# Cours A&G – Automates minimaux

## 1 Langage associé à un automate, à un état $q$ de l'automate

$\mathscr{L}(A) \stackrel{def}{=} \{\omega \mid \exists\ (q_0, \omega) \xrightarrow[A]{*} (q_n, \epsilon)\ \wedge\ q_0 \in Init(A)\ \wedge\ q_n \in Acc(A)\}$

$\mathscr{L}(\{q\}_A) \stackrel{def}{=} \{\omega \mid \exists\ (q, \omega) \xrightarrow[A]{*} (q_n, \epsilon)\ \wedge\ q_n \in Acc(A)\}$

*autrement dit, on prend $q$ comme état initial.*

## 2 Minimisation

**Principe général des algorithmes de minimisation** Les algorithmes de minimisation sont utilisées pour simplifier un problème ou optimiser un système :

- on fusionne les équations équivalentes ($x = 2y$ et $x = 4y - x$) d'un système avant de le résoudre
- on fusionne les contraintes équivalentes ($x \in \mathbb{N}$, $x > 0$ et $x \geq 1$) pour éliminer les contraintes redondantes
- en compilation on réduit le nombre de variables en fusionnant celles qui ont la même valeur pendant une partie de l'exécution (dans les boucles en particuler) on économise ainsi le nombre de regsitres utilisés et les accés mémoires.

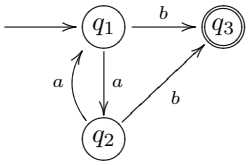
Tous ces algorithmes de minimisation sont basés sur le même principe : fusionner ce qui est équivalent. Pour minimiser un automate, on va fusionner les états équivalents, c'est-à-dire les états qui reconnaissent les mêmes mots.

**Définition de l'équivalence entre états** $q \simeq q' \stackrel{def}{=} \mathscr{L}(\{q\}_A) = \mathscr{L}(\{q'\}_A)$

Deux états $q$ et $q'$ de l'automate $A$ seront considérés comme équivalents si le langage reconnu par $A$ en prenant $q$ comme état initial (noté $L_q$) est le même que le langage reconnu par $A$ en prenant $q'$ comme état initial (noté $L_{q'}$).

**Classes d'équivalence associée à $\simeq$** Puisqu'on a défini une équivalence $\simeq$ entre états, une classe d'équivalence de $\simeq$ est – par définition – un ensemble d'état équivalent : ceux qu'on va fusionner.

**Exemple :** Considérons l'automate A =



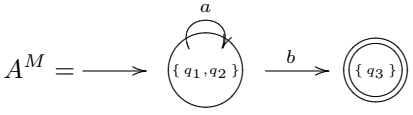

En étudiant les états $q_1$ et $q_2$ on constate que $L_{q_1} = L_{q_2}$ donc $q_1 \simeq q_2$ et que $L_{q_1} \neq L_{q_3} \neq L_{q_2}$, donc $q_1, q_2 \not\simeq q_3$.

L'ensemble $\mathcal{Q} = \{q_1, q_2, q_3\}$ des états de $A$ peut donc être partionné en deux classes d'équivalences $\{q_1, q_2\} \sqcup \{q_3\}$. Ce qui est une manière de dire qu'on fusionne les états $q_1$ et $q_2$.

**Automate minimal** Les états de l'automate minimale $A^M$ sont des ensembles d'états de $A$, ce sont les classes des états équivalents de $A$. Les transitions de $A^M$ sont celle de $A$ (mais entre ensemble d'états).

Autrement dit, la minimisation consiste à reconstruire la structure des transitions sur des états qui sont les classes d'équivalences de l'automate d'origine.

**Exemple (suite) :**



## 3 Algorithme de construction des classes d'équivalences

On considère une partition $\mathscr{P} \stackrel{def}{=} \{Q_0, Q_1, \ldots, Q_n\}$ de l'ensemble $\mathcal{Q}$ des états de l'automate $A$ – c'est-à-dire $\mathcal{Q} = Q_0 \sqcup Q_1 \sqcup \ldots \sqcup Q_n$. La partition $\mathscr{P}$ est *compatible* avec la relation de transition de $A$ si

$$\forall Q_i \in \mathscr{P},\ \forall q,p \in Q_i, \quad q,p \textit{ se comportent de la même manière, c'est-à-dire}$$
$$q \in Acc(A) \Leftrightarrow p \in Acc(A)$$
$$\bigwedge \quad \forall s \in \Sigma, \quad \left.\begin{array}{c} q \xrightarrow{s} q' \in A \\ p \xrightarrow{s} p' \in A \end{array}\right\} \Rightarrow \exists Q_j \in \mathscr{P},\ q', p' \in Q_j$$

Les classes d'équivalences de $\simeq$ correspondent à la plus petite partition $\mathscr{P}$ de $\mathcal{Q}$ (celle avec les moins de $Q_i$ possible) compatible avec la relation de transition.

**Principe de minimisation** On cherche à construire la partition de $\mathcal{Q}$ qui regroupe le plus d'états. On commence avec la partition la plus grossière $\mathscr{P} = \{\mathcal{Q}\}$ constituée du seul sous-ensemble $\mathcal{Q}$ et on la redécoupe jusqu'à obtenir une partition compatible avec la relation de transition de l'automate. L'algorithme de construction des classes d'équivalences est fourni en OCAML en fin de chapître.

**Exemple d'exécution sur l'automate précédent :**

```
#use "minimisation.ml" ;;

val aut1 : automaton =
  {alphabet = ['a'; 'b']; initial = [1]; accepting = [3];
   transitions = [ (1, 'a', 2) ; (2, 'a', 1) ;
                   (1, 'b', 3) ; (2, 'b', 3) ]}

initial partition = {{1,2,3}}

 states 1 ~/~ 3 : NOT same accepting status
So, {1,2,3}  is splitted into  {1,2} |_| {3}

final partition = {{1,2},{3}}
```

**Exemple d'exécution sur un automate plus conséquent :**

```
#use "minimisation.ml" ;;

val aut2 : automaton =
  {alphabet = ['z'; 'u']; initial = [1]; accepting = [5; 6];
   transitions =
    [(1, 'z', 1); (2, 'z', 1); (3, 'z', 4); (4, 'z', 5); (5, 'z', 5);
     (6, 'z', 5); (1, 'u', 2); (2, 'u', 3); (3, 'u', 3); (4, 'u', 3);
     (5, 'u', 6); (6, 'u', 5)]}

initial partition = {{1,2,3,4,5,6}}

 states 1 ~/~ 5 : NOT same accepting status
So, {1,2,3,4,5,6}  is splitted into  {1,2,3,4} |_| {5,6}

 states 1 ~/~ 4 : NOT same behavior on symbol 'z'
So, {1,2,3,4}  is splitted into  {1,2,3} |_| {4}

 states 1 ~/~ 3 : NOT same behavior on symbol 'z'
So, {1,2,3}  is splitted into  {1,2} |_| {3}
```

```
 states 1 ~/~ 2 : NOT same behavior on symbol 'u'
So, {1,2}  is splitted into  {1} |_| {2}

final partition = {{1},{2},{3},{4},{5,6}}
```

**Implantation en OCAML de la construction des classes d'états équivalents** Voir site web A&G → outils → `minimization.ml`