

IN 101 - Cours 06

14 octobre 2011



présenté par

Matthieu Finiasz

✘ Des logs de serveur web ressemblent à cela :

```
127.0.0.1      - - [13/Oct/2010:09:40:02 +0000] "GET /server-status?auto HTTP/1.1" 200 439 "-" "libwww-perl/5.836"
182.46.10.154 - - [13/Oct/2010:09:40:07 +0000] "POST /m HTTP/1.1" 200 20 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 6.0; fr; rv:1.9.1.13) Gecko/20100914 Firefox/3.5.13 (.NET CLR 3.5.30729; .NET4.0C)"
182.46.10.154 - - [13/Oct/2010:09:40:08 +0000] "POST /m HTTP/1.1" 200 854 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 6.0; fr; rv:1.9.1.13) Gecko/20100914 Firefox/3.5.13 (.NET CLR 3.5.30729; .NET4.0C)"
182.53.208.200 - - [13/Oct/2010:09:40:27 +0000] "POST /m HTTP/1.1" 200 1112 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 5.1; fr; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10"
182.46.10.154 -- [13/Oct/2010:09:41:07 +0000] "GET /gt?tb=676&a=head HTTP/1.1" 200 9462 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 6.0; fr; rv:1.9.1.13) Gecko/20100914 Firefox/3.5.13 (.NET CLR 3.5.30729; .NET4.0C)"
184.7.230.122 - - [13/Oct/2010:10:42:24 +0000] "GET /thumb?i=13622448 HTTP/1.1" 200 6805 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 5.1; fr; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10 GTB7.1 (.NET CLR 3.5.30729)"
184.7.230.122 - - [13/Oct/2010:10:42:24 +0000] "GET /thumb?i=13622349 HTTP/1.1" 200 7094 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 5.1; fr; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10 GTB7.1 (.NET CLR 3.5.30729)"
184.7.230.122 - - [13/Oct/2010:10:48:35 +0000] "GET /m?_=1286966914300&mt=feed&f=a_refresh&tm=1286966735&mid=5228&mid=5202&mid=5246&mi
184.7.230.122 - - [13/Oct/2010:10:48:35 +0000] "POST /m HTTP/1.1" 200 695 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 5.1; fr; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10 GTB7.1 (.NET CLR 3.5.30729)"
182.53.208.200 - - [13/Oct/2010:11:25:05 +0000] "POST /m HTTP/1.1" 200 1038 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 5.1; fr; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10"
184.7.230.122 - - [13/Oct/2010:11:25:35 +0000] "POST /m HTTP/1.1" 200 689 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 5.1; fr; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10 GTB7.1 (.NET CLR 3.5.30729)"
191.21.9.195  - - [13/Oct/2010:11:42:22 +0000] "GET /fav?i=4102 HTTP/1.1" 304 - "http://site.com/" "Mozilla/5.0 (X11; U;
      Linux i686; en-US; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10"
191.21.9.195  - - [13/Oct/2010:11:42:22 +0000] "GET /thumb?i=13612162 HTTP/1.1" 304 - "http://site.com/" "Mozilla/5.0 (X11; U;
      Linux i686; en-US; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10"
182.24.187.141 - - [13/Oct/2010:12:13:32 +0000] "GET /thumb?i=13622016 HTTP/1.1" 200 2645 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 6.1; fr; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3"
182.24.187.141 - - [13/Oct/2010:12:13:32 +0000] "GET /thumb?i=13622463 HTTP/1.1" 200 5960 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 6.1; fr; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3"
182.24.187.141 - - [13/Oct/2010:13:02:25 +0000] "POST /m HTTP/1.1" 200 1262 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 6.1; fr; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3"
182.24.187.141 - - [13/Oct/2010:13:02:28 +0000] "POST /m HTTP/1.1" 200 20 "http://site.com/" "Mozilla/5.0 (Windows; U;
      Windows NT 6.1; fr; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3"
```

✘ Des logs de serveur web ressemblent à cela :

```
127.0.0.1      - - [13/Oct/2010:09:40:02 +0000] "GET /server-status?auto HTTP/1.1" 200 439 "-" "libwww-perl/5.836"
182.46.10.154 - - [13/Oct/2010:09:40:07 +0000] "POST /m HTTP/1.1" 200 20 "http://site.com/" "Mozilla/5.0 (Windows; U;
Windows NT 6.0; fr; rv:1.9.1.13) Gecko/20100914 Firefox/3.5.13 (.NET CLR 3.5.30729; .NET4.0C)"
182.46.10.154 - - [13/Oct/2010:09:40:08 +0000] "POST /m HTTP/1.1" 200 854 "http://site.com/" "Mozilla/5.0 (Windows; U;
Windows NT 6.0; fr; rv:1.9.1.13) Gecko/20100914 Firefox/3.5.13 (.NET CLR 3.5.30729; .NET4.0C)"
182.53.208.200 - - [13/Oct/2010:09:40:27 +0000] "POST /m HTTP/1.1" 200 1112 "http://site.com/" "Mozilla/5.0 (Windows; U;
Windows NT 5.1; fr; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10"
182.46.10.154 -- [13/Oct/2010:09:41:07 +0000] "GET /gt?tb=676&a=head HTTP/1.1" 200 9462 "http://site.com/" "Mozilla/5.0 (Windows; U;
Windows NT 6.0; fr; rv:1.9.1.13) Gecko/20100914 Firefox/3.5.13 (.NET CLR 3.5.30729; .NET4.0C)"
184.7.230.122 - - [13/Oct/2010:10:42:24 +0000] "GET /thumb?i=13622448 HTTP/1.1" 200 6805 "http://site.com/" "Mozilla/5.0 (Windows; U;
Windows NT 5.1; fr; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10 GTB7.1 (.NET CLR 3.5.30729)"
184.7.230.122 - - [13/Oct/2010:10:42:24 +0000] "GET /thumb?i=13622349 HTTP/1.1" 200 7094 "http://site.com/" "Mozilla/5.0 (Windows; U;
Windows NT 5.1; fr; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10 GTB7.1 (.NET CLR 3.5.30729)"
```

✘ On veut pouvoir facilement et **efficacement** trouver :

- ✘ les lignes contenant une adresse IP,
 - ✘ les adresses IP ayant accédé à une page donnée,
 - ✘ la proportion de requêtes utilisant Linux...
- dans un fichier de plusieurs millions de lignes.

✘ La commande `grep` permet de faire cela, mais comment ?

Recherche de motifs

IN101 - 2011-2012

- ✘ On veut savoir si un certain motif M apparaît dans un texte T
 - ✘ souvent, trouver toutes les instances du motif et leurs positions.
- ✘ Motif : $M = \text{"our"}$.
- ✘ Texte : $T =$

Partir un jour sans retour,
Effacer notre amour,
Sans se retourner ne pas regretter
Garder les instants qu'on a volés.
Partir un jour sans bagages,
Oublier ton image,
Sans se retourner ne pas regretter
Penser à demain, recommencer.

- × On veut savoir si un certain motif M apparaît dans un texte T
 - × souvent, trouver toutes les instances du motif et leurs positions.
- × Motif : $M = \text{"our"}$.
- × Texte : $T =$

Partir un jour sans retour,
Effacer notre amour,
Sans se retourner ne pas regretter
Garder les instants qu'on a volés.
Partir un jour sans bagages,
Oublier ton image,
Sans se retourner ne pas regretter
Penser à demain, recommencer.

