TP1 : expressions régulières

1 Manipulations d'expressions régulières

On représente les expressions régulières sur $\Sigma =$ `char` à l'aide du type suivant :

```
type regexp =
    | Vide
    | Epsilon
    | Lettre of char
    | Concat of regexp * regexp
    | Union of regexp * regexp
    | Etoile of regexp
;;
```

Question 1

Définir en OCaml les expressions régulières suivantes :

1. $e_1 = (a|b|c)*$
2. $e_2 = (ab|b)*$
3. $e_3 = (-|\varepsilon)\,(1|2|3)\,(0|1|2|3)*$

On remarque qu'il n'est pas simple de rédiger les expressions ainsi, ni même de les lire.

Question 2

Écrire une fonction `print_regexp : regexp -> unit` affichant à l'écran de manière plus lisible une expression régulière.

Question 3

On souhaite faciliter l'écriture des expressions régulières :

1. Écrire une fonction `mot : string -> regexp` qui construit une expression régulière dénotant le langage ne contenant qu'un seul mot. L'argument est supposé non égal à la chaîne de caractères vide.
2. Écrire une fonction `concatl : regexp list -> regexp` permettant de former une expression régulière comme la concaténation d'un ensemble non vide d'expressions régulières.
3. Faire de même pour `unionl : regexp list -> regexp`.

2 Langage dénoté

Question 4

1. Écrire une fonction `denote_vide : regexp -> bool` qui détermine si une expression régulière dénote le langage $\varnothing$.
2. Écrire une fonction `denote_epsilon : regexp -> bool` qui détermine si une expression régulière dénote un langage **contenant** ε.

Question 5

Écrire une fonction `exemple : regexp -> string` qui construit un mot appartenant au langage dénoté par l'expression régulière, ou qui déclenche une exception `Langage_vide` si le langage dénoté est vide. Tester cette fonction sur e_1, e_2 et e_3.

Question 6

Écrire une fonction `simplifie_vide : regexp -> regexp` qui élimine un maximum de symboles $\varnothing$ de l'expression régulière passée en argument tout en conservant l'équivalence.

3 Expressions régulières étendues

On définit des expressions régulières étendues ainsi :

```
type regexp_etendue =
    | Evide
    | Eepsilon
    | Elettre of char
    | Erange of char * char (* intervalle de caractères *)
    | Econcat of regexp_etendue list
    | Eunion of regexp_etendue list
    | Eetoile of regexp_etendue
    | Eplus of regexp_etendue (* motif repete au moins 1 fois *)
    | Eoption of regexp_etendue (* motif optionnel *)
    | Erepete of regexp_etendue * int (* exactement n fois *)
    | Eplusde of regexp_etendue * int (* au moins n fois *)
    | Emoinsde of regexp_etendue * int (* au plus n fois *)
```

```
;;
```

Question 7

Définir des expressions régulières étendues pour représenter :

- une valeur littérale entière dans un langage C (en base 10 seulement)
- une adresse mail en .fr
- un numéro de téléphone au format "06-01-02-03-04"
- un numéro INE (soit 10 chiffres et 1 lettre, soit 9 chiffres et 2 lettres)

Question 8

Écrire une fonction `traduire : regexp_etendue -> regexp` qui produit une expression régulière équivalente à une expression régulière étendue.

Utiliser les résultats précédents pour générer des exemples d'adresses mail, numéros de téléphone, etc.

4 Premières, dernières lettres et facteurs

Soit L un langage, on note $D(L)$ l'ensemble des lettres qui peuvent apparaître au début d'un mot de L, $F(L)$ l'ensemble des lettres qui peuvent apparaitre en position finale d'un mot de L et $T(L)$ l'ensemble des facteurs de longueur 2 qui existent dans au moins un mot de L.

Question 9

Écrire une fonction `debut : regexp -> char list` qui retourne $D(\mu(e))$ en fonction de e. Écrire de même une fonction `fin : regexp -> char list` qui retourne $F(\mu(e))$.

Question 10

Écrire une fonction

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

calculant le produit cartésien de deux listes.

Question 11

Écrire une fonction `facteurs2 : regexp -> (char * char) list` retournant $T(\mu(e))$ en fonction de e. $T(\mu(e))$ sera représenté à l'aide d'une liste de couple de lettres. Par exemple si $('a','b')$ apparaît dans le résultat cela signifie qu'au moins un mot de $\mu(e)$ admet pour facteur ab.

5 Générer tous les mots du langage dénoté

On souhaite écrire une fonction `langage : regexp -> int -> char list -> string list` retournant la liste des mots de $\mu(e)$ de longueur n, dont les lettres appartiennent à un alphabet fixé.

Question 12

1. Réfléchir à un algorithme pour générer ces mots.
2. Réfléchir au problème des doublons et de leur élimination.
3. Implémenter votre solution