# TP-Colle-3 (MPI) :

## Environnement

Nous travaillerons en OCaml mode interprété pour ce TP-colle. Nous vous laissons le choix de l'environnement, selon vos habitudes. Vous pouvez utiliser :

- L'interpréteur en ligne BetterOCaml : https://BetterOCaml.ml/
- Emacs, muni du mode Tuareg.

## Consignes et mise en place

- Vous répondrez aux questions de programmation en écrivant le code dans votre outil.
- Vous répondrez aux questions de cours à l'aide de la fiche-réponse à la fin du sujet.
- Vous répondrez aux questions à développer à l'oral lors de votre passage au tableau (pour les concernés). Utilisez un brouillon pour vous préparer des notes.

# I Préliminaires

1. *(Question de cours)* Montrez que l'intersection de deux langages reconnaissables par un automate est reconnaissable par un automate. Discutez rapidement du cas de l'union et de la concaténation de deux langages reconnus par un automate.
2. Définissez un type `'a automate`, comme un enregistrement qui contient les champs suivants :
   - `nb`, un entier tel que tous les états de l'automate sont des `int` entre `0` et `nb-1`.
   - `sigma`, un tableau d'éléments de type `'a` qui seront les lettres de l'alphabet.
   - `i` qui est la liste des états initiaux.
   - `f` qui est la liste des états finaux.
   - `delta` l'ensemble des transitions. Vous êtes libres de l'implémenter comme vous le souhaitez (liste de triplets, table de hachage, matrice de transition où chaque coefficient contient la liste des lettres qui permettent de prendre une transition, etc...)

# II Problème de l'inclusion

Etant donné deux automates $\mathcal{A}_1$ et $\mathcal{A}_2$, on souhaite pouvoir déterminer si $L(\mathcal{A}_1)$ est inclus dans $L(\mathcal{A}_2)$. Pour des raisons de simplicité, nous allons supposer qu'on ne travaille qu'avec des automates déterministes dans cette partie.

On va pour cela se baser sur l'identité suivante : pour tous ensembles $A$ et $B$, on a $A \subseteq B$ si et seulement si $A \backslash B = \emptyset$.

1. Ecrire une fonction `completer : 'a automate -> 'a automate` qui prend en entrée un automate, et renvoie un automate complet qui renvoie le même langage.
2. `(Question à développer à l'oral)` Quelle est la complexité de la fonction précédente ?
3. Ecrire une fonction `complementaire : 'a automate -> 'a automate` qui prend en entrée un automate déterministe, et renvoie l'automate reconnaissant le langage complémentaire de l'automate d'entrée.
4. *(Question à développer à l'oral)* Soient $m$ et $n$ deux entiers non nuls. Construisez une bijection entre $[\![0; m-1]\!] \times [\![0; n-1]\!]$ et $[\![0; mn-1]\!]$. Justifiez que votre application est bijective.

5. Ecrire une fonction `intersection : 'a automate -> 'a automate -> 'a automate` qui prend en entrée deux automates déterministes sur le même alphabet et construit l'automate reconnaissant l'intersection de leurs langages.
6. *(Question à développer à l'oral)* A partir des fonctions précédentes, étant donné deux automates déterministes $\mathcal{A}_1$ et $\mathcal{A}_2$, comment peut-on construire un automate qui reconnaît le langage $L(\mathcal{A}_1)\backslash L(\mathcal{A}_2)$ ? Quelle complexité demande sa construction ?
7. *(Question à développer à l'oral)* Que faut-il maintenant vérifier sur l'automate construit à la question précédente, pour être capable de dire si $L(\mathcal{A}_1) \subset L(\mathcal{A}_2)$ ? Proposez une méthode pour effectuer cette vérification.

# III Minimiser un automate

On s'intéresse dans cette section au problème suivant : étant donné un automate déterministe, on souhaite construire un automate déterministe reconnaissant le même langage, mais dont le nombre d'états est minimal.

Soit $\mathcal{A} = (Q, \Sigma, \{q_0\}, F, \delta)$ un automate déterministe.

1. *(Question à développer à l'oral)* Justifier qu'on peut considérer $\delta$ comme une fonction partielle de $Q \times \Sigma$ dans $Q$. Que vaut $\delta(q, a)$ pour $q \in Q, a \in A$ ?
   Proposez une extension de $\delta$ sur l'ensemble $Q \times \Sigma^*$ et donnez alors un moyen d'obtenir la valeur $\delta(q, u)$.

