Licence Sciences et Technologies
Univ. Grenoble Alpes

Examen à mi-parcours
INF 302 : Langages et Automates
L2, 2025/2026

Rappel à propos des consignes et quelques conseils et remarques

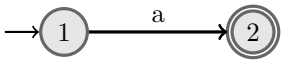
— Durée : 2 heures.
— Aucune sortie avant 30 minutes. Aucune entrée après 30 minutes.
— 1 feuille A4 R/V autorisée.
— Tout dispositif électronique est interdit (calculatrice, téléphone, tablette, montre connectée, etc.).
— Le soin de la copie sera pris en compte.
— Les exercices sont indépendants et en ordre croissant de difficulté.
— Le barème est donné à titre indicatif.
— L'examen est sur 22 points, vous devez obtenir 20 points pour obtenir la note maximale.

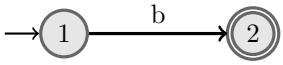
Exercice 1 (Vrai ou Faux - 4 points)

Répondre par vrai ou faux aux questions suivantes. Justifier soigneusement et de façon concise vos réponses (sans preuve). Si une proposition est fausse, répondre par un contre-exemple.

1. **(0,5 pt)** Pour construire un automate déterministe reconnaissant le complémentaire d'un langage L reconnu par un automate A, il suffit d'inverser les états accepteurs et non-accepteurs de A.
2. **(0,5 pt)** Pour tout langage à états L, si $\text{Pref}(L) = \Sigma^*$, alors $L = \Sigma^*$.
3. **(0,5 pt)** Si L_1 et L_2 sont des langages reconnus par des automates déterministes à n_1 et n_2 états respectivement, alors, en ne comptant pas ses états non-accessibles, l'automate produit reconnaissant $L_1 \cap L_2$ possède exactement $n_1 \times n_2$ états.
4. **(0,5 pt)** Pour tout automate fini déterministe, l'automate minimal équivalent est unique à un renommage des états près.
5. **(0,5 pt)** Si deux états p et q d'un automate déterministe sont équivalents alors pour tout $s \in \Sigma$, $\delta(p, s)$ est équivalent à $\delta(q, s)$.
6. **(0,5 pt)** Si un AEFD possède un cycle accessible, alors le langage qu'il reconnaît est infini.
7. **(0,5 pt)** L'automate déterministe résultant de l'algorithme de déterminisation appliqué à un automate non-déterministe est minimal.
8. **(0,5 pt)** Pour deux automates non-déterministes reconnaissant le même langage, les automates obtenus après déterminisation puis minimisation sont les mêmes, à un renommage des états prêts et en ignorant les états non-accessibles.

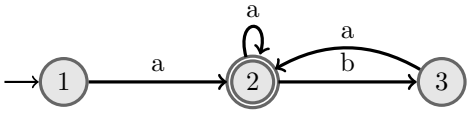
Solution de l'exercice 1

1. Faux. Il faudrait également que l'automate soit complet. Sinon, l'algorithme donné est incorrect.
2. Faux. Prenons Σ tel que $\{a\} \subseteq \Sigma$ et $L = \Sigma^* \cdot \{a\}$, nous avons $L \neq \Sigma^*$ et $\text{Pref}(L) = \Sigma^*$.
3. Faux. Prenons les automates suivants :
   — Automate déterministe reconnaissant L_1 :
   

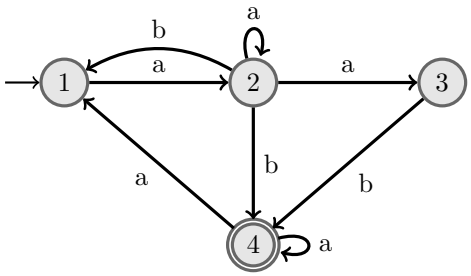
   — Automate déterministe reconnaissant L_2 :
   

   — Automate produit des deux automates précédents :
   

4. Vrai. (Argument simplifié). Tout autre automate déterministe de la même taille mais différent aurait soit un état marqué comme final ou non-final différemment de l'automate minimal ou alors aurait



(a) Automate pour l'Exercice 2.

