TD 10

Langages

Exercice 1 : Un langage L est dit :

- récursivement énumérable s'il existe un algorithme qui énumère tous les mots de L ;
- récursif s'il existe un algorithme qui, prenant un mot u de A^* , retourne true si u est dans L et false sinon.

Pour les langages suivants, écrire la fonction `test_i` qui prend en argument un mot et vérifie l'appartenance au langage L_i .

`test_i : string -> bool`

- (a) $L_1 = \{w \in A^* \mid \exists |w|\}$
- (b) $L_2 = \{w \in A^* \mid aabb \text{ est un sous-mot}\}$
- (c) $L_3 = \{w \in A^* \mid aabb \text{ est un facteur}\}$
- (d) $L_4 = \{w \in A^* \mid agaga \text{ est un préfixe}\}$
- (e) $L_5 = \{w \in A^* \mid \text{est un paladrome}\}$
- (f) $L_6 = \{a^n b^n \mid n \in \mathbb{N}\}$

Exercice 2 : On définit le type expression rationnelle par :

```
type 'a exp_rat =
  | Vide
  | Epsilon
  | Lettre of 'a
  | Plus of ('a exp_rat )*( 'a exp_rat )
  | Concat of ('a exp_rat )*( 'a exp_rat )
  | Etoile of 'a exp_rat
;;
```

Définir les expressions rationnelles suivantes :

$$e_1 = a.b + b.a$$

$$e_2 = (a + b)^*(aabb)$$

On souhaite construire une fonction qui vérifie si un mot donné est issu d'une expression rationnelle donnée.

1. Écrire une fonction `test_1` qui renvoie vrai si le mot vérifie une expression rationnelle simple (soit une lettre, soit une somme)

Par exemple pour l'expression

```
let e3 = Plus ( Lettre 'a', Lettre 'b' );;
```

2. Écrire une fonction `test_2` qui renvoie vrai si le mot vérifie une expression rationnelle issue d'une concaténation. (on rappelle la fonction : `String.sub w n p` qui renvoie le sous-mot extrait de w)

```
let e4 = Concat( Concat( Concat ( Lettre 'a' , Lettre 'b'),Lettre 'a'),
Lettre 'a' );;
```

3. Écrire une fonction `test` qui renvoie vrai si le mot vérifie une expression rationnelle quelconque.

Par exemple :

```
let e5 = Concat( Etoile( Plus( Lettre 'a' , Lettre 'b')),
Concat( Concat( Concat ( Lettre 'a' , Lettre 'a'),Lettre 'b'), Lettre 'b'
));;
```

Exercice 3: Expressions rationnelles linéaires

On conserve le type précédente.

On pourra travailler avec l'exemple suivant :

```
let e = Concat(Etoile ( Plus (Lettre 'a', Lettre 'b')),Concat(
Concat(Concat(Lettre 'c', Lettre 'd'), Lettre 'e'), Lettre 'f'));;
```

1. Écrire une fonction `est_lineaire` qui renvoie vrai si l'expression rationnelle entrée en argument est linéaire. (on pourra écrire au préalable la fonction `appartient`, et la fonction `lineaire` qui renvoie le couple de type `bool * 'a list`, la liste étant la liste des lettres composant l'expression.)

```
appartient : 'a -> 'a list -> bool
lineaire : 'a exp_rat -> bool * 'a list
est_lineaire : 'a exp_rat -> bool
```

2. Écrire une fonction `mot_vide` qui renvoie vrai si le mot vide appartient au langage dénotée par une expression rationnelle entrée en argument.

```
mot_vide : 'a exp_rat -> bool
```

3. Écrire une fonction `prefixe` qui renvoie la liste de première lettres possibles pour une expression rationnelle linéaire entrée en argument.

```
prefixe : 'a exp_rat -> 'a list
```

4. Écrire une fonction `suffixe` qui renvoie la liste de dernières lettres possibles pour une expression rationnelle linéaire entrée en argument.

```
suffixe : 'a exp_rat -> 'a list
```

5. Écrire la fonction `produit` qui renvoie le produit cartésien de deux listes.

```
produit : 'a list -> 'b list -> ('a * 'b) list
```

6. Écrire la fonction `facteur` qui renvoie la liste des facteurs de deux lettres sous la forme d'une liste de couples.

```
facteur : 'a exp_rat -> ('a * 'a) list
```