch (etat) {
        case 3:
            switch (t) {
                case 'a': etat=1; break;
                case 'b': etat=0; break;
                case 'c': etat=4; break;
                default: printf("erreur");
            } break;
        ...
        case 0:
            switch (t) { ... } break;
    }
    scanf("%c", &t);
}
if (etat == 4) printf("reconnu !");
```

Implémentation d'un DFA

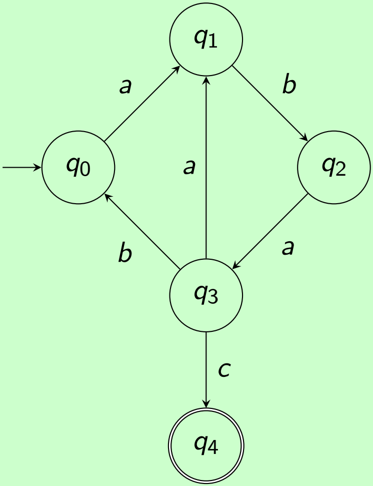
Exemple




```
int etat = 0; char t;
scanf("%c", &t);
while (t<='z' && t >='a') {
    switch (t) {
        case 'a':
            switch (etat) {
                case '0': etat=1; break;
                case '2': etat=3; break;
                case '3': etat=1; break;
                default: printf("erreur");
            } break;
        ...
        case 'c':
            switch (t) { ... } break;
    }
    scanf("%c", &t);
}
if (etat == 4) printf("reconnu !");
```

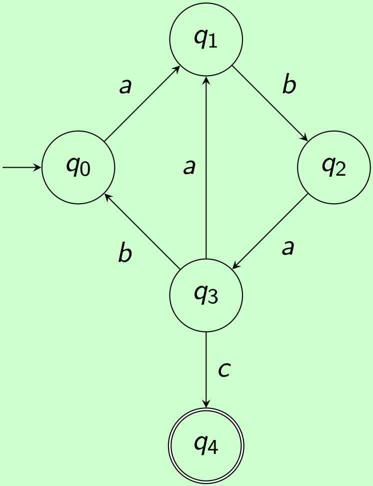
Implémentation d'un DFA

Exemple



Implémentation d'un DFA

Exemple




```
const int delta[5][3] = {
    { 1,-1,-1},
    {-1, 2,-1},
    { 3,-1,-1},
    { 1, 0, 4},
    {-1,-1,-1}};
int etat = 0;
char t;
scanf("%c", &t);
while (t<='z' && t >='a') {
    etat = delta[etat][t-'a'];
    if (etat == -1) printf("erreur!");
    scanf("%c", &t);
}
if (etat == 4) printf("reconnu !");
```

Complétion d'un DFA

Définition (Complétude)

Un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ est dit **complet** si δ est définie pour tout $q \in Q$ et tout $a \in \Sigma$ (δ est une fonction totale).

Complétion d'un DFA

Définition (Complétude)

Un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ est dit **complet** si δ est définie pour tout $q \in Q$ et tout $a \in \Sigma$ (δ est une fonction totale).

- Pour rendre un automate complet, une méthode est de rajouter un état « erreur » q_{err} qui prend en compte les comportements imprévus ;
- Si, pour $q \in Q$ et $a \in \Sigma$, $\delta(q, a)$ n'est pas défini alors on définit $\delta(q, a) = q_{err}$;
- C'est fait implicitement dans les implémentations précédentes.

Premier Bilan

- On sait qu'un langage régulier peut-être reconnu par un DFA ;
- On sait implémenter un DFA ;
- On sait exprimer un langage régulier par une expression régulière ;
- Étant donné un DFA, on sait calculer son langage

Premier Bilan

- On sait qu'un langage régulier peut-être reconnu par un DFA ;
- On sait implémenter un DFA ;
- On sait exprimer un langage régulier par une expression régulière ;
- Étant donné un DFA, on sait calculer son langage

Reste :

- Comment savoir si un langage donné est régulier ?
- Comment construire l'automate reconnaissant une RE particulière ?

Prouver qu'un langage est régulier

Pour prouver qu'un langage $\mathcal{L}$ est régulier :

- Exprimer $\mathcal{L}$ comme le résultat d'opérations régulières sur des langages réguliers

Prouver qu'un langage est régulier

Pour prouver qu'un langage $\mathcal{L}$ est régulier :

- Exprimer $\mathcal{L}$ comme le résultat d'opérations régulières sur des langages réguliers
- Construire l'automate qui reconnaît $\mathcal{L}$ (preuve constructive)

Prouver qu'un langage est régulier

Pour prouver qu'un langage $\mathcal{L}$ est régulier :

- Exprimer $\mathcal{L}$ comme le résultat d'opérations régulières sur des langages réguliers
- Construire l'automate qui reconnaît $\mathcal{L}$ (preuve constructive)

Exercice

- Montrer que $\mathcal{L} = \{a^m b^n | m, n \in \mathbb{N}\}$ est régulier ;
- Montrer que tout langage fini est régulier ;
- Montrer que si $\mathcal{L}$ est un langage régulier alors son inverse (l'ensemble de ses éléments inversés) est un langage régulier (p.ex. l'inverse de *abc* est *cba*) ;
- Idem pour son complémentaire.

