# TP 6— AUTOMATES

http://www.liafa.jussieu.fr/~nath/tp6/tp6.pdf

Les questions sont les bienvenues et peuvent être envoyées à nathanael.fijalkow@gmail.com.

Pour ce premier TP de l'année, nous allons implémenter des automates. L'objectif de ce TP est d'écrire les fonctions utiles que nous utiliserons plus tard pour manipuler nos automates.

## 1 MOTS ET LANGAGES

L'alphabet latin (ISO/IEC 8859-1) contient 256 caractères, et est le plus largement utilisé pour des raisons historiques, malgré l'émergence d'un nouvel alphabet plus complet (UTF-8, qui contient 95 000 caractères). En Caml light, l'alphabet en 256 caractères (le plus souvent ISO/IEC 8859-1) est manipulé par le type `char` (un caractère) et le type `string` (un mot).

Étant donné un mot $u$ (donc un élément de type `string`), on accède à sa $i$-ème lettre par $u.[i]$, et comme pour les vecteurs les positions sont indexées à partir de $0$. La longueur du mot $u$ est donnée par `string_length u`.

## 2 AUTOMATES DÉTERMINISTES

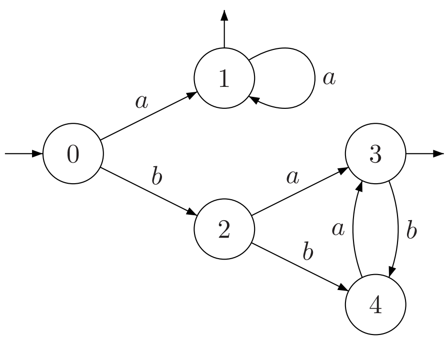
Les automates que l'on considère dans cette partie sont déterministes (pour chaque couple (état,lettre), il y a *au plus* une transition).

L'enregistrement `taille` donne le nombre d'états. L'état initial est donné par l'enregistrement `initial`, et les états finaux sont donnés par le vecteur de booléens `final`. Le tableau `transitions` contient l'ensemble des transitions.

La taille des tableaux `transitions` et `final` doit être `taille`, mais ceci n'est pas spécifié dans le type.

```
type automate =
  { taille :  int ;
    initial :  int ;
    transitions :  (char * int) list vect ;
    final :  bool vect } ;;
```

▷ **Question 1.** Écrire une fonction `calcul_det` qui étant donné un mot et un automate supposé déterministe, détermine si l'automate accepte ce mot. Quelle est sa complexité ? ◁



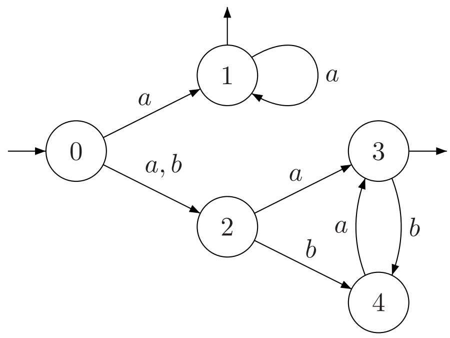

▷ **Question 2.** Définir l'automate ci-dessus, et vérifier sur les exemples $aa$, $aba$ et $bab$ que la fonction `calcul_det` est correcte. ◁

▷ **Question 3.** Écrire une fonction `accessible` qui supprime les états inaccessibles d'un automate. Pour cela, on doit renuméroter les états, on pourra maintenir deux tableaux `tab_conv` et `tab_inv` qui gère la correspondance entre nouveaux et anciens états. ◁

## 3 AUTOMATES NON-DÉTERMINISTES

Considérons à présent les automates non-déterministes, toujours avec le type `automate`.

