

Cours 8: Expressions régulières. Automates finis.

Olivier Bournez

bournez@lix.polytechnique.fr
LIX, Ecole Polytechnique

INF421-b

Bases de la programmation et de l'algorithmique

Aujourd'hui

Expressions régulières

Un peu d'Unix

Un tout petit peu de JAVA

Automates finis déterministes

Automates finis non-déterministes

Déterminisation

Chercher un mot ou un motif: Unix

- Dans un texte: commande Unix

`egrep <motif> <nom-fichier>`

- ▶ `egrep -x 'v.ca..se' francais.txt`

réponse *vocalise*

- ▶ `egrep 'i.*i.*i.*i.*i.*i.*' francais.txt`

réponse *indivisibilité*

indivisibilités

inintelligibilité

inintelligibilités

- Dans une arborescence: commande Unix `ls <motif>`

- ▶ `ls *.java`

Mots

- Un *alphabet* Σ est un ensemble fini.
 - ▶ Exemple 1: $\Sigma_1 = \{0, 1\}$, l'alphabet binaire.
 - ▶ Exemple 2: $\Sigma_2 = \{a, b, \dots, z\}$, l'alphabet des lettres minuscules.
 - ▶ Exemple 3: Σ_3 l'ensemble des caractères Unicode.
- Un élément de Σ est appelé un *caractère*, ou une *lettre*.
- Un mot w sur Σ est une suite finie de caractères de Σ .
 - ▶ w s'écrit $w = a_1 a_2 \dots a_n$, avec $a_i \in \Sigma$.
 - ▶ $n \geq 0$ est *la longueur du mot*.
 - ▶ Le mot vide, noté ϵ , est (l'unique) mot de longueur 0.
- Σ^* dénote l'ensemble des mots sur Σ .
 - ▶ Exemple: $101101 \in \Sigma_1^*$.
 - ▶ Exemple: $vocalise \in \Sigma_2^*$.
 - ▶ Exemple: $vczeerrztx \in \Sigma_2^*$.
 - ▶ Exemple: $bonne\ année \in \Sigma_3^*$.

Langages

- Un *Langage* sur Σ est un ensemble de mots sur Σ .
 - ▶ Exemple: $\{10, 11, 1\}$ pour Σ_1 .
 - ▶ Exemple: les mots binaires codant un entier pour Σ_1 .
 - ▶ Exemple: les mots de francais.txt pour Σ_3 .

Quelques opérations sur les mots

- Etant donné deux mots m_1 et m_2 , la concaténation de m_1 et de m_2 , notée $m_1.m_2$, ou notée m_1m_2 , est le mot obtenu en mettant m_2 à la fin de m_1 .
 - ▶ Exemple: $aba.ba$ est le mot $ababa$ sur $\Sigma = \{a, b\}$.
- La concaténation est une opération associative, avec un élément neutre, le mot vide ϵ .
- si $m = m_1m_2$, m_1 est appelé un *préfixe* de m et m_2 un *suffixe* de m .

Quelques opérations sur les langages

- Union ensembliste.

$$L_1 \cup L_2$$

- Concaténation.

$$L_1.L_2 = \{m_1.m_2 \mid m_1 \in L_1 \wedge m_2 \in L_2\}$$

- Itération.

$$L^0 = \{\epsilon\}$$

$$L^{n+1} = L^n.L = L.L^n$$

$$L^* = \bigcup_{n \in \mathbb{N}} L^n$$

Autrement dit,

- $m \in L^n$ si et seulement si $m = m_1 m_2 \cdots m_n$ avec $m_i \in L$ pour $1 \leq i \leq n$.
- $m \in L^*$ si et seulement si $m = m_1 m_2 \cdots m_n$ avec $m_i \in L$ pour un certain n .

Les expressions régulières: le principe

- L'ensemble vide \emptyset représente le langage vide.
- Le mot vide ϵ représente le langage $\{\epsilon\}$.
- Un caractère c de Σ représente le langage $\{c\}$.
- L'*alternative* $p_1|p_2$ représente l'union des langages représentés par p_1 et p_2 .
- La *concaténation* $p_1.p_2$ représente la concaténation des langages représentés par p_1 et p_2 .
- La *répétition* p^* représente l'itération du langage représenté par p .

Proprement

On va distinguer

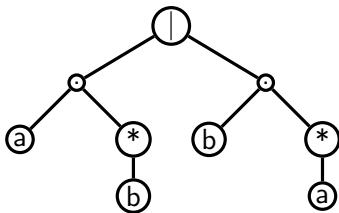
- la *syntaxe* (= motif)
- de la *sémantique* (= langage correspondant)

Les expressions régulières: syntaxe

- Définition inductive:
 - ▶ Le symbole \emptyset est un motif.
 - ▶ Le mot vide ϵ est un motif.
 - ▶ Un caractère c de Σ est un motif.
 - ▶ Si p_1 et p_2 sont des motifs, alors *l'alternative* $p_1|p_2$ est un motif.
 - ▶ Si p_1 et p_2 sont des motifs, alors la *concaténation* $p_1.p_2$ est un motif.
 - ▶ Si p est un motif, alors *la répétition* p^* est un motif.
- Bref: un motif est un arbre.

$$M = \{\epsilon, \emptyset\} \uplus \Sigma \uplus (M \times \{ |, . \} \times M) \uplus (M \times \{ * \}).$$

Exemple



- Ecriture possible: $(a.(b*))|(b.(a*))$.
- Comme pour les expressions arithmétiques, on fixe des conventions sur les priorités entre opérateurs pour éviter certaines parenthèses.
 - ▶ Autre écriture de cet arbre: $ab * |ba*$

Priorités

- Analogie avec les expressions arithmétiques, en interprétant | par +, . par \times , * par une mise à la puissance.
- Formellement: par ordre décroissant de priorité
 1. les parenthèses (comme les parenthèses)
 2. * (comme ^)
 3. . (comme .)
 4. | (comme +)
- Le symbole de concaténation peut être omis.
- Exemples:
 - ▶ ab^* correspond à $a.(b^*)$.
 - ▶ $ab|c$ correspond à $(a.b)|c$.
 - ▶ $ab * |c$ correspond à $(a.(b^*))|c$.

Priorités

- Analogie avec les expressions arithmétiques, en interprétant | par +, . par \times , * par une mise à la puissance.
- Formellement: par ordre décroissant de priorité
 1. les parenthèses (comme les parenthèses)
 2. * (comme \wedge)
 3. . (comme .)
 4. | (comme +)
- Le symbole de concaténation peut être omis.
- Exemples:
 - ▶ ab^* correspond à $a.(b^*)$.
 - ▶ $ab|c$ correspond à $(a.b)|c$.
 - ▶ $ab * |c$ correspond à $(a.(b^*))|c$.
- Remarque: il reste des ambiguïtés (exemple: abc est $(ab)c$ ou $a(bc)$?) mais sans importance par associativité des opérateurs.

Sémantique

- D'un motif vers un langage: La fonction $\llbracket \cdot \rrbracket$ associe à chaque motif un langage.

$$\begin{aligned}\llbracket \emptyset \rrbracket &= \emptyset \\ \llbracket \epsilon \rrbracket &= \{\epsilon\} \\ \llbracket c \rrbracket &= \{c\} \\ \llbracket p_1 | p_2 \rrbracket &= \llbracket p_1 \rrbracket \cup \llbracket p_2 \rrbracket \\ \llbracket p_1 \cdot p_2 \rrbracket &= \llbracket p_1 \rrbracket \cdot \llbracket p_2 \rrbracket \\ \llbracket p^* \rrbracket &= \llbracket p \rrbracket^*\end{aligned}$$

- Un langage L est dit *régulier* s'il existe un motif p tel que

$$\llbracket p \rrbracket = L.$$

- Remarque: tout langage n'est pas régulier (preuve simple: les motifs sont dénombrables, les mots sont dénombrables, mais les parties de \mathbb{N} et donc les langages ne le sont pas).