Prouver qu'un langage n'est pas régulier

Lemme (Lemme de l'étoile (Pumping Lemma))

Pour tout langage régulier $\mathcal{L}$, il existe un entier n tel que pour tout mot w de longueur $|w|$ supérieure à n, il existe des mots x, u et y de Σ^* tels que $u \neq \epsilon$, $|xu| \leq n$, $w = xuy$ et $\forall k \geq 0, xu^k y \in \mathcal{L}$.

Exercice

Prouver que $\{a^n b^n, n \in \mathbb{N}\}$ n'est pas régulier

Prouver qu'un langage n'est pas régulier

Lemme (Lemme de l'étoile (Pumping Lemma))

Pour tout langage régulier $\mathcal{L}$, il existe un entier n tel que pour tout mot w de longueur $|w|$ supérieure à n, il existe des mots x, u et y de Σ^* tels que $u \neq \epsilon$, $|xu| \leq n$, $w = xuy$ et $\forall k \geq 0, xu^k y \in \mathcal{L}$.

Exercice

Prouver que $\{a^n b^n, n \in \mathbb{N}\}$ n'est pas régulier

Exercice

1. Prouver avec le lemme de l'étoile que $\{a^m b^n, m > n\}$ n'est pas régulier ;
2. Prouver avec le lemme de l'étoile que $\{a^m b^n, m \neq n\}$ n'est pas régulier ;
3. Prouver que si L_1 et L_2 sont réguliers alors $L_1 \cap L_2$ aussi ;
4. En déduire une autre preuve que $\{a^m b^n, m \neq n\}$ n'est pas régulier.

Plan

Traduire une RE en un DFA

Théorème (Theorème de Kleene (rappel))

Les expressions régulières décrivent exactement les langages réguliers.

Traduire une RE en un DFA

Théorème (Theorème de Kleene (rappel))

Les expressions régulières décrivent exactement les langages réguliers.

Définition (Langages réguliers (rappel))

Un langage est dit **régulier** s'il est **reconnu** par un **automate fini**.

Traduire une RE en un DFA

Théorème (Theorème de Kleene (rappel))

Les expressions régulières décrivent exactement les langages réguliers.

Définition (Langages réguliers (rappel))

Un langage est dit **régulier** s'il est **reconnu** par un **automate fini**.

Traduire une RE en DFA n'est pas pratique : on va passer par les **automates finis non déterministes**.

Automates Finis Non Déterministes (NFA)

Définition (NFA)

Un automate fini non déterministe (NFA) est un quintuplet $(Q, \Sigma, \delta, Q_0, F)$ où :

- Q est un ensemble fini d'**états** ;
- Σ est un **alphabet** fini ;
- $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathbf{2^Q}$ est une fonction partielle appelée **fonction de transition** ;
- $Q_0 \subseteq Q$ est l'**ensemble des états initiaux** ;
- $F \subseteq Q$ est l'ensemble des **états accepteurs** (ou terminaux).

Automates Finis Non Déterministes (NFA)

Définition (NFA)

Un automate fini non déterministe (NFA) est un quintuplet $(Q, \Sigma, \delta, Q_0, F)$ où :

- Q est un ensemble fini d'**états** ;
- Σ est un **alphabet** fini ;
- $\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathbf{2^Q}$ est une fonction partielle appelée **fonction de transition** ;
- $Q_0 \subseteq Q$ est l'**ensemble des états initiaux** ;
- $F \subseteq Q$ est l'ensemble des **états accepteurs** (ou terminaux).

