INF421-a
Bases de la programmation et de l'algorithmique

(Bloc 8/ 9)

Philippe Baptiste

CNRS LIX, École Polytechnique

13 octobre 2006

Aujourd'hui

Automates

Objets “mathématiques”, très utilisés en informatique, qui permettent de modéliser un grand nombre de systèmes (informatiques).

Quelques exemples classiques d'utilisation d'automates :

- Vérification d'un circuit électronique
- Recherche d'occurrence dans un texte (moteur de recherches sur le web, *etc.*)
- Vérification de protocoles de communication
- Compression de données
- Compilation
- Biologie (génomique)

En dehors de ces utilisations : modéliser les ordinateurs et comprendre ce qu'un ordinateur peut faire (décidabilité) et ce qu'il sait faire efficacement (complexité).

Symboles, mots, langage

- L'alphabet Σ est un ensemble de caractères (ou symboles).
- Un mot est une suite de caractères.
- L'ensemble des mots sur Σ est noté Σ^*.
- Un *langage* est un sous-ensemble de Σ^*, c'est-à-dire un ensemble particulier de mots.
- Parmi les mots de Σ^* on distingue le mot vide noté ϵ. Le mot vide est l'unique mot de longueur zéro.

Symboles, mots, langage

- La concaténation de deux mots m_1 et m_2 est le mot obtenu en mettant m_1 à la fin de m_2.
- On note $\cdot$ l'opérateur de concaténation (on omet souvent cet opérateur) : $m_1 \cdot m_2$ ou $m_1 m_2$.
- La concaténation est une opération associative qui possède un élément neutre : le mot vide ϵ.
- Dans l'écriture $m = m_1 m_2$, m_1 est le *préfixe* du mot m, tandis que m_2 est le *suffixe* du mot m

symboles, mots, langage

- La concaténation s'étend aux ensembles de mots, on note $L_1 \cdot L_2$ le langage obtenu en concaténant tous les mots de L_1 avec les mots de L_2.

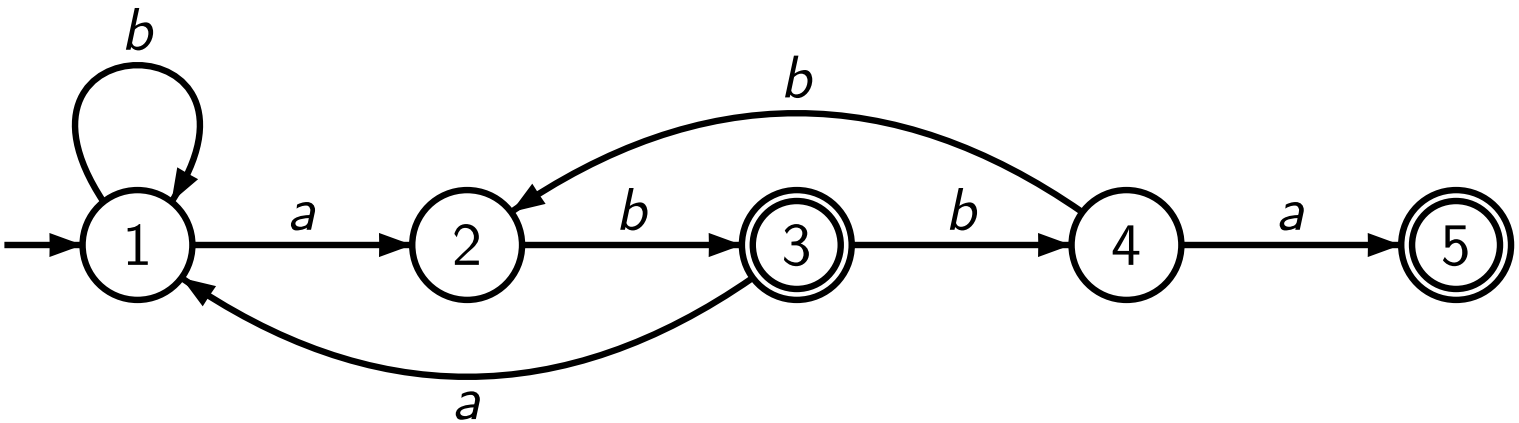
$$L_1 \cdot L_2 = \{m_1 \cdot m_2 \mid m_1 \in L_1 \wedge m_2 \in L_2\}$$

- On note $L^\star$ le langage obtenu en concaténant les mots de L.
  - $L^0 = \{\epsilon\}$
  - $L^{n+1} = L^n \cdot L$
  - $L^\star = \bigcup_{i \in \mathbb{N}} L^i$

  C'est-à-dire qu'un mot m de L^* est la concaténation de n mots ($n \geq 0$) $m_1, \ldots m_n$, où les m_i sont tous des mots de L.

Aujourd'hui

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 $(Q, \Sigma, \delta, q_0, F)$ constitué des éléments suivants

- un alphabet fini (Σ)
- un ensemble fini d'états (Q)
- une fonction de transition ($\delta : Q \times \Sigma \to Q$)
- un état de départ ($q_0 \in Q$)
- un ensemble d'états finaux (ou acceptant) $F \subseteq Q$

Fonctionnement d'un automate fini déterministe

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

- Le processus commence à l'état de départ q_0
- Les symboles du mot sont lus les uns après les les autres.
- À la lecture de chaque symbole, on emploie la fonction de transition δ pour se déplacer vers le prochain état
- le mot est reconnu si et seulement si le dernier état est un état de F.

Le langage associé à un automate est constitué de l'ensemble des mots qu'il reconnaît.

Formalisons : fonction de transition étendue aux mots

La *fonction de transition étendue aux mots*, $\hat{\delta}$ se définit récursivement comme suit

- À partir d'un état q en lisant le mot vide ϵ on reste dans l'état q, *i.e.*, $\forall q \in Q, \hat{\delta}(q, \epsilon) = q$
- Étant donné un mot c se terminant par $a \in \Sigma$ (*i.e.*, $c = c'a$ avec $c' \in \Sigma \cup \{\epsilon\}$), et un état q de Q, $\hat{\delta}(q, c) = \hat{\delta}(q, c'a) = \delta(\hat{\delta}(q, c'), a)$

Nous pouvons maintenant définir le langage $L(A)$ accepté par un automate fini déterministe $A = (Q, \Sigma, \delta, q_0, F)$.

$$L(A) = \{c | \hat{\delta}(q_0, c) \in F\}$$

Des représentations "compactes" des automates

On peut associer à un automate une *table de transition* qui décrit de manière extensive la fonction de transition δ :

- Une colonne correspond à un caractère de l'alphabet.
- Une ligne correspond à un état de l'automate (l'état initial est précédé d'une flèche "$\rightarrow$" ; l'état final d'une étoile "$*$")

La valeur $\delta(q, a)$ pour $q \in Q, a \in \Sigma$ correspond à l'état indiqué à l'intersection de la ligne q et de la colonne a.

