Licence Sciences et Technologies
Univ. Grenoble Alpes

Examen à mi-parcours
INF 302 : Langages et Automates
L2, 2019/2020

Rappel à propos des consignes et quelques conseils et remarques

— Durée : 2 heures.
— Aucune sortie avant 30 minutes.
— Aucune entrée après 30 minutes.
— 3 feuilles A4 R/V autorisées.
— Tout dispositif électronique est interdit (calculatrice, téléphone, tablette, montre connectée, etc.).
— Le soin de la copie sera pris en compte.
— 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 - 5 points)

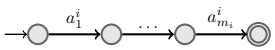
Répondre par vrai ou faux aux questions suivantes. Justifier soigneusement et de façon concise vos réponses (sans preuve). Si une proposition est fausse, répondre par un contre-exemple.

1. Tout langage fini est un langage à états.
2. Tout langage à états est un langage fini.
3. Un automate déterministe et complet reconnaît le langage universel.
4. Si un automate a un cycle, le langage qu'il reconnaît est infini.
5. Si un automate a un cycle accessible, le langage qu'il reconnaît est infini.
6. La différence ensembliste entre deux langages à états est un langage à états.
7. La concaténation entre deux langages à états est un langage à états.
8. Pour un automate A, on note $|A|$ le nombre d'états de A. Soient A_1 et A_2 deux automates à états finis et déterministes. $|A_1 \times A_2| \leq |A_1| \times |A_2|$.
9. Si un langage est à états, alors tout sous-ensemble de ce langage est un langage à états.
10. Si un langage est à états, alors tout sur-ensemble de ce langage est un langage à états.

Solution de l'exercice 1

Pour chacune des questions, nous donnons une explication lorsque la réponse est vraie et un contre-exemple lorsque la réponse est fausse.

1. Vrai. Soit L un langage est fini, alors il peut se décrire en extension sous la forme $\{w_1, \ldots, w_n\}$ où les w_i sont des mots. Pour chaque mot w_i qui s'écrit $a_1^i \cdot \ldots \cdot a_{m_i}^i$, nous pouvons trouver un automate comme ci-dessous qui le reconnait :



   Ainsi, comme $L = \cup_{i=0}^{n} \{w_i\}$ et que les langages à états sont fermés par union (car ils sont fermés par négation et intersection), nous en déduisons le résultat.
2. Faux. Nous pouvons prendre un automate déterministe et complet ne contenant pas d'état accepteur, par exemple.
3. Faux. Le langage sur l'alphabet $\{a, b\}$ des mots ne contenant que des a est infini, mais c'est un langage à états.
4. Faux. On peut prendre tout automate possédant un cycle mais sans état accepteur et plus généralement tout automate dont les cycles ne sont pas co-accessibles.
5. Faux. Même justification que pour la précédente question.
6. Vrai. Soient L et L' deux langages (à états), alors $L \setminus L' = L \cap \overline{L'}$. Comme les langages à états sont fermés par négation et intersection, alors $l \setminus L'$ est un langage à états si L et L' sont des langages à états.
7. Vrai. A partir des automates reconnaissant L et L' deux langages à états, nous pouvons obtenir un automate qui reconnaît $L \cdot L'$ en construisant l'automate contenant les mêmes états et transitions que les automates de L et L'. Par ailleurs, nous gardons comme états accepteurs ceux de l'automate reconnaissant L' et ajoutons une ϵ-transition entre chaque état accepteur de l'automate reconnaissant L vers l'état initial l'automate qui reconnaît L'.
8. Vrai. Par définition, l'ensemble des états de l'automate produit est le produit cartésien de l'ensemble d'états des deux automates utilisés. Le cardinal du produit cartésien de deux ensembles est le produit des cardinaux.
9. Faux. Par exemple, nous pouvons prendre le langage des mots sur l'alphabet $\{a, b\}$ qui contiennent autant de a que de b qui n'est pas un langage à états. Ce langage est un sous-ensemble du langage universel sur l'alphabet $\{a, b\}$.
10. Faux. Par exemple, nous pouvons prendre le langage des mots sur l'alphabet $\{a, b\}$ qui contiennent autant de a que de b qui n'est pas un langage à états. Ce langage est un sur-ensemble du langage vide.

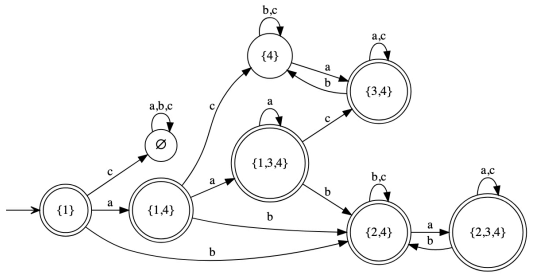
Exercice 2 (Déterminisation, minimisation - 4 points)

Soit $\Sigma = \{a, b, c\}$. Considérons l'automate de la Figure 1a.