Quelques exemples de langages réguliers

- Tout langage fini est régulier:

- ▶ Exemple: $\{ab, b, aba\} = \llbracket ab|b|aba \rrbracket$.
- ▶ Cas général:

$$L = \{w_1, w_2, \dots, w_n\} = \llbracket w_1|w_2|\dots|w_n \rrbracket$$

- Si l'on définit le français comme les mots d'un dictionnaire de référence (celui de l'Académie), le français aussi.
- Ecriture décimale des entiers:

$$0|(1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*$$

- Mots constitués de 0 et de 1 alternés:
 - ▶ Commence par 0, finit par 1: $(01)^*$
 - ▶ Commence par 0, finit par 0: $(01)^*0$
 - ▶ etc.

$$(01)^*|(01)^*0|(10)^*|(10)^*1$$

Quelques exemples de langages réguliers

- Tout langage fini est régulier:

- ▶ Exemple: $\{ab, b, aba\} = \llbracket ab|b|aba \rrbracket$.
- ▶ Cas général:

$$L = \{w_1, w_2, \dots, w_n\} = \llbracket w_1|w_2|\dots|w_n \rrbracket$$

- Si l'on définit le français comme les mots d'un dictionnaire de référence (celui de l'Académie), le français aussi.
- Ecriture décimale des entiers:

$$0|(1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)^*$$

- Mots constitués de 0 et de 1 alternés:

- ▶ Commence par 0, finit par 1: $(01)^*$
- ▶ Commence par 0, finit par 0: $(01)^*0$
- ▶ etc.

$$(01)^* | (01)^*0 | (10)^* | (10)^*1$$

- Plus concis:

$$(\epsilon|1)(01)^*(\epsilon|0)$$

Aujourd'hui

Expressions régulières

Un peu d'Unix

Un tout petit peu de JAVA

Automates finis déterministes

Automates finis non-déterministes

Déterminisation

Un peu de sucre syntaxique I

Les constructions suivantes n'étendent pas la classe des langages réguliers (ce sont des raccourcis de notations de motifs).

- Le *joker* `.`

- ▶ Pour $\Sigma = \{a_1, \dots, a_n\}$, `.` est un synonyme de $a_1|a_2|\dots|a_n$
- ▶ $[[.]] = \Sigma$

- Exemple: `egrep -x 'v.ca..se' francais.txt`

- L' *option* (zero ou une fois) `p?`, synonyme de $p|\epsilon$.

- Exemple: `egrep -x '1?(01)*0?' fichier.java`

- La *répétition au moins une fois* `p+`, synonyme de pp^* .

- Exemple: `egrep -x 'ab+a' fichier.txt`

Un peu de sucre syntaxique II

Notations Unix (man egrep)

- Les ensembles de caractères écrits entre crochets [...].

Exemples:

- ▶ [aeiou] synonyme de a|e|i|o|u
 - ▶ [a-z] synonyme de a|b|c|...|z (on utilise l'ordre ASCII des caractères)
 - ▶ [a-zA-Z0-9], synonyme de tout caractère alphanumérique.
- Les complémentaires, notés [^...].

Exemple:

- ▶ [^a-zA-Z0-0] (caractère non-alphanumérique)
- Caractères spéciaux:
 - ▶ \n (retour chariot), \t (tabulation)
 - ▶ *, \+, \{, \|, \[, \), \., \. (les caractères *,+,{,|,(,). respectivement)

Exemple: extraire les commentaires de JAVA

- Commentaire de la forme `// texte`:
 - ▶ Commence par `//`, et se termine en fin de ligne.
 - ▶ Expression régulière: `//[\n]*\n`
 - ▶ Unix: `egrep '//[\n]*\n' fichier.java`

- Commentaire de la forme `/* texte */`:
 - ▶ Commence par `/*`, se termine par `*/`, imbrications non-autorisées.
 - ▶ Expression régulière: `/*([\^*]|*+[\^*/])**+\/.`
 - ▶ Unix: `egrep '/*([\^*]|*+[\^*/])**+\/' fichier.java`

Dans l'interprète de commande Unix

- L'interprète de commande Unix (le *shell*) accepte quelques expressions régulières pour les noms de fichiers du répertoire courant, avec une syntaxe spéciale.
 - ▶ L'expression `*` désigne `.*` (n'importe quel nom de fichier)
 - ▶ Le joker est noté `?`
 - ▶ L'alternative est notée `{...,...}`
- Exemples:
 - ▶ `rm *.class`: effacer tous les fichiers `.class`.
 - ▶ `ls {?,??,???:}`: liste les fichiers dont le nom est de longueur 1, 2 ou 3.
 - ▶ `wc -l *.java`: compter les lignes des sources JAVA.

Aujourd'hui

Expressions régulières

Un peu d'Unix

Un tout petit peu de JAVA

Automates finis déterministes

Automates finis non-déterministes

Déterminisation

Utiliser les expressions régulières en JAVA

- Les objets de la classe `Pattern` de `java.util.regex.*` implémentent les motifs:
- Les objets de la classe `Matcher` de `java.util.regex.*` implémentent la confrontation d'un `Pattern p` à un mot `m`:
- Chaque `Matcher` possède une méthode `matches()` et une méthode `find()` qui renvoient des booléens.

```
import java.util.regex.*;
...
System.out.print(
    Pattern.compile("ab?a").matcher("baba").find() +", "+
    Pattern.compile("ab?a").matcher("baba").matches());
```

Affiche: *true,false*

Utiliser les expressions régulières en JAVA

- Les objets de la classe `Pattern` de `java.util.regex.*` implémentent les motifs:
 - ▶ `Pattern.compile(s)` crée un motif à partir de la chaîne de caractère `s`.
- Les objets de la classe `Matcher` de `java.util.regex.*` implémentent la confrontation d'un `Pattern p` à un mot `m`:
- Chaque `Matcher` possède une méthode `matches()` et une méthode `find()` qui renvoient des booléens.

```
import java.util.regex.*;
...
System.out.print(
    Pattern.compile("ab?a").matcher("baba").find() + ", "+
    Pattern.compile("ab?a").matcher("baba").matches());
```

Affiche: *true,false*

Utiliser les expressions régulières en JAVA

- Les objets de la classe `Pattern` de `java.util.regex.*` implémentent les motifs:
 - ▶ `Pattern.compile(s)` crée un motif à partir de la chaîne de caractère `s`.
- Les objets de la classe `Matcher` de `java.util.regex.*` implémentent la confrontation d'un `Pattern p` à un mot `m`:
 - ▶ `p.matcher(m)` fabrique un `Matcher` pour le mot `m`.
- Chaque `Matcher` possède une méthode `matches()` et une méthode `find()` qui renvoient des booléens.

```
import java.util.regex.*;
...
System.out.print(
    Pattern.compile("ab?a").matcher("baba").find() + ", "+
    Pattern.compile("ab?a").matcher("baba").matches());
```

Affiche: *true,false*

Utiliser les expressions régulières en JAVA

- Les objets de la classe `Pattern` de `java.util.regex.*` implémentent les motifs:
 - ▶ `Pattern.compile(s)` crée un motif à partir de la chaîne de caractère `s`.
- Les objets de la classe `Matcher` de `java.util.regex.*` implémentent la confrontation d'un `Pattern p` à un mot `m`:
 - ▶ `p.matcher(m)` fabrique un `Matcher` pour le mot `m`.
- Chaque `Matcher` possède une méthode `matches()` et une méthode `find()` qui renvoient des booléens.
 - ▶ `matches()` retourne vrai si `p` filtre `m`, et `find` retourne vrai si `p` filtre un sous-mot de `m`.

```
import java.util.regex.*;
...
System.out.print(
    Pattern.compile("ab?a").matcher("baba").find() + ", "+
    Pattern.compile("ab?a").matcher("baba").matches());
```

Affiche: *true,false*

En dire plus?

■ Nous n'en dirons pas plus

- ▶ en renvoyant à la documentation JAVA pour
 - toutes les possibilités des classes `Pattern` et `Matcher`.
 - toutes les syntaxes autorisées dans les motifs: exemple `\p{L}` représente n'importe quelle lettre.
- ▶ en renvoyant au poly pour une reprogrammation (simple) de ces classes.
- ▶ et au TD de cet après midi...

