TP D'OPTION INFORMATIQUE 1
Expressions rationnelles

On utilise pour implémenter en Ocaml les expressions rationnelles (standards) le type :

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

où le paramètre de type `'s` correspond au type des éléments de Σ (usuellement `char`).

1. Représenter l'expression $(a+b)^*ca$ dans ce type.
2. Écrire une fonction `taille : 'b regexp -> int` prenant en argument une expression rationnelle et renvoyant sa taille, ie le nombre de constructeurs y apparaissant.
3. On veut à présent calculer les ensembles de 1-préfixes, 1-suffixes, 2-facteurs d'une expression rationnelle. L'algorithme vu en cours nécessite de pouvoir calculer des unions disjointes et des concaténations d'ensembles de lettres.

   Écrire une fonction `union : 'a list -> 'a list -> 'a list` prenant en argument deux listes représentant des ensembles, supposés disjoints, et renvoyant leur union disjointe.

   (b) Écrire une fonction `concat : 'a list -> 'b list -> ('a * 'b) list` prenant en argument deux listes représentant des ensembles de lettres, et renvoyant la liste de couples représentant la concaténation de ces ensembles, obtenue par produit cartésien.

   Par exemple, `concat ['a'; 'b'] ['c'; 'd'; 'e']` doit renvoyer
   `[('a', 'c'); ('a', 'd'); ('a', 'e'); ('b', 'c'); ('b', 'd'); ('b', 'e')]`

   (c) Écrire une fonction `psf : 'b regexp -> bool * 'b list * 'b list * ('b * 'b) list` prenant en argument une expression rationnelle supposée linéaire et renvoyant le quadruplet (b, P, S, F) caractérisant le langage local dénoté par cette expression.

   Par exemple, le quadruplet renvoyé sur l'expression $(a+b)^*c$ doit être

   ```
   false,
   ['a'; 'b'; 'c'],
   ['c'],
   [('a', 'a'); ('a', 'b'); ('b', 'a'); ('b', 'b'); ('a', 'c'); ('b', 'c')]
   ```

   (d) Rappeler la complexité de la fonction précédente.
4. Écrire une fonction

   `reconnu_local : bool * char list * char list * (char * char) list -> string -> bool`

   prenant en argument un quadruplet (b, P, S, F) caractérisant un langage local, et un mot m, et décidant si m appartient à ce langage local. Quelle est la complexité de cette fonction relativement à la longueur de m ?
5. Écrire une fonction `est_lineaire : 'b regexp -> bool` prenant en argument une expression rationnelle et déterminant si elle est linéaire. On demande une complexité linéaire en la taille de l'expression. On pourra utiliser les fonctions suivantes, en $O(1)$:
   - `Hashtbl.create 0` : crée un dictionnaire vide
   - `Hashtbl.mem h a` : teste si la clé `a` apparait dans le dictionnaire `h`
   - `Hashtbl.add h a b` : ajoute la valeur `b` associée à la clé `a` dans le dictionnaire `h`
6. Ecrire une fonction `enumerer : 'b regexp -> int -> 'b list list` prenant en argument une expression rationnelle et un entier n, et renvoyant la liste des mots d'au plus n lettres reconnus par cette expression.