Différences avec les DFA :

- On peut avoir plusieurs états initiaux et plusieurs transitions avec la même étiquette sortant d'un même état ;
- Les transitions ϵ ne changent pas le mot qui est reconnu

Langage d'un NFA

- La relation de transition d'un NFA est :

$$\delta : Q \times \Sigma \cup \{\epsilon\} \to \mathbf{2^Q}$$

Langage d'un NFA

- La relation de transition d'un NFA est :

$$\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathbf{2^Q}$$

- Et on a un ensemble d'états initiaux :

$$Q_0 \subseteq Q$$

Langage d'un NFA

- La relation de transition d'un NFA est :

$$\delta : Q \times \Sigma \cup \{\epsilon\} \to \mathbf{2^Q}$$

- Et on a un ensemble d'états initiaux :

$$Q_0 \subseteq Q$$

- En exécutant l'automate sur un mot w on obtient donc un **ensemble** d'états :

$$\delta(Q_0, w) \in 2^Q$$

Langage d'un NFA

- La relation de transition d'un NFA est :

$$\delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathbf{2^Q}$$

- Et on a un ensemble d'états initiaux :

$$Q_0 \subseteq Q$$

- En exécutant l'automate sur un mot w on obtient donc un **ensemble** d'états :

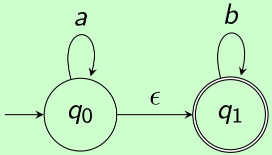
$$\delta(Q_0, w) \in 2^Q$$

- Le mot est accepté si **au moins** l'un de ces états est final :

$$\delta(Q_0, w) \cap F \neq \emptyset$$

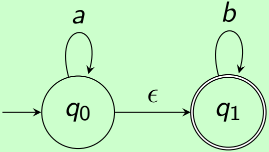
Automates Finis Non Déterministes (NFA)

Exemple



Automates Finis Non Déterministes (NFA)

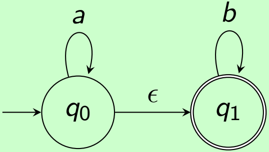
Exemple



$$\mathcal{L}(\mathcal{A}) = a^* b^*$$

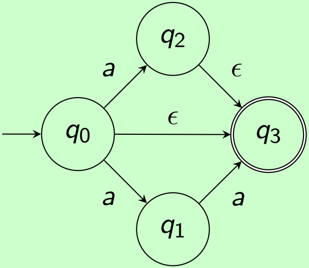
Automates Finis Non Déterministes (NFA)

Exemple



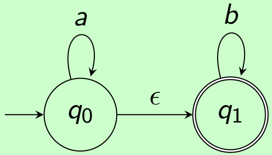
$\mathcal{L}(\mathcal{A}) = a^* b^*$

Exemple



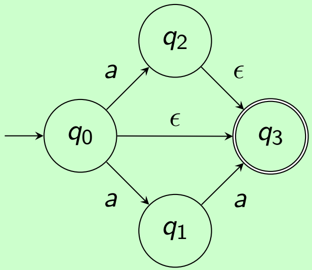
Automates Finis Non Déterministes (NFA)

Exemple



$\mathcal{L}(\mathcal{A}) = a^* b^*$

Exemple



$\mathcal{L}(\mathcal{A}) = \epsilon|a|aa$

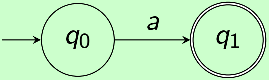
Des RE vers les NFA

- Les **atomes** des RE se codent aisément avec les DFA et donc les NFA :

Des RE vers les NFA

- Les **atomes** des RE se codent aisément avec les DFA et donc les NFA :

Exemple

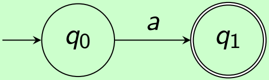


$\mathcal{L}(\mathcal{A}) = a$

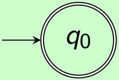
Des RE vers les NFA

- Les **atomes** des RE se codent aisément avec les DFA et donc les NFA :

Exemple



$\mathcal{L}(\mathcal{A}) = a$

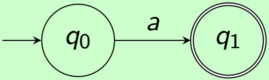


$\mathcal{L}(\mathcal{A}) = \epsilon$

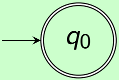
Des RE vers les NFA

- Les **atomes** des RE se codent aisément avec les DFA et donc les NFA :

Exemple



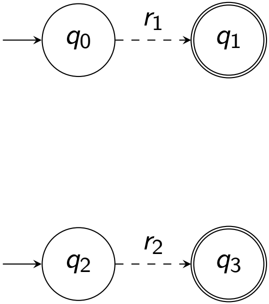
$\mathcal{L}(\mathcal{A}) = a$



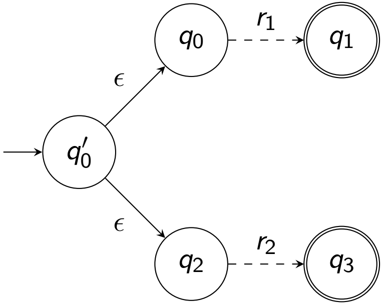
$\mathcal{L}(\mathcal{A}) = \epsilon$

- Restent à coder : les **opérations régulières**

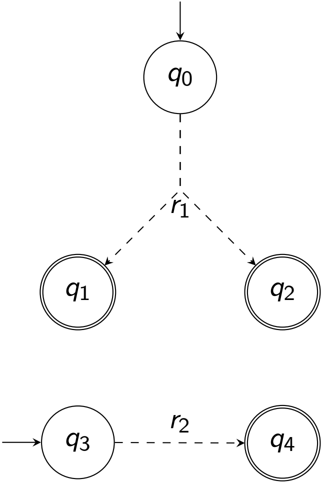
Union : $r_1 | r_2$



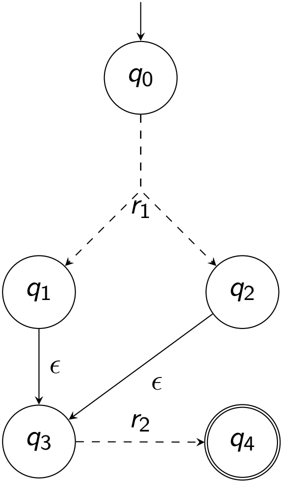