■ Un point avec lequel il faut parfois batailler pour certains motifs:

- ▶ si le motif est donné par une chaîne JAVA, il faut tenir compte des règles d'écriture dans les chaînes JAVA.
 - Par exemple, `\p{L}` s'écrit dans une chaîne JAVA `"\\p{L}"`, car pour citer un `\` il faut écrire `\\`.

Aujourd'hui

Expressions régulières

Un peu d'Unix

Un tout petit peu de JAVA

Automates finis déterministes

Automates finis non-déterministes

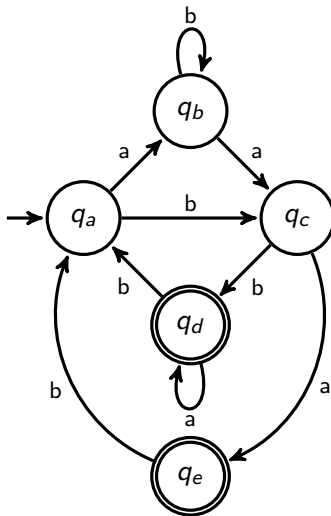
Déterminisation

Automates

- Les automates apparaissent ici via les expressions régulières,
- ... mais ont leur intérêt propre dans de nombreux domaines:
 - ▶ Vérification de circuits électroniques (on les décrit par des automates finis ou des automates “étendus”)
 - ▶ Vérification de protocoles de communications (chaque agent est décrit par un automate)
 - ▶ Description de propriétés temporelles (logiques temporelles)
 - ▶ Compilation (reconnaissance de mots du langage compilé)
 - ▶ Biologie (motifs dans les séquences d’ADN)
 - ▶ etc...
- et aussi sont primordiaux en décidabilité/complexité: ils permettent de comprendre ce qu’un ordinateur peut résoudre (décidabilité), ou résoudre efficacement (complexité), avec une mémoire limitée (finie).

Un automate fini déterministe

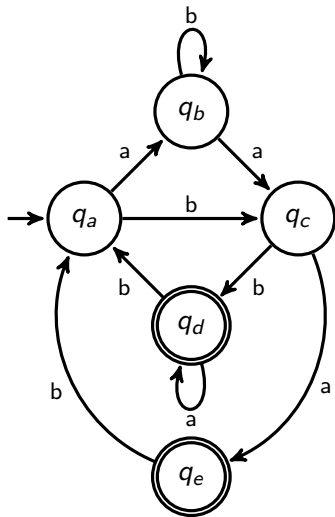
- Un automate ...



- ... prend en entrée un mot et l'accepte ou le rejette.

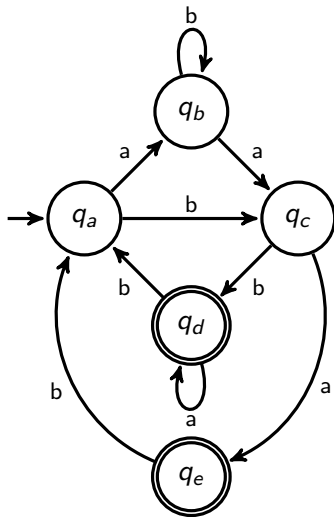
Automate fini déterministe

- Un automate fini déterministe est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où
 - ▶ Σ est un alphabet.
 - ▶ Q est un ensemble fini d'états.
 - ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
 - ▶ q_0 est l'état initial.
 - ▶ $F \subset Q$ est un ensemble d'états finaux.



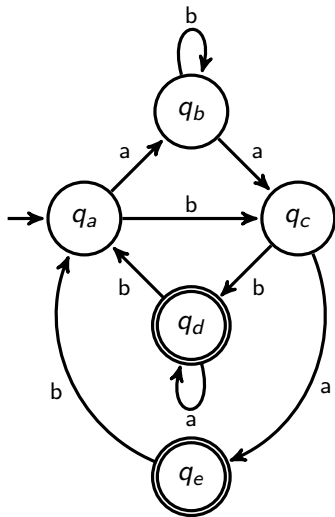
- Sur l'exemple:

Automate fini déterministe



- Un automate fini déterministe est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où
 - ▶ Σ est un alphabet.
 - ▶ Q est un ensemble fini d'états.
 - ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
 - ▶ q_0 est l'état initial.
 - ▶ $F \subset Q$ est un ensemble d'états finaux.
- Sur l'exemple:
 - ▶ $\Sigma = \{a, b\}$.

Automate fini déterministe



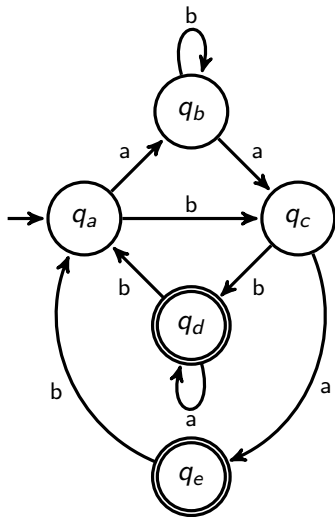
- Un automate fini déterministe est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où

- ▶ Σ est un alphabet.
- ▶ Q est un ensemble fini d'états.
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
- ▶ q_0 est l'état initial.
- ▶ $F \subset Q$ est un ensemble d'états finaux.

- Sur l'exemple:

- ▶ $\Sigma = \{a, b\}$.
- ▶ $Q = \{q_a, q_b, q_c, q_d, q_e\}$.

Automate fini déterministe



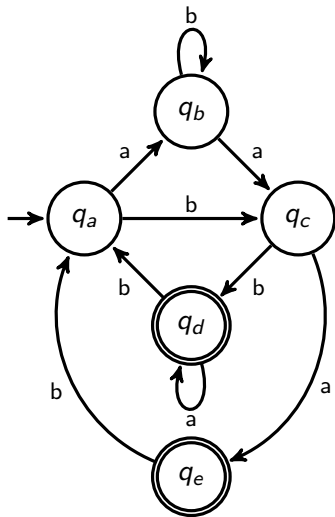
- Un automate fini déterministe est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où

- ▶ Σ est un alphabet.
- ▶ Q est un ensemble fini d'états.
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
- ▶ q_0 est l'état initial.
- ▶ $F \subset Q$ est un ensemble d'états finaux.

- Sur l'exemple:

- ▶ $\Sigma = \{a, b\}$.
- ▶ $Q = \{q_a, q_b, q_c, q_d, q_e\}$.
- ▶ $\delta(q_a, a) = q_b$ par exemple.

Automate fini déterministe



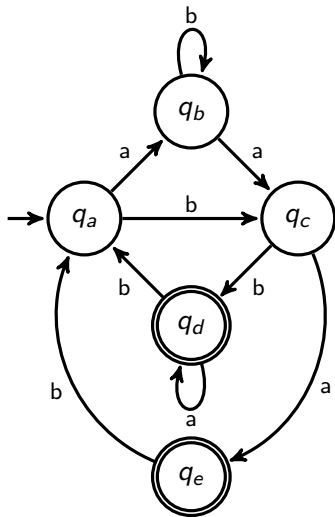
- Un automate fini déterministe est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où

- ▶ Σ est un alphabet.
- ▶ Q est un ensemble fini d'états.
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
- ▶ q_0 est l'état initial.
- ▶ $F \subset Q$ est un ensemble d'états finaux.

- Sur l'exemple:

- ▶ $\Sigma = \{a, b\}$.
- ▶ $Q = \{q_a, q_b, q_c, q_d, q_e\}$.
- ▶ $\delta(q_a, a) = q_b$ par exemple.
- ▶ $q_0 = q_a$

Automate fini déterministe



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

- ▶ Σ est un alphabet.
- ▶ Q est un ensemble fini d'états.
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
- ▶ q_0 est l'état initial.
- ▶ $F \subset Q$ est un ensemble d'états finaux.