- ✘ Plus formellement, on a un alphabet (bits, char...) et deux tableaux T et M de taille n et m d'éléments de cet alphabet
 - ✘ on cherche tous les **décalages** $s \in [0, n - m]$ tels que :

$$\forall j \in [0, m - 1] \quad M[j] = T[s + j]$$

- ✘ Applications :
 - ✘ traitement de texte,
 - ✘ la commande grep,
 - ✘ recherche de séquences de gènes...

- ✘ Pour chaque s , on teste l'égalité avec le motif.

```
1 for (int s=0; s<n-m+1; s++) {
2   for (int j=0; j<m; j++) {
3     if (M[j] != T[s+j]) {
4       break;
5     }
6   }
7   if (j == m) {
8     printf("%d\n", s);
9   }
10 }
```

- ✘ Complexité dans le pire cas $\Theta(nm)$.
 - ✘ En moyenne \rightarrow dépend de la taille de l'alphabet, mais $\Theta(n)$,
 - ⚠ pour ce problème on est très rarement dans un cas moyen !

- × Principe : on aime mieux manipuler des nombres,
 - × alphabet de taille $d \rightarrow$ codage du motif/texte en base d .
- × On doit alors comparer les entiers :

$$p = M[0] + M[1] \times d + \dots + M[m-1] \times d^{m-1},$$

et pour un décalage de s :

$$t_s = T[s] + T[s+1] \times d + \dots + T[s+m-1] \times d^{m-1}.$$

- × Le point important est que t_{s+1} se déduit facilement de t_s :

$$t_{s+1} = \frac{t_s - T[s]}{d} + d^{m-1}T[s+m] = \left\lfloor \frac{t_s}{d} \right\rfloor + d^{m-1}T[s+m].$$

- ✘ Recherche de séquence d'ADN :
 - ✘ l'alphabet est $\{A, T, C, G\}$ et $d = 4$.
 - ✘ on choisit : $A = 0$, $T = 1$, $C = 2$ et $G = 3$.
- ✘ On prend $T = ATACGGACT$ et $M = GGA$
 - ✘ on trouve alors :

$$p = 3 + 3 \times d + 0 \times d^2 = 15$$

$$t_0 = 0 + 1 \times d + 0 \times d^2 = 4$$

$$t_1 = 1 + 0 \times d + 2 \times d^2 = 33 = \left\lfloor \frac{t_0}{d} \right\rfloor + 2 \times d^2$$

$$t_2 = 0 + 2 \times d + 3 \times d^2 = 56 = \left\lfloor \frac{t_1}{d} \right\rfloor + 3 \times d^2$$

$$t_3 = 2 + 3 \times d + 3 \times d^2 = 62 = \left\lfloor \frac{t_2}{d} \right\rfloor + 3 \times d^2$$

$$t_4 = 3 + 3 \times d + 0 \times d^2 = 15 = \left\lfloor \frac{t_3}{d} \right\rfloor + 0 \times d^2$$

$$t_5 = 3 + 0 \times d + 2 \times d^2 = 35 = \left\lfloor \frac{t_4}{d} \right\rfloor + 2 \times d^2$$

$$t_6 = 0 + 2 \times d + 1 \times d^2 = 24 = \left\lfloor \frac{t_5}{d} \right\rfloor + 1 \times d^2$$

```
1 void Rabin_Karp (int* T, int n, int* M, int m, int d) {
2   int i,h,p,t;
3   /* on calcule d^(m-1) une fois pour toute */
4   h = pow(d,m-1);
5   /* on calcule p */
6   p=0;
7   for (i=m-1; i>=0; i--) {
8     p = M[i] + d*p;
9   }
10  /* on calcule t_0 */
11  t=0;
12  for (i=m-1; i>=0; i--) {
13    t = T[i] + d*t;
14  }
15  /* on teste tous les décalages */
16  for (i=0; i<n-m; i++) {
17    if (t == p) {
18      printf("Motif trouvé à la position %d\n", i);
19    }
20    t = (t/d) + h*T[i+m];
21  }
22  if (t == p) {
23    printf("Motif trouvé à la position %d\n", n-m);
24  }
25 }
```

- ✘ Complexités des différentes étapes :
 - ✘ calcul de h en $\Theta(\log m)$ (ou simplement $\Theta(m)$),
 - ✘ calcul de p et t_0 en $\Theta(m)$,
 - ✘ $n - m$ calculs des t_s en $\Theta(1)$ donc $\Theta(n - m)$ au total.

- ✘ Complexité globale dans le pire cas optimale : $\Theta(n)$.

 on considère que tous les calculs se font en $\Theta(1)$.

- ✘ Plus d^m est grand, plus les calculs sont longs :
 - ✘ si $d^m < 2^{32}$ les calculs sont en $\Theta(1)$ (sur un CPU 32 bits),
 - ✘ sinon la complexité est $\Theta(\log(d^m)) = \Theta(m \log d)$
 - $\log d$ est constant et on retrouve la complexité naïve : $\Theta(nm)$.

- ✘ Pour améliorer cela on fait les calculs modulo un entier adapté comme 65521 :
 - ✘ nombre premier,
 - ✘ plus petit que 2^{16} → produits possibles en temps $\Theta(1)$.

- ✘ Si $t_s = p \pmod{65521}$ on fait la comparaison naïve pour vérifier.

Algorithme de Rabin-Karp avec modulo

✘ Code similaire avec des modulo en plus.

(cf. poly p.111-112)

✘ Complexité :

✘ dans le pire cas : $\Theta(nm)$

→ tous les modulo sont égaux...

✘ en moyenne : $\Theta(nm)$ aussi !

→ avec une constante intéressante : $n + m \times (\mathcal{N}_{sol} + \frac{n}{65521})$.

✘ En pratique, si $m \ll 65521$ cet algorithme est très efficace :

✘ contrairement à l'algorithme naïf le pire cas n'arrive pas...

✘ le modulo agit sur le motif complet → moins sujet à des biais.

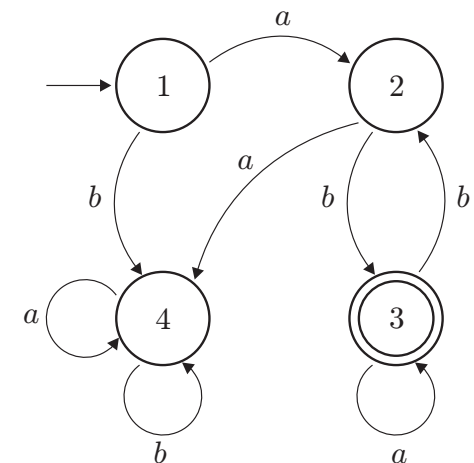
Automates finis

IN101 - 2011-2012

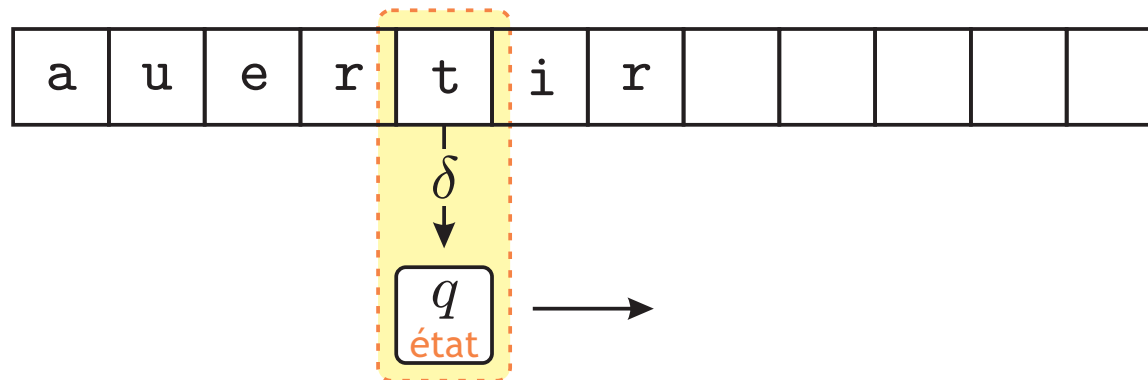
- ✘ Un **automate fini** est une « machine » pouvant se trouver dans un ensemble fini de configurations internes appelées **états**.
 - ✘ Son but : reconnaître mécaniquement des « mots ».