Des représentations "compactes" des automates

	a	b
$\rightarrow 1$	1	2
$* 2$	1	2

- $Q = \{1, 2\}$
- $\Sigma = \{a, b\}$
- $\delta(1, a) = 1$, $\delta(1, b) = 2$, $\delta(2, a) = 1$, $\delta(2, b) = 2$
- $q_0 = 1$
- $F = \{2\}$

Le langage de cet automate est constitué exactement des mots composés de ...

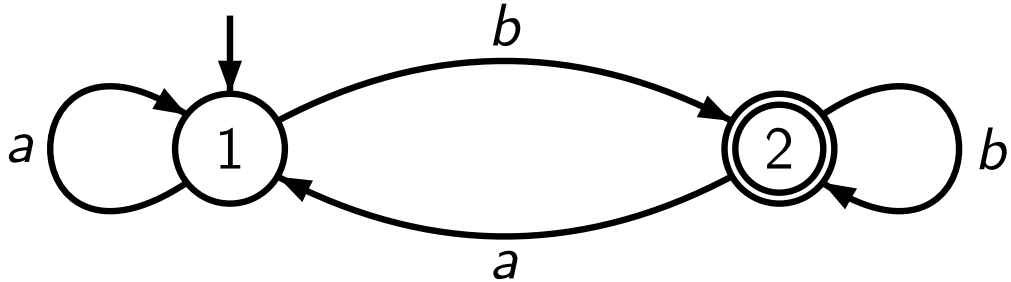
Des représentations "compactes" des automates

Le graphe de transition est constitué des éléments suivants :

- Un ensemble de sommets (chaque sommet représente un élément de Q).
- Un ensemble d'arcs entre les sommets valués par un symbole de σ (un arc entre les états q et q' valué par le symbole s signifie que $\delta(q, s) = q'$).
- L'état initial q_0 est marqué par une flèche entrante.
- Les états finaux F sont entourés d'une double ligne.

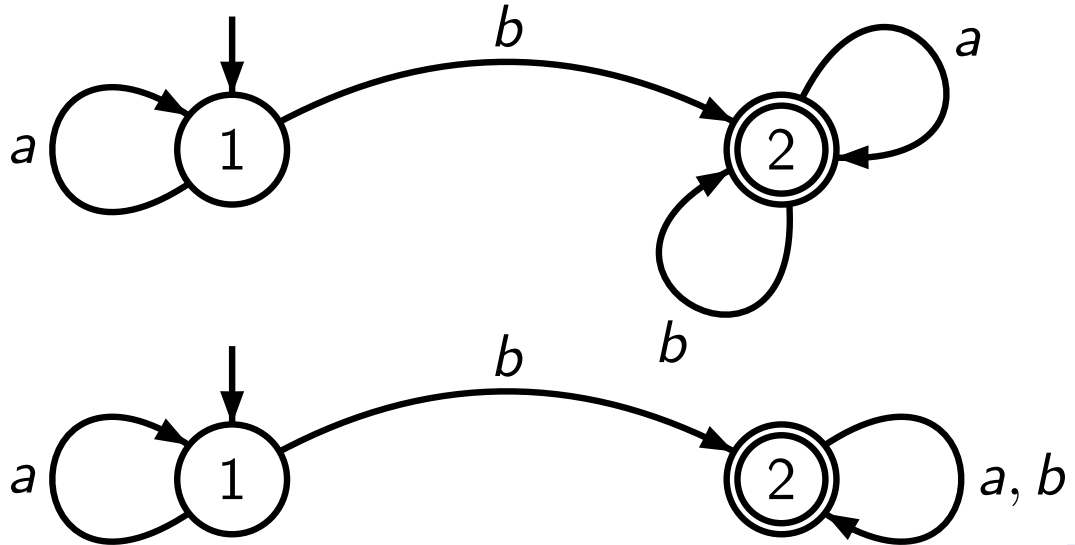
Des représentations “compactes” des automates

	a	b
$\rightarrow 1$	1	2
$* 2$	1	2



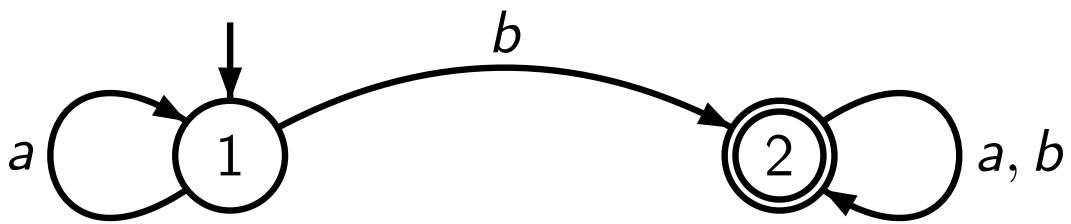
Des représentations "compactes" des automates

Pour simplifier encore cette représentation, un arc entre deux sommets q, q' peut être valué par plusieurs symboles $s_1, ..., s_n$ séparés par des virgules. Cette dernière convention signifie simplement que $\forall i \leq n, \delta(q, s_i) = q'$