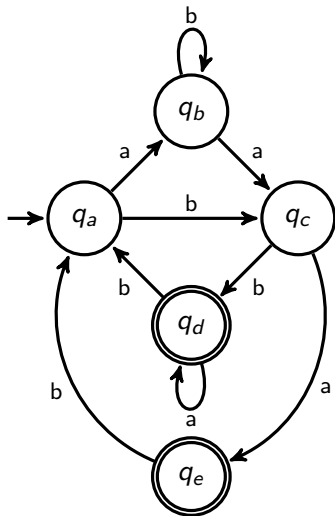
- Sur l'exemple:

- ▶ $\Sigma = \{a, b\}$.
- ▶ $Q = \{q_a, q_b, q_c, q_d, q_e\}$.
- ▶ $\delta(q_a, a) = q_b$ par exemple.
- ▶ $q_0 = q_a$
- ▶ $F = \{q_d, q_e\}$

Représentation graphique

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

- ▶ Σ est un alphabet.
- ▶ Q est un ensemble fini d'états.
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
- ▶ q_0 est l'état initial.
- ▶ $F \subset Q$ est un ensemble d'états finaux.

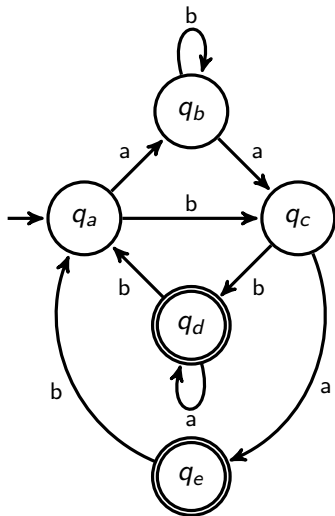


- Représentation graphique:

Représentation graphique

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

- ▶ Σ est un alphabet.
- ▶ Q est un ensemble fini d'états.
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
- ▶ q_0 est l'état initial.
- ▶ $F \subset Q$ est un ensemble d'états finaux.



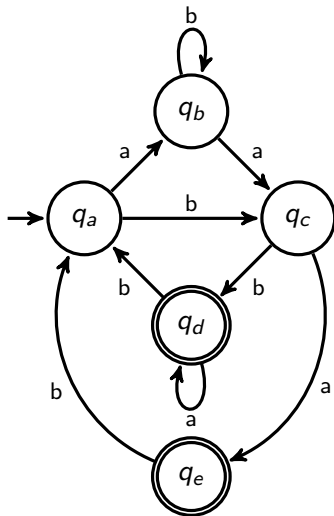
- Représentation graphique:

- ▶ Les états de Q sont les sommets d'un (multi-)graphe.

Représentation graphique

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

- ▶ Σ est un alphabet.
- ▶ Q est un ensemble fini d'états.
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
- ▶ q_0 est l'état initial.
- ▶ $F \subset Q$ est un ensemble d'états finaux.



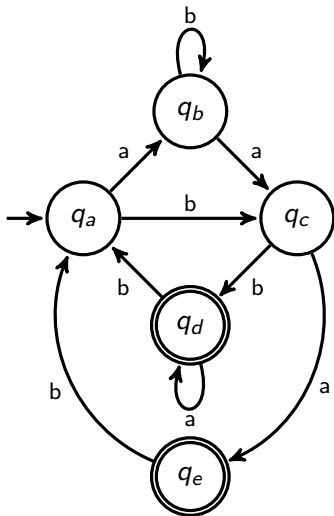
- Représentation graphique:

- ▶ Les états de Q sont les sommets d'un (multi-)graphe.
- ▶ $\delta(q, a) = q'$ est codé par un arc de q vers q' étiqueté par a .

Représentation graphique

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

- ▶ Σ est un alphabet.
- ▶ Q est un ensemble fini d'états.
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
- ▶ q_0 est l'état initial.
- ▶ $F \subset Q$ est un ensemble d'états finaux.



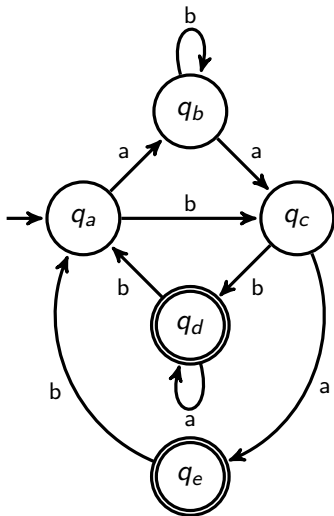
- Représentation graphique:

- ▶ Les états de Q sont les sommets d'un (multi-)graphe.
- ▶ $\delta(q, a) = q'$ est codé par un arc de q vers q' étiqueté par a .
- ▶ L'état initial est marqué par une flèche entrante.

Représentation graphique

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

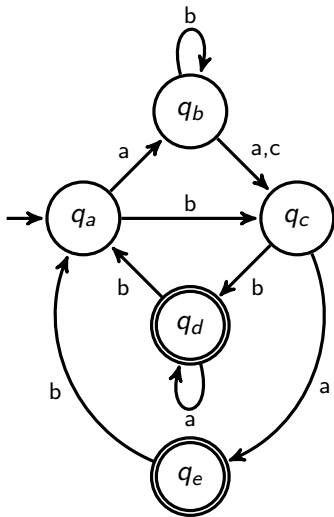
- ▶ Σ est un alphabet.
- ▶ Q est un ensemble fini d'états.
- ▶ $\delta : Q \times \Sigma \rightarrow Q$ est la fonction (partielle) de transition.
- ▶ q_0 est l'état initial.
- ▶ $F \subset Q$ est un ensemble d'états finaux.



- Représentation graphique:

- ▶ Les états de Q sont les sommets d'un (multi-)graphe.
- ▶ $\delta(q, a) = q'$ est codé par un arc de q vers q' étiqueté par a .
- ▶ L'état initial est marqué par une flèche entrante.
- ▶ Les états finaux sont marqués par un double cercle.

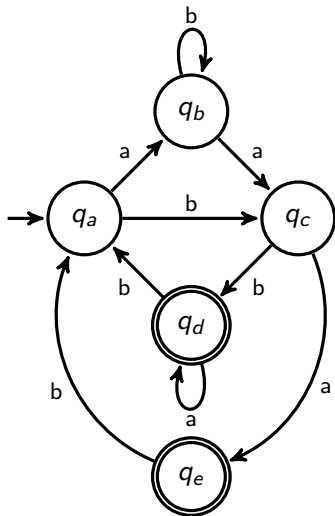
Représentation graphique: remarque



■ Remarque graphique:

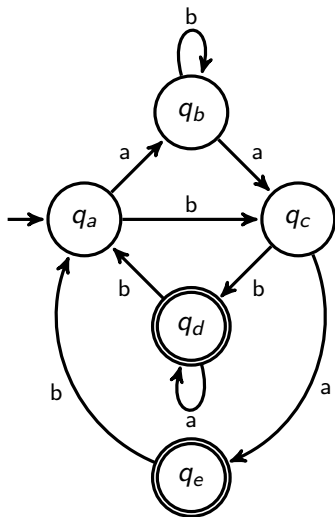
- ▶ Entre deux états, il peut y avoir plusieurs arcs: on peut avoir $\delta(q, a) = \delta(q, b) = q'$.
- ▶ Il s'agit donc d'un multi-graphe plutôt que d'un graphe (un graphe autorisant plusieurs arcs/arêtes entre sommets).
- ▶ On note souvent plusieurs arcs par un seul arc étiqueté par des caractères séparés par des virgules: un arc étiqueté par a, b par exemple dénote un arc étiqueté par a , et un arc étiqueté par b .

Langage accepté par un automate: Informellement



- Un automate prend en entrée un mot w et l'accepte ou le rejette:
 - ▶ On part à l'état q_0
 - ▶ On lit les caractères du mot w un à un en suivant la transition correspondante (si il n'y en a pas, on se bloque = échec).
 - ▶ Lorsque tous les caractères sont lus, on accepte si l'on est sur un état final, on rejette sinon.
- Exemple:
 - ▶ Le mot *abbbab* est accepté.
 - ▶ Le mot *babba* est accepté.
 - ▶ Le mot *ababab* n'est pas accepté.