- ✘ Un automate fini déterministe est la donné de :

- Σ un alphabet fini, $\{a, b\}$
- Q un ensemble fini d'états, $\{1, 2, 3, 4\}$
- $q_0 \in Q$ un état initial, 1
- $F \subset Q$ un ensemble d'états finaux, $\{3\}$
- $\delta : Q \times \Sigma \rightarrow Q$ une **fonction de transition**.

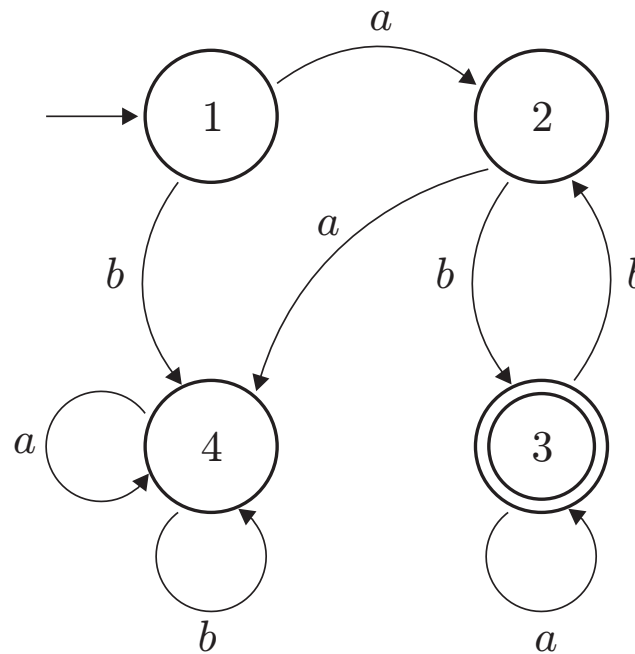


- ✘ **Principe** : l'automate démarre en q_0 , lit un mot de Σ^* et suit la fonction de transition. On regarde si l'état d'arrivée est dans F .
- ✘ Dans l'état $q \in Q$, en lisant $t \in \Sigma$, l'automate passe dans l'état $\delta(q, t)$ et avance d'une case.

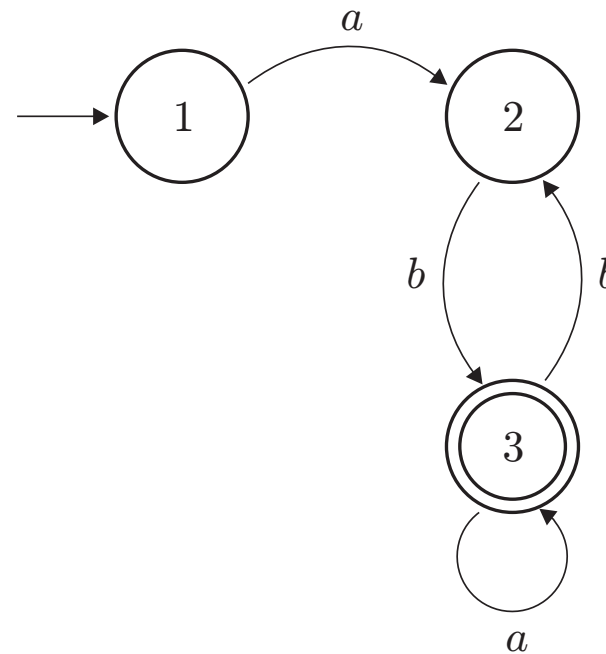


- ✘ Le langage \mathcal{L} reconnu par un automate est l'ensemble des mots qui l'amène dans un état final :

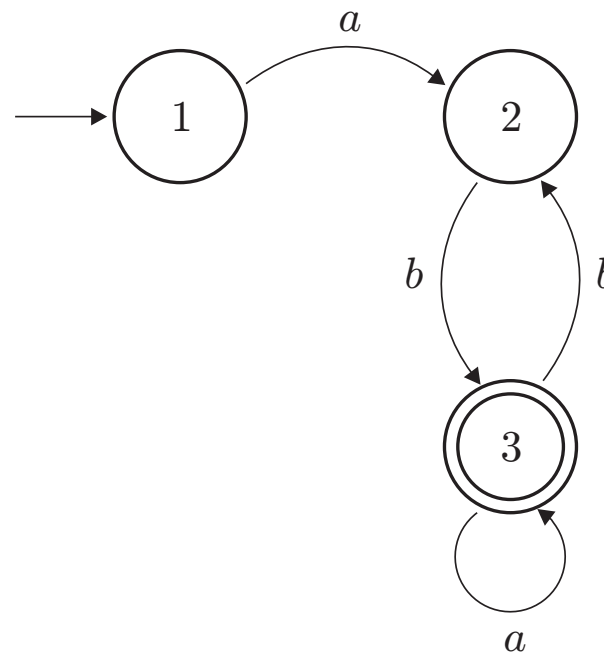
$$\mathcal{L} = \{W \in \Sigma^*, q_0 \xrightarrow{W} f, f \in F\}$$



- ✘ Reconnaît par exemple les mots ab , aba , $abbb$ et $ababbaa$.
- ✘ L'état 4 est un **état rebut** :
 - ✘ ce n'est pas un état final,
 - ✘ on n'en ressort jamais.

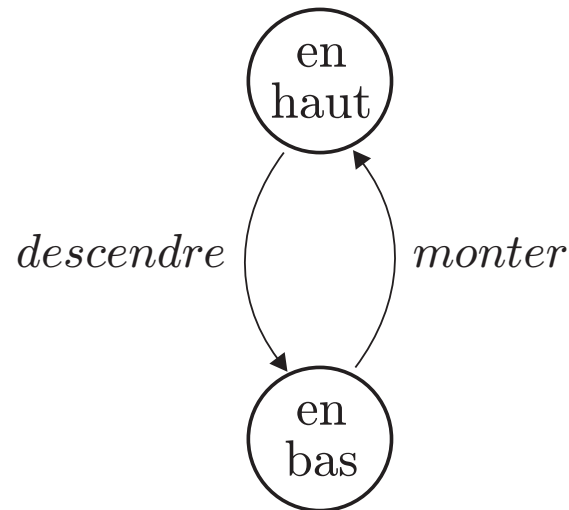


- ✘ Reconnaît par exemple les mots *ab*, *aba*, *abbb* et *ababbaa*.
- ✘ L'état 4 est un **état rebut** :
 - ✘ ce n'est pas un état final,
 - ✘ on n'en ressort jamais.
- ✘ L'automate peut se simplifier.

