TP 7 - Langages, expressions régulières et automates -

Exercice 1. Expression régulières binaires... ou pas

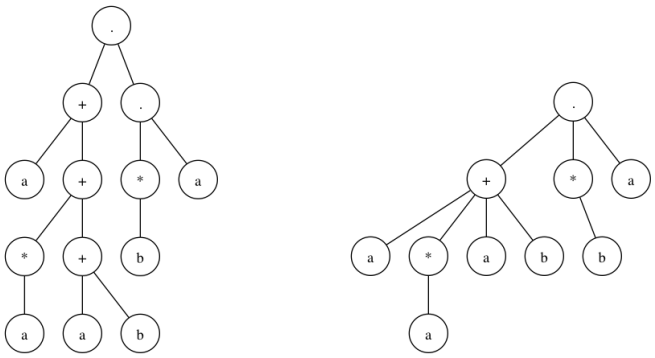
Dans cet exercice, on définit les expressions régulières utilisant des opérateurs binaires de la façon suivante :

```
type bexp =
  Epsilon
  | Char of char
  | Union of bexp * bexp
  | Concat of bexp * bexp
  | Star of bexp
```

1. Écrire une fonction `string_of_bexp` qui permet d'obtenir l'écriture parenthésée d'une expression régulière binaire.
2. Écrire une fonction `gen_bexp` qui prend en paramètre une taille n et permet d'engendrer une expression régulière aléatoire sur l'alphabet $\{a, b\}$, sans ε et comportant n symboles. La génération se fait récursivement. Les lettres sont tirées à pile ou face et pour $n > 2$ on choisit une des 3 opérations uniformément au hasard. Pour les opération binaires, on tire également les tailles des sous-expressions au hasard.

   ```
   # let bea=gen_bexp 10;;
   val bea : bexp =
     Union (Concat (Char 'b', Star (Char 'b')),
      Star (Union (Char 'b', Star (Char 'a'))))
   # string_of_bexp bea;;
   - : string = "((bb*)+(b+a*)*)"
   ```

3. Définir un type `regexp` pour des expressions régulières générales telles que la concaténation et l'union ne sont plus nécessairement binaires.
4. Écrire une fonction `regexp_of_bexp` qui permet de transformer une expression régulière binaire en expression régulière générale. Par exemple, l'arbre binaire à gauche devient l'arbre général à droite.



5. Écrire une fonction `string_of_regexp` qui permet d'obtenir l'écriture parenthésée d'une expression régulière générale.

Exercice 2. Reconnaissance d'un mot par un automate

Dans cet exercice, on utilisera le type suivant pour définir un automate dont les états sont numérotés de 1 à n et tel que 1 est l'état initial et n est l'état final. Le type `automaton` correspond au couple (n, liste des transitions).

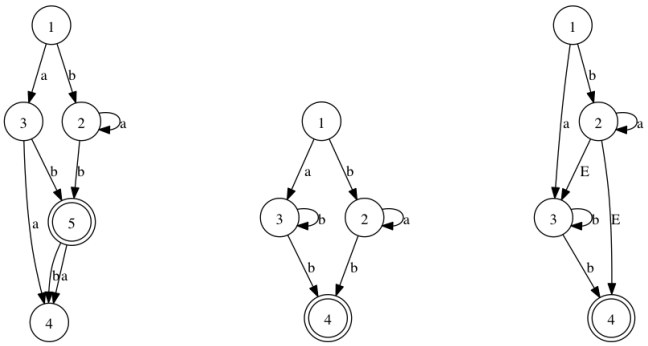
```
type terminal = Epsilon | Char of char ;;
type automaton = int * (int * int * terminal) list;;
```

Voici quelques exemples d'automates :

```
let my_dfa = 5,[1,3,Char 'a'; 1,2,Char 'b'; 3,5,Char 'b'; 2,2,Char 'a';
                2,5, Char 'b'; 3,4, Char 'a'; 5,4, Char 'a'; 5,4, Char 'b'];;

let my_ndfa = 4,[1,3,Char 'a'; 1,2,Char 'b'; 3,4,Char 'b'; 2,2,Char 'a';
                 3,3,Char 'b'; 2,4, Char 'b'];;

let my_ndfa2 = 4,[1,3,Char 'a'; 1,2,Char 'b'; 3,4,Char 'b'; 2,2,Char 'a';
                  3,3,Char 'b'; 2,3, Epsilon; 2,4, Epsilon];;
```