Langage accepté par un automate: Formellement I



- On introduit la *fonction $\hat{\delta}$ de transition étendue aux mots* :

- ▶ A partir d'un état q en lisant le mot vide ϵ on reste dans l'état q ,

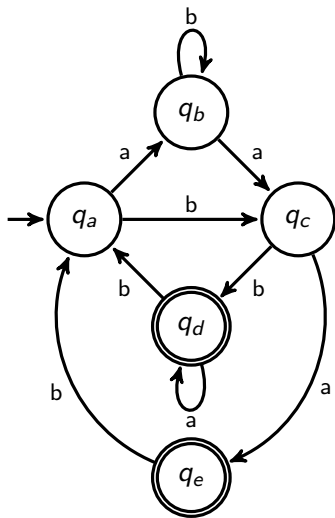
$$\forall q \in Q, \hat{\delta}(q, \epsilon) = q.$$

- ▶ A partir d'un état q , en lisant le mot $c = wa$ se terminant par $a \in \Sigma$, on a d'abord lu w , puis effectué la transition correspondante à a

$$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a),$$

lorsque le membre droit existe.

Langage accepté par un automate: Formellement II



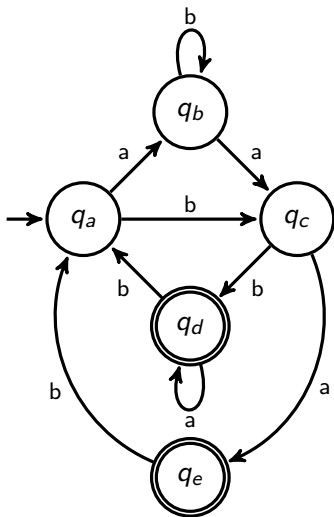
- Le langage $L(A)$ accepté par un automate fini déterministe $A = (Q, \Sigma, \delta, q_0, F)$ est défini par

$$L(A) = \{m \mid \hat{\delta}(q_0, m) \in F\}.$$

- Un mot est dit *accepté* si et seulement il appartient à $L(A)$.

Représentation compacte des automates

- Par un tableau:



	<i>a</i>	<i>b</i>
→ <i>q_a</i>	<i>q_b</i>	<i>q_c</i>
<i>q_b</i>	<i>q_c</i>	<i>q_b</i>
<i>q_c</i>	<i>q_e</i>	<i>q_d</i>
* <i>q_d</i>	<i>q_d</i>	<i>q_a</i>
* <i>q_e</i>		<i>q_a</i>

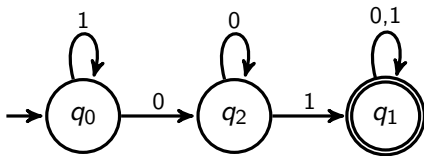
- ▶ Chaque ligne correspond à un état de l'automate
 - un état final est précédé d'une étoile *
 - l'état initial est précédé d'une flèche →
- ▶ Chaque colonne correspond à une lettre de l'alphabet
- ▶ L'élément à la ligne i et à la colonne j donne la valeur de $\delta(i, j)$.

Lien avec les expressions régulières

- Théorème de Kleene: Tout langage reconnu par un automate fini déterministe est régulier, et réciproquement.
- Autrement dit:
 - ▶ Le langage accepté par tout automate fini déterministe est régulier.
 - ▶ Tout langage régulier est accepté par un automate fini déterministe.

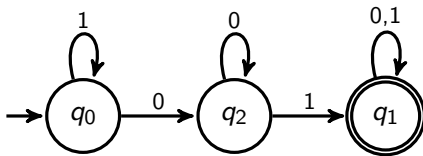
Exemple 1: sous-mot 01

- Construire un automate fini déterministe (*DFA*) qui reconnaît le langage des mots sur l'alphabet $\{0, 1\}$ qui contiennent le sous-mot 01.
- Expression régulière: $(0|1)^* 01(0|1)^* .$
- Automate:



Exemple 1: sous-mot 01

- Construire un automate fini déterministe (*DFA*) qui reconnaît le langage des mots sur l'alphabet $\{0, 1\}$ qui contiennent le sous-mot 01.
- Expression régulière: $(0|1)^* 01(0|1)^* .$
- Automate:

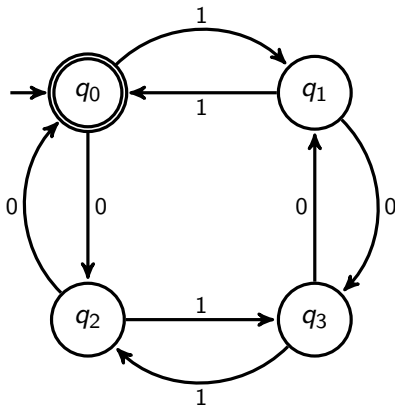


- Autre représentation:

	0	1
$\rightarrow *q_0$	q_2	q_0
q_1	q_1	q_1
q_2	q_2	q_1

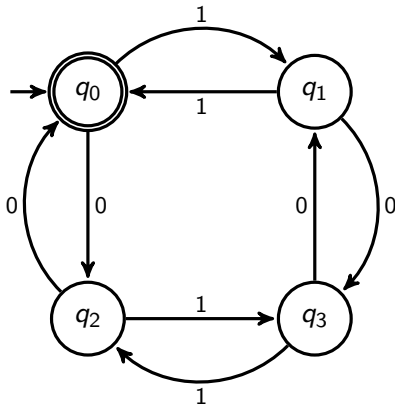
Exemple 2: nombre de 0,1 pair

- Construire un *DFA* qui reconnaît le langage des mots sur l'alphabet $\{0, 1\}$ avec un nombre pair de 0 et un nombre pair de 1.
- Automate:



Exemple 2: nombre de 0,1 pair

- Construire un *DFA* qui reconnaît le langage des mots sur l'alphabet $\{0, 1\}$ avec un nombre pair de 0 et un nombre pair de 1.
- Automate:



Autre représentation:

	0	1
$\rightarrow *q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Aujourd'hui

Expressions régulières

Un peu d'Unix

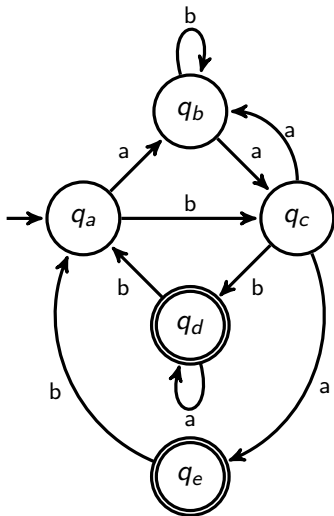
Un tout petit peu de JAVA

Automates finis déterministes

Automates finis non-déterministes

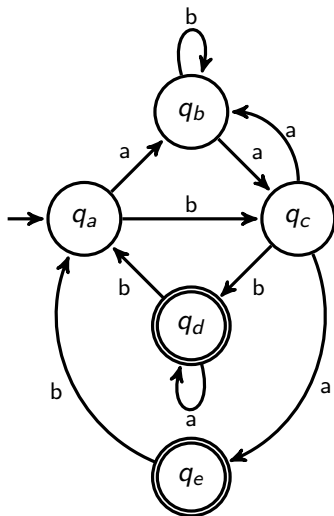
Déterminisation

Automate fini non-déterministe (NFA= Nondeterministic Finite Automaton)



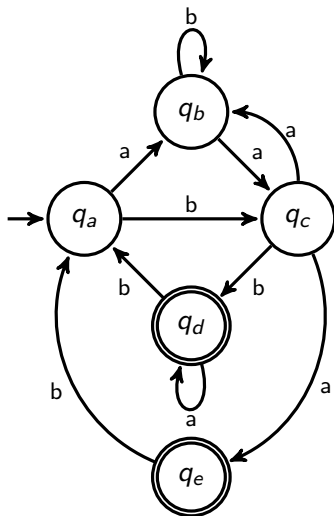
- Le principe: on autorise plusieurs transitions avec le même symbole.
- δ n'est plus une fonction partielle de $Q \times \Sigma$ vers Q , mais une fonction de $Q \times \Sigma$ vers les parties de Q .
- Ici:
 - ▶ $\delta(q_c, a) = \{q_e, q_b\}$.
 - ▶ $\delta(q_a, a) = \{q_b\}$.
 - ▶ $\delta(q_e, a) = \emptyset$.