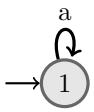


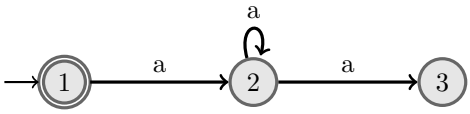
(b) Automate pour l'Exercice 4.

FIGURE 1 – Automates pour les exercices de complémentation et de déterminisation.

une transition en plus ou en moins entre les états (qui sont tous accessibles et co-accessibles) et ajouterait ou enlèverait un chemin vers un état accepteur.

5. Vrai. Si deux états p et q sont équivalents, alors $\forall w \in \Sigma^* : \delta^*(p, w) \in F \Leftrightarrow \delta^*(q, w) \in F$. Nous en déduisons $\forall w \in \{s\} \cdot \Sigma^* : \delta^*(p, w) \in F \Leftrightarrow \delta^*(q, w) \in F$ puis $\forall w \in \Sigma^* : \delta^*(\delta(p, s), w) \in F \Leftrightarrow \delta^*(\delta(q, s), w) \in F$. Ceci signifie que $\delta(p, s)$ et $\delta(q, s)$ sont équivalents.
6. Faux. Il faut également que ce cycle soit co-accessible. Voici deux contre-exemples :





7. Faux. Voir par exemple l'exemple utilisé dans le cours sur la déterminisation.
8. Vrai. L'automate minimal est unique à un rennommage des états prêt.

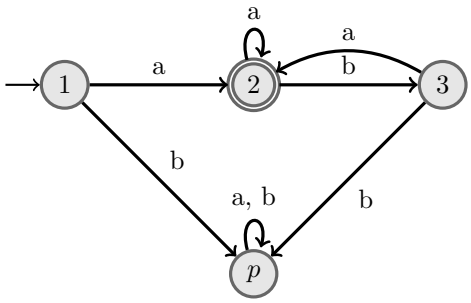
Exercice 2 (Automate complémentaire - 2 points)

Nous considérons l'automate représenté sur la Figure 1a.

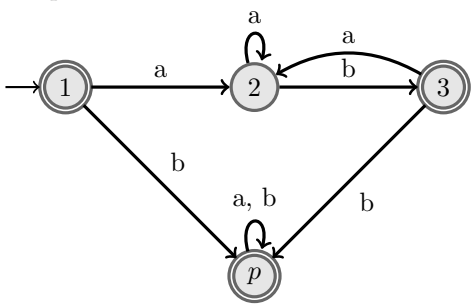
1. Calculer et représenter l'automate complémentaire de cet automate.

Solution de l'exercice 2

1. Nous commençons par compléter l'automate.



Dans cet automate complet, nous inversons états accepteurs et non-accepteurs. L'automate complémentaire est représenté ci-dessous :



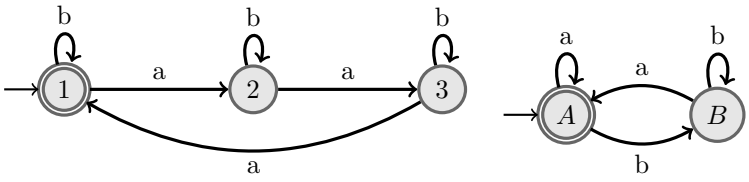


FIGURE 2 – Automates dont il faut calculer le produit.

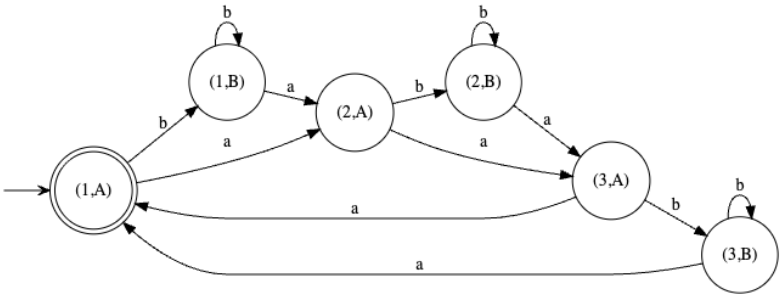
Exercice 3 (Automate produit et exécution - 3 points)

Nous considérons les automates représentés sur la Figure 2.

