TP D'OPTION INFORMATIQUE 3
Implémentation des automates
Niveau Bonus

L'objectif de cette partie bonus est d'obtenir une implémentation des automates pour laquelle la fonction de transition est définie par une table, ce qui garantit que chaque lecture de lettre dans l'automate est en $O(1)$. En effet les fonctions de transition des automates obtenus précédemment font appel à des fonctions coûteuses, comme `union` ou `bordure`.

1 Automates définis par une table

Le nouveau type d'automates (déterministes complets) que nous allons utiliser est le suivant :

```
type 'a adc2 = {
    tab_alphabet : 'a array;
    hash_alphabet : ('a, int) Hashtbl.t;
    table : int array array;
    finaux : int list
    };;
```

La table étant une matrice d'entier, on note qu'il est nécessaire de stocker en mémoire l'alphabet, sous forme de tableau pour passer de l'entier à la lettre, et sous forme de table de hachage pour le sens réciproque. On notera que dans cette représentation, les états sont des entiers : il y en a exactement autant que de lignes dans la matrice `table`, et l'état initial est toujours 0. L'état atteint depuis l'état i en lisant la lettre d'indice j est alors donné par `table.(i).(j)`. Le type `'a` est le type des lettres de l'alphabet.

1. Représenter graphiquement l'automate suivant :

   ```
   let ex2 =
       let t = [| [|1;0|] ; [|0;1|] |] in
       let h = Hashtbl.create 2 in
       Hashtbl.add h 'a' 0;
       Hashtbl.add h 'b' 1;
       {
       tab_alphabet = [|'a';'b'|];
       hash_alphabet = h;
       table = t;
       finaux = [0]
       };;
   ```

2. Écrire une fonction `indice_lettre` prenant en argument un automate à table et une lettre, et renvoyant l'entier associé à cette lettre.
   On utilisera la fonction `Hashtbl.find : ('a, 'b) Hashtbl.t -> 'a -> 'b` .
3. Implémenter les fonctions `delta_etoile` et `reconnait` pour ce type.
4. Écrire une fonction traduisant un automate défini dans le type `adc2` en un automate équivalent du type `adc`.
5. La traduction réciproque demande un peu plus de travail. L'idée est de procéder à un parcours en profondeur des états de l'automate pour leur associer à chacun un entier. Pour garder en mémoire les états déjà atteints, nous allons avoir besoin d'une structure de dictionnaire.
   (a) Implémenter une structure de dictionnaire. Nous n'utiliserons que les opérations créant un dictionnaire, ajoutant un couple (clé, valeur), testant si une clé est présente, et donnant la valeur associée à une clé. On pourra dans un premier temps utiliser une simple

implémentation par référence de liste, qui pourra toujours être remplacée plus tard par une implémentation plus efficace par arbre binaire de recherche voire table de hachage dynamique.

(b) Écrire la fonction traduisant un automate `adc` en automate `adc2`. Cette fonction prendra également l'alphabet en argument. On procédera de la façon suivante :
- Un premier parcours en profondeur des états permettra de remplir un dictionnaire dont les clés seront les états de l'automate `adc`, et les valeurs sont les entiers qu'on leur attribuera. À l'issue de ce premier parcours, le nombre d'états accessibles distincts est connu.
- Un second parcours servira à remplir la table de transition et la liste des états finaux.

6. Écrire une fonction `produit2 : 'a adc2 -> 'a adc2 -> 'a adc2` prenant en argument deux automates à table, supposé de même alphabet, et renvoyant leur automate produit.

2 Double renversement de Brzozowski

Le double renversement de Brzozowski est une méthode pour minimiser un automate : il suffit de le transposer, de le déterminiser, et à nouveau le transposer et déterminiser.

1. Écrire une fonction `transpose` prenant en argument un automate `adc2` et renvoyant son automate transposé, c'est-à-dire l'automate dans lequel le sens de toutes les flèches sont inversées, les transitions mais aussi les états initiaux et finaux. On notera bien que l'automate transposé est en général non déterministe, on construira donc un automate du type `(int, 'a) nda`.
2. En déduire une fonction `minimisation` prenant en argument un automate `adc2` et renvoyant son automate `adc2` minimal obtenu par double renversement de Brzozowski. On utilisera la fonction déterminisant un automate `nda` en automate `adc`.
3. Écrire une fonction `glushkov2` prenant en argument une expression rationnelle et renvoyant l'automate `adc2` minimal reconnaissant cette expression.

3 Reconnaissance des occurrences d'un mot dans un texte

L'objectif ici est d'obtenir un programme efficace déterminant toutes les occurrences d'un mot dans un texte, à l'aide de l'automate des occurrences construit dans le type `adc2`.

1. Nous allons utiliser comme alphabet l'ensemble des lettres présentes dans le mot. Écrire une fonction `lettres_presentes` prenant en argument une chaîne de caractères, et renvoyant un tableau contenant chaque caractère distinct présent dans la chaîne. On pourra travailler sur une référence de liste puis utiliser la fonction `Array.of_list`.
2. Écrire une fonction `construire_aut_occ` prenant en argument une chaîne de caractère correspondant à un mot, et renvoyant l'automate `adc2` des occurrences associé.
3. On suppose que l'automate des occurrences est construit, et on désire à présent en déduire une fonction donnant toutes les occurrences du mot, en complexité linéaire en la longueur du texte. Écrire une fonction `occurrences_motif` prenant en argument l'automate des occurrences et le texte, et renvoyant la liste de tous les indices terminant une occurrence du mot dans le texte. Par exemple, si `aut` est l'automate des occurrences de `"ana"`, `occurrences_motif aut "ananas banane"` renverra `[2; 4; 10]`. On prendra garde à traiter le cas où la lettre lue dans le texte n'apparaît pas dans le mot, et n'appartient donc pas à l'alphabet (on pourra faire en sorte que la fonction `codage` renvoie une valeur particulière dans ce cas).