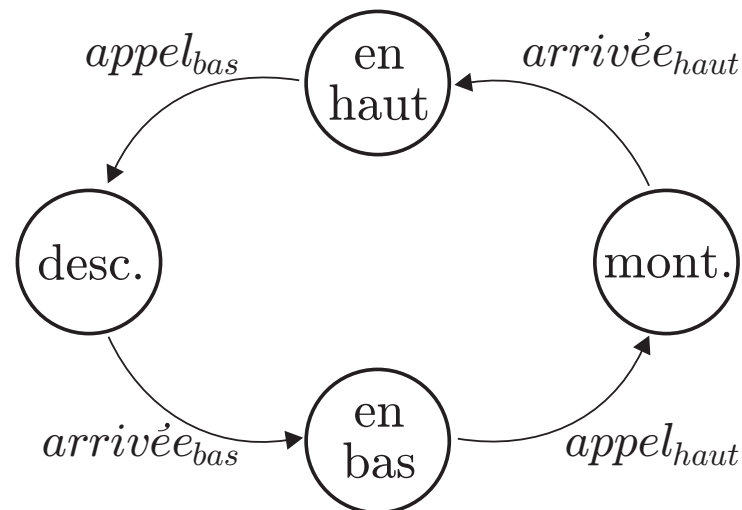


- ✘ Cet automate reconnaît le langage $\mathcal{L} = ab(a|bb)^* = a(ba^*b)^*ba^*$:
 - ✘ ab signifie un a suivi d'un b ,
 - ✘ $(a|bb)$ signifie a ou bb ,
 - ✘ a^* signifie un nombre quelconque de a , y compris 0,
 - ✘ a^+ signifie un nombre strictement positif de a .

- ✘ La modélisation la plus simple d'un ascenseur à 2 étages contient :
 - ✘ 2 états : en haut ou en bas,
 - ✘ 2 transitions possibles : monter ou descendre.
- ✘ On obtient un automate très simple :



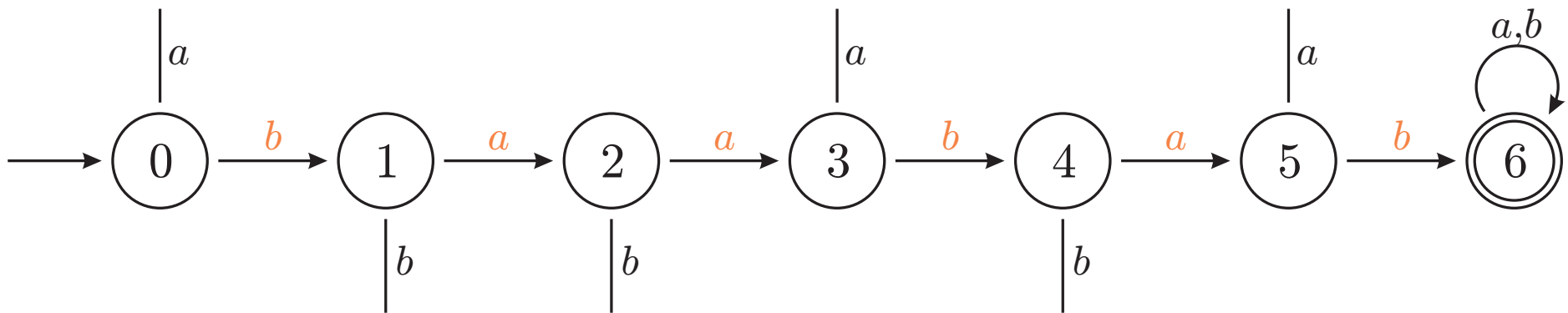
- ✘ Pour quelque chose de plus réaliste on complique un peu :
 - ✘ 4 états : en haut, ou en bas, montant ou descendant,
 - ✘ 4 transitions possibles : appel en haut, appel en bas, arrivée en haut ou arrivée en bas.
- ✘ On obtient un automate qui correspond aux vieux ascenseurs sans mémoire :



- ✘ Pour tout motif $M \in \Sigma^*$ de longueur m , on peut construire un automate A qui reconnaît $\mathcal{L} = \Sigma^* M \Sigma^*$.

- ✘ Construction de l'automate :
 - l'alphabet de A est Σ ,
 - A possède $m + 1$ états numérotés $\{0, 1, \dots, m\}$,
 - A arrive dans l'état i en lisant les i premières lettres de M ,
 - 0 est l'état initial,
 - m est l'unique état final (dont on ne ressort jamais),
 - reste à construire la fonction de transition δ ...

- ✘ On part d'un squelette qui suffit à reconnaître le mot *baabab*
 - ✘ reste à définir le reste de δ .



- ✘ Soit M_i le préfixe de longueur i du motif M :
 - ✘ on est dans l'état i quand les i derniers caractères lus sont M_i ,
 - ✘ quand on lit le caractère a :
 - si $M_i a = M_{i+1}$ on va dans l'état $i + 1$,
 - sinon, on retourne dans l'état λ tel que M_λ est « le plus long suffixe de $M_i a$ ».

$$\lambda = \max \{j \in [0, i], \exists w \in \Sigma^+, wM_j = M_i a\}.$$

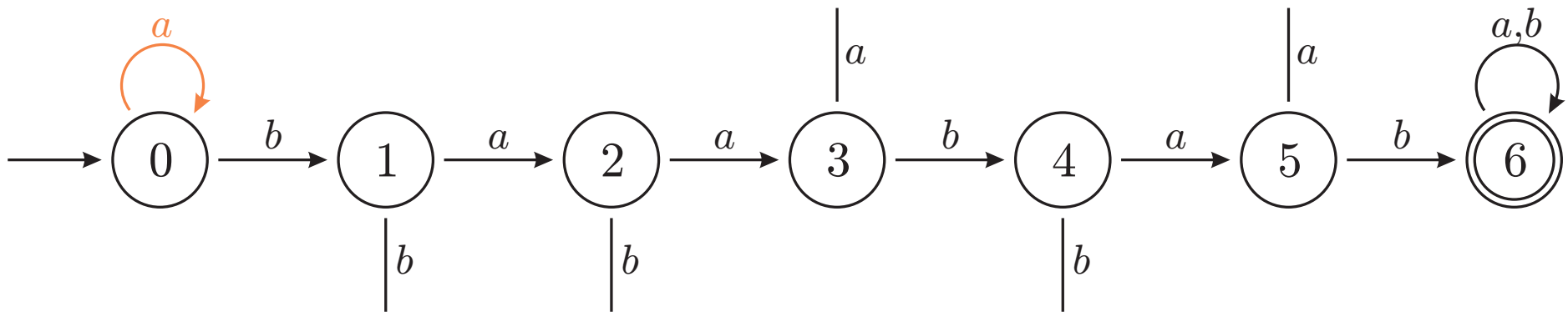
✘ On a alors $\delta(i, a) = \lambda$.