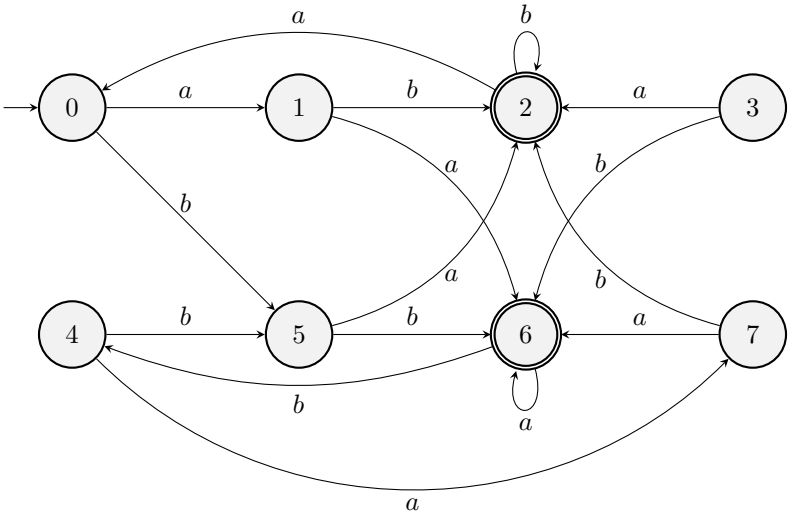
Pour minimiser l'automate, une idée est de fusionner des états qui jouent le même rôle (c'est-à-dire depuis lesquels on peut faire les mêmes opérations), et de laisser séparés ceux qui doivent être séparés. On propose l'algorithme suivant, qui est plutôt élémentaire[1], pour résoudre ce problème :

(i) Au début, on va considérer chaque paire d'état $(p, q) \in Q \times Q$ comme des états *séparables* si exactement un état de la paire est final.

(ii) On répète cette opération :

- pour toute paire $(p, q) \in Q \times Q$ et toute lettre $a \in \Sigma$
  - on calcule la paire d'états $(p', q')$ avec $p' = \delta(p, a)$ et $q' = \delta(q, a)$, pourvu que ces deux états existent.
  - si la paire $(p', q')$ est séparable, alors la paire $(p, q)$ devient séparable.
- si on a marqué au moins une nouvelle paire comme étant séparable, on recommence. Sinon, on sort de cette boucle.

2. *(Question à développer à l'oral)* Appliquer l'algorithme précédent à cet exemple. Vous pouvez représenter les paires marquées dans un tableau à double entrée, qui évoluera pendant l'algorithme.




[1] Du moins, en comparaison à l'algorithme de Moore ou à celui de Hopcroft

Dans la suite, à chaque étape de l'algorithme considéré, on représentera les états séparés dans une matrice de booléen, qu'on appellera *matrice de séparation* : si on l'appelle `t`, on aura `t.(p).(q)` à `true` si la paire `(p, q)` est séparable, sinon `false`.

3. Implémentez une fonction `init_separation : 'a automate -> bool array array`, qui renvoie la matrice de séparation initialisée par l'étape (i) de l'algorithme.
4. Implémentez une fonction `etape_de_separation : 'a automate -> bool array array -> bool` qui réalise une itération de la boucle donnée en (ii). Cette fonction doit recevoir en argument la matrice de séparation et la modifier. Elle devra renvoyer un booléen indiquant si une nouvelle paire a été marquée.
5. Implémentez une fonction `separation : 'a automate -> bool array array` qui réalise tout l'algorithme, et renvoie la matrice de séparation finale correspondant à l'automate donné.
6. *(Question à développer à l'oral)* Vérifiez que l'algorithme réalisé termine. Donnez sa complexité.

On définit pour tout état $p$, $L_p(\mathcal{A}) = \{v \in \Sigma^* \mid \delta(p, v) \in F\}$ (c'est le langage des mots qui permettent d'atteindre un état acceptant depuis $p$).
On admet pour l'instant la propriété suivante : à la fin de cet algorithme, on a l'équivalence

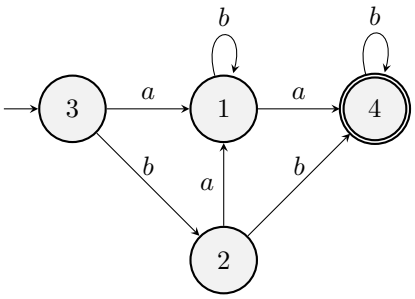
$$q \text{ et } q' \text{ ne sont pas séparables si et seulement si } L_q(\mathcal{A}) = L_{q'}(\mathcal{A})$$

7. *(Question à développer à l'oral)* Proposez une méthode pour construire l'automate minimal à partir de la matrice de séparation. Donner une description formelle de l'automate obtenu.
8. *(Question à développer à l'oral)* Montrez la propriété précédente.

# IV Des automates vers les expressions régulières par le Lemme d'Arden

Nous allons voir ici une méthode alternative pour dériver une expression régulière reconnaissant le même langage qu'on automate déterministe donné, basé sur la résolution d'un système d'équations sur les langages.

Etant donné un état $q_i$ d'un automate déterministe, on notera $X_i$ le langage des mots qui permettent de passer de $q_i$ à un état acceptant dans l'automate.