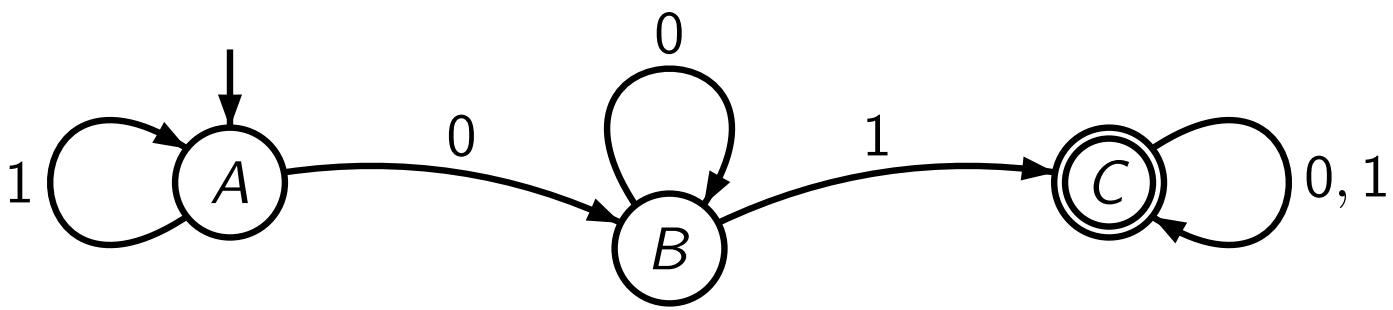
Exercice

Quel est le langage reconnu par



Exercice

Écrire la table de transition de l'automate suivant. Quel est le langage reconnu ?



Exercice

Soit l'automate fini déterministe $(\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$ donné par la table

δ	0	1
$\longrightarrow * q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Dessiner l'automate et montrer qu'il accepte "110101".

Exercice

Construire un automate fini déterministe qui reconnaît le langage

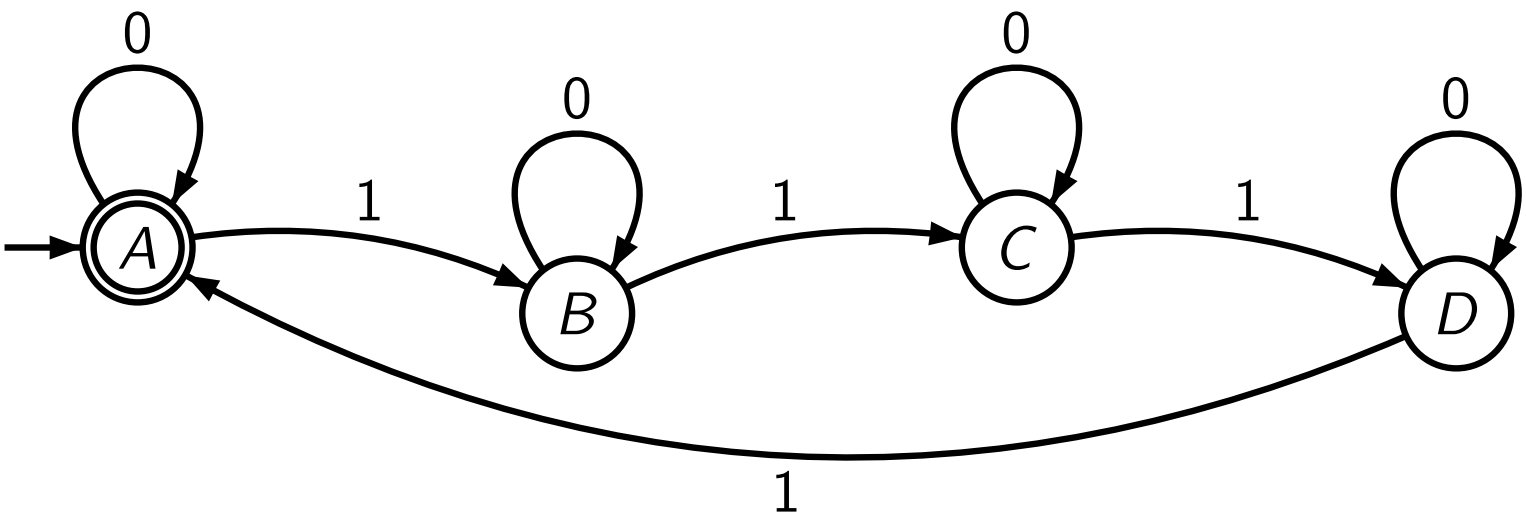
$$L = \{x \in \{0,1\}^* | n_1(x) \equiv 0 \mod 4\}$$

où $n_1(x)$ est le nombre d'occurrence du symbole 1 dans le mot x.

Exercice (suite)

Construire un automate fini déterministe qui reconnaît le langage

$$L = \{x \in \{0,1\}^* | n_1(x) \equiv 0 \bmod 4\}$$



Exercice

Construire les automates finis déterministes qui reconnaissent les langages suivants

$L_1 = \{m \in (a+b)^* | \text{chaque } a \text{ de } m \text{ est imm. précédé et suivi d'un } b\}$
$L_2 = \{m \in (a+b)^* | m \text{ contienne à la fois } ab \text{ et } ba\}$
$L_3 = \{m \in (a+b)^* | m \text{ contienne exactement une occurrence de } aaa\}$

Aujourd'hui

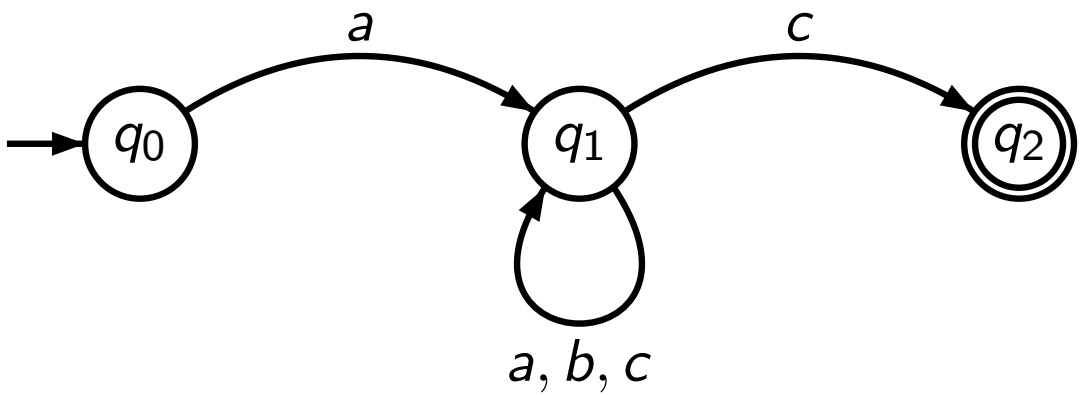
Automate fini non-déterministe

Il peut y avoir *plusieurs* transitions avec le même symbole. Le fonctionnement d'un tel automate n'est donc PAS totalement « DÉTERMINÉ », car on ne sait pas quel état l'automate va choisir.