Union : $r_1 | r_2$



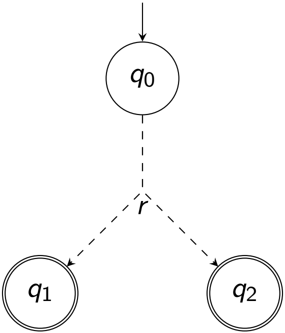
Concaténation : $r_1 r_2$



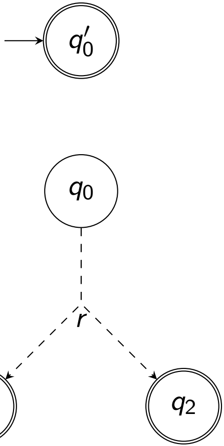
Concaténation : $r_1 r_2$



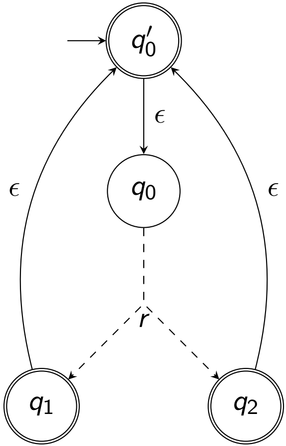
Fermeture (Étoile) de Kleene : r^*



Fermeture (Étoile) de Kleene : r^*

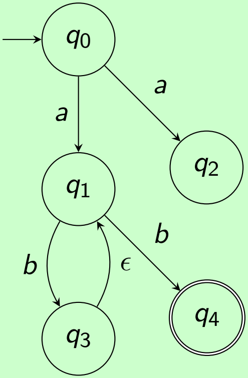


Fermeture (Étoile) de Kleene : r^*



Implémentation d'un NFA

Exemple




```
list<int> T[5][3], E, nE;
list<int>::iterator i; char t;
E.push_back(0);
T[0][0].push_back(1); // q0 -a-> q1
T[0][0].push_back(2); // q0 -a-> q2
T[1][1].push_back(3); // q1 -b-> q3
T[1][1].push_back(4); // q1 -b-> q4
T[3][2].push_back(1); // q3 -ϵ-> q1
cin >> t;
while (t<='z' && t >='a') {
    for (i=E.begin(); i != E.end(); i++) {
       nE.append(T[*i][t-'a']); // mais append
       epsilon-fermeture(nE); // n'existe pas :-(
    }
    E = nE; nE.clear();
    cin >> t;
}
if (find(E.begin(), E.end(), 4) != E.end())
    cout << "reconnu" << endl;
```

Équivalence NFA - DFA

Théorème

Pour tout automate fini non déterministe $\mathcal{A}$, il existe un automate fini déterministe $\Delta(\mathcal{A})$ qui reconnaît le même langage que $\mathcal{A}$.

Équivalence NFA - DFA

Théorème

Pour tout automate fini non déterministe $\mathcal{A}$, il existe un automate fini déterministe $\Delta(\mathcal{A})$ qui reconnaît le même langage que $\mathcal{A}$.

Calculer $\Delta(\mathcal{A})$ est appelé ≪ Déterminisation de $\mathcal{A}$ ≫.

Déterminisation d'un NFA

Le principe est le même que celui utilisé pour l'implémentation précédente.
Soit $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$. On définit $\Delta(\mathcal{A}) = (Q', \Sigma, \delta', q_0', F')$

- $Q' = 2^Q$;

Déterminisation d'un NFA

Le principe est le même que celui utilisé pour l'implémentation précédente.
Soit $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$. On définit $\Delta(\mathcal{A}) = (Q', \Sigma, \delta', q'_0, F')$

- $Q' = 2^Q$;
- $q'_0 = Q_0$;

Déterminisation d'un NFA

Le principe est le même que celui utilisé pour l'implémentation précédente.
Soit $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$. On définit $\Delta(\mathcal{A}) = (Q', \Sigma, \delta', q'_0, F')$

- $Q' = 2^Q$;
- $q'_0 = Q_0$;
- $F' = \{S \in 2^Q \mid F \cap S \neq \emptyset\}$;

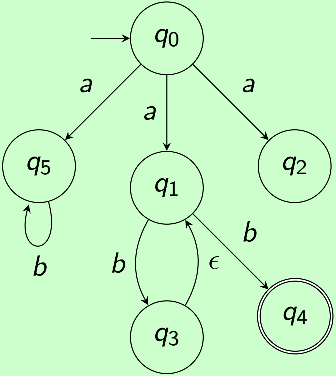
Déterminisation d'un NFA

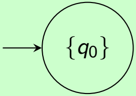
Le principe est le même que celui utilisé pour l'implémentation précédente. Soit $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$. On définit $\Delta(\mathcal{A}) = (Q', \Sigma, \delta', q'_0, F')$

- $Q' = 2^Q$;
- $q'_0 = Q_0$;
- $F' = \{S \in 2^Q \mid F \cap S \neq \emptyset\}$;
- $\forall q' \in Q', \forall a \in \Sigma, \delta'(q') = \epsilon - \mathcal{F}(\bigcup_{q \in q'} \delta(q, a))$ où $\epsilon - \mathcal{F}$ est le point fixe de la fonction $S \mapsto S \cup \{q | \exists q', q \in \delta(q', \epsilon)\}$.

