Opérations sur les automates

Consignes

Sauf mention contraire, les questions de ce sujet doivent être traitées dans l'ordre. Chaque question est caractérisée par un ou plusieurs "types", signalés par un pictogramme :

- Les questions marquées avec ⌨ nécessitent d'écrire un programme dans le langage demandé. **Le code produit doit compiler, s'exécuter correctement et être testé même lorsque l'énoncé ne le demande pas explicitement.** La clarté du code est également importante : il faut pouvoir en expliquer le fonctionnement à l'examinateur ou l'examinatrice à sa demande.
- Les questions marquées 👆 sont à traiter à l'oral. Elles peuvent néanmoins être préparées en amont, y compris à l'aide d'un support écrit si vous en avez besoin.
- Les questions marquées ✎ sont aussi à traiter à l'oral mais à l'aide d'un support écrit. Il est demandé pour ces questions d'écrire la réponse ou les grandes lignes de votre raisonnement sur une feuille et de s'appuyer dessus lors de l'explication à l'examinateur ou examinatrice.

Lors du passage de l'examinateur ou de l'examinatrice, vous devez présenter les réponses apportées aux questions traitées depuis le dernier passage. Il est toujours possible de solliciter un passage en levant la main, que ce soit pour vérifier une solution, discuter d'une idée ou demander de l'aide. Une fois que votre appel a été vu, vous pouvez aborder les questions suivantes en attendant.

Le sujet demande de compléter le code compagnon Operations_automates.ml. Ce dernier fournit du code qui sera présenté au fil de l'énoncé. Sauf mention contraire, il est interdit d'utiliser des boucles ou des références. Il est en revanche permis d'utiliser toutes les fonctions du module `List`.

- - - - - - -

L'objectif de ce sujet est de résoudre algorithmiquement le problème $\mathcal{P}$ suivant :

$$\begin{cases} \textbf{Entrée :} \text{ Deux automates finis potentiellement non déterministes } A \text{ et } B. \\ \textbf{Sortie :} \text{ Oui si } A \text{ et } B \text{ reconnaissent le même langage ; non sinon.} \end{cases}$$

Les automates seront manipulés en OCaml via le type suivant, fourni dans le code compagnon :

```
type automate = {alphabet : char list;
                 nb_etats : int;
                 initiaux : int list;
                 finaux : int list;
                 transitions : (int * char * int) list}
```

Les états d'un automate seront toujours numérotés de 0 à `nb_etats`-1. Les transitions de l'automate sont stockées sous forme de liste dans le champ `transitions`. Par exemple, si `(0,'a',3)` fait partie de la liste `transitions`, cela signifie qu'il existe une transition depuis l'état 0 vers l'état 3 étiquetée par la lettre a. **On supposera dans tout le sujet que les automates manipulés utilisent un même alphabet Σ. Pour les exemples et les tests, on aura $\Sigma = \{a, b\}$.**

1. ✎ ⌨ Dessiner un automate non déterministe à trois états qui reconnaît le langage $(a+b)^\star ab(a+b)^\star$ sur $\{a, b\}$. Puis, définir dans le code un automate `ab` permettant de le représenter en OCaml.
2. 👆 Le code compagnon définit un automate `test_1`. Quel langage reconnaît-il ? Proposer une façon pertinente de renommer cet automate. Reprendre ces questions avec l'automate `test_2`.

Pour savoir si deux automates A et B reconnaissent le même langage, on se propose de construire un automate qui reconnaît $\mathcal{L}(A)\Delta\mathcal{L}(B)$ où Δ représente la différence symétrique. On rappelle que si E et F sont deux ensembles, $E\Delta F = (E \cup F) \setminus (E \cap F)$.

3. ✎ Rappeler comment exprimer une différence symétrique à l'aide d'unions, d'intersections et de complémentaires. Expliquer comment le calcul de $\mathcal{L}(A)\Delta\mathcal{L}(B)$ permettra de répondre au problème $\mathcal{P}$.

4. ✎ Dessiner un automate A_1 reconnaissant le complémentaire du langage reconnu par l'automate `test_1` défini dans le code compagnon. Puis, dessiner un automate A' reconnaissant l'intersection du langage reconnu par A_1 et du langage reconnu par l'automate `ab` défini en question 1.

5. ⌨ Écrire une fonction `union : automate -> automate -> automate` permettant de faire l'union de deux automates. L'automate résultant de cette opération devra bien avoir ses états numérotés de 0 à son nombre d'états moins un.

La renumérotation des états devient plus délicate dans le cas d'une intersection où d'une déterminisation. Le code compagnon fournit une fonction `correspondance :'a list -> ('a, int) Hashtbl.t` telle que `correspondance etats` renvoie un dictionnaire dont les valeurs numérotent tous les éléments de `etats` (qui sont donc les clés de ce dictionnaire) consécutivement à partir de 0.

6. ☝ Peut-on effectivement renommer les éléments d'une liste quel que soit le type de ses éléments avec la fonction `correspondance` ? Justifier.

7. ⌨ Écrire `renommage_etat : ('a, int) Hashtbl.t -> 'a -> int` telle que `renommage_etat dico e` renvoie le numéro de `e` selon la correspondance établie par le dictionnaire `dico`.

   Écrire de même une fonction `renommage_liste_etats : ('a, int) Hashtbl.t -> 'a list -> int list` qui renomme tous les états d'une liste puis une fonction `renommage_transitions : ('a, int) Hashtbl.t -> ('a*char*'a) list -> (int*char*int) list` qui renomme tous les états dans une liste de transitions.

Le code compagnon fournit une fonction `automate_parties` qu'il n'est pas nécessaire de comprendre. Elle prend en entrée un automate A. Elle renvoie des informations sur l'automate A' obtenu à partir de A par déterminisation accessible (avec complétion), à savoir, dans cet ordre :

- La liste des états de A'.
- Son état initial.
- La liste de ses états finaux.
- La liste de ses transitions.

Les états dans la sortie ne sont en revanche pas numérotés consécutivement à partir de 0.

8. ⌨ En utilisant la question 7, écrire une fonction `determinisation : automate -> automate` qui déterminise l'automate en entrée. Vérifier la cohérence du résultat obtenu sur l'automate `test_1`.

9. ⌨ ☝ Écrire une fonction `complementaire : automate -> automate` permettant de complémentariser l'automate en entrée. Justifier brièvement la correction de votre algorithme.

10. ⌨ Décommenter la fonction `intersection` fournie dans le code compagnon puis écrire une fonction `difference_symetrique` réalisant la différence symétrique de deux automates.

11. ☝ Combien d'états a l'automate résultant de la différence symétrique entre `test_2` défini dans le code compagnon et A' construit à la question 4 ? D'où pourrait provenir ce phénomène ?

12. ⌨ Déduire des questions précédentes une fonction `equivalents` indiquant si les deux automates en entrée reconnaissent le même langage. On pourra s'aider de fonctions intermédiaires.

13. ⌨ ☝ Les automates `test_2` et A' sont-ils équivalents ? Est-ce surprenant ?