✘ Par exemple : $M_5 a = baabaa$

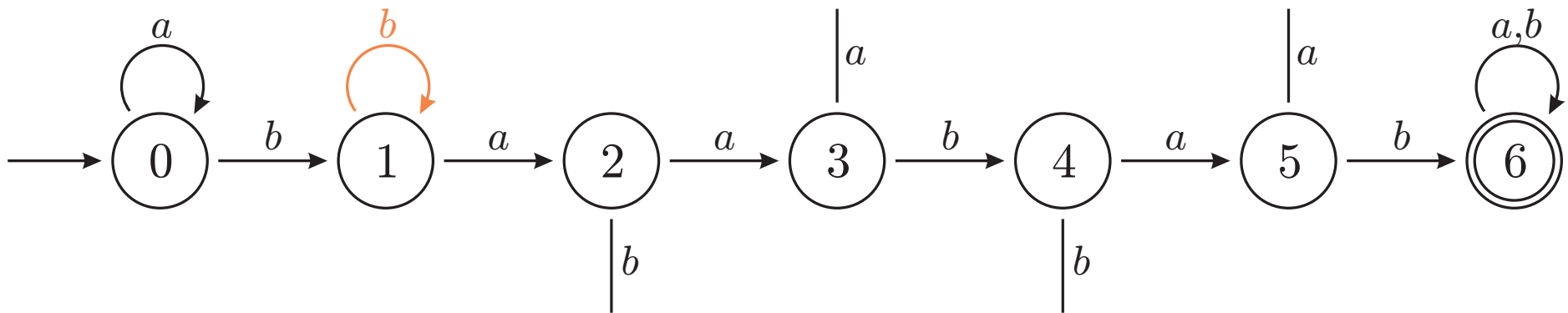
$$baabab \rightarrow \delta(5, a) = 3.$$

- ✘ Pour calculer ce λ on utilise la partie de l'automate déjà construite :
 - ✘ en lisant les i derniers caractères de $M_i a$ il se retrouve en λ .

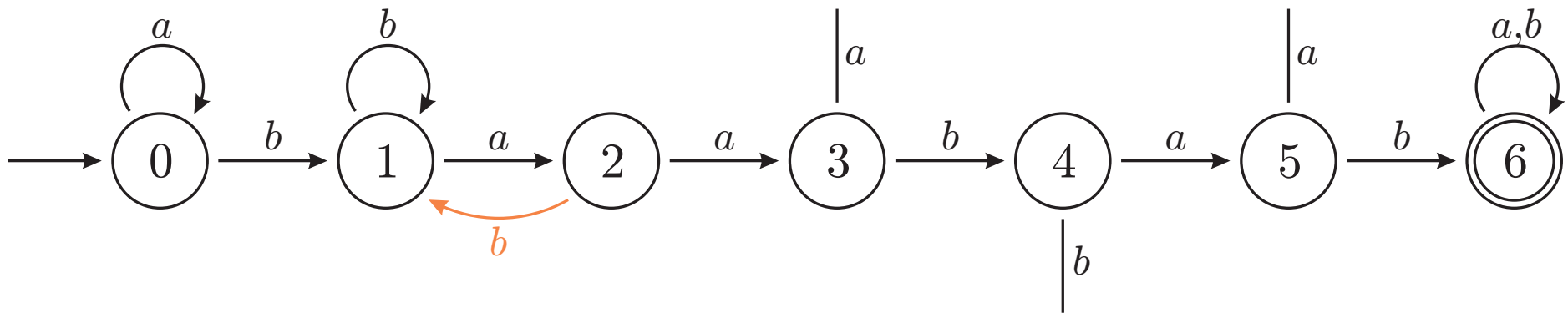
- ✘ Pour l'état 0, tout ce qui n'amène pas en 1 fait rester en 0.



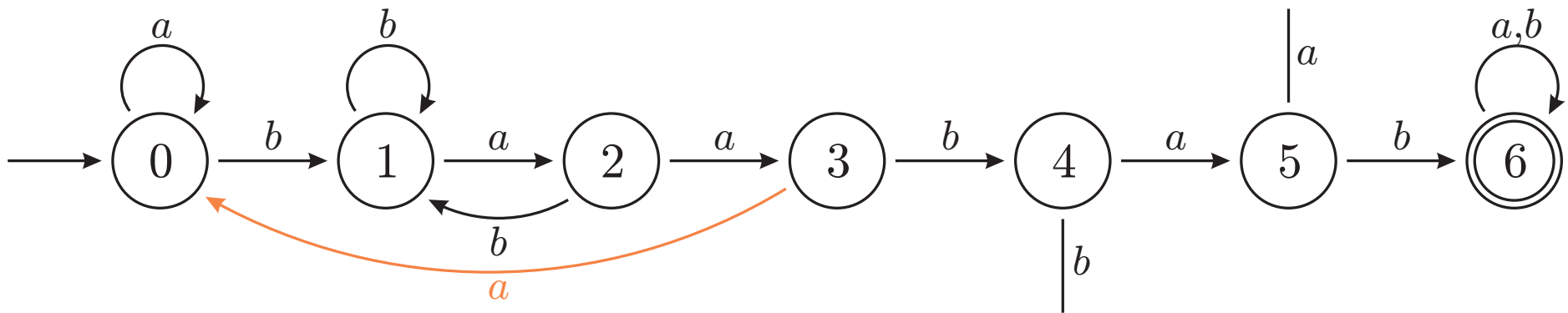
- ✘ Pour l'état 1, si on lit b on doit regarder comment se comporte le dernier caractère de $M_1 b = bb$.



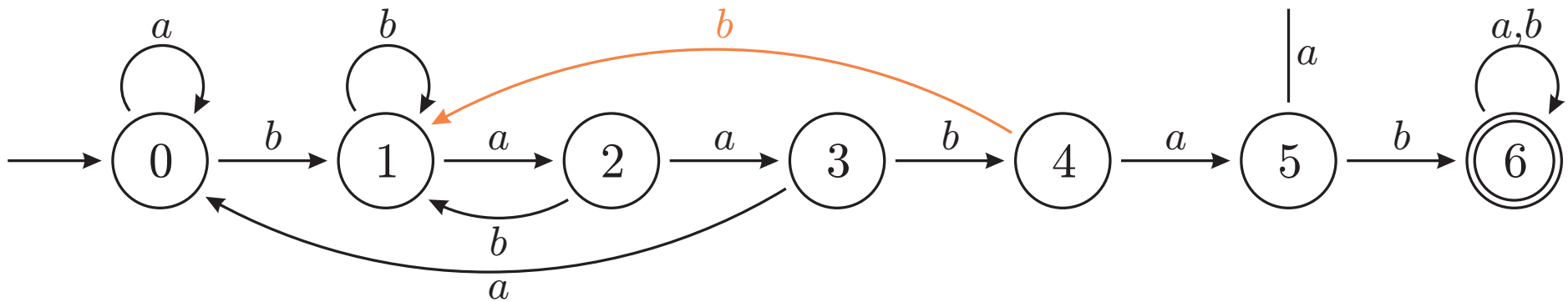
- ✘ Pour l'état 2, si on lit b on doit regarder comment se comportent les 2 derniers caractères de $M_2b = bab$.