NFA: formellement



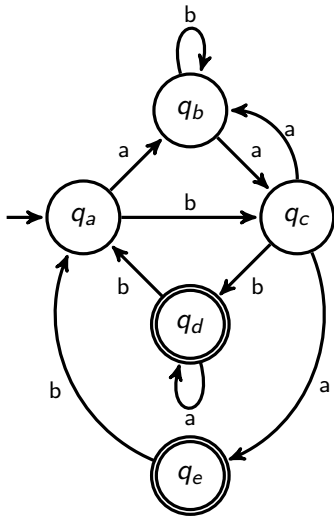
- Un automate fini non-déterministe est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où
 - ▶ Σ est un alphabet.
 - ▶ Q est un ensemble fini d'états.
 - ▶ $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ est une fonction de $Q \times \Sigma$ vers les parties Q . δ associe à tout état $q \in Q$, et symbole $c \in \Sigma$ un sous-ensemble $\delta(q, c)$ de Q .
 - ▶ q_0 est l'état initial.
 - ▶ $F \subset Q$ est un ensemble d'états finaux.

NFA: formellement



- Un automate fini non-déterministe est un quintuplet $(Q, \Sigma, \delta, q_0, F)$ où
 - ▶ Σ est un alphabet.
 - ▶ Q est un ensemble fini d'états.
 - ▶ $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ est une fonction de $Q \times \Sigma$ vers les parties Q . δ associe à tout état $q \in Q$, et symbole $c \in \Sigma$ un sous-ensemble $\delta(q, c)$ de Q .
 - ▶ q_0 est l'état initial.
 - ▶ $F \subset Q$ est un ensemble d'états finaux.
- Pourquoi la terminologie "non-déterministe":
 - ▶ l'état courant et le symbole lu ne "détermine" pas le prochain état.

Représentation compacte des automates

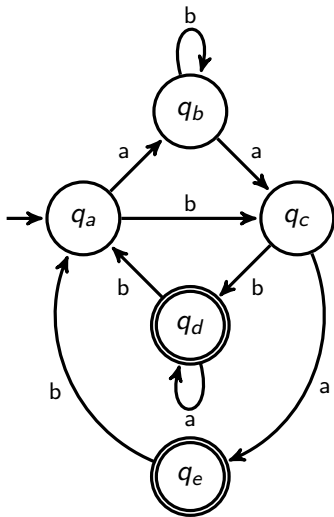


■ Par un tableau:

	<i>a</i>	<i>b</i>
$\rightarrow q_a$	$\{q_b\}$	$\{q_c\}$
q_b	$\{q_c\}$	$\{q_b\}$
q_c	$\{q_b, q_e\}$	$\{q_d\}$
$*q_d$	$\{q_d\}$	$\{q_a\}$
$*q_e$	\emptyset	$\{q_a\}$

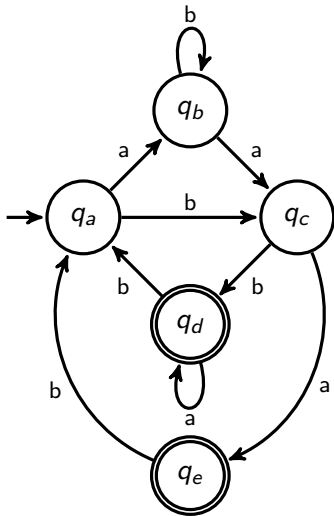
■ L'élément à la ligne q et à la colonne j donne la valeur de $\delta(q, j)$.

Une remarque



- Tout *DFA* peut être vu comme un *NFA*.
- Principe: remplacer $\delta(q, a) = q'$ par $\delta(q, a) = \{q'\}$ pour tout q, a, q' .

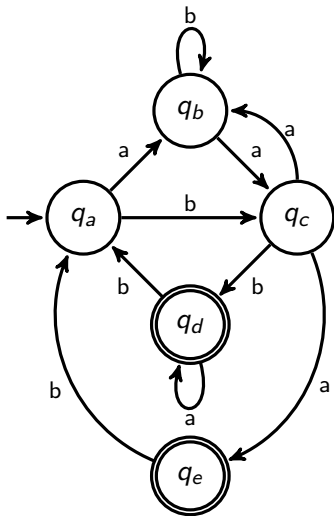
Rappel. Langage accepté par un *DFA*: informellement



- Un automate prend en entrée un mot w et l'accepte ou le rejette:
 - ▶ On part à l'état q_0
 - ▶ On lit les caractères du mot w un à un en suivant la transition correspondante (si il n'y en a pas, on se bloque = échec)
 - ▶ Lorsque tous les caractères sont lus, on accepte si l'on est sur un état final, on rejette sinon.
- Exemple:
 - ▶ Le mot *abbbab* est accepté.
 - ▶ Le mot *babba* est accepté.
 - ▶ Le mot *ababab* n'est pas accepté.

Langage accepté par un *NFA*: informellement

- Un automate prend en entrée un mot w et l'accepte ou le rejette:



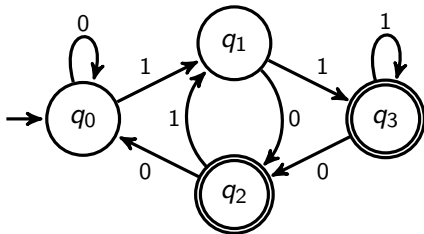
- ▶ On part à l'état q_0
- ▶ On essaye de lire les caractères du mot w un à un (lorsqu'il y a plusieurs possibilités, on en choisit une/ s'il n'y en a pas, on se bloque = échec).
- ▶ S'il existe une façon de lire tous les caractères et d'arriver sur un état final, on accepte le mot.
- ▶ Sinon (il n'y a pas de façon de lire tous les caractères et d'arriver sur un état final), on rejette.

- Exemple:

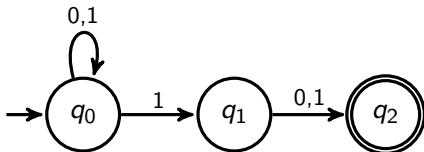
- ▶ *aaabab* est accepté.
- ▶ *bab* n'est pas accepté.

Exercices sur $\Sigma = \{0, 1\}$

- Construire un *DFA* qui reconnaît le langage des mots tels que le 2^{ème} symbole en partant de la droite soit un 1.

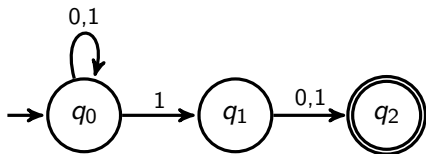


- Même exercice avec un *NFA*.



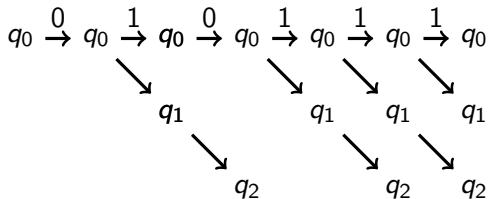
Une façon de voir les exécutions

- Exemple:



sur $w = 010111$.

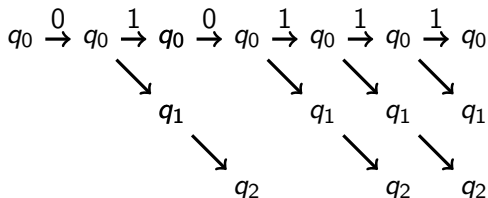
- Exécutions:



- On accepte un mot si et seulement s'il y a une exécution sur ce mot partant de l'état initial qui termine en un état de F .

Une autre façon de voir

■ Exécutions:



■ Fonction $\hat{\delta}$ (fonction de transition étendue aux mots):

- ▶ $\hat{\delta}(q_0, \epsilon) = \{q_0\}$
- ▶ $\hat{\delta}(q_0, 0) = \{q_0\}$
- ▶ $\hat{\delta}(q_0, 01) = \{q_0, q_1\}$
- ▶ $\hat{\delta}(q_0, 010) = \{q_0, q_2\}$
- ▶ $\hat{\delta}(q_0, 0101) = \{q_0, q_1\}$
- ▶ $\hat{\delta}(q_0, 01011) = \{q_0, q_1, q_2\}$
- ▶ $\hat{\delta}(q_0, 010111) = \{q_0, q_1, q_2\}$

■ On accepte un mot w si et seulement si $\hat{\delta}(q_0, w) \cap F \neq \emptyset$.