1. **(1 pt)** Calculer et représenter l'automate produit de ces deux automates.
2. **(1 pt)** Décrire le langage reconnu par l'automate produit.
3. **(1 pt)** Est-ce que le mot *abaab* est accepté par l'automate produit ? Justifier en donnant l'exécution de ce mot sur cet automate.

Solution de l'exercice 3

1. L'automate produit est représenté ci-dessous :



2. Le premier automate reconnaît l'ensemble des mots avec un ombre d'occurrences du symbole 'a' multiple de trois. Deuxième auto, l'automate reconnaît l'ensemble des mots qui termine par le symbole 'a'. L'automate produit reconnaissant l'intersection des langages, il reconnaît donc l'ensemble des mots avec un nombre d'occurrences du symbole 'a' qui est multiple de trois et qui terminent par le symbole 'a' s'il y a au moins une occurrence de 'a' ; c'est-à-dire que le langage contient également ϵ. Alternativement, le langage reconnu est l'ensemble des mots avec un nombre d'occurrences de 'a' multiple de 3 et qui ne terminent pas par 'b'.
3. Le mot *abaab* n'est pas accepté par l'automate. L'exécution de ce mot est donnée ci-dessous :

$$(1A, abaab) \cdot (2A, baab) \cdot (2B, aab) \cdot (3B, ab) \cdot (1A, b) \cdot (1B, \epsilon)$$

(Nous dénotons l'état (X, Y) par XY.)
L'état $1B$ n'étant pas accepteur, le mot n'est pas accepté.

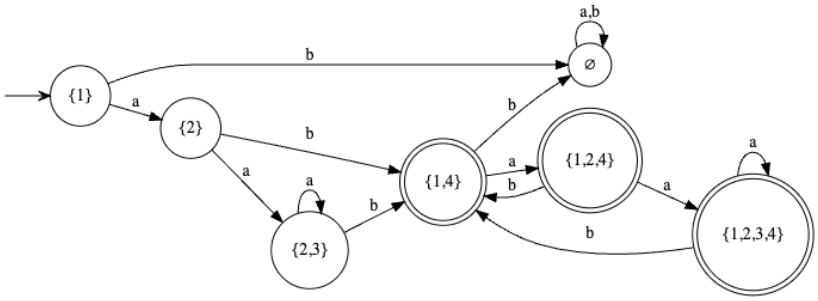
Exercice 4 (Automate déterminisé - 4 points)

Nous considérons l'automate représenté sur la Figure 1b.

1. Calculer et représenter le déterminisé de cet automate.

Solution de l'exercice 4

1. Le déterminisé est représenté ci-après.



Exercice 5 (Algorithme : impasse - 2 points)

Nous définissons une *impasse* dans un automate comme un état dans l'automate que l'on peut atteindre depuis l'état initial, mais qui ne mène à aucun état accepteur.

1. **(1 pt)** Exprimer la condition d'être sans impasse pour un automate en utilisant les états accessibles et co-accessibles.
2. **(1 pt)** Donner un algorithme, en moins de 5 lignes, qui détermine si un automate donné en entrée est sans impasse. Vous pouvez réutiliser tous les algorithmes vus en cours sans les redéfinir.

Solution de l'exercice 5

1. Un automate est sans impasse si et et seulement si tous ses états accessibles sont aussi co-accessibles.

2. L'algorithme est donné ci-après.

Algorithme 1 : Vérifier l'absence d'impasse dans un automate

Entrée : Un automate $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$
Sortie : Vrai si et seulement si A est sans impasse
Accessibles $\leftarrow$ accessibles(A)
co_Accessibles $\leftarrow$ co_accessibles(A)
retourner Accessibles $\subseteq$ co_Accessibles

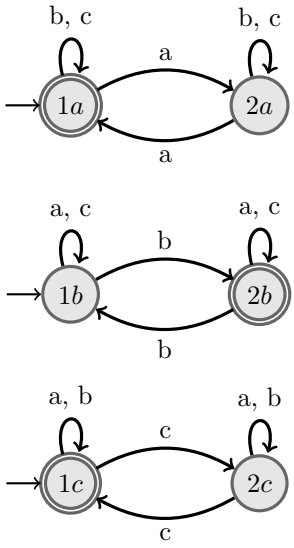
Exercice 6 (Automates avec plusieurs états initiaux - 4 points)

Nous souhaitons définir une variante d'automates à états finis non-déterministes où le non-déterminisme est introduit par la possibilité d'avoir plusieurs états initiaux. Les transitions restant comme dans le cas des automates déterministes. Lorsqu'il s'exécute, l'automate peut démarrer simultanément dans plusieurs états initiaux.