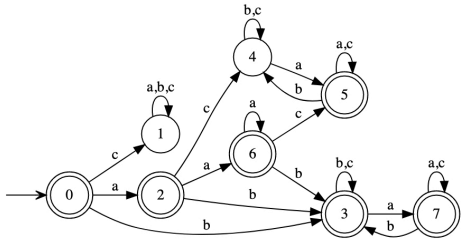
1. Déterminiser l'automate.
2. Minimiser l'automate obtenu.

Solution de l'exercice 2

1. Le determinisé de l'automate représenté dans la Figure 1a est donné ci-dessous.



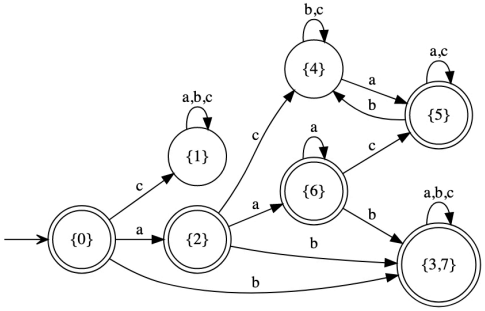
2. Nous renommons les états de l'automate pour obtenir l'automate ci-dessous.



Ensuite, nous appliquons l'algorithme de minimisation (l'automate est complet).

$\equiv_0$	$\equiv_1$	$\equiv_2$	$\equiv_3$
0	0	0	0
2	2	2	2
3	3	3	3
5	7	7	7
6	5	5	5
7	6	6	6
1	1	1	1
4	4	4	4

L'automate n'est pas minimal, les états 3 et 7 peuvent être fusionnés. Nous obtenons l'automate ci-dessous.



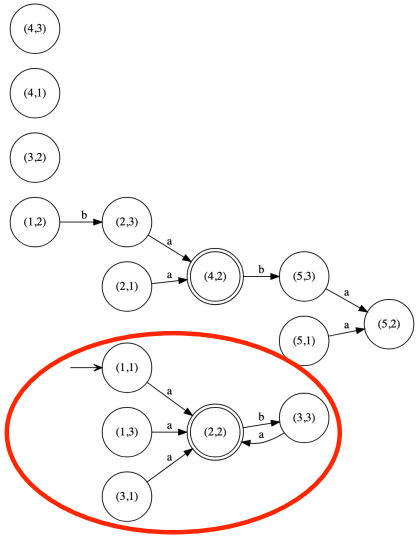
Exercice 3 (Produit d'automates - 4 points)

Soit $\Sigma = \{a, b\}$. Considérons les automates des Figures 1b et 1c.

1. Calculer le produit de ces deux automates.

Solution de l'exercice 2

1. Le produit des deux automates est donné ci-dessous. Tous les états sont représentés. En calculant le produit à la volée, nous obtenons uniquement la partie dans l'ovale rouge.



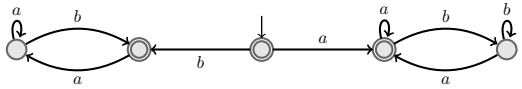
Exercice 4 (Un langage à états ? - 3 points)

Soit $\Sigma = \{a, b\}$. Considérons le langage L des mots sur Σ qui contiennent autant d'occurrences du facteur $a \cdot b$ que du facteur $b \cdot a$.

1. Donner deux mots dans L et deux mots dans $\Sigma^* \setminus L$.
2. Est-ce que L est un langage à états ? Si oui, donner un automate qui le reconnait, sinon expliquer de manière informelle pourquoi ce langage n'est pas un langage à états.

Solution de l'exercice 4

1. Deux mots dans L sont aba et b. Deux mots dans $\Sigma^* \setminus L$ sont $abab$ et ab.
2. Oui. Un automate reconnaissant L est donné ci-dessous.



Exercice 5 Mot formé par les états dans le parcours en profondeur - 3 points

Soit $(Q, \Sigma, q_0, \delta, F)$ un automate à états fini déterministe dont tous les états sont accessibles. Supposons un ordre $\prec$ sur les symboles dans Σ. Nous considérons l'ensemble des états Q de l'automate comme un alphabet, que nous nommons Σ_Q. Nous considérons les mots formés sur l'alphabet Σ_Q. Ces mots sont dans l'ensemble Σ_Q^*. Par exemple si $Q = \{1, 2, 3\}$, alors un mot sur Σ_Q est $3 \cdot 2 \cdot 2 \cdot 2 \cdot 1$ où $\cdot$ est le symbole de la concaténation.

1. Donner un algorithme qui pour un automate retourne le mot de Q^* de longueur $|Q|$ dont l'ordre des symboles dans le mot est le même que l'ordre obtenu par le parcours en profondeur de l'automate

dans lequel l'ordre $\prec$ est utilisé pour déterminer la priorité entre symboles.

