Licence Sciences et Technologies
Univ. Grenoble Alpes

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

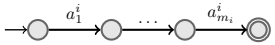
Rappel à propos des consignes et quelques conseils et remarques

— 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.

Solution de l'exercice ??

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

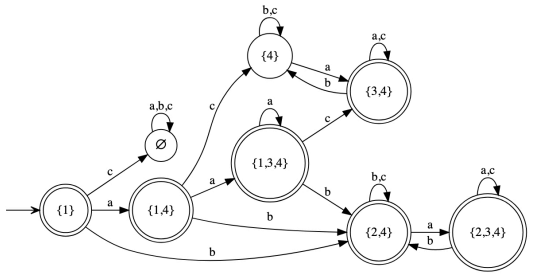
1. Vrai. Soit L un langage est fini, alors il peut se décrire en extension sous la forme $\{w_1, \ldots, w_n\}$ où les w_i sont des mots. Pour chaque mot w_i qui s'écrit $a_1^i \cdot \ldots \cdot a_{m_i}^i$, nous pouvons trouver un automate comme ci-dessous qui le reconnait :



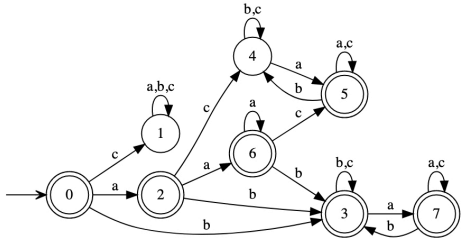
   Ainsi, comme $L = \cup_{i=0}^{n} \{w_i\}$ et que les langages à états sont fermés par union (car ils sont fermés par négation et intersection), nous en déduisons le résultat.
2. Faux. Nous pouvons prendre un automate déterministe et complet ne contenant pas d'état accepteur, par exemple.
3. Faux. Le 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.

Solution de l'exercice ??

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



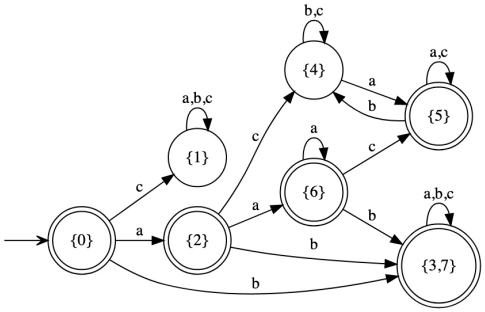
2. Nous renommons les états de l'automate pour obtenir l'automate ci-dessous.



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

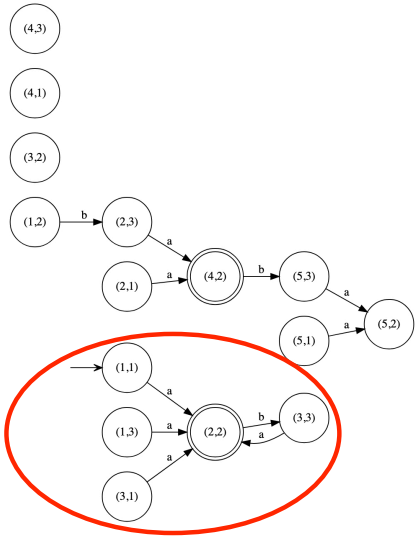
$\equiv_0$	$\equiv_1$	$\equiv_2$	$\equiv_3$
0	0	0	0
2	2	2	2
3	3	3	3
5	7	7	7
6	5	5	5
7	6	6	6
1	1	1	1
4	4	4	4

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



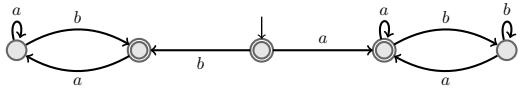
Solution de l'exercice ??

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.



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.



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 := $\emptyset$; (* initialement, rien n'est visité *)
3: **mot** m := ϵ ; (* initialement, le mot est vide *)
4: **tant que** $\neg$ 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** $q \notin m$ **alors**
7: m := $m \cdot q$;
8: **fin si**
9: Déjà_visités := Déjà_visités $\cup \{q\}$; (* l'état q vient d'être visité *)
10: À_visiter.*empiler*(Succ(q) $\setminus$ 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 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 $\prec$.

Y. Falcone (UGA - Inria) INF 302 : Langages & Automates Année Académique 2019 - 2020 1 / 1

Solution de l'exercice 6

1. L'algorithme est donné ci-dessous.

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

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

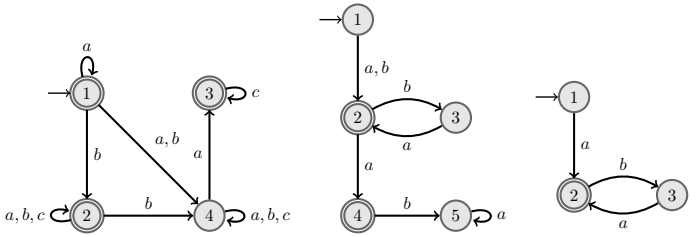
Algorithme 1 Profondeur d'un état dans un automate suivant 1 ordre $\prec$ entre les symboles

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 **??**

(c) Automate pour l'exercice **??**

FIGURE 1: Automates pour les exercices.