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

- un alphabet fini (Σ)
- un ensemble fini d'états (Q)
- une fonction de transition δ qui associe à tout état $q \in Q$ et tout symbole $s \in \Sigma$ un sous ensemble de Q noté $\delta(q, s)$.
- un état de départ (q_0)
- un ensemble d'états finaux (ou acceptant) F

Fonctionnement d'un automate fini non-déterministe



	a	b	c
$\rightarrow q_0$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_1	$\{q_1\}$	$\{q_1\}$	$\{q_1, q_2\}$
$* q_2$	$\emptyset$	$\emptyset$	$\emptyset$

Fonctionnement d'un automate fini non-déterministe

Comme pour les automates déterministes, nous introduisons la *fonction de transition étendue aux mots*, $\hat{\delta}$.

- À partir d'un état q en lisant le mot vide ϵ, on reste dans l'état q, *i.e.*, $\forall q \in Q, \hat{\delta}(q, \epsilon) = \{q\}$
- Étant donné un mot c se terminant par $a \in \Sigma$ (*i.e.*, $c = c'a$ avec $c' \in \Sigma \cup \{\epsilon\}$), et un état q de Q,

$$\hat{\delta}(q, c) = \hat{\delta}(q, c'a) = \bigcup_{p \in \hat{\delta}(q, c')} \delta(p, a)$$

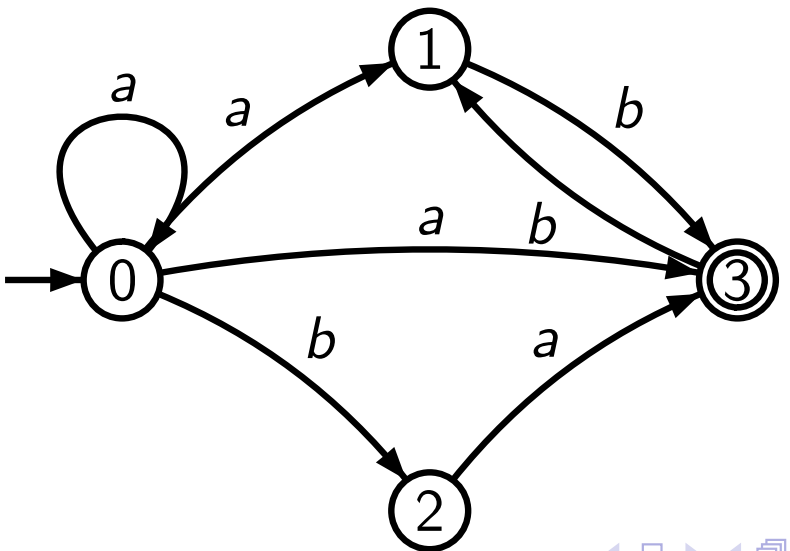
Nous pouvons maintenant définir le langage $L(A)$ accepté par un automate fini non-déterministe $A = (Q, \Sigma, \delta, q_0, F)$.

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

Exercice

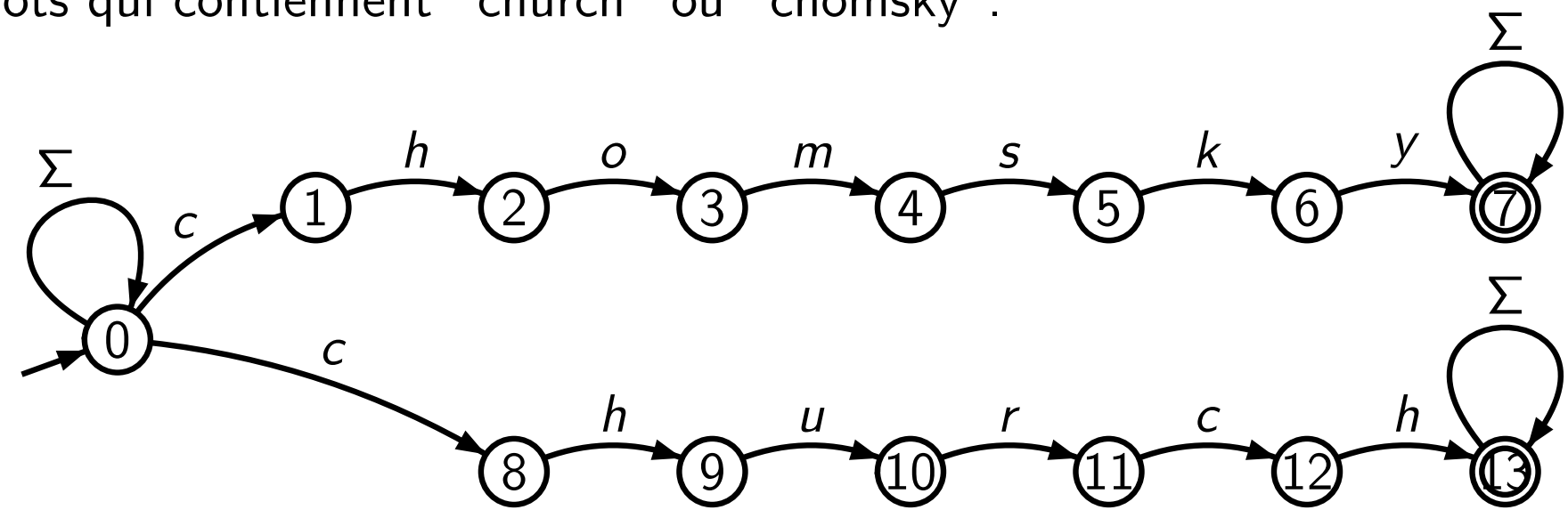
Construire l'automate fini non-déterministe associé à

	a	b
$\rightarrow 0$	$\{0,1,3\}$	$\{2\}$
1	$\emptyset$	$\{3\}$
2	$\{3\}$	$\emptyset$
$* 3$	$\emptyset$	$\{1\}$