Déterminisation d'un NFA : Exemple

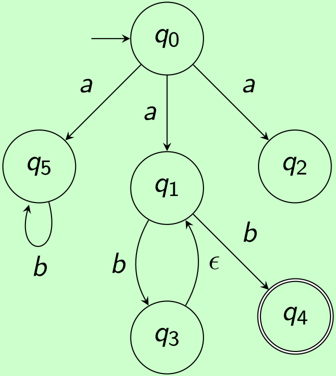
Exemple

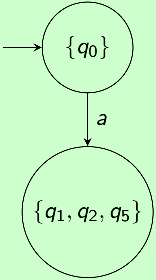




Déterminisation d'un NFA : Exemple

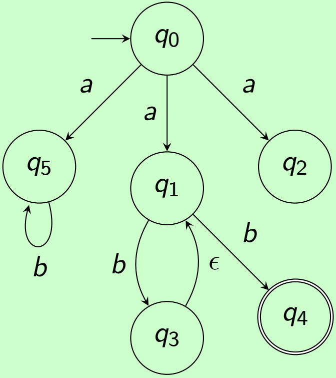
Exemple

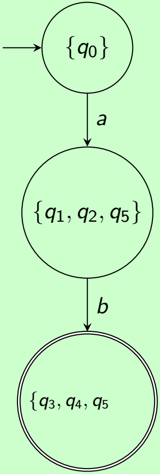




Déterminisation d'un NFA : Exemple

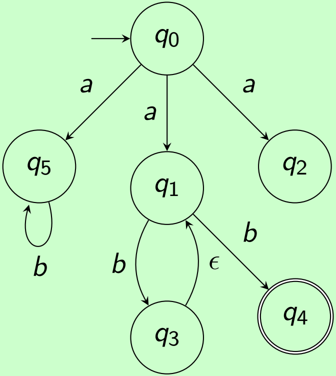
Exemple

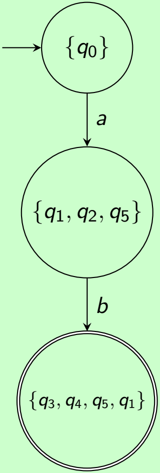




Déterminisation d'un NFA : Exemple

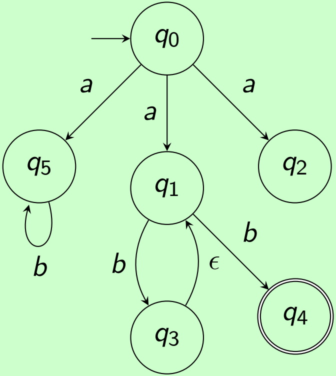
Exemple

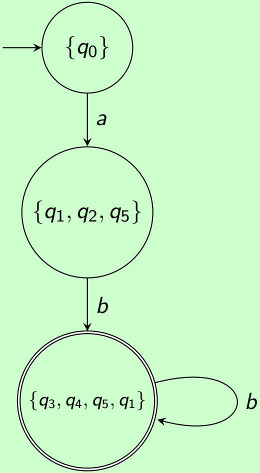




Déterminisation d'un NFA : Exemple

Exemple

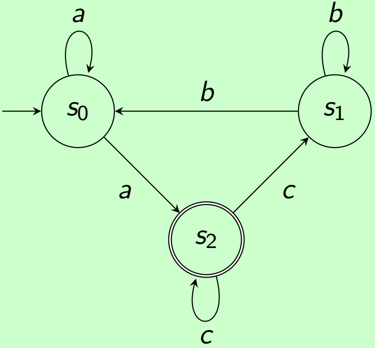


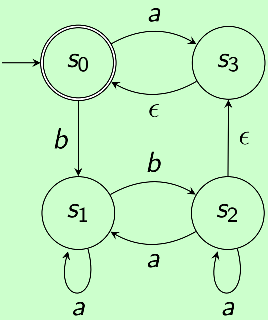


Déterminisation d'un NFA : Exercice

Exercice

Déterminisez les automates suivants :





Exercices

Exercice

Construisez un automate fini déterministe et complet qui reconnaît le langage $(a\Sigma^* b\Sigma^* a)^*$ pour $\Sigma = \{a, b\}$.

Exercices

Exercice

Construisez un automate fini déterministe et complet qui reconnaît le langage $(a\Sigma^* b\Sigma^* a)^*$ pour $\Sigma = \{a, b\}$.

Exercice

1. Construisez un NFA sur $\Sigma = \{a\}$ qui reconnaît le langage $\{(aaa)^m(aaaa)^n \mid m, n \geq 0\}$;
2. Construisez un DFA qui reconnaît son complémentaire ;
3. Déduisez-en l'ensemble des nombres qui ne s'écrivent pas $3m + 4n$ avec m, n entiers naturels.

Minimisation d'un DFA

- La déterminisation fournit des automates potentiellement très gros ;
- On définit une **relation d'équivalence** entre les états de l'automate ;

Minimisation d'un DFA

- La déterminisation fournit des automates potentiellement très gros ;
- On définit une **relation d'équivalence** entre les états de l'automate ;

Définition (relation $\equiv$)

On dit que deux états q et q' de $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ sont équivalents si $\forall w \in \Sigma^*, \delta(q, w) \in F \Leftrightarrow \delta(q', w) \in F$. On note alors $q \equiv q'$.

Minimisation d'un DFA

- La déterminisation fournit des automates potentiellement très gros ;
- On définit une **relation d'équivalence** entre les états de l'automate ;

Définition (relation $\equiv$)

On dit que deux états q et q' de $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ sont équivalents si $\forall w \in \Sigma^*, \delta(q, w) \in F \Leftrightarrow \delta(q', w) \in F$. On note alors $q \equiv q'$.

- Les états d'une même classe d'équivalence pourront être confondus.

Minimisation d'un DFA

- La déterminisation fournit des automates potentiellement très gros ;
- On définit une **relation d'équivalence** entre les états de l'automate ;

Définition (relation $\equiv$)

On dit que deux états q et q' de $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ sont équivalents si $\forall w \in \Sigma^*, \delta(q, w) \in F \Leftrightarrow \delta(q', w) \in F$. On note alors $q \equiv q'$.

- Les états d'une même classe d'équivalence pourront être confondus.
- Pour construire les classes d'équivalence de $\equiv$, on définit $\equiv_n$:

Définition (relation $\equiv_n$)

On dit que deux états q et q' de $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ sont n-équivalents si $\forall w \in \Sigma^*$ t.q. $|w| \leq n$, $\delta(q, w) \in F \Leftrightarrow \delta(q', w) \in F$. On note alors $q \equiv_n q'$.

Minimisation d'un DFA

- De façon assez immédiate, la partition de Q induite par $\equiv_0$ est $\{F, Q \setminus F\}$;

Minimisation d'un DFA

- De façon assez immédiate, la partition de Q induite par $\equiv_0$ est $\{F, Q \setminus F\}$;
- Supposons qu'on a une partition $\mathcal{P}_n$ pour $\equiv_n$. Soit $P, P' \in \mathcal{P}_n$. Alors $\forall a \in \Sigma, \{q \in P | \delta(q, a) \in P'\}$ définit un élément de $\mathcal{P}_{n+1}$;

Minimisation d'un DFA

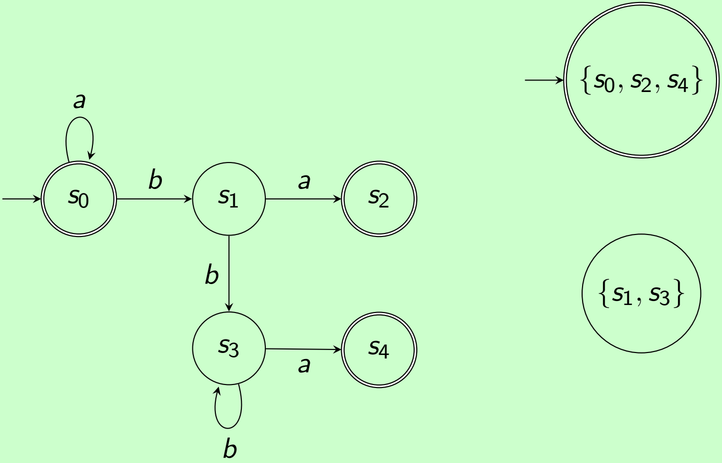
- De façon assez immédiate, la partition de Q induite par $\equiv_0$ est $\{F, Q \setminus F\}$;
- Supposons qu'on a une partition $\mathcal{P}_n$ pour $\equiv_n$. Soit $P, P' \in \mathcal{P}_n$. Alors $\forall a \in \Sigma, \{q \in P | \delta(q, a) \in P'\}$ définit un élément de $\mathcal{P}_{n+1}$;
- Le nombre d'éléments de $\mathcal{P}_{n+1}$ est supérieur ou égal à celui de $\mathcal{P}_n$;

Minimisation d'un DFA

- De façon assez immédiate, la partition de Q induite par $\equiv_0$ est $\{F, Q \setminus F\}$;
- Supposons qu'on a une partition $\mathcal{P}_n$ pour $\equiv_n$. Soit $P, P' \in \mathcal{P}_n$. Alors $\forall a \in \Sigma, \{q \in P | \delta(q, a) \in P'\}$ définit un élément de $\mathcal{P}_{n+1}$;
- Le nombre d'éléments de $\mathcal{P}_{n+1}$ est supérieur ou égal à celui de $\mathcal{P}_n$;
- Le nombre de façons de partitionner Q est fini, donc $\exists k$ t.q. $\mathcal{P}_{k+1} = \mathcal{P}_k$. Alors $\equiv_k = \equiv$.

