TP7. Algorithme de Berry-Sethi en OCaml

Dans ce TP, on implémente l'algorithme de Berry-Sethi, qui permet d'exhiber un automate reconnaissant le langage d'une expression régulière.

I. Manipulation d'ensembles finis

On représentera un ensemble fini par une liste **triée** :

```
type 'a ensf = 'a list
```

1. Définir une fonction `inter : 'a ensf -> 'a ensf -> 'a ensf` retournant l'intersection des deux ensembles qui lui sont passées en argument. On utilisera le fait que les listes passées en argument sont bien triées pour faire en sorte que l'appel (`inter e1 e2`) s'exécute en temps $O(n)$ où n est le maximum des cardinaux de `e1` et `e2`.

2. Définir une fonction `union : 'a ensf -> 'a ensf -> 'a ensf` retournant l'union des deux ensembles qui lui sont passées en argument. On utilisera le fait que les listes passées en argument sont bien triées pour faire en sorte que l'appel (`union e1 e2`) s'exécute en temps $O(n)$ où n est le maximum des cardinaux de `e1` et `e2`.

II. Expressions régulières linéaires

On définit le type des expressions régulières comme suit :

```
type 'a expr =
    | Vide
    | Epsilon
    | Lettre of 'a
    | Conc of 'a expr * 'a expr
    | Union of 'a expr * 'a expr
    | Etoile of 'a expr
;;
```

1. Écrire une fonction `alphabet : 'a expr -> 'a ensf` retournant l'alphabet employé dans une expression rationnelle, c'est-à-dire l'ensemble des `x` pour lesquels `Lettre (x)` apparaît dans l'expression.

2. Écrire une fonction `contient_epsilon : 'a expr -> bool` disant si une expression régulière linéaire contient le mot vide.

3. Écrire une fonction `pref : 'a expr -> 'a ensf` retournant l'ensemble des préfixes de longueur 1 d'une expression régulière linéaire.

4. Écrire de même une fonction `suf : 'a expr -> 'a ensf` retournant l'ensemble des suffixes de longueur 1.

5. Écrire une fonction `facteur : 'a expr -> 'a -> 'a ensf` telle que `facteur e x` retourne l'ensemble des lettres `y` telles que `xy` soit un facteur du langage associé à `e`.

6. Écrire une fonction `linearise : 'a expr -> ('a * int)expr` telle que `linearise e` retourne une expression régulière linéaire obtenue en remplaçant chaque lettre `x` de `e` par une lettre `(x, i)` où `i` est un entier, de façon à ce que les entiers soient tous distincts.
Indication : On pourra utiliser une référence qu'on incrémentera.

III. Automates locaux

On représente les automates finis non déterministes par le type suivant :

```
type ('a, 'b) automate = {
    sigma : 'a ensf;
    etats : 'b ensf;
    initiaux : 'b ensf;
    finaux : 'b ensf;
    delta : ('b * 'a * 'b) ensf;
}

type 'a mot = 'a list
```

1. Écrire une fonction `aut_local : 'a expr -> ('a, 'a option)automate` telle que `aut_local e`, où le langage associé à `e` est supposé être local, retourne l'automate local associé à ce langage. L'ensemble des états sera représenté par le type `'a option` : l'état initial sera représenté par `None` et pour tout lettre `a`, les transitions étiquetées par `a` arriveront sur l'état `Some a`.

2. Écrire une fonction
`applique_morphisme : ('a * 'b)automate -> ('a -> 'c)-> ('c * 'b)automate` telle que `applique_morphisme a f` retourne l'automate ayant les mêmes états que `a` mais dans lequel chaque transition étiquetée par une lettre `x` a été remplacée par une transition étiquetée par `f x`.

3. Écrire une fonction `berry_sethi : 'a expr -> ('a * ('a * int)option)automate` retournant l'automate de Glushkov associé à une expression rationnelle.

4. Écrire une fonction `execute_lettre : ('a, 'b)automate -> 'b -> 'a -> 'b ensf` qui, à partir d'un automate A, d'un état q et d'une lettre x calcule l'ensemble $\delta(q, x)$.

5. Écrire une fonction `execute : ('a, 'b)automate -> 'b ensf -> 'a mot -> 'b ensf` qui, à partir d'un automate A, d'un ensemble d'états E et d'un mot w calcule l'ensemble, noté $\delta^*(E, w)$, constitué des états pouvant être atteints à partir d'un état de E en lisant le mot w.

6. Écrire une fonction `est_reconnu : ('a, 'b)automate -> 'a mot -> bool` indiquant si le mot donné est reconnu par l'automate donné.