ALGORITHME DE BERRY-SETHI

Vous trouverez sur le site du lycée le code déjà vu sur les automates déterministes légèrement modifié, la définition du type regexp et des fonctions utiles.

1 Automates : rappels

L'alphabet étant connu, un automate (déterministe ou non) est défini par :
- ses états ;
- son ou ses états initiaux ;
- ses états finaux ;
- ses transitions, qui sont des triplets (p, a, q) où p, q sont deux états et a une lettre de l'alphabet.

Autrement dit, avec cette définition, un automate est un quadruplet de 4 ensembles.

Pour représenter l'alphabet, on peut prendre n'importe quel ensemble. Cela peut être l'alphabet à deux lettres a et b. Mais on peut aussi choisir un alphabet plus gros à n lettres $x_0, \ldots, x_{n-1}$: dans ce cas, on représente chaque lettre par leur numéro.

```
type lettre = a | b;;
let alphabet = [a; b];;
```

On peut facilement représenter un automate en CAML par un enregistrement, dont les champs sont des listes. Pour permettre avec le même type de manipuler aussi bien les automates déterministes ou les non-déterministes et laisser le choix de l'alphabet, on donne deux paramètres de type, qui précise la nature des états et la nature de l'alphabet :

```
type ('a, 'b) automate = {
  etats   : 'a list;
  initial : 'a list;
  final   : 'a list;
  trans   : ('a * 'b * 'a) list;
```

Pour distinguer un automate déterministe d'un non-déterministe, il suffit de vérifier que le champ initial est une liste à un élément et que dans le champ trans, il n'y a pas deux triplets (p, a, q) et (p', a', q') tels que $p = p'$ et $a = a'$.

Dans le cadre de ce TP, les états seront toujours des entiers.

2 Expressions régulières

On représente les expressions régulières par le type suivant, paramétré par le type de l'alphabet :

```
type 'a regexp =
   | Eps
   | Ltr of 'a
   | Union of 'a regexp * 'a regexp
   | Conc of 'a regexp * 'a regexp
   | Et of 'a regexp;;
```

Ltr(x) représente la lettre x, les autres constructeurs représentent l'union, la concaténation et l'étoile.

Par exemple, l'expression régulière $e = a^*(ba)^*$ est représentée par l'expression CAML

```
let e = Conc(
  Et(Ltr(a)),
  Et(
    Conc(
       Ltr(b),
       Ltr(a)
    )
  )
);;
```

On veut écrire une fonction qui calcule la linéarisée d'une expression régulière : pour cela, on procède un peu différemment de l'idée du cours. On parcourt l'expression et on remplace chaque lettre par un numéro (ce qui revient à changer chaque lettre par une lettre x_i).

Par exemple, la linéarisée de l'expression ci-dessus est :

```
Conc (
  Et (Ltr 0),
  Et (
    Conc (
      Ltr 1,
      Ltr 2
    )
  )
)
```

Mais ça ne suffit pas pour appliquer l'algorithme de Berry-Sethi, puisqu'à la fin, il faut revenir aux lettres de l'expression de départ. On a donc aussi besoin d'une liste de couples (on appelle une telle liste une liste d'association), qui permet de savoir par quelle lettre remplacer un numéro : `[(0, a); (1, b); (2, a)]`

Écrivez une fonction `linearise` qui calcule à partir d'une expression régulière sa linéarisée, la liste d'association et un entier naturel décrit ci-dessous. Pour cela, procédez récursivement à l'aide d'une fonction encapsulée qui prend en paramètre l'expression e et un numéro k : ce numéro indique que le premier disponible pour la numérotation des lettres de e est k ; le troisième entier calculé est alors le numéro qui suit le dernier utilisé pour numéroter les lettres de e (donc le premier disponible pour une numérotation future).

Par exemple, si l'expression est `Ltr(x)`, alors `lin e k = (Ltr(k), [k,x], k+1)`.

3 Automate local

La deuxième étape de l'algorithme de Berry-Sethi est le calcul des ensembles P, S, F et du booléen V associés à une expression linéaire (voir notations du cours). On a vu dans le cours que de simples listes suffisent pour représenter ces ensembles, la réunion de deux listes étant la concaténation des listes.

Écrivez une fonction `calcul e` qui calcule le quadruplet `(p,s,f,v)`. Vous aurez sans doute besoin de quelques petites fonctions auxiliaires qui manipulent des listes.

Puis écrivez une fonction `automate_local (p,s,f,v) n` qui calcule l'automate local reconnaissant le langage local déterminé par les 4 objets p, s, f, v et un entier n qui est simplement le nombre d'états de l'automate sauf l'état initial : comme on a numéroté les lettres par des entiers naturels à partir de 0, ces états sont donc des entiers positifs ou nuls, donc on peut représenter l'état initial par l'entier -1. Là aussi, n'hésitez pas à créer des petites fonctions auxiliaires.

4 Automate de Glushkov

La dernière étape est le remplacement des numéros des transitions de l'automate local par les lettres de l'expression initiale : cela se fait simplement grâce à la liste d'association.

Terminez par l'écriture d'une fonction `glushkov e` qui calcule à partir d'une expression régulière e un automate (non-déterministe) qui reconnaît le langage décrit par e.