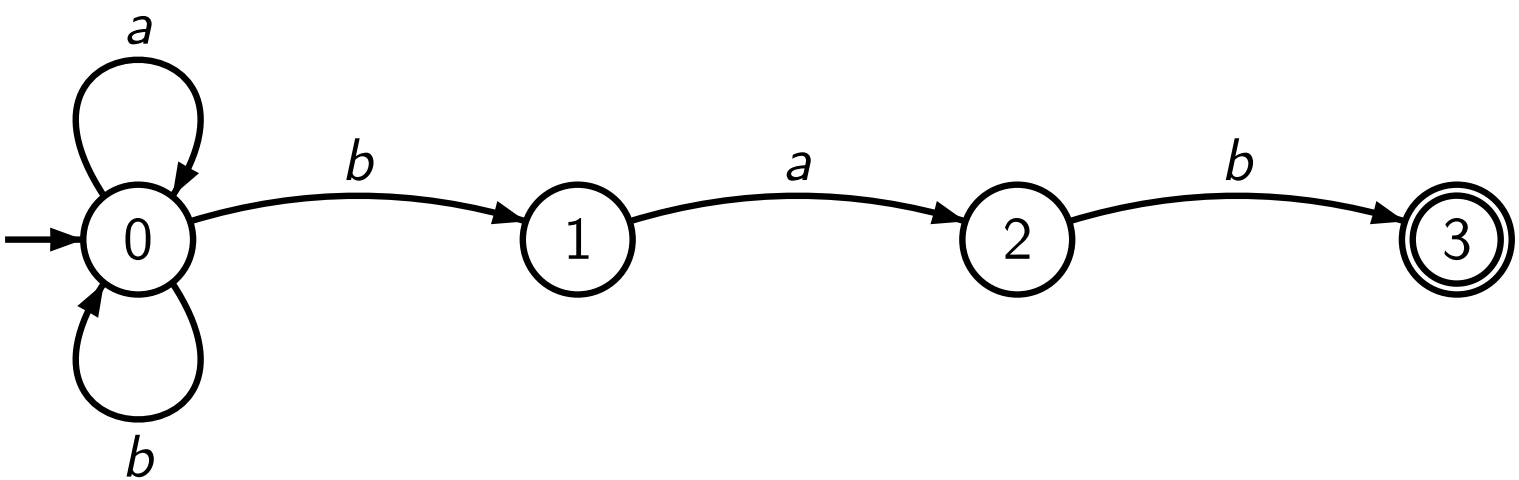
Exercice

Construire un automate fini non-déterministe qui reconnaît les mots qui contiennent “church” ou “chomsky”.



Exercice

Construire un automate fini non-déterministe qui reconnaît les mots de l'alphabet $\{a, b\}$ qui terminent par bab.



Exercice

Construire un automate fini non-déterministe qui reconnaît les nombres dont le dernier chiffre n'apparaît qu'une fois.

Déterminisation d'un automate fini non-déterministe (Rabin-Scott)

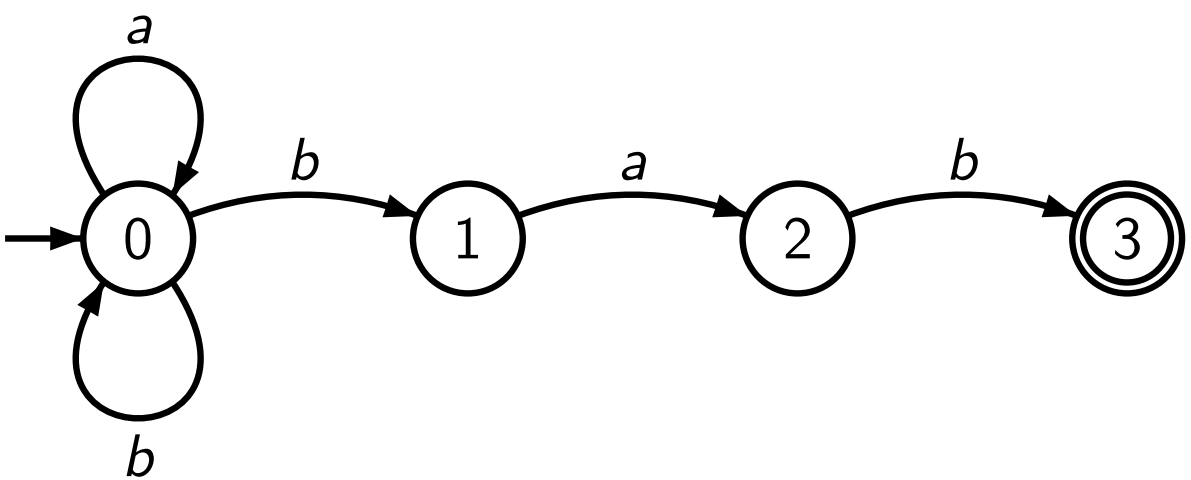
Considérons un automate fini non-déterministe $A_n = (Q_n, \Sigma, \delta_n, q_0, F_n)$ et construisons un automate fini déterministe $A_d = (Q_d, \Sigma, \delta_d, \{q_0\}, F_d)$ équivalent.

- Les alphabets de A_n et de A_d sont identiques.
- Les états de départ sont respectivement q_0 et le singleton $\{q_0\}$.
- Q_d est constitué de tous les sous-ensembles de Q_n.
- F_d est l'ensemble des sous-ensembles de Q_n qui contiennent au moins un élément de F_n.
- Étant donné un sous-ensemble S de Q_n et un symbole $a \in \Sigma$, on définit ainsi la fonction de transition $\delta_d(S, a)$

