TP D'OPTION INFORMATIQUE 2
Implémentation des automates

L'objectif de ce TP est de définir et d'exploiter une structure de données implémentant les automates.

1 Automates déterministes complets

On utilisera pour implémenter un automate déterministe complet le type :

```
type ('a, 'b) adc = {
    initial : 'b ;
    est_final : 'b -> bool ;
    delta : 'b -> 'a -> 'b};;
```

L'ensemble des états est donc une partie du type `'b`, et l'alphabet est une partie du type `'a`, ni l'un ni l'autre n'étant stockés explicitement dans l'automate. Il serait plus efficace mais plus compliqué de coder lettres et états par des indices et de représenter la fonction de transition par une table, on ne fait pas ce choix ici.

1. Voici un exemple d'automate codé dans ce type :

   ```
   let ex =
       let d etat lettre = match etat, lettre with
            0, 'a' -> 1
           |0, 'b' -> 0
           |1, 'a' -> 0
           |1, 'b' -> 1
           |_ -> 2
       in
       let f = function
           0 -> true
           |_ -> false
       in
       {initial = 0;
       est_final = f;
       delta = d};;
   ```

   Représenter cet automate graphiquement.
2. Implémenter dans ce type un automate `aut1` reconnaissant le langage dénoté par l'expression $(a+b)^*c+b$ sur l'alphabet correspondant au type `char`.
3. Écrire une fonction `list_of_string : string -> char list` prenant en argument une chaîne de caractères et renvoyant la liste de caractères correspondante. On pourra utiliser cette fonction pour alléger l'écriture des tests des fonctions suivantes.
4. Écrire une fonction `delta_etoile : ('a, 'b) adc -> 'b -> 'a list -> 'b` calculant la fonction de transition étendue d'un automate.
5. Écrire une fonction `reconnait : ('a, 'b) adc -> 'a list -> bool` décidant si un automate reconnait un mot.

   Tester cette fonction sur les deux automates précédents :
   - `reconnait ex (list_of_string "ababaa")` doit renvoyer `true` ;
   - `reconnait ex (list_of_string "abaabaa")` doit renvoyer `false` ;
   - `reconnait aut1 (list_of_string "baac")` doit renvoyer `true` ;
   - `reconnait aut1 (list_of_string "ba")` doit renvoyer `false`.
6. Écrire une fonction `produit : ('a, 'b) adc -> ('a, 'c) adc -> ('a, 'b * 'c) adc` prenant en argument deux automates et renvoyant l'automate produit reconnaissant l'intersection de leurs langages.

2 Automates non déterministes

On utilisera pour implémenter un automate non déterministe le type :

```
type ('a, 'b) nda = {
    initiaux : 'b list;
    est_final_nd : 'b -> bool;
    delta_nd : 'b -> 'a -> 'b list };;
```

Dans toute la suite, les ensembles d'états seront représentés par des listes **triées** d'états (sans doublons), de façon à ce que l'union puisse être calculée efficacement.

1. Écrire une fonction `union : 'a list -> 'a list -> 'a list` prenant en argument deux listes triées sans doublons représentant deux ensembles, et renvoyant la liste triée sans doublons représentant l'union de ces ensembles. La complexité devra être linéaire en la somme des longueurs des deux listes.

   Par exemple, `union [2;4;5] [1;4;5;6]` doit renvoyer `[1;2;4;5;6]`.
2. Écrire la fonction `delta_etoile_nd : ('a, 'b) nda -> 'b list -> 'a list -> 'b list`.
3. Écrire la fonction `reconnait_nd : ('a, 'b) nda -> 'a list -> bool`.
4. Écrire une fonction `determinise : ('a, 'b) nda -> ('a, 'b list) adc` déterminisant un automate non déterministe.

3 Passage d'une expression rationnelle à un automate

On reprend pour les expressions rationnelles (standards, non vides) le type :

```
type 'a regexp =
    Epsilon |
    Lettre of 'a |
    Plus of 'a regexp * 'a regexp |
    Concat of 'a regexp * 'a regexp |
    Etoile of 'a regexp ;;
```

1. Écrire une fonction `marquage : 'a regexp -> ('a * int) regexp` prenant en argument une expression rationnelle et la marquant (il suffit d'associer à chaque occurrence de lettre un entier strictement positif distinct).
2. Écrire une fonction `glushkov : 'a regexp -> ('a, int list) adc` implémentant la méthode vue en cours (dite algorithme de Glushkov) pour calculer un automate déterministe complet reconnaissant le langage dénoté par l'expression rationnelle prise en argument. On utilisera la fonction `psf` du TP précédent.

4 Automate des occurrences

1. Écrire une fonction `est_prefixe : 'a list -> 'a list -> bool` prenant en argument deux mots u et m et déterminant si u est préfixe de m.
2. Écrire une fonction `bordure : 'a list -> 'a list` prenant en argument un mot non vide et renvoyant sa bordure.
3. Écrire une fonction `aut_occ : 'a list -> ('a, 'a list) adc` prenant en argument un mot m et renvoyant l'automate des occurrences reconnaissant $\Sigma^* m$.