1. **(1 pt)** Donner un exemple en représentation graphique d'un tel automate avec moins de 6 états pour reconnaître l'ensemble des mots qui contiennent un nombre pair de a ou un nombre impair de b ou un nombre pair de c.
2. **(1 pt)** Donner une définition pour de tels automates en prenant soin de spécifier les contraintes sur chaque élément impliqué dans la définition.
3. **(1 pt)** Définir les configurations de tels automates.
4. **(1 pt)** Définir la relation de dérivation entre configurations de tels automates.

Solution de l'exercice 6

1. Un automate qui reconnaît l'ensemble des mots avec un nombre pair de a ou un nombre impair de b ou un nombre pair de c est représenté ci-après :



2. Un automate avec plusieurs états initiaux est donné par un 5-tuple $(Q, \Sigma, Q_{\text{init}}, \delta, F)$ tel que $Q_{\text{init}} \subseteq Q$ et les autres éléments sont définis comme pour les automates déterministes.
3. Une configuration de l'automate est un couple formé par un état et un mot, comme dans le cas des automates déterministes, non-déterministes et non-déterministes avec ϵ-transitions.
4. La relation de dérivation est définie comme dans le cours.
   Note : Pour les deux dernières questions, on acceptera les réponses différentes mais cohérentes ; par exemple les configurations où il y a un ensemble d'états et une relation de dérivation qui calcule l'ensemble des successeurs.

Exercice 7 (Algorithme : langage à longueur fixe - 3 points)

Un langage L est dit à longueur fixe s'il existe $n \in \mathbb{N} \setminus \{0\}$ tel que tous les mots de L sont de longueur n.

1. **(1 pt)** Donner un exemple et un contre-exemple de langage à longueur fixe.
2. **(2 pt)** Donner un algorithme qui détermine si un automate déterministe donné en entrée est à longueur fixe. Indice : il faut s'inspirer de l'algorithme de parcours en largeur.

Solution de l'exercice 7

1. — Exemple de langage : $\{aaa, bbb, aab\}$.
   — Contre-exemple de langage : $\{a, bb, ccc\}$.
2. L'algorithme est donné ci-après.

Algorithme 2 : Vérifier si un automate déterministe est à longueur fixe

Entrée : Un automate déterministe $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$
Sortie : Vrai si et seulement si A est à longueur fixe

// Construction de l'automate émondé
$E \leftarrow \text{emonde}(A)$

// Si l'automate possède un cycle accessible et co-accessible, alors le langage est infini
si *detection_cycle*(E) **alors**
    retourner faux

// Parcours en largeur dans l'automate sans cycle, sans mémorisation des états déjà visités
A_visiter : file d'états $\leftarrow$ file vide
A_visiter.ajouter(q_{init})

// Fonction Distances associant à chaque état l'ensemble de ses distances à l'état initial
// Chaque ensemble est fini car l'automate est acyclique
Distances : $Q \to 2^{\mathbb{N}}$
Distances$(q_{\text{init}}) \leftarrow \{0\}$

tant que A_visiter $\neq \emptyset$ **faire**
    $q \leftarrow A$_visiter.defiler()
    pour $q' \in \mathit{Succ}(q)$ **faire**
        Anciennes_distances $\leftarrow$ Distances(q')
        Distances$(q') \leftarrow$ Distances$(q') \cup \{\, d + 1 \mid d \in \text{Distances}(q) \,\}$
        si *Distances*$(q') \neq$ *Anciennes_distances* **alors**
            A_visiter.enfiler(q')

longueurs_acceptees $\leftarrow \bigcup_{q \in F} \text{Distances}(q)$
retourner $|\mathit{longueurs\_acceptees}| = 1$ *et* $\mathit{longueurs\_acceptees} \neq \{0\}$