Formellement: langage accepté par un automate

- On introduit la *fonction $\hat{\delta}$ de transition étendue aux mots* :

- ▶ A partir d'un état q en lisant le mot vide ϵ on reste dans l'état q ,

$$\forall q \in Q, \hat{\delta}(q, \epsilon) = \{q\}$$

- ▶ A partir d'un état q , en lisant le mot $m = wa$ se terminant par $a \in \Sigma$, on a d'abord lu w , puis effectué les transitions correspondantes à a

$$\hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q, w)} \delta(p, a).$$

- Le langage $L(A)$ accepté par un automate fini non-déterministe $A = (Q, \Sigma, \delta, q_0, F)$ est défini par

$$L(A) = \{m \mid \hat{\delta}(q_0, m) \cap F \neq \emptyset\}$$

Exercices sur $\Sigma = \{0, 1\}$

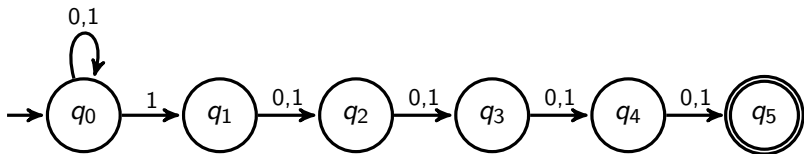
- Construire un *DFA* qui reconnaît le langage des mots tels que le 5ième symbole en partant de la droite soit un 1.

Exercices sur $\Sigma = \{0, 1\}$

- Construire un *DFA* qui reconnaît le langage des mots tels que le 5^{ème} symbole en partant de la droite soit un 1.
compliqué (il n'y en a pas avec moins de $32 = 2^5$ états).

Exercices sur $\Sigma = \{0, 1\}$

- Construire un *DFA* qui reconnaît le langage des mots tels que le 5^{ème} symbole en partant de la droite soit un 1.
compliqué (il n'y en a pas avec moins de $32 = 2^5$ états).
- Même exercice avec un *NFA*.



Un NFA en JAVA: coder un automate

- δ peut se coder par un tableau doublement indexé.
- Chaque ensemble d'états $\delta[q][a]$ peut se coder par une liste.

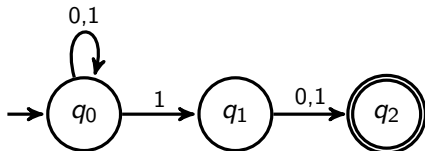
```
class Automate {  
  // Etat initial:  
  int q0;  
  // Fonction de transition:  
  Liste[] [] delta;  
  // Etats finaux:  
  Liste finaux;  
  
  // Constructeur:  
  Automate(int q, Liste f,  
           Liste[] []d) {  
    q0 = q;  
    delta = d;  
    finaux = f; }  
  
  ...  
}
```

```
class Liste {  
  int contenu;  
  Liste suivant;  
  Liste (int x, Liste a) {  
    contenu = x;  
    suivant = a;}}}
```

```
static int longueur(Liste l) {  
  if (l == null) return 0;  
  return 1 + longueur(l.suivant);}  
  
static boolean Dans(int x, Liste a) {  
  if (a == null) return false;  
  if (a.contenu == x) return true;  
  return Dans(x, a.suivant);  
}
```

Un NFA en JAVA: exemple

- L'automate:



- Son codage en JAVA:

```
Liste[] [] delta = new Liste[3][128];
delta[0][(int)'0'] = new Liste(0, null);
delta[0][(int)'1'] = new Liste(0, new Liste(1, null));
delta[1][(int)'0'] = null;
delta[1][(int)'1'] = new Liste(2, null);
delta[2][(int)'0'] = null;
delta[2][(int)'1'] = null;
Automate a = new Automate(0, new Liste(2, null), delta);
```

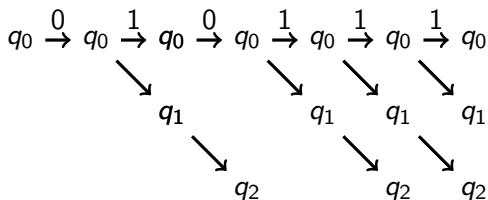
- Remarque JAVA: (**int**)'0' retourne le code ASCII du caractère "0".

Un *NFA* en JAVA: déterminer si un mot est accepté I

- Par *backtracking* (essai et retour en arrière): en essayant toutes les possibilités.

```
boolean Accepte(String w) {  
    return essai(w,0,q0);  
}
```

- Exemple: pour l'automate précédent sur $w = 010111$.



Un *NFA* en *JAVA*: déterminer si un mot est accepté II

- Solution récursive naturelle:

```
boolean essai(String w, int i, int q) {
    if (i == w.length())
        return Dans(q, finaux);
    int c = w.charAt(i); // le caractère courant
    Liste l = delta[q][c]; // les prochains états possibles
    for (; l != null; l = l.suivant) {
        if (essai(w,i+1,l.contenu)){
            // on a trouvé une exécution vers un état final
            return true;
        }
    }
    // si on a tout essayé, mais rien trouvé
    return false;
}
```

Aujourd'hui

Expressions régulières

Un peu d'Unix

Un tout petit peu de JAVA

Automates finis déterministes

Automates finis non-déterministes

Déterminisation

Déterminisation

- Théorème: Tout *NFA* peut être simulé par un *DFA*.

Déterminisation

■ Théorème: Tout *NFA* peut être simulé par un *DFA*.

■ Preuve:

▶ Soit $(Q, \Sigma, \delta, q_0, F)$ un *NFA*.

▶ Il reconnaît le même langage que le *DFA*

$$(Q', \Sigma, \delta', \{q_0\}, F')$$

avec:

- $Q' = \mathcal{P}(Q)$: Q' est constitué de tous les sous-ensembles de Q .
- F' est constitué de tous les sous-ensembles de Q avec au moins un élément commun avec F .
- Pour tout $S \in Q'$, et pour tout $a \in \Sigma$,

$$\delta'(S, a) = \cup_{q \in S} \delta(q, a).$$

Déterminisation

■ Théorème: Tout *NFA* peut être simulé par un *DFA*.

■ Preuve:

▶ Soit $(Q, \Sigma, \delta, q_0, F)$ un *NFA*.

▶ Il reconnaît le même langage que le *DFA*

$$(Q', \Sigma, \delta', \{q_0\}, F')$$

avec:

- $Q' = \mathcal{P}(Q)$: Q' est constitué de tous les sous-ensembles de Q .
- F' est constitué de tous les sous-ensembles de Q avec au moins un élément commun avec F .
- Pour tout $S \in Q'$, et pour tout $a \in \Sigma$,

$$\delta'(S, a) = \cup_{q \in S} \delta(q, a).$$

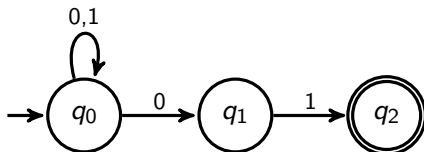
▶ Pourquoi: il est fait exactement pour que

$$\hat{\delta}'(q_0, w) = \hat{\delta}(\{q_0\}, w)$$

pour tout mot w .