$$\delta_d(S, a) = \bigcup_{q \in S} \delta_n(q, a).$$

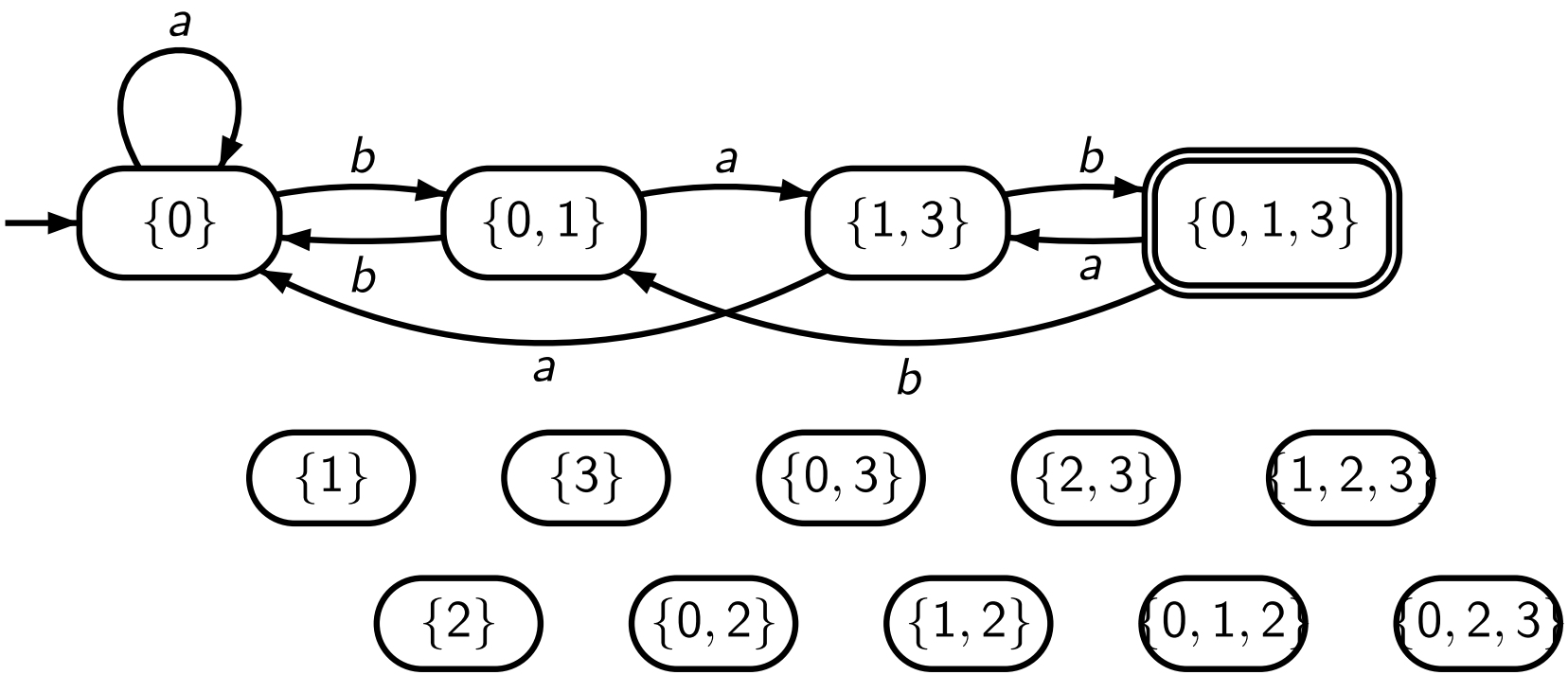
Déterminisation (Rabin-Scott)

Construire un automate fini non-déterministe reconnaissant les mots de l'alphabet $\{a, b\}$ qui terminent par bab.



Déterminisation (Rabin-Scott)

Essayons maintenant de le déterminiser en construisant un nouvel état à partir de chaque sous-ensemble d'état possible.



Remarquons que les états $\{1\}$, $\{2\}$, $\{3\}$, $\{0,2\}$, $\{0,3\}$, $\{1,2\}$, $\{2,3\}$, $\{0,1,2\}$, $\{1,2,3\}$, $\{0,2,3\}$ sont inatteignables

Déterminisation (Rabin-Scott)

En pratique, lors de la conversion, on ne crée pas immédiatement tous les états de l'automate fini déterministe. Les états "utiles" sont crées quand on en a besoin en suivant la méthode de construction ci-dessous :

- Q_d est initialisé à $\emptyset$ et soit E un ensemble d'états initialisé à $E = \{\{q_0\}\}$
- Tant que E est non vide,
  - choisir un élément S de E (S est donc un sous-ensemble de Q_n),
  - ajouter S à Q_d,
  - pour tout symbole $a \in \Sigma$,
    - calculer l'état $S' = \bigcup_{q \in S} \delta_n(q, a)$
    - si S' n'est pas déjà dans Q_d, l'ajouter à E
    - ajouter un arc sur l'automate entre S et S' et la valuer par a

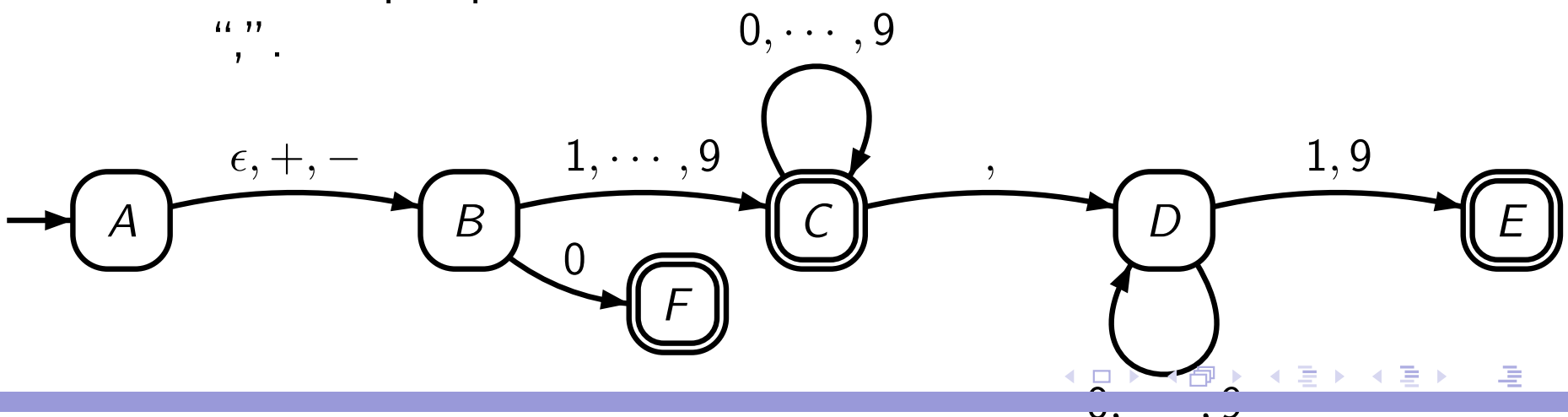
Les ϵ transitions

Une ϵ transition (notée ϵ sur l'arc d'un automate) permet de passer d'un état à l'autre d'un automate sans lire de symbole.

Cette facilité permet de programmer facilement des automates complexes.

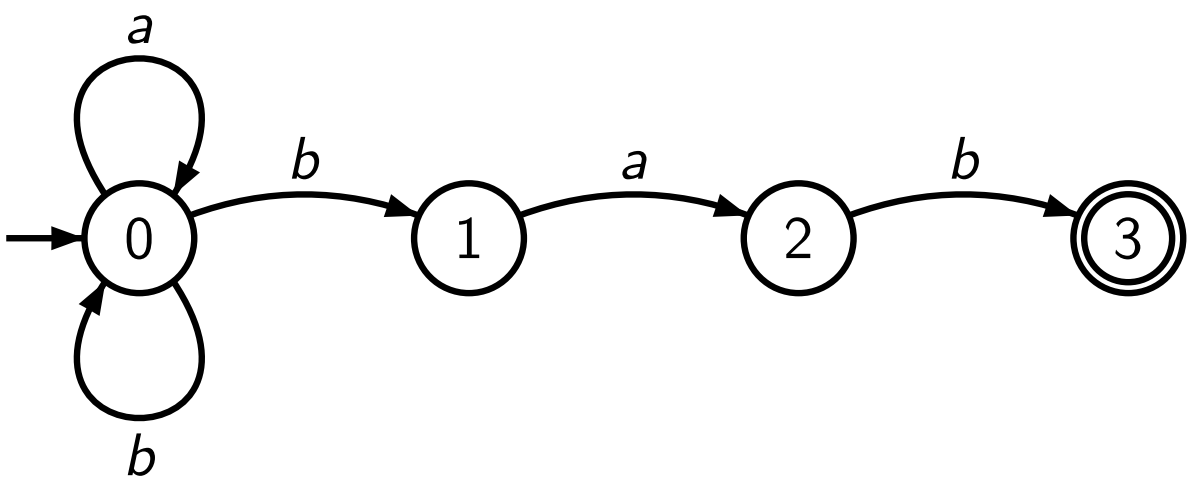
Les ϵ transitions : les décimaux

- Un nombre décimal = un nombre réel qui est le quotient d'un entier relatif par une puissance de dix.
- On souhaite pouvoir écrire le nombre décimal en commençant par un "+" ou un "-', suivi d'une suite de chiffres, d'une virgule et d'une suite de chiffres
- Le "+" ou le "-" sont optionnels, la première chaîne de chiffres ne peut pas être vide et ne commence pas par "0" (sauf si le nombre décimal est 0). La seconde chaîne ne se termine pas par "0". Si seconde chaîne est vide, on omet la ",".



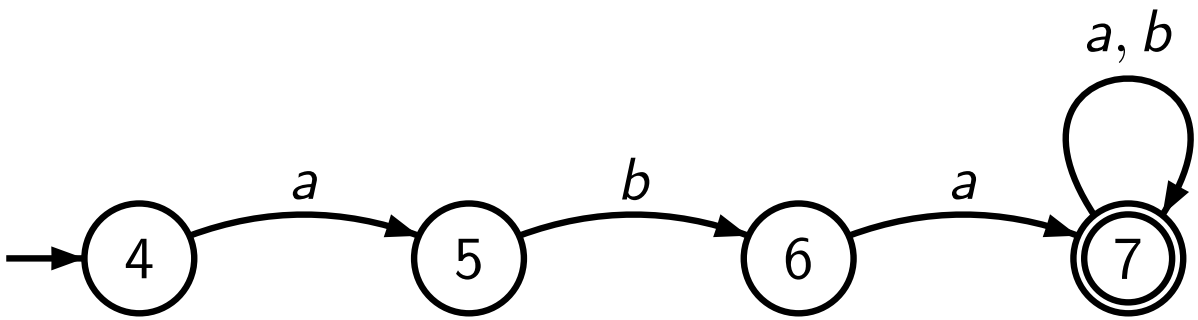
Exercice

On cherche à construire un automate qui reconnaît les mots qui se terminent par *bab* OU qui commencent par *aba*.
On sait construire un automate qui reconnaît les mots qui se terminent par *bab*



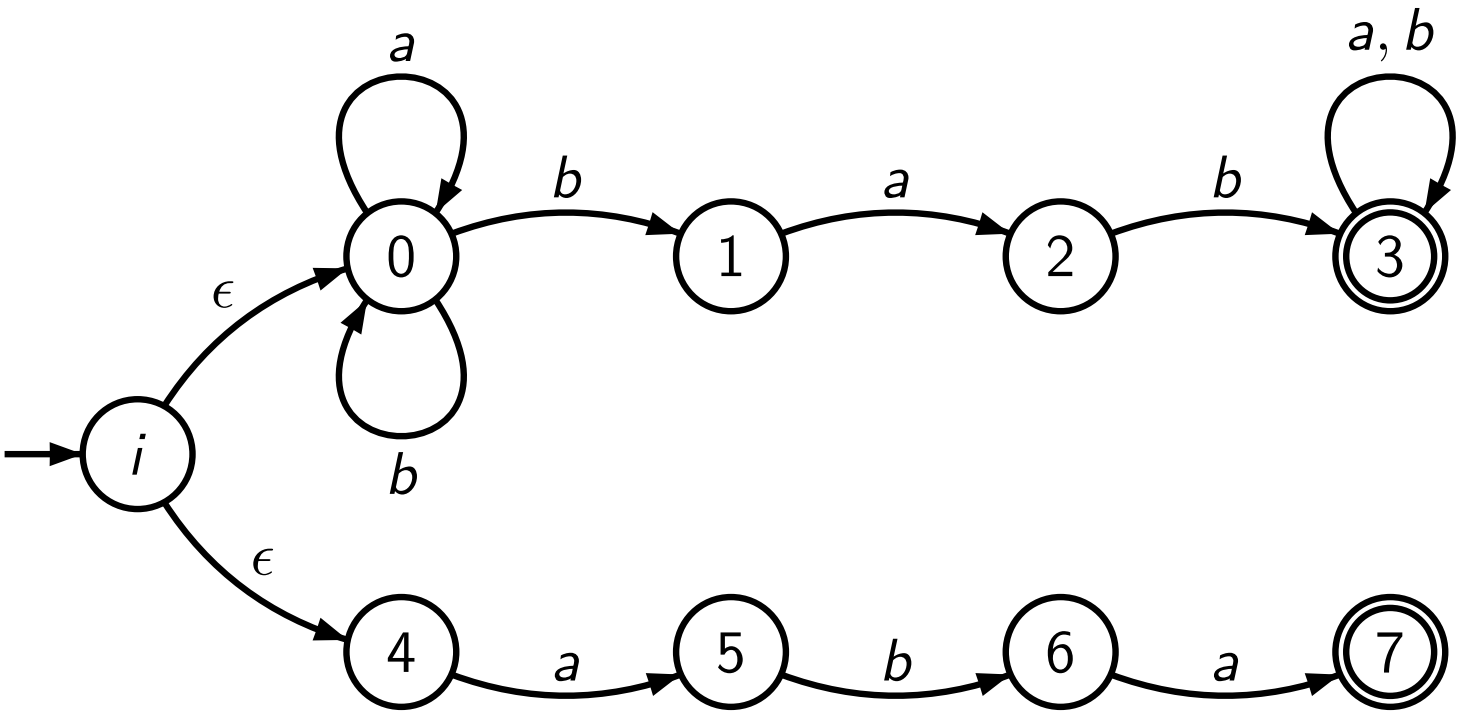
Exercice

Il est facile de construire un automate qui reconnaît les mots qui commencent par *aba*.



Exercice

Il suffit alors d'assembler ces automates avec une simple ϵ transition.



Aujourd'hui

Java

Objectif

- Modéliser un automate fini non-déterministe (sans ϵ transition)
- Écrire un programme qui détermine si une chaîne de caractère est reconnue par l'automate.

Sans perte de généralité, nous allons supposer que l'alphabet est l'ensemble des caractères ASCII et que les états sont *numérotés* à partir de 0.

Modèle

Le modèle de données est alors très simple :

- L'état initial de l'automate est indiqué par un entier `q0`.
- La table de transition `delta` est un tableau bidimensionnel de listes d'entiers (la liste d'entier étant ici le moyen le plus simple de représenter un ensemble d'états). Ainsi `delta[q][c]` est la liste des états atteignables à partir de `q` en lisant le caractère `c`.
- L'ensemble des états `finaux` est une liste d'entiers.
- Enfin, le mot que l'on cherche à reconnaître est une chaîne de caractères `String mot`.

Modèle

Soit donc en Java les deux classes `Liste` et `Automate`.

```
class Liste {
    int val;
    Liste suivant;
    Liste(int v, Liste x) {
        val = v; suivant = x; }
}
class Automate {
    int q0;              // état initial
    Liste[][] delta;     // fonction de transition
    Liste finaux;        // états finaux
    Automate(int q, Liste f, Liste[][]d) {
        q0 = q;
        delta = d;
        finaux = f;
    }
}
```

Algorithme de recherche

Nous aurons besoin de quelques fonctions classiques sur les listes

```
static int longueur(Liste x) {              // La longueur d'une liste
    if (x == null) return 0;
    else return 1 + longueur(x.suivant);
}
static int kieme(Liste x, int k) {          // Le k ème élément
    if (k == 1) return x.val;
    else return kieme(x.suivant, k-1);
}
static boolean estDans(Liste x, int v) { // Le test d'appartenance
    if (x == null) return false;
    else return x.val == v || estDans(x.suivant, v);
}
```

Algorithme de recherche

La fonction `accepter(String mot, Automate a)` qui permet de vérifier qu'un mot `mot` est accepté par l'automate `a` appelle la fonction `static boolean accepter(String mot, Automate a, int i, int q)`.

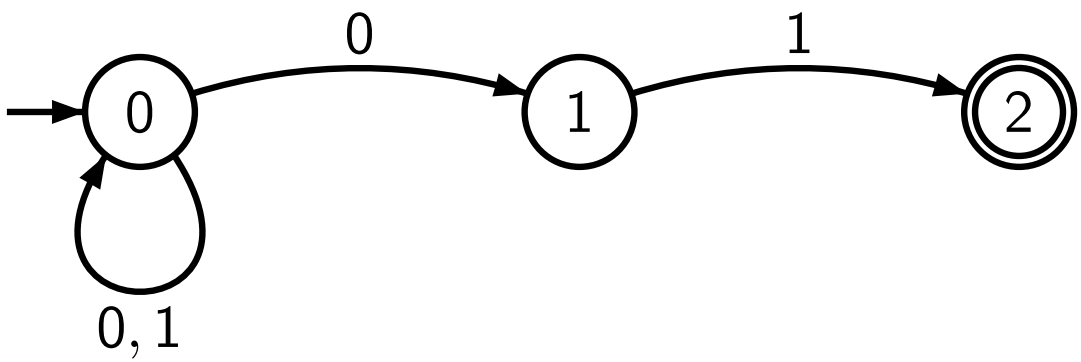
```
static boolean accepter(String mot, Automate a) {
    return accepter(mot, a, 0, a.q0);
}
```

Algorithme de recherche

```
static boolean accepter(String mot, Automate a, int i, int q) {
    if (i == mot.length()) return Liste.estDans(a.finaux, q);
    else {
        boolean resultat = false;
        int c = mot.charAt(i);   // le code ASCII du caractère courant
        for (Liste nv_q = a.delta[q][c];
             nv_q ≠ null;
             nv_q = nv_q.suivant)
            resultat = resultat || accepter(mot, a, i+1, nv_q.val);
        return resultat;
    }
}
```

Mise en œuvre sur un automate

Considérons l'automate suivant qui accepte toutes les chaînes qui se terminent par "01".



La table associée à cet automate est alors :

	0	1
$\rightarrow 0$	$\{0,1\}$	$\{0\}$
1	$\emptyset$	$\{2\}$
$* 2$	$\emptyset$	$\emptyset$

Mise en œuvre sur un automate

```
public static void main(String [] arg) {
    Liste[][] delta = new Liste[3][128];
    delta[0][(int)'0'] = new Liste(0, new Liste(1, null));
    delta[0][(int)'1'] = new Liste(0, 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);
    System.out.println("accepter = " + accepter(arg[0], a));
}
```

Remarquons que le code ASCII du caractère `'0'` est obtenu par `(int)'0'`.