Solution de l'exercice 5

1. L'algorithme est donné ci-dessous.

Univ. Grenoble Alpes, Département Licence Sciences et Technologies, Licence deuxième année

Mot formé par les états dans le parcours en profondeur

Algorithme 1 Mot formé par les états dans le parcours en profondeur

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AEFD *)

```
1:  pile d'états À_visiter := ensemble d'états de départ ;
2:  ens d'états Déjà_visités := ∅ ;                    (* initialement, rien n'est visité *)
3:  mot m := ϵ ;                                       (* initialement, le mot est vide *)
4:  tant que ¬ À_visiter.est_vide() faire              (* tant qu'il reste des états à visiter *)
5:      état q := À_visiter.dépiler() ;                (* prendre un état à visiter *)
6:      si q ∉ m alors
7:          m := m · q ;
8:      fin si
9:      Déjà_visités := Déjà_visités ∪ {q} ;           (* l'état q vient d'être visité *)
10:     À_visiter.empiler(Succ(q) \ Déjà_visités) ;
                  (* les nouveaux états à visiter sont les successeurs de q non visités *)
11: fin tant que
```

- Cet algorithme suppose que la structure de pile « se comporte comme un ensemble », c'est à dire que l'empilage d'un élément déjà présent dans la pile est sans effet.
- La structure de données pour Déjà_visités n'importe pas.
- La ligne 11 peut être remplacée par une itération sur les éléments de $\text{Succ}(q)$ dans l'ordre inverse de $\prec$.
- L'empilage des successeurs se fait dans *l'ordre inverse* de la priorité entre successeurs donnée par $\prec$.

Exercice 6 Profondeur d'un automate - 3 points

On appelle chemin dans l'automate, une séquence de transitions partant de l'état initial et suivant la fonction ou relation de transitions (c'est à dire que deux transitions consécutives dans la séquence sont telles que l'état d'arrivé de la première est l'état de départ de la seconde). La longueur d'un chemin est le nombre d'éléments dans la séquence. On appelle profondeur d'un état, la longueur du plus petit chemin dans l'automate telle que la dernière transition du chemin ait pour état d'arrivé cet état. On appelle profondeur d'un automate la profondeur maximale de ses états.

1. Donner un algorithme qui donne la profondeur d'un état $q \in Q$ dans un automate à états non-déterministe $(Q, \Sigma, q_0, \Delta, F)$.
2. Donner un algorithme qui donne la profondeur d'un automate à états non-déterministe $(Q, \Sigma, q_0, \Delta, F)$.

Solution de l'exercice 6

1. L'algorithme est donné ci-dessous.

Profondeur d'un état basé sur le parcours en **largeur**

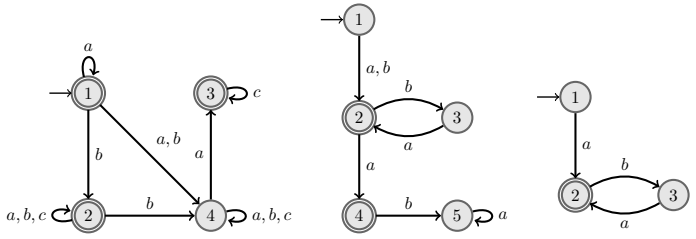
Algorithme 1 Profondeur d'un état dans un automate suivant 1 ordre $\prec$ entre les symboles

Entrée : $A = (Q, \Sigma, q_{\text{init}}, \delta, F)$ (* un AEFD *)
Entrée : $q_p \in Q$ (* un état dont on cherche la profondeur *)
Sortie : la profondeur de q_p dans A

```
1:  file d'états À_visiter := {(q_init, 0)} ;
2:  ens d'états Déjà_visités := ∅ ;                    (* initialement, rien n'est visité *)
3:  état q, entier prof ;                              (* temporaires *)
4:  tant que ¬ À_visiter.est_vide() faire              (* tant qu'il reste des états à visiter *)
5:      (q, prof) := À_visiter.défiler() ;             (* prendre un (état, profondeur) à visiter *)
6:      si q = q_p alors retourner prof
7:      fin si
8:      Déjà_visités := Déjà_visités ∪ {q} ;           (* l'état q vient d'être visité *)
9:      pour q_s ∈ Succ(q) \ Déjà_visités faire         (* suivant ≺ *)
10:         À_visiter.enfiler(q_s, prof + 1) ;
11:     fin pour
12: fin tant que
```

- Cet algorithme suppose que la structure de file « se comporte comme un ensemble », c'est à dire que l'enfilage d'un état déjà présent dans la file est sans effet.
- La structure de données pour Déjà_visités n'importe pas.
- L'ordre d'enfilage des successeurs n'importe pas.

2. Un algorithme (peu efficace) est de calculer la profondeur de chaque état. Il est possible de le faire en un passe avec un parcours en largeur et en enregistrant la profondeur maximale rencontrée.



(a) Automate pour l'exercice 2 (b) Automate pour l'exercice 3 (c) Automate pour l'exercice 3

FIGURE 1: Automates pour les exercices.