Exemple 1

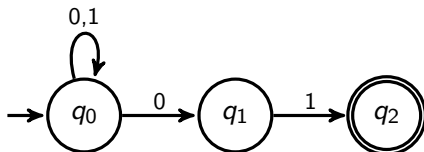
- Un *NFA*:



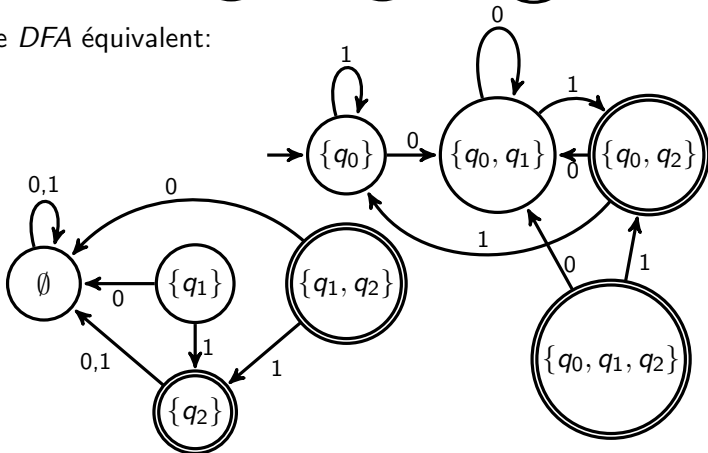
- Le *DFA* équivalent:

Exemple 1

- Un *NFA*:

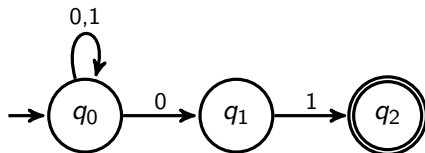


- Le *DFA* équivalent:

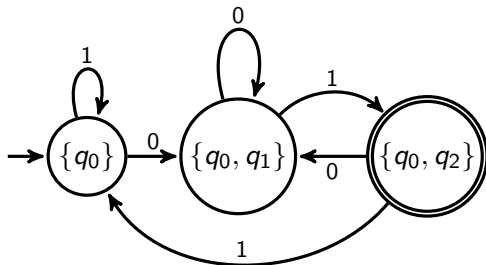


Exemple 1: suite

- Le *DFA*:

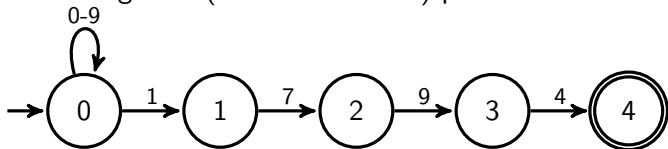


- La partie atteignable du *DFA* équivalent:



Exemple: un digicode

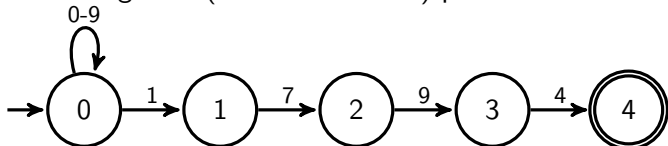
- Réaliser un digicode (code secret 1794) par un *NFA*:



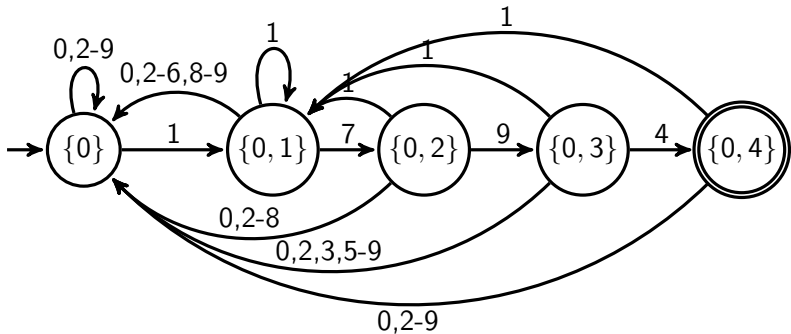
- Le *DFA* obtenu (circuit électronique correspondant):

Exemple: un digicode

- Réaliser un digicode (code secret 1794) par un *NFA*:

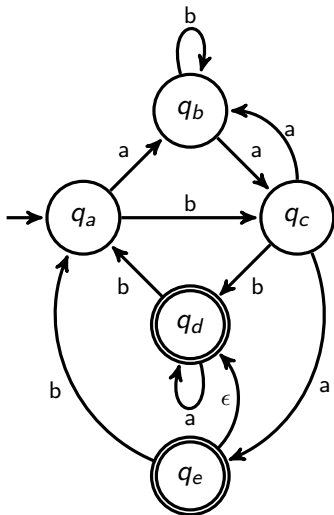


- Le *DFA* obtenu (circuit électronique correspondant):



- Tous les autres sous-ensembles de $\{0, 1, 2, 3, 4\}$ ne sont pas atteignables: ils ne sont pas représentés ici.

Encore une extension: automate fini non-déterministe avec ϵ -transitions (*NFA* – ϵ)



■ Le principe:

1. on autorise toujours plusieurs transitions avec le même symbole. (non-déterminisme)
2. mais aussi les transitions “spontanées”:
 - certaines transitions peuvent être étiquetées par le mot vide ϵ .
 - Elles peuvent être prises sans lire de lettre.

■ Exemple: le mot *baaaaa* est accepté.

NFA – ϵ : formalisations

- Formalisation de la notion d'automate:

- Formalisation de la notion de mot reconnu:

NFA – ϵ : formalisations

- Formalisation de la notion d'automate:

▶ ... $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$...

- Formalisation de la notion de mot reconnu:

NFA – ϵ : formalisations

- Formalisation de la notion d'automate:

▶ ... $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$...

- Formalisation de la notion de mot reconnu:

▶ ... $\hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q, w)} \epsilon\text{-cl\^oture}(\delta(p, a)).$

$$\epsilon\text{-cl\^oture}(q) = \left\{ \begin{array}{l} \text{tous les \^etats joignables \^a partir de } q \\ \text{par une suite de } n \geq 0 \text{ transitions} \\ \text{\'etiquett\'ees par } \epsilon \end{array} \right\} \dots$$

NFA – ϵ : formalisations

- Formalisation de la notion d'automate:

▶ ... $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$...

- Formalisation de la notion de mot reconnu:

▶ ... $\hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q, w)} \epsilon\text{-cl\^oture}(\delta(p, a)).$

$$\epsilon\text{-cl\^oture}(q) = \left\{ \begin{array}{l} \text{tous les \^etats joignables \^a partir de } q \\ \text{par une suite de } n \geq 0 \text{ transitions} \\ \text{\'etiquetees par } \epsilon \end{array} \right\} \dots$$

- Tout *DFA* peut \^etre vu comme un *NFA*, tout *NFA* peut \^etre vu comme un un *NFA – ϵ* .

NFA – ϵ : formalisations

- Formalisation de la notion d'automate:

▶ ... $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$...

- Formalisation de la notion de mot reconnu:

▶ ... $\hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q, w)} \epsilon\text{-cl\^oture}(\delta(p, a)).$

$$\epsilon\text{-cl\^oture}(q) = \left\{ \begin{array}{l} \text{tous les \^etats joignables \^a partir de } q \\ \text{par une suite de } n \geq 0 \text{ transitions} \\ \text{\'etiquett\'ees par } \epsilon \end{array} \right\} \dots$$

- Tout *DFA* peut \^etre vu comme un *NFA*, tout *NFA* peut \^etre vu comme un un *NFA* – ϵ .
- Th\'eor\^eme: Tout *NFA* – ϵ peut \^etre simul\'e par un *DFA*.

NFA – ϵ : formalisations

- Formalisation de la notion d'automate:

▶ ... $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$...

- Formalisation de la notion de mot reconnu:

▶ ... $\hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q, w)} \epsilon\text{-cl\^oture}(\delta(p, a)).$

$$\epsilon\text{-cl\^oture}(q) = \left\{ \begin{array}{l} \text{tous les \^etats joignables \^a partir de } q \\ \text{par une suite de } n \geq 0 \text{ transitions} \\ \text{\'etiquetees par } \epsilon \end{array} \right\} \dots$$

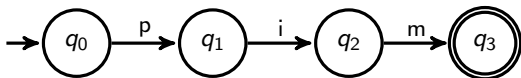
- Tout *DFA* peut \^etre vu comme un *NFA*, tout *NFA* peut \^etre vu comme un un *NFA – ϵ* .
- Th\'eor\^eme: Tout *NFA – ϵ* peut \^etre simul\'e par un *DFA*.

▶ ... m\^eme principe ...

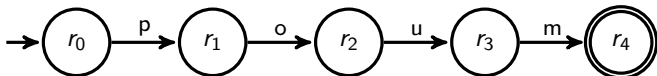
Un intérêt: permettre de combiner directement des automates.

■ Exemple:

- ▶ à partir d'un automate qui reconnaît *pim*



- ▶ et d'un automate qui reconnaît *poum*



- ▶ construire un automate qui reconnaît *pim|poum* à coup de copier coller:

