INF 421 Luc Maranget

Automates

Luc.Maranget@inria.fr
http://www.enseignement.polytechnique.fr/profs/informatique/Luc.Maranget/421/

Automates

Très utilisés en informatique.

- Vérifications de circuits (quand ce sont des automates !).
- Recherche efficace dans le texte (moteur de recherche sur le web, `egrep`, ...).
- Vérification des protocoles de communication.
- Compilation (reconnaissance des mots du langage compilé).
- Biologie (motifs dans les séquences d'ADN).
- Modélisation des calculs possibles (décidabilité), Ici les automates (finis) définissent la classe de ce qu'il est possible de faire simplement (disons sans écrire dans la mémoire).

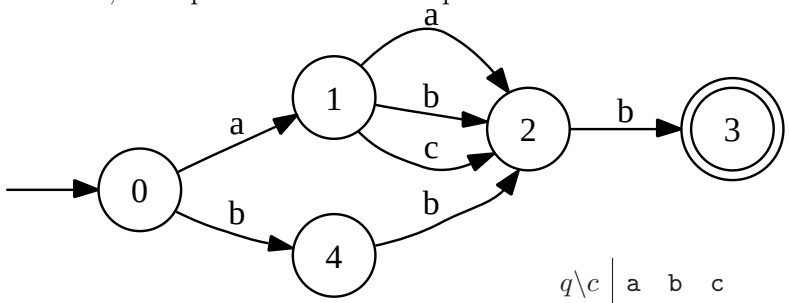
Automate finis déterministes (DFA)

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

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

Exemple simple de DFA

Idéalement, on exprime les automates par des dessins :



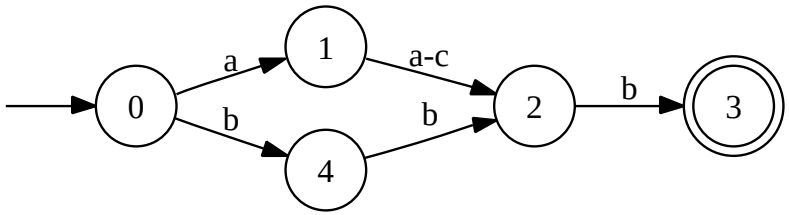
Ici $Q = \{0, 1, 2, 3, 4\}$, $q_0 = 0$, $F = \{3\}$, et $\delta =$

$q \backslash c$	a	b	c
0	1	4	
1	2	2	2

...

Notation compacte

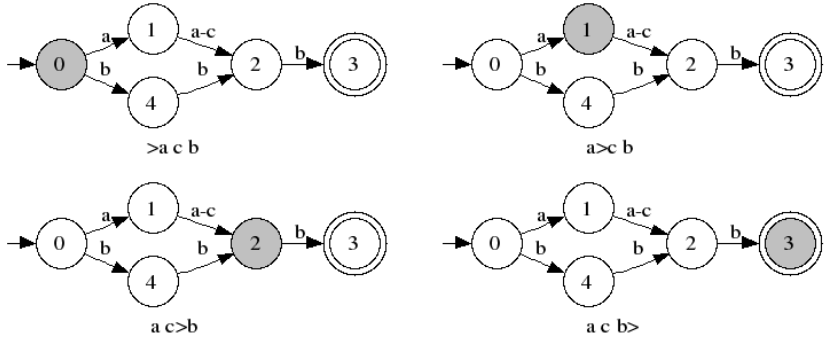
Transitions étiquetées par des ensembles de caractères.



La transition notée `a-c` compte pour trois.

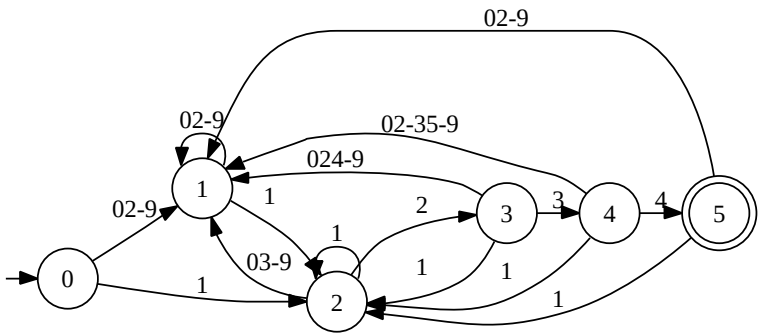
Exécution d'un DFA

- Démarrer dans l'état initial.
- Lire les caractères de l'entrée en effectuant les transitions (NB : si blocage, échec).
- À la fin le mot lu est reconnu si l'état courant est final.



Exemple (utile ?) de DFA

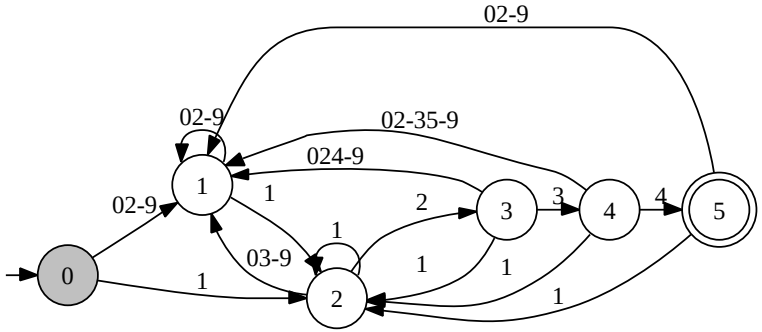
Cet automate est un digicode cablé !



L'automate reconnaît un code à quatre chiffres, lequel ? `1234`.

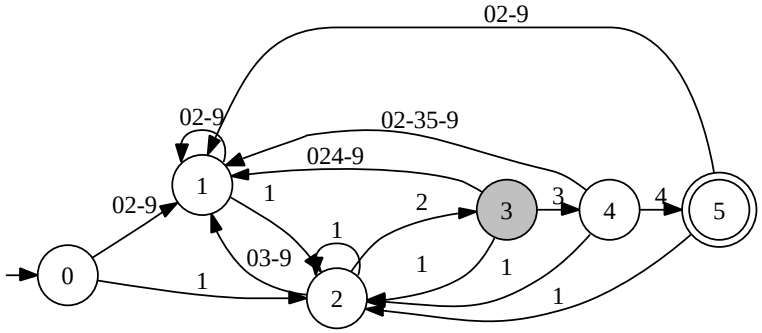
Reconnaissance de la séquence `1211234`

Initialement :



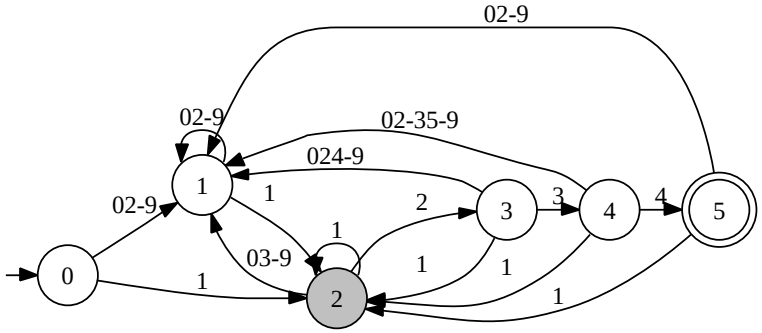
>1 2 1 1 2 3 4

Lecture des deux premiers chiffres



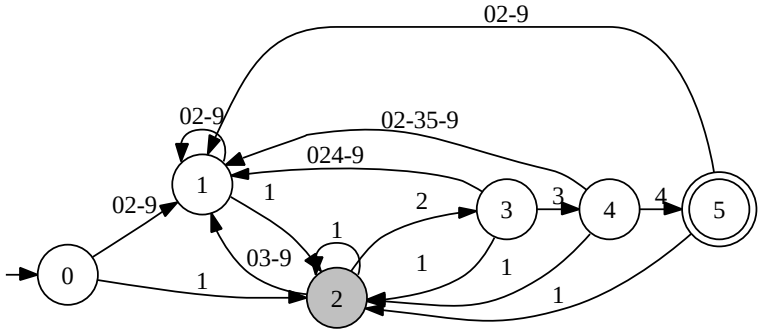
1 2>1 1 2 3 4

Lecture de 1, retour en arrière



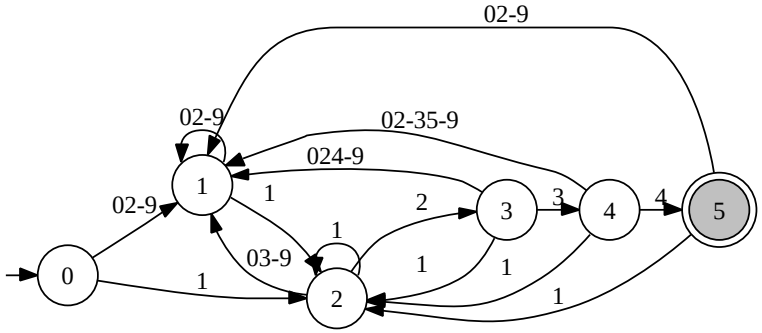
1 2 1>1 2 3 4

Lecture de 1, on ne bouge plus



1 2 1 1>2 3 4

Lecture du reste du code et succès



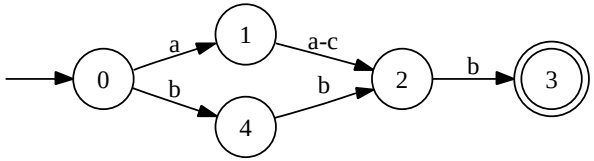
1 2 1 1 2 3 4>

Définition formelle de l'exécution

Si $\delta(q, c) = q'$, on note la transition effectuée :

$$q \xrightarrow{c} q'$$

Ainsi la lecture de `abc` par



s'écrit :

$$0 \xrightarrow{\texttt{a}} 1 \xrightarrow{\texttt{c}} 2 \xrightarrow{\texttt{b}} 3$$

C'est une reconnaissance parce que 0 est initial et 3 final.

Consommation des mots

Soit un mot $m = a_0 a_1 \cdots a_{n-1}$ et une lecture

$$q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots q_{n-1} \xrightarrow{a_{n-1}} q_n$$

On abrège en :

$$q_0 \xrightarrow{m*} q_n$$

Un mot est reconnu par l'automate si et seulement si

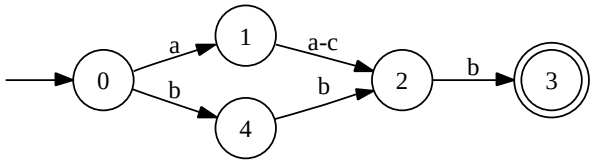
$$q_0 \xrightarrow{m*} q_n, \quad \text{avec } q_n \in F$$

Langage reconnu (défini) par un DFA

Tout simplement l'ensemble des mots reconnus par le DFA.

$$\{m \in \Sigma^\star \mid q_0 \xrightarrow{m*} q, q \in F\}$$

Le langage reconnu par



est : $\{\,\texttt{aab}, \texttt{abb}, \texttt{acb}, \texttt{bbb}\,\}$.

Implémentation possible des DFA

Une classe `Auto`.

```
class Auto {
  State init ;
  Set<State> end ;
  ... }
```

Classe des états :

```
class State {
  /* Identifiant unique (pratique) */
  int id ;
  /* Tableau des transitions, indexé par char (dans [0..256[) */
  State [] delta ;
  ... }
```

Structure « creuse » pour les transitions ? `Map<Character,State>`.

Implémentation de la reconnaissance

Méthode des objets `Auto`.

```
boolean isAccepted(String txt) {
  State q = init ;
  for (int k = 0 ; k < txt.length() ; k++) {
    char c = txt.charAt(k) ;
    if (q.delta[c] == null) return false ; // Bloqué
    q = q.delta[c] ;
  }
  return end.contains(q) ;
}
```


Automates finis non-deterministes (NFA)

Un NFA est un quintuplet $(\Sigma, Q, \delta, q_0, F)$ où

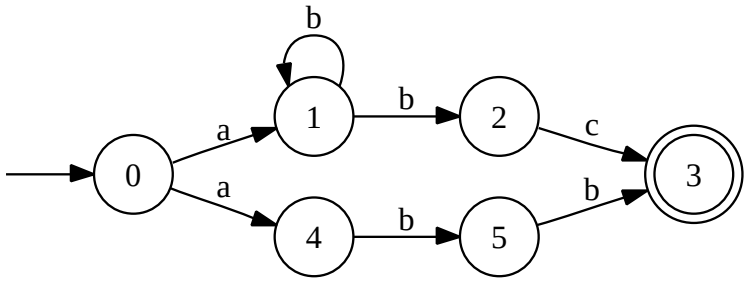
- Σ est un alphabet;
- Q est un ensemble fini d'états;
- $\delta : Q \times \Sigma \to Q$ est la relation de transition;
- q_0 est l'état initial;
- $F \subseteq Q$ est un ensemble d'états finaux.

Changement : Fonction $\to$ Relation.

On peut avoir :

$$q \xrightarrow{c} q', \quad q \xrightarrow{c} q'', \quad \text{et } q' \neq q''$$

Exemple simple de NFA



Ou sont les ambiguïtés ?

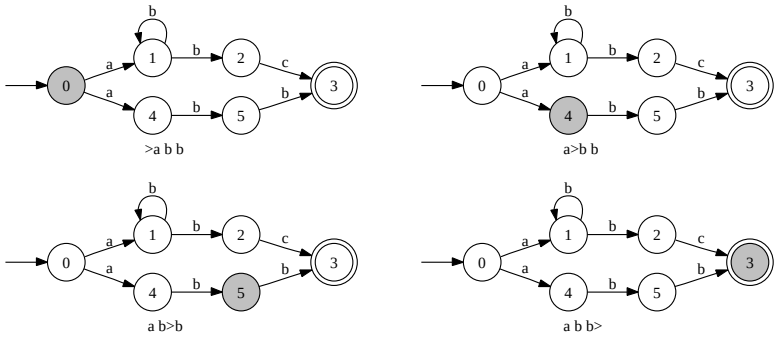
- On a $0 \xrightarrow{\mathtt{a}} 1$ et $0 \xrightarrow{\mathtt{a}} 4$.
- On a $1 \xrightarrow{\mathtt{b}} 1$ et $1 \xrightarrow{\mathtt{b}} 2$.

Reconnaissance par un NFA

La définition ne change pas.

$$\{m \in \Sigma^\star \mid q_0 \xrightarrow{m}{}^* q, q \in F\}$$

Exemple, reconnaissance de `abb`.



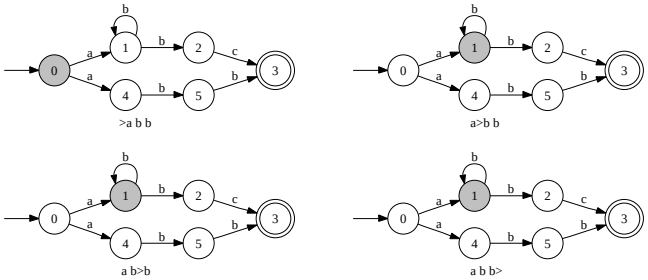
Mais alors, qu'est-ce qui change ?

Il devient plus délicat, étant donnés un automate et un mot, de décider si l'automate reconnaît le mot.

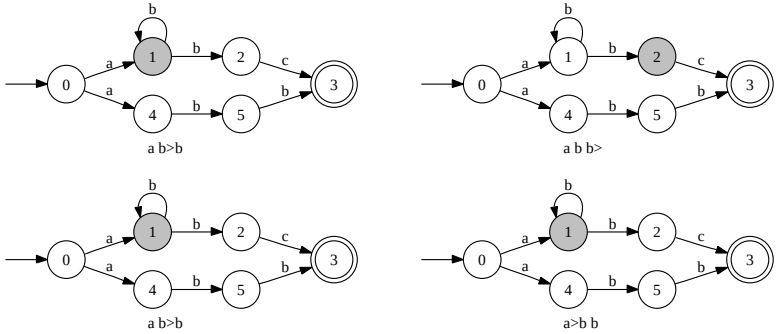
On procède par essai et retour en arrière (*backtrack*).

Exemple, essayer de reconnaître `abb`. Quatre premières étapes.

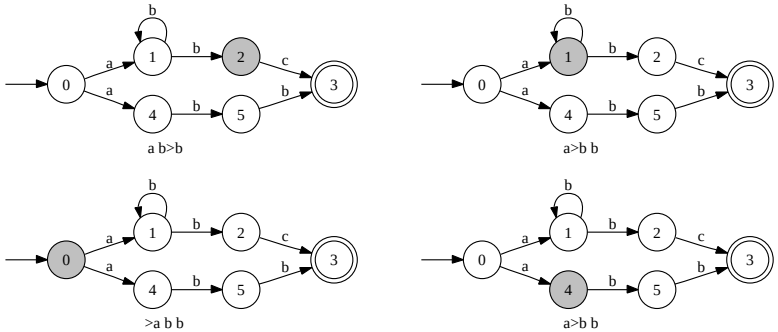
$0 \xrightarrow{\texttt{a}} 1 \xrightarrow{\texttt{b}} 1 \xrightarrow{\texttt{b}} 1 \xrightarrow{\texttt{b}} 1$, échec, car 1 n'est pas final



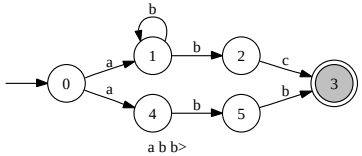
$1 \xleftarrow{\texttt{b}} 1 \xrightarrow{\texttt{b}} 2$ (échec) $1 \xleftarrow{\texttt{b}} 1 \xleftarrow{\texttt{b}} 1$



$1 \xrightarrow{\texttt{b}} 2$ (bloqué) $1 \xleftarrow{\texttt{b}} 1 \xleftarrow{\texttt{a}} 0 \xrightarrow{\texttt{a}} 4$



Et c'est bien parti...



Implémentation de la reconnaissance par NFA

Représentation de la relation δ : une fonction vers un ensemble.

```
class State {
  ...
  Set<State> [] delta ;
  ...
}
```

Essai et retour en arrière, veut dire tout essayer.

```
boolean isAccepted(String txt) {
  return run(txt, 0, init) ;
}
```

Implémentation du *backtracking*

Récursivement bien sûr.

```
boolean run(String txt, int k, State q) {
  if (k >= txt.length()) {
    return end.contains(q) ;
  } else {
    char c = txt.charAt(k) ;
    // Pour tous les états n tels que q --c--> n.
    for (State n : q.delta[c]) {
      if (run(txt, k+1, n))
        return true ; // Trouvé (chemin vers état final)
    }
    return false ; // Pas trouvé
  }
}
```


Encore une sorte d'automate, le NFA-ϵ

Un automate fini non déterministe avec transitions spontanées (ϵ-transitions) est un sextuplet $(\Sigma, Q, \delta, q_0, F, \epsilon)$ où

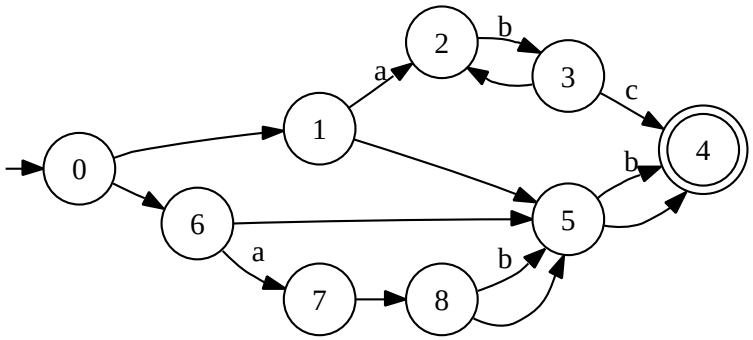
- $(\Sigma, Q, \delta, q_0, F)$ est un NFA.
- ϵ est une relation entre états.

Il y a des transitions spontanées. Si q et q' sont en relation par ϵ, c'est-à-dire sans consommer de caractère, on note $q \longrightarrow q'$.

Attention Dans la littérature, on appelle souvent les NFA-ϵ, NFA tout court.

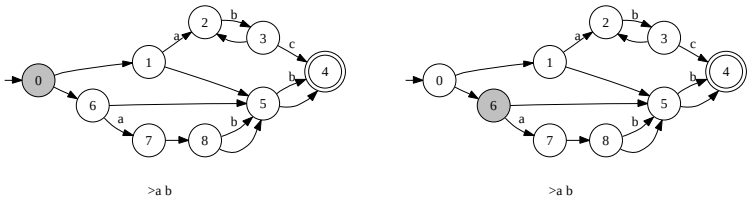
Exemple simple de NFA-ϵ

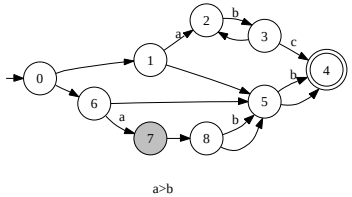
On dessine les transitions spontanées sans aucune étiquette.

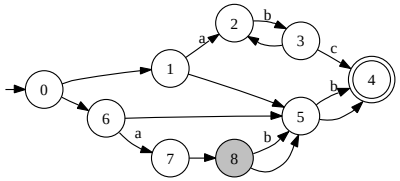


Reconnaissance par un NFA-ϵ

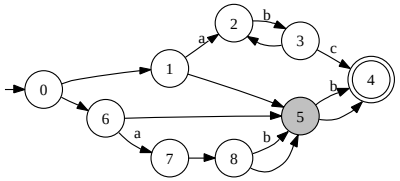
Exemple, reconnaissance de `ab`. $0 \longrightarrow 6 \xrightarrow{\texttt{a}} 7 \longrightarrow 8 \longrightarrow 5 \xrightarrow{\texttt{b}} 4$



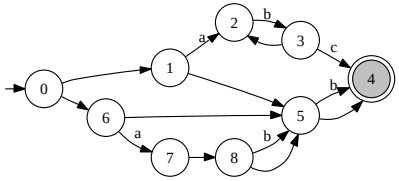




a>b



a>b



a b>

Définitions formelles

- La lecture des mots, notée $q \xrightarrow{m}{}^{*} q'$ tient compte des deux sortes de transitions.

$$\textsc{Read}\ \frac{q \xrightarrow{c} q'' \qquad q'' \xrightarrow{m}{}^{*} q'}{q \xrightarrow{cm}{}^{*} q'} \qquad \textsc{Epsilon}\ \frac{q \longrightarrow q'' \qquad q'' \xrightarrow{m}{}^{*} q'}{q \xrightarrow{m}{}^{*} q'}$$

$$\textsc{Init}\ \ q \xrightarrow{\epsilon}{}^{*} q$$

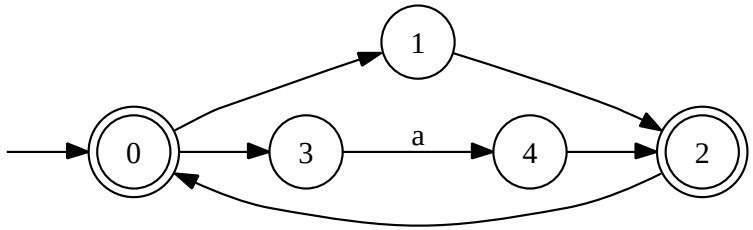
- Langage reconnu, rien de neuf.

$$\{m \in \Sigma^{\star} \mid q_0 \xrightarrow{m}{}^{*} q, q \in F\}$$

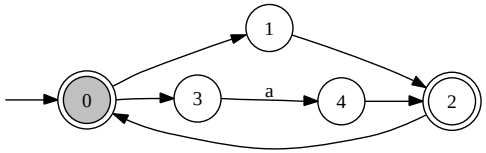
Algorithme de reconnaissance

Le backtracking fonctionne, mais est plus délicat que dans les NFA.

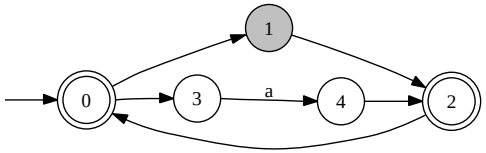
À cause des boucles de transitions spontanées.



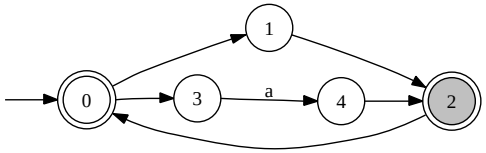
$$0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 0 \longrightarrow 1 \longrightarrow \ldots$$



>a



>a



>a

Une nouvelle technique d'implémentation

Considérons des transitions entre ensembles d'états.

- Transitions étiquetées, $Q \xrightarrow{c} Q'$, avec

$$Q' = \{q' \mid \exists q \in Q, q \xrightarrow{c} q'\}$$

  Extension naturelle aux ensembles de $\xrightarrow{c}$.

- Transitions spontanées $Q \longrightarrow^* Q'$, avec

$$Q' = \{q' \mid \exists q \in Q, q \longrightarrow \cdots \longrightarrow q'\}$$

  Extension aux ensembles de la fermeture transitive de $\rightarrow$.

Les symboles $\xrightarrow{c}$ et $\longrightarrow^*$ traduisent des fonctions entre ensembles d'états (notées $c(Q)$ et $\epsilon^*(Q)$).

Lecture des mots avec les ensembles d'états

Définie comme :

$$\textsc{Read}\ \frac{Q \xrightarrow{c} Q'' \qquad Q'' \xrightarrow{m}{}^* Q'}{Q \xrightarrow{cm}{}^* Q'} \qquad \textsc{Epsilon}\ \frac{Q \longrightarrow^* Q'' \qquad Q'' \xrightarrow{m}{}^* Q'}{Q \xrightarrow{m}{}^* Q'}$$

$$\textsc{Init}\ \ Q \xrightarrow{\epsilon}{}^* Q$$

En outre, on peut se contenter d'alterner les étapes $\longrightarrow^*$ et $\xrightarrow{c}$, parce que

$$Q \longrightarrow^* Q' \longrightarrow^* Q'' \quad \Rightarrow \quad Q' = Q'' = \epsilon^*(Q)$$

Donc, soient Q et m, Q' tel que $Q \xrightarrow{m}{}^* Q'$ est uniquement déterminé (commencer et finir par une étape $\longrightarrow^*$).

Reconnaissance des mots avec ensembles d'états

Le mot $m = a_0 a_1 \cdots a_{n-1}$ est reconnu si et seulement si.

$$\{\, q_0 \,\} \longrightarrow^* Q_1 \xrightarrow{a_0} Q_2 \longrightarrow^* Q_3 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} Q_{2n} \longrightarrow^* Q_{2n+1}$$

Avec $Q_{2n+1} \cap F \neq \emptyset$. C'est-a-dire

$$\epsilon^*(a_{n-1}(\cdots a_1(\epsilon^*(a_0(\epsilon^*(\{q_0\}))))\cdots)) \cap F \neq \emptyset$$

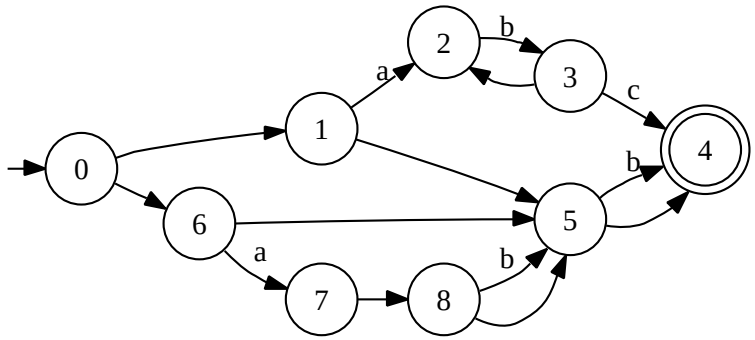
Correct, car équivalent par définition des transitions sur les ensembles à l'existence de :

$$q_0 \longrightarrow \cdots \longrightarrow q_1 \xrightarrow{a_1} q_2 \longrightarrow \cdots \longrightarrow q_3 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} q_{2n} \longrightarrow \cdots \longrightarrow q_{2n+1}$$

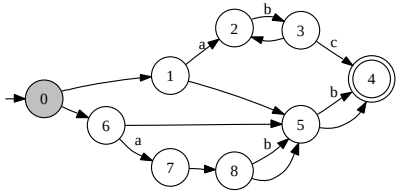
Avec $q_{2n+1} \in F$

Effectuer toutes les transitions à la fois

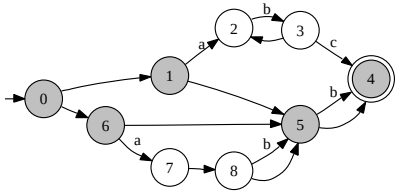
Exemple, reconnaissance de `ab`.



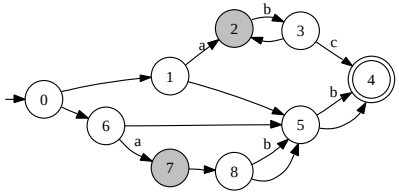
$$\{0\} \longrightarrow^* \{0,1,4,5,6\} \xrightarrow{\texttt{a}} \{2,7\}$$



>a b

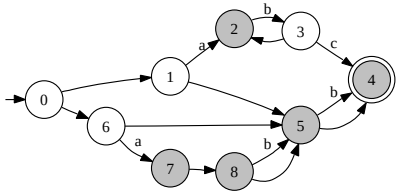


>a b

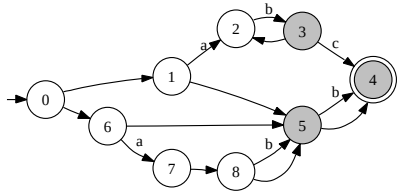


a>b

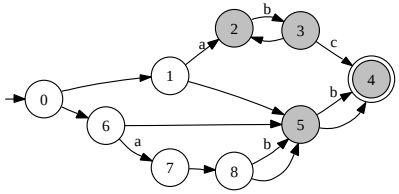
$$\{2,7\} \longrightarrow^* \{2,4,5,7,8\} \xrightarrow{\texttt{b}} \{3,4,5\} \longrightarrow^* \{2,3,4,5\}$$



a>b



a b>



a b>

Implémentation des transitions sur les ensembles

- Transitions étiquetées, relativement facile :

```
static Set<State> doTransition(char c, Set<State> qs) {
  Set<State> r = new TreeSet<State> () ; // Pas de classe Set
  for (State q : qs) {
    Set<State> ds = q.delta[c] ; // On suppose ds != null
    r.addAll(ds) ; // R ← R ∪ D
  }
  return r
}
```

- Un peu plus dur (vu en TP), attention à ne pas boucler.

$$U^0 = Q \qquad U^{n+1} = \{q' \mid \exists q \in U^n, q \longrightarrow q'\}$$

On a $Q \longrightarrow^* \bigcup_{n \leq N} U^n$ pour N assez grand, dès que U_N n'a pas d'élément qui ne soit déjà dans U_0, U_1, $\ldots U_{N-1}$.

Trois sortes d'automates, bilan provisoire

- DFA, efficacité de la reconnaissance (linéaire en n la taille de m).
- NFA, backtracking facile (peut être exponentiel en n, fonctionne souvent en pratique).
- NFA-ϵ, on y vient !

Pour ce qui est de la classe des langages définis :

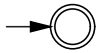
$$\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{NFA}) \subseteq \mathcal{L}(\text{NFA-}\epsilon)$$

Évident, car les DFA sont des cas particuliers de NFA qui sont des cas particuliers de NFA-ϵ.

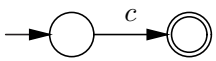
Rapport avec les expressions régulières

Étant donnée une expressions régulière p, on construit facilement un NFA-ϵ qui reconnaît (définit) $[\![p]\!]$.

- Motif ϵ (motif vide).



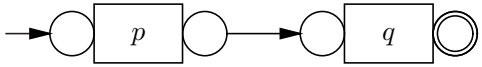
- Motif caractère.



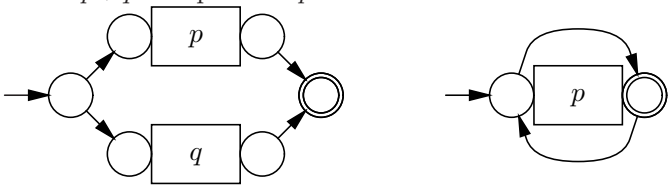
Et inductivement

Soient p et q deux motifs, donc deux automates.

- Séquence pq.



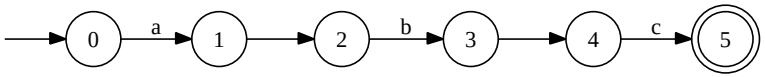
- Alternative $p \mid q$ et répétition $p*$.



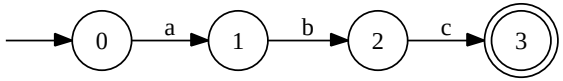
Nous avons *compilé* les motifs (haut-niveau) vers les automates (bas-niveau).

Compilations alternatives

Soit $p =$ `abc`, si on suit la règle, on a :



Alors que l'automate suivant est plus simple et convient.

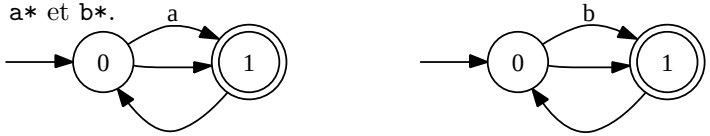


Cela revient à modifier la règle de la concaténation (confondre l'état final de p et l'état initial de q)

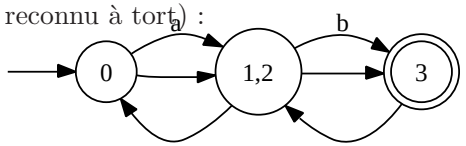
Optimiser mais rester correct

La règle « optimisée » de la concaténation est parfois fausse.

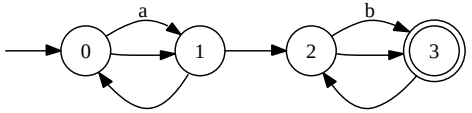
Motifs `a*` et `b*`.



Faux (mot `abab` reconnu à tort) :

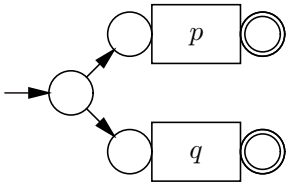


Correct :

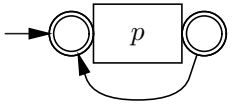


Compilations alternatives

De l'alternative

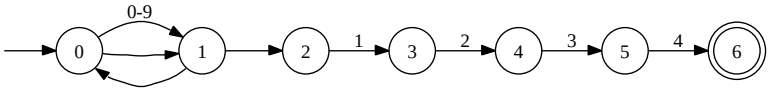


De la répétition.

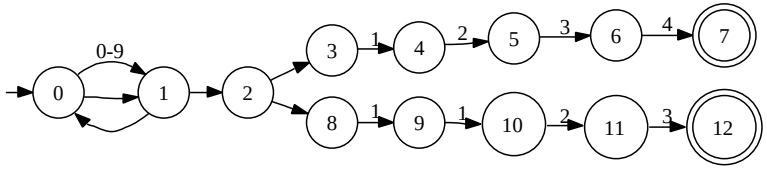


Exemple : le digicode 1234

Le motif est `[0-9]*1234`.



Plus vicieux, le digicode à deux codes.



Tout est dans tout (et réciproquement)

Théorème (Rabin-Scott) Étant donné un NFA-ϵ il existe un DFA qui reconnaît exactement le même langage.

Corollaire : $\mathcal{L}(\text{NFA-}\epsilon) \subseteq \mathcal{L}(\text{DFA})$

Conséquence :

$$\mathcal{L}(\text{DFA}) = \mathcal{L}(\text{NFA}) = \mathcal{L}(\text{NFA-}\epsilon)$$

Théorème (Kleene) Étant donné un DFA, il existe une expression régulière p telle que le langage par lui reconnu est exactement $[\![p]\!]$.

Conséquence :

$$\mathcal{L}(\text{DFA}) = \mathcal{L}(\text{NFA}) = \mathcal{L}(\text{NFA-}\epsilon) = \text{langages réguliers}$$

Principe de la déterminisation

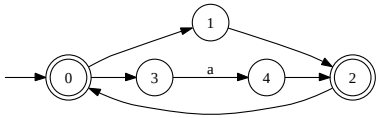
Soit $(\Sigma, Q, \delta, q_0, F, \epsilon)$ un NFA-ϵ. Le DFA équivalent est :

- Ensemble d'états : parties de Q fermées par $\longrightarrow^*$ ($\epsilon^*(U) = U$)
- État initial $\epsilon^*(\{q_0\})$.
- États finaux $\{U \mid U \cap F \neq \emptyset\}$.
- Transitions : $U \xrightarrow{c} \epsilon^*(c(U))$.

L'algoritme de déterminisation est assez malin, mais il peut être exponentiel (le DFA peut avoir $2^{|Q|}$ états).

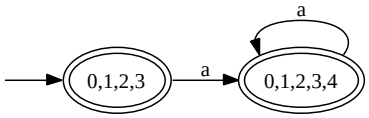
L'algorithme est important en pratique (implémentation efficace des expressions régulières).

Exemple facile de déterminisation

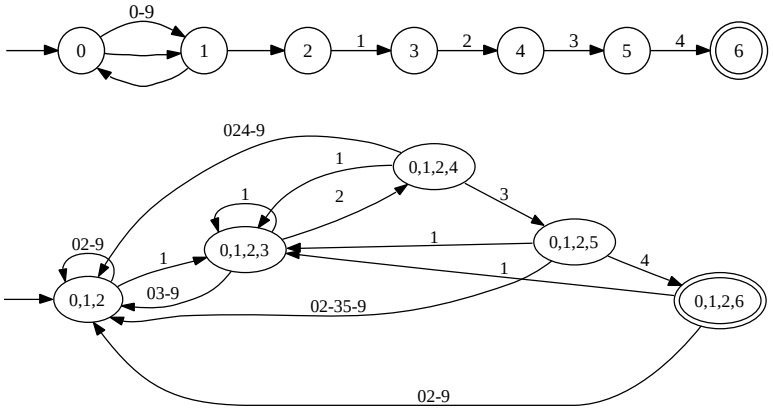


- Initialement : $Q_0 = \epsilon^*(\{\,0\,\}) = \{\,0, 1, 2, 3\,\}$.
- Autre état $Q_1 = \epsilon^*(\mathtt{a}(Q_0)) = \{\,0, 1, 2, 3, 4\,\}$.
- Et par ailleurs $Q_1 = \epsilon^*(\mathtt{a}(Q_1))$.

Soit :



Plus raide, le digicode



L'algorithme de Kleene

- Est intéressant (il prouve que les langages définis par les automates sont les langages réguliers), mais sans intérêt en pratique.
- Il produit des expressions régulières, assez affreuses.
- Alors, faute de temps... (abordé en INF-421-a[a])

[a] `http://www.enseignement.polytechnique.fr/informatique/INF421/c9-2x2.pdf`

Le mot de la fin

- Les points important du cours (⇒ INF-431)
  - ▷ Programmer plus et mieux, utiliser la librairie, (⇒ Héritage, Projet).
  - ▷ Les structures de données dynamiques et surtout les arbres (⇒ Graphes).
  - ▷ Les automates (⇒ Graphes, Grammaires, Analyse syntaxique, Calculabilité).
  - ▷ ⇒ Réseaux.
- Contrôle
  - ▷ Documents (poly, copie des transparents) autorisés et même recommandés.
  - ▷ Réviser tout ! Penser à revoir aussi les TP.