Par exemple, voici le système d'équations qu'on peut dériver de l'automate ci-contre :

$$\begin{aligned} X_1 &= bX_1 \mid aX_4 \\ X_2 &= aX_1 \mid bX_4 \\ X_3 &= aX_1 \mid bX_2 \\ X_4 &= bX_4 \mid \varepsilon \end{aligned}$$

Nous allons représenter chaque équation par une paire d'expressions régulières étendues (que nous abrégerons en ERE), chaque membre de la paire étant un membre de l'équation. Nous définirons les ERE pour un alphabet $\Sigma$ et un ensemble de variables $V$ ainsi :

- $\emptyset$ est une ERE.
- pour tout $a \in \Sigma$, $a$ est une ERE.
- pour tout $v \in V$, $v$ est une ERE.
- si $e_1$ et $e_2$ sont des ERE, alors $e_1 \mid e_2$, $e_1 e_2$ et $e_1^*$ sont des ERE.

Pour un automate donné, on construira des ERE avec $V = \{x_q \mid q \in Q\}$. Ainsi, dans l'exemple ci-dessous, la première équation est représentée par la paire d'expressions $(x_1, bx_1 \mid ax_4)$

1. Déclarez un type `expreg` d'expressions régulières étendues.
2. Implémentez une fonction `equation : 'a automate -> int -> ('a ere)*('a ere)` telle que `equation a q` renvoie l'équation associée à l'état `q` dans l'automate `a`, par la méthode donnée en exemple ci-dessus. La première ERE de la paire est le membre gauche de l'équation, et la deuxième est le membre droit.

3. Implémentez une fonction `systeme : 'a automate -> (('a ere)*('a ere)) list`, qui a un automate donné associe son système d'équations.

Pour résoudre un tel système, on utilise le Lemme d'Arden :

Si $A$ et $B$ sont deux langages sur $\Sigma$ et que $\varepsilon \notin A$, alors l'équation $X = AX \mid B$ (dont l'inconnue est le langage $X$, donc un ensemble) admet pour unique solution $A^*B$.

Prouvons ce lemme.

4. *(Question à développer à l'oral)* Montrer que $A^*B$ est bien solution de l'équation donnée.
5. *(Question à développer à l'oral)* Montrez que si $X$ est un langage solution de l'équation, on a nécessairement que pour tout $n \geq 0$, $X = A^{n+1}X \mid A^nB \mid \ldots \mid AB \mid B$. Déduisez-en que $A^*B \subseteq X$.
6. *(Question à développer à l'oral)* Réciproquement, montrez que si $u \in X$, alors il existe un $m$ tel que $u \in A^mB$. Déduisez-en que $x \subseteq A^*B$.

7. *(Question à développer à l'oral)* Par substitutions successives, résolvez le système d'équations de l'exemple donné plus haut.
8. *(Question à développer à l'oral)* Proposez un algorithme qui permette de résoudre un système d'équation issu d'un automate déterministe. Si vous en avez le temps, implémentez le.

# Annexes

Vous trouverez ici des rappels concernant certains traits du langage OCaml

## Enregistrements

- Déclaration : `type nom = { champ_1 : type_1 ; ...; champ_k : type_k }`
- Pour un champ mutable : `mutable champ_1 : type_1`
- Créer un objet : `{ champ_1 = val_1 ; ...; champ_k = val_k }`
- Accéder à la valeur d'un champ : `objet.champ`
- Changer la valeur d'un champ : `objet.champ <- valeur`

## Module List

- `List.length l` renvoie la longueur de la liste`l`
- `List.hd l` récupère la tête de la liste `l`
- `List.tl l` récupère la queue de la liste `l`
- `[a0;a1;a2;...]` pour construire une liste d'exemple à la main
- `a::t` construit la liste de tête `a` et de queue `t`
- `List.mem x l` renvoie un booléen spécifiant si `x` apparaît dans `l`
- `List.map f l` renvoie la liste `[ f(a.0) ; f(a1) ; ...]` si `l = [a0 ; a1 ; ...]`
- `List.iter f l` exécute dans cet ordre `[ f(a.0) ; f(a1) ; ...]` si `l = [a0 ; a1 ; ...]`

## Tables de hachage

Pour manipuler des tables d'association (dictionnaires):

- `Hashtbl.create n` Renvoie une table de hachage de taille $n$.
- `Hashtbl.add table cle valeur` Ajoute l'association clé → valeur à la table de hachage.
- `Hashtbl.replace table cle valeur` Remplace la valeur courante associée à `cle` dans la table par `valeur`.
- `Hashtbl.mem table cle` renvoie `true` ssi la cle a une valeur associée dans la table.
- `Hashtbl.find table cle` renvoie la valeur associée à la clé dans la table.
- `Hashtbl.find_opt table cle` renvoie une option sur la valeur associée à la clé dans la table (c'est-à-dire `None` s'il n'en existe pas, et `Some v` si la valeur `v` a été trouvée.

# Fiche réponse

NOM :
Prénom :
Répondez ici aux questions de cours, en rédigeant vos réponses.