1. Écrire une fonction `list_of_string` qui prend en paramètre une chaîne de caractères et la transforme en liste de caractères.
2. Écrire une fonction `find_transition` qui prend en paramètre un état s, un terminal t et un automate déterministe complet et renvoie l'état dans lequel on arrive en lisant t depuis l'état s.
3. Écrire une fonction `recognize_dfa` qui prend en paramètre un mot w et un automate déterministe complet et teste si le mot w est reconnut par l'automate.
4. Écrire une fonction `find_transitions` qui prend en paramètre un état s, un terminal t et un automate non déterministe (sans ε-transitions) et renvoie la liste des états dans lesquels on arrive en lisant t depuis l'état s.
5. Écrire une fonction `recognize` qui prend en paramètre un mot w et un automate non déterministe (sans ε-transitions) et teste si le mot w est reconnut par l'automate.
6. ★ Modifier `find_transitions` et `recognize` pour qu'elles fonctionnent sur des automates comportant des ε-transitions.

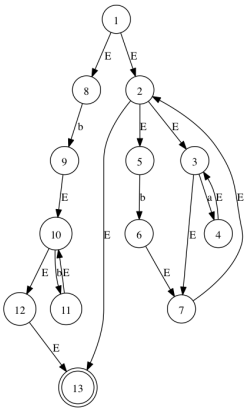
```
# recognize_dfa "aa" my_dfa;;
- : bool = false
# recognize_dfa "ab" my_dfa;;
- : bool = true
# recognize "ba" my_ndfa2;;
- : bool = true
```

Exercice 3. De l'expression régulière à l'automate : l'algorithme de Thompson

1. Écrire une fonction `change_state` qui prend en paramètre un état s, une liste de transitions et un entier n et transforme l'état s en n dans toutes les transitions.
2. Écrire une fonction `shift_automaton` qui prend en paramètre un automate et un entier n et décale tous les états de l'automate de n.
3. Écrire une fonction `automaton_of_bexp` qui prend en paramètre une expression régulière binaire et la transforme en automate en utilisant l'algorithme de Thompson.

   Rappel : `http://www.lsv.ens-cachan.fr/~schmitz/teach/2012_agreg/notes-r62.pdf`.

```
# let toto = automaton_of_bexp bea;;
val toto : automaton =
  (13,
   [(1, 8, Epsilon); (12, 13, Epsilon); (8, 9, Char 'b'); (9, 10, Epsilon);
    (10, 12, Epsilon); (11, 10, Epsilon); (10, 11, Char 'b');
    (1, 2, Epsilon); (2, 13, Epsilon); (7, 2, Epsilon); (2, 5, Epsilon);
    (6, 7, Epsilon); (5, 6, Char 'b'); (2, 3, Epsilon); (3, 7, Epsilon);
    (4, 3, Epsilon); (3, 4, Char 'a')])
```



Pour afficher les automates, vous pouvez utiliser le code suivant qui produit un fichier `toto.dot` que vous pouvez convertir en pdf avec la commande : `dot -Tpdf toto.dot > toto.pdf`

```
let rec dottify a =
  let rec aux = function
    []  ->  ""
    | (s1,s2,Epsilon)::q -> (string_of_int s1) ^ " -> " ^ (string_of_int s2)
                            ^  "[label = \"E\"]; "  ^(aux q)
    | (s1,s2,Char c)::q ->(string_of_int s1) ^ " -> " ^ (string_of_int s2)
                            ^  "[label = " ^ (Char.escaped c) ^ "]; "  ^(aux q)
   in "digraph G { node [shape = circle] ; " ^ aux (snd a) ^ (string_of_int (fst a))
        ^ " [shape = doublecircle];} ";;

let dotfile a =
  let s = dottify a in
    let oc = open_out "toto.dot" in
      let v = Printf.fprintf oc "%s\n" s in close_out oc;;
```

Exercice 4. Pour aller plus loin

1. Passer de l'expression régulière à l'automate en utilisant l'algorithme de Glushkov.
2. Passer de l'automate à l'expression régulière (méthode des équations de langages).