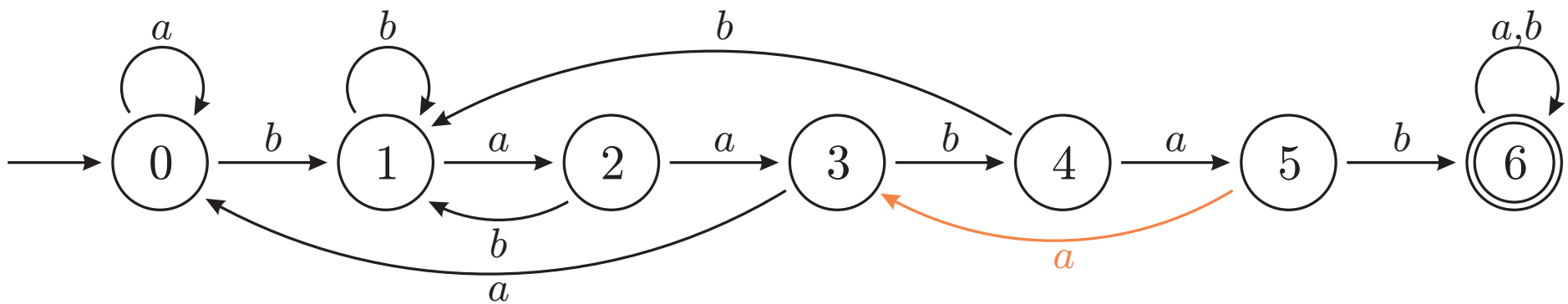
- ✘ Pour l'état 3, si on lit a on doit regarder comment se comportent les 3 derniers caractères de $M_3a = baaa$.



- ✘ Pour l'état 4, si on lit b on doit regarder comment se comportent les 4 derniers caractères de $M_4b = baabb$.



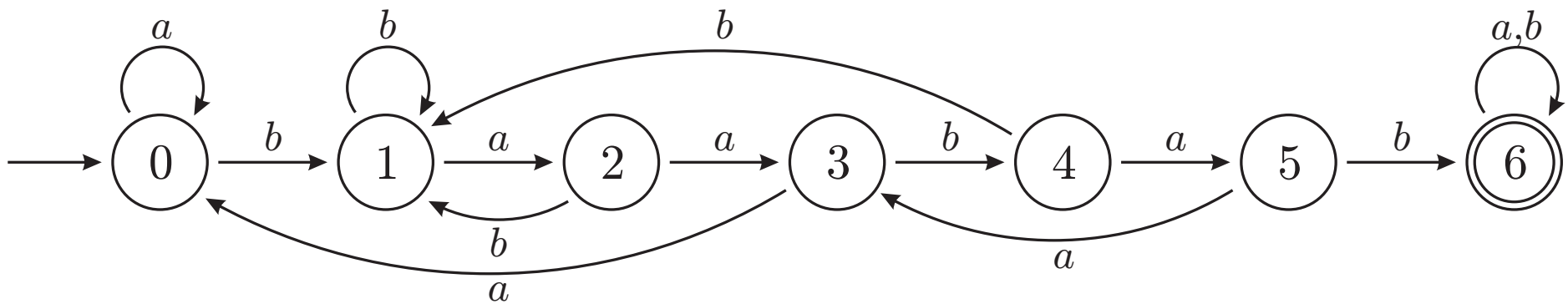
- ✘ Pour l'état 5, si on lit a on doit regarder comment se comportent les 5 derniers caractères de $M_5a = baabaa$.



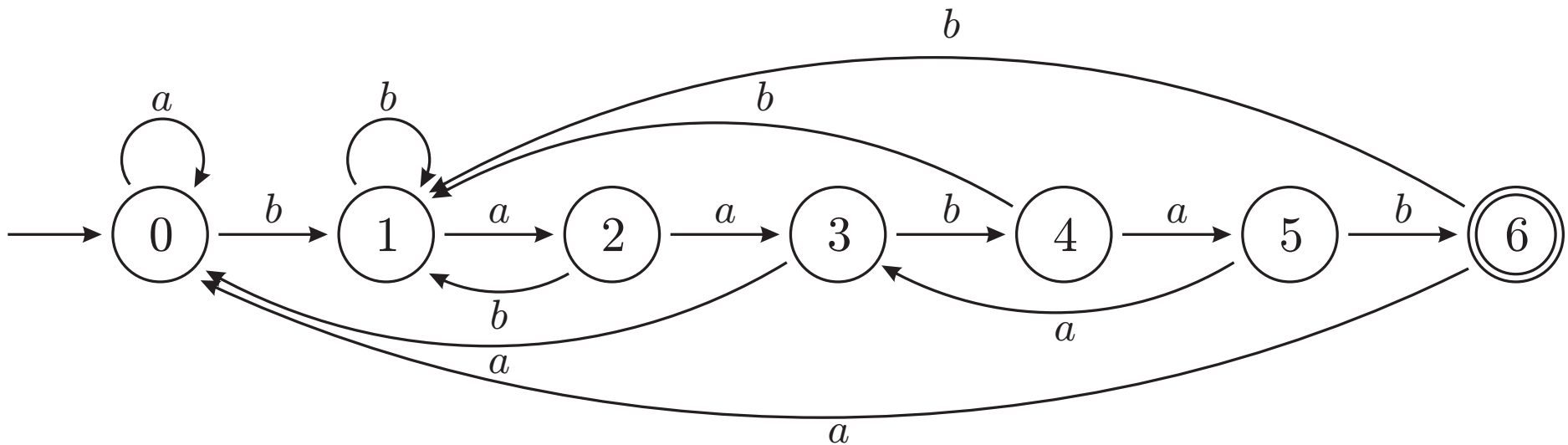
- ✘ Un automate peut se représenter par un état courant q et la fonction de transition δ
 - ✘ l'état courant est un *int*,
 - ✘ la fonction de transition est un tableau de taille $m + 1 \times |\Sigma|$:
`int d[m+1][TAILLE_ALPHABET];`
`d[q][a]` vaut $\delta(q, a)$.
- ✘ Bien optimisée, la complexité de la construction de l'automate (le remplissage de la table `d`) est $\Theta(m|\Sigma|)$.
- ✘ La complexité en mémoire est aussi $\Theta(m|\Sigma|)$.

(cf. poly p.115)

- ✘ Cet automate permet de savoir si le motif *baabab* est présent.



- ✘ Cet automate permet de savoir si le motif *baabab* est présent.
- ✘ On le modifie un peu :
 - dès qu'il **pass**e dans l'état final le motif a été trouvé.



```
1 void pattern_search (in m, in alph_size, in n, int* T, int** d) {
2   int q = 0;
3   for (int s=0; s<n; s++) {
4     q = d[q][T[s]];
5     if (q == m) {
6       printf("Motif trouvé à la position %d\n", s-m+1);
7     }
8   }
9 }
```

- ✘ La recherche est aussi simple que pour l'algorithme naïf.
 - ✘ la complexité est optimale $\Theta(n)$.
- ✘ Il faut quand même ajouter à cela le coût de construction :
 - ✘ temps en $\Theta(n + m|\Sigma|)$, espace en $\Theta(m|\Sigma|)$.

- ✘ Sont utilisés pour programmer tous les appareils ayant un nombre fini d'états possibles :
 - ✘ ascenseur, machine à café...
- ✘ Utilisables pour tout ce qui a un état interne :
 - ✘ retenue d'un additionneur...
- ✘ Pour la recherche de motifs/reconnaissance d'un langage :
 - ✘ analyseur lexical, recherche de séquence ADN...