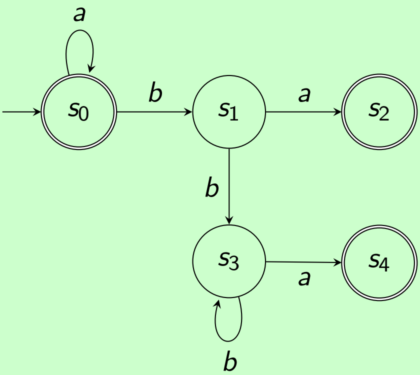
Minimisation d'un DFA

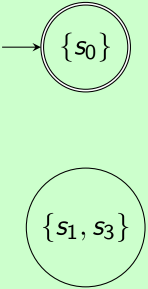
Exemple



Minimisation d'un DFA

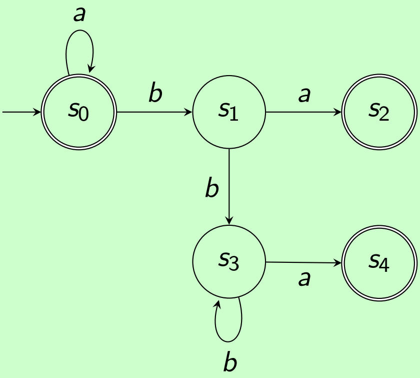
Exemple

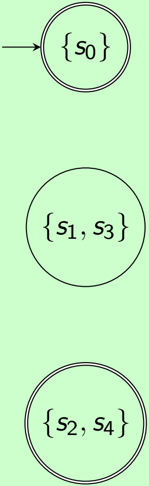




Minimisation d'un DFA

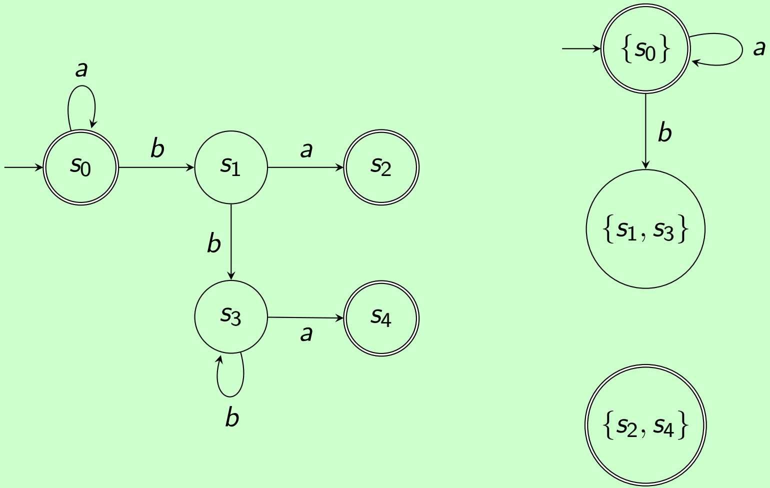
Exemple





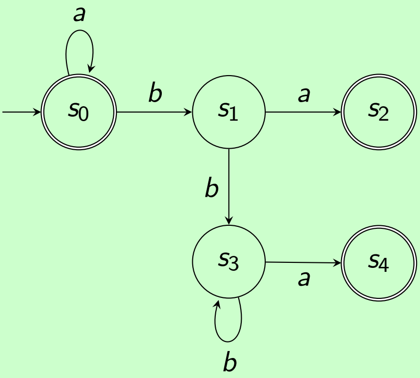
Minimisation d'un DFA

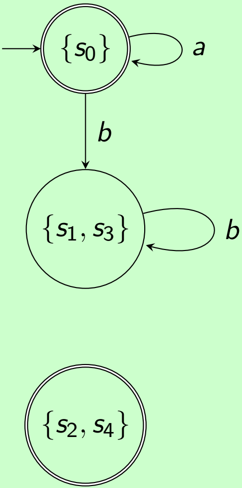
Exemple



Minimisation d'un DFA

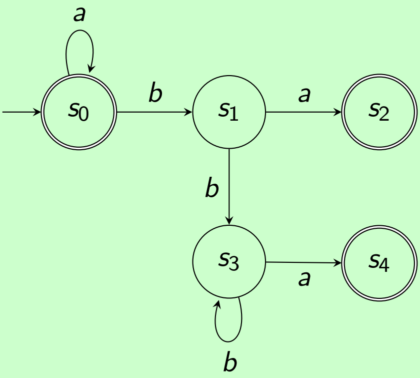
Exemple

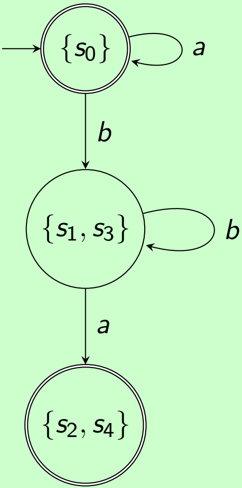




Minimisation d'un DFA

Exemple

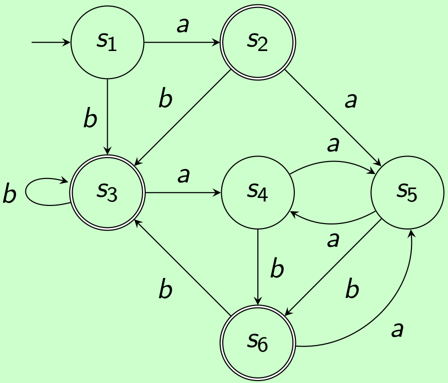


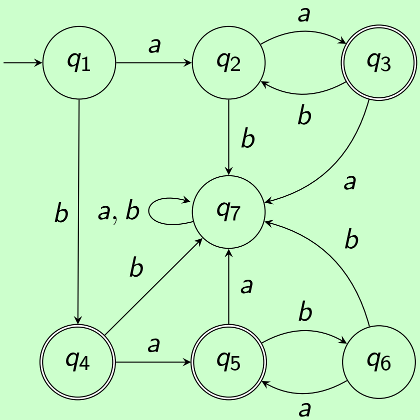


Minimisation d'un DFA : Exercice

Exercice

Minimisez les automates suivants :





Plan

Un constructeur d'analyseur lexical : Lex

- **lex** est un constructeur d'analyseur lexical pour Unix ;

Un constructeur d'analyseur lexical : Lex

- **lex** est un constructeur d'analyseur lexical pour Unix ;
- Il génère un **automate fini déterministe** ;

Un constructeur d'analyseur lexical : Lex

- **lex** est un constructeur d'analyseur lexical pour Unix ;
- Il génère un **automate fini déterministe** ;
- Un outil ayant ses fonctionalités fait partie du standard **POSIX** ;

Un constructeur d'analyseur lexical : Lex

- **lex** est un constructeur d'analyseur lexical pour Unix ;
- Il génère un **automate fini déterministe** ;
- Un outil ayant ses fonctionalités fait partie du standard **POSIX** ;
- En pratique, on en utilise souvent une implémentation *open source* : **flex**.