▷ **Question 4.** Écrire une fonction `calcul_nondet` qui étant donné un mot et un automate, détermine si l'automate accepte le mot. Quelle est sa complexité ? ◁




▷ **Question 5.** Définir l'automate représenté ci-dessus, et vérifier sur les exemples $aa$, $aba$ et $bab$ que la fonction `calcul_nondet` est correcte. ◁

## 4 Déterminisation

Pour déterminiser un automate, on construit l'automate des parties : étant donné un automate non-déterministe $\mathcal{A} = (Q = \{0, \ldots, n-1\}, 0, \delta, F)$, son déterminisé est $\widehat{\mathcal{A}} = (2^Q, \{0\}, \delta', F')$, où

$$\delta'(S, a) = \{q' \in Q \mid \exists q \in S, (q, a, q') \in \delta\}$$

et

$$F' = \{S \subseteq Q \mid S \cap F \neq \emptyset\} .$$

Pour représenter les états de $\widehat{\mathcal{A}}$, on code les sous-ensembles $S \subseteq \{0, \ldots, n-1\}$ par des entiers. Par exemple, $\{0, 2, 3\}$ est représenté par le tableau de booléens `[| true ; false ; true ; true |]`, et de manière équivalente par l'entier $2^0 + 2^2 + 2^3 = 13$.

▷ **Question 6.** Écrire les fonctions de conversion `tab2int` et `int2tab`. La fonction `int2tab` prend en argument l'entier $k$ à convertir et la taille $n$ du tableau attendu. ◁

▷ **Question 7.** Écrire une fonction `determinise` qui calcule l'automate déterminisé. ◁

Le nombre d'états de $\widehat{\mathcal{A}}$ est exponentiel en le nombre d'états de $\mathcal{A}$; ceci rend impraticable la déterminisation dès que $\mathcal{A}$ est gros. En pratique, on préfère calculer seulement la partie accessible de $\widehat{\mathcal{A}}$. En effet, *souvent*, le déterminisé a (beaucoup) moins que $2^n$ états. Cependant, dans certains cas cette borne est atteinte :

▷ **Question 8.** Construire un automate non-déterministe reconnaisant $L_n = A^* \cdot a \cdot A^{n-1}$. Montrer que tout automate déterministe reconnaissant $L_n$ possède au moins $2^n$ états. ◁

## 5 Minimisation

Pour minimiser un automate (déterministe et complet), on calcule l'équivalence de Nérode. Étant donné $\mathcal{A} = (Q = \{0, \ldots, n-1\}, 0, \delta, F)$, c'est la relation d'équivalence sur $Q$ définie par

$$p \sim q \iff \forall w \in A^*,\ (p \cdot w \in L(\mathcal{A}) \Leftrightarrow q \cdot w \in L(\mathcal{A})) .$$

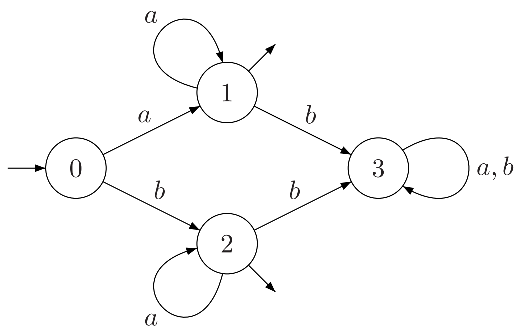
On la calcule par approximations successives : on définit les relations $\sim_k$ pour $k \in \mathbb{N}$ par

$$p \sim_k q \iff \forall w \in A^{\leq k},\ (p \cdot w \in L(\mathcal{A}) \Leftrightarrow q \cdot w \in L(\mathcal{A})) .$$

$A^{\leq k}$ est l'ensemble des mots de longueur au plus $k$.

▷ **Question 9.** Montrer que $\sim\, = \,\sim_{n-2}$. ◁

Pour maintenir ces relations successives, on utilise un tableau `tab_partition` d'entiers de taille $n$. La valeur `tab_partition.(i)` est un état $j \leq i$ qui est en relation avec $i$. La fonction `classe` calcule le représentant minimal de la classe de $i$, et met à jour les valeurs des états considérés. Pour tester si deux états sont en relation, il suffit de calculer les représentants minimaux de leurs deux classes, et de les comparer.




▷ **Question 10.** Écrire une fonction `minimise` qui calcule l'automate minimal. Tester sur l'automate ci-dessus. ◁