- ✘ Leur puissance de calcul est limitée :
 - ✘ ne peuvent pas reconnaître $\{a^n b^n, n \geq 0\}$,
 - ✘ pas d'analyse syntaxique (parenthèse bien formées...).
 - les **machines de Turing** permettent de faire tout cela

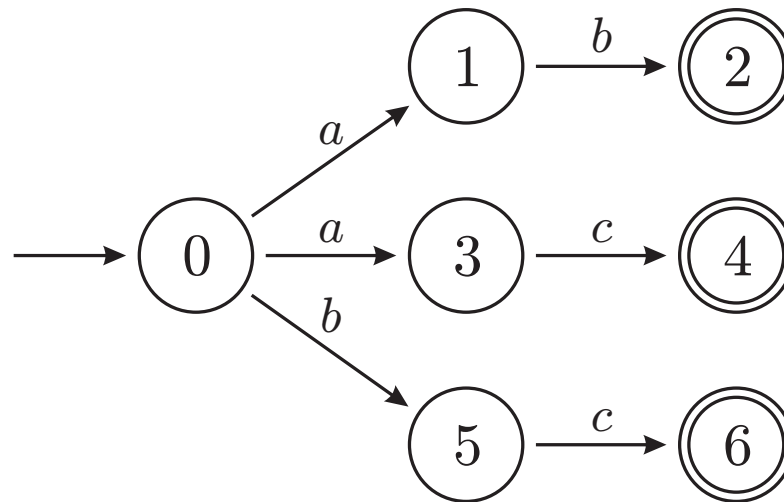
Automates non-déterministes

- ✘ Un automate **non-déterministe** peut avoir plusieurs transitions pour un même symbole et un même état :
 - ✘ plusieurs états d'arrivée possible pour un même mot lu,
 - ✘ un mot est reconnu si **l'un des états possibles** est final.

- ✘ Les deux familles permettent de reconnaître les mêmes langages
 - ✘ souvent, l'automate non-déterministe est plus facile à écrire.

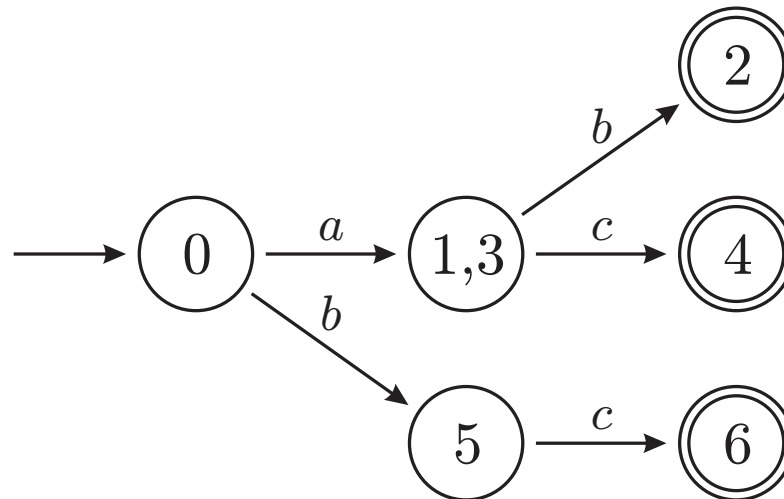
- ✘ Un automate non-déterministe peut toujours être remplacé par un automate déterministe équivalent
 - ✘ un algorithme permet de faire facilement la conversion,
 - ✘ pour un automate non-déterministe à n états, le déterministe peut avoir jusqu'à 2^n états.

- ✘ Voici un automate qui reconnaît $(ab|ac|bc)$:



- ✘ Au lieu d'un seul état, on a un ensemble d'états courants $\subset Q$:
 - ✘ en lisant aa on passe par les états $\{0\}, \{1, 3\}, \emptyset$
 - on arrive sur l'ensemble vide, le motif n'est pas reconnu,
 - ✘ en lisant ac on passe par les états $\{0\}, \{1, 3\}, \{4\}$
 - 4 est un état final, le motif est reconnu.

- ✘ On procède comme pour le parcours non-déterministe :
 - ✘ chaque état du nouvel automate est un sous-ensemble de Q ,
 - ✘ l'état initial reste le même,
 - ✘ tous les états contenant un état final sont finaux.
- ✘ Pour l'automate précédent on obtient :

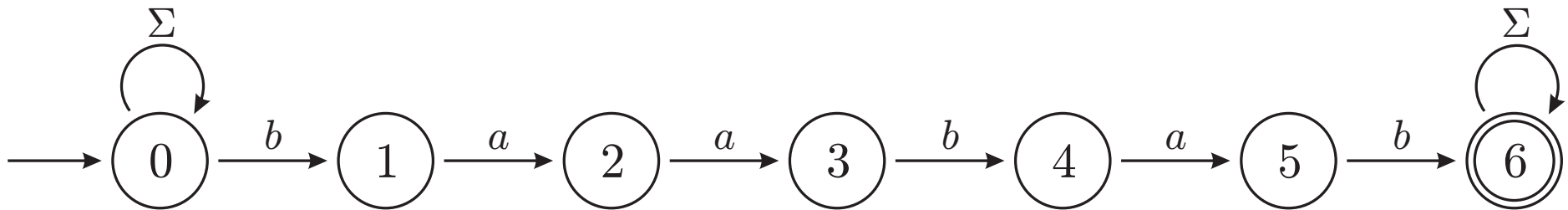


- ✘ On peut écrire de façon plus systématique un automate pour un langage donné.

Automate non-déterministe pour la recherche de motif

Tout devient simple...

- ✘ L'écriture de l'automate pour le motif *baabab* est instantanée :