Un constructeur d'analyseur lexical : Lex

- **lex** est un constructeur d'analyseur lexical pour Unix ;
- Il génère un **automate fini déterministe** ;
- Un outil ayant ses fonctionalités fait partie du standard **POSIX** ;
- En pratique, on en utilise souvent une implémentation *open source* : **flex**.

Syntaxe du fichier de définition :

```
Définitions
%%
Règles (Expressions régulières)
%%
Code C
```

Lex – Exemple : eval.l

```
%{
#include "..."
%}
BLANC [ \n\t]
%%
[0-9]+          yylval = atoi(yytext); return NOMBRE;
[-+*/=]         return yytext[0]; /* car. unités lexicales *
{BLANC}+        ;
.               fprintf(stderr, "Erreur"); /* tout le reste *
```

Plan

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :
  - puissance d'expression ;

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :
  - puissance d'expression ;
  - et efficacité.

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :
  - puissance d'expression ;
  - et efficacité.
- On peut en dériver **automatiquement** un DFA qui s'implémente facilement et efficacement ;

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :
  - puissance d'expression ;
  - et efficacité.
- On peut en dériver **automatiquement** un DFA qui s'implémente facilement et efficacement ;
- Il existe des outils pour créer ces analyseurs lexicaux ;

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :
  - puissance d'expression ;
  - et efficacité.
- On peut en dériver **automatiquement** un DFA qui s'implémente facilement et efficacement ;
- Il existe des outils pour créer ces analyseurs lexicaux ;
- Les automates finis sont également très utilisés pour modéliser ou analyser des **systèmes à événements discrets** :

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :
  - puissance d'expression ;
  - et efficacité.
- On peut en dériver **automatiquement** un DFA qui s'implémente facilement et efficacement ;
- Il existe des outils pour créer ces analyseurs lexicaux ;
- Les automates finis sont également très utilisés pour modéliser ou analyser des **systèmes à événements discrets** :
  - programmes ;

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :
  - puissance d'expression ;
  - et efficacité.
- On peut en dériver **automatiquement** un DFA qui s'implémente facilement et efficacement ;
- Il existe des outils pour créer ces analyseurs lexicaux ;
- Les automates finis sont également très utilisés pour modéliser ou analyser des **systèmes à événements discrets** :
  - programmes ;
  - systèmes biologiques ;

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :
  - puissance d'expression ;
  - et efficacité.
- On peut en dériver **automatiquement** un DFA qui s'implémente facilement et efficacement ;
- Il existe des outils pour créer ces analyseurs lexicaux ;
- Les automates finis sont également très utilisés pour modéliser ou analyser des **systèmes à événements discrets** :
  - programmes ;
  - systèmes biologiques ;
  - systèmes de production ;

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :
  - puissance d'expression ;
  - et efficacité.
- On peut en dériver **automatiquement** un DFA qui s'implémente facilement et efficacement ;
- Il existe des outils pour créer ces analyseurs lexicaux ;
- Les automates finis sont également très utilisés pour modéliser ou analyser des **systèmes à événements discrets** :
  - programmes ;
  - systèmes biologiques ;
  - systèmes de production ;
  - scénarios (de jeux) ;

Conclusion

- Les expressions rationelles fournissent un bon **compromis** entre :
  - puissance d'expression ;
  - et efficacité.
- On peut en dériver **automatiquement** un DFA qui s'implémente facilement et efficacement ;
- Il existe des outils pour créer ces analyseurs lexicaux ;
- Les automates finis sont également très utilisés pour modéliser ou analyser des **systèmes à événements discrets** :
  - programmes ;
  - systèmes biologiques ;
  - systèmes de production ;
  - scénarios (de jeux) ;
  - ...