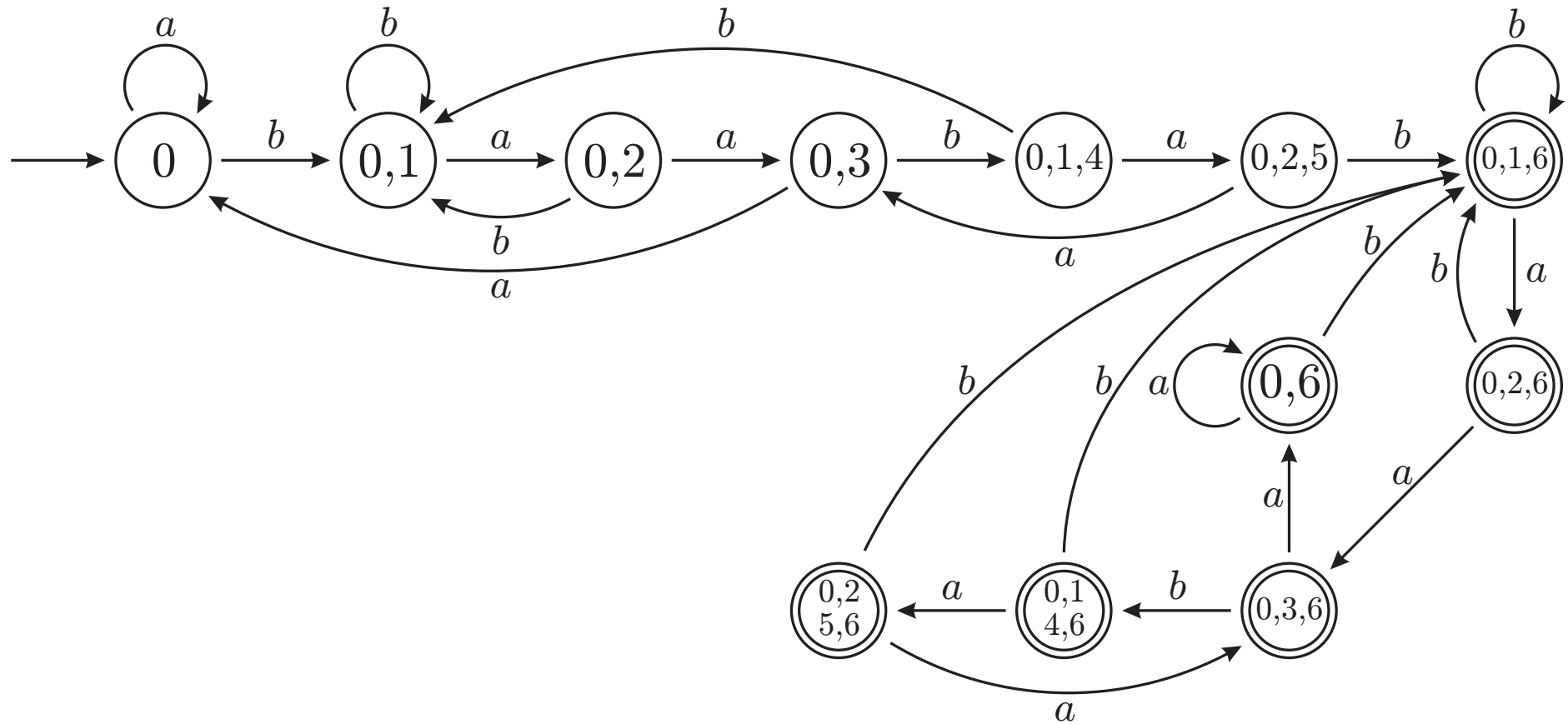


- ✘ En revanche, parcourir un tel automate est plus cher :
 - ✘ la table de transition ne contient pas juste un état, mais un ensemble d'états (jusqu'à $|Q| = m + 1$),
 - ✘ on ne part pas d'un seul état, mais de plusieurs,
 - chaque étape du parcours coûte $\Theta(m^2)$ au lieu de $\Theta(1)$.
- ✘ On préfère déterminer cet automate.

Automate non-déterministe pour la recherche de motif

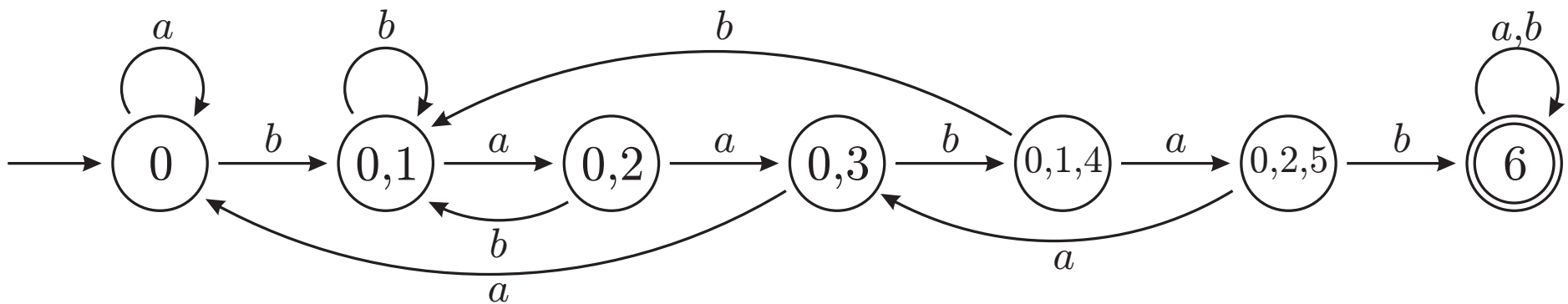
...ou presque

✘ L'automate que l'on obtient n'est pas aussi simple que l'on voudrait :



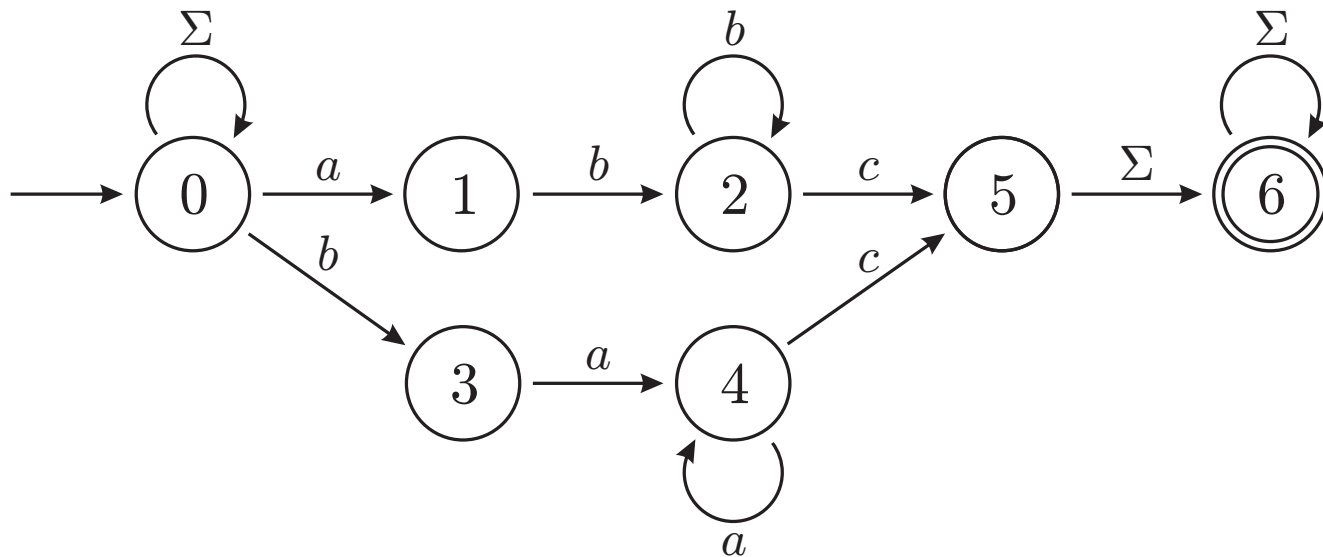
✘ Il faut donc le simplifier...

- ✘ Quelques règles simples existent :
 - ✘ deux états (de même nature) dont toutes les transitions sont identiques peuvent être fusionnés,
 - ✘ un état duquel aucune suite de symboles ne permet d'atteindre un état final peut être supprimé,
 - ✘ un état final duquel toute suite de symbole ramène sur un état final peut être bouclé sur lui-même.



Construction automatique d'automates non-déterministes

- ✘ Pour reconnaître le langage $\Sigma^*(ab^+|ba^+)c\Sigma^+$, il suffit de “coller” des petits automates.



- ✘ Il ne reste plus qu'à le déterminer...

- ✘ La recherche de motif est un problème fréquent en informatique :
 - ✘ un bon algorithme doit être **linéaire** en la taille du texte,
 - ✘ le coût de la recherche ne doit pas dépendre de la taille du motif,
 - ✘ les automates offrent une solution optimale.

- ✘ Les automates ont aussi d'autres applications :
 - ✘ représentation de toute « machine à états finis ».

- ✘ Pour construire un automate on commence souvent avec un automate **non-déterministe** :
 - ✘ la construction est souvent plus directe/plus simple,
 - ✘ la détermination est nécessaire, mais facilement **automatisable**.