TD2 - Langages réguliers (éléments de réponses)

Exercice 1

Question 1. b^*ab^*

Question 2. $(b|ab)^*$

Question 3. $(a|c)^*(b(b|c)^*|\varepsilon)$. On peut lire simplement l'expression de gauche à droite pour comprendre son fonctionnement. Les lettres se trouvant avant le premier b sont une répétition arbitraire de a ou c. Ensuite, soit b n'est pas présent du tout (cas géré par $|\varepsilon$) auquel cas l'expression se termine. Soit un b est présent. Dans ce cas, les lettres qui le suivent ne peuvent pas être un a.

Question 4. $(^*b|c)(\varepsilon|a((b|c)(b|c)(b|c)a)^*(b|c)^*)$. L'expression commence par une suite arbitraire de b ou c. Elle peut se terminer par ε (cas où il n'y a pas de a dans la séquence). S'il y a un a, soit il n'y a plus de a ensuite, le mot se termine par une suite de b ou c. Soit il y a un a, et donc se dernier est séparé de trois caractères. Ce même a s'il est lui même suivi d'un a doit être aussi séparé de trois caractères.

Exercice 2

L'expression régulière suivante convient : $(a+b)((a+b+0+1+\_)^*(a+b+0+1)+\varepsilon)$.

Exercice 3

Pas difficile.

Exercice 4

Par construction, l'expression $a^*ba^*ba^*$ est régulière et décrit le langage.

Exercice 5

- $(\varepsilon|\Sigma)\,(\varepsilon|\Sigma)$ dénote l'ensemble des mots ayant au plus deux lettres.
- $\left(\Sigma^2\right)^*$ dénote le langage des mots ayant un nombre pair de lettres.
- $(b|ab)^*(a|\varepsilon)$ dénote le langage des mots dans lesquels n'existe pas deux a consécutifs.
- $(ab^*a|b)^*$ dénote le langage des mots qui contiennent un nombre pair de a.

Exercice 6

Question 1.

$$(a^2|b^2)^*$$

Question 2. Le langage dénoté par l'expression régulière $(a|b)^*b$ est l'ensemble des mots construits sur l'alphabet $\{a,b\}$ se terminant par b. Σ^* est l'ensemble de tous les mots qu'il est possible de constuire sur Σ. C'est un langage qui peut être dénoté par l'expression régulière $(a|b)^*$, expression qui peut se mettre sous la forme $\varepsilon|(a|b)^+$ ou encore $\varepsilon|(a|b)^*(a|b) = \varepsilon|(a|b)^*a|(a|b)^*b$. L'expression régulière recherchée prend la forme $\varepsilon|(a|b)^*a$.

Exercice 7

Soit deux entiers n et m tels que $n+m \equiv 0 \mod 2$. Deux situations sont possibles pour obtenir une telle somme.

- Soit n et m sont tous les deux pairs. Alors il existe p et q tels que $n=2p$ et $m=2q$. Puis :

  $$a^n b^m \equiv a^{2p}b^{2q}$$

  Par associativité de l'opérateur de concaténation, on a :

  $$a^{2p} = (a^2)^p \qquad b^{2q} = (b^2)^q$$

  Ainsi :

  $$a^n b^m \equiv (a^2)^p (b^2)^q$$

  Ce résultat étant vrai pour tous les couples (n,m) formés d'entiers tous les deux pairs, l'expression régulière $(a^2)^*(b^2)^*$ appartient au langage L.

- Soit n et m sont tous les deux impairs. Alors il existe p et q tels que $n = 2p + 1$ et $m = 2q + 1$. Puis :

$$a^n b^m \equiv a^{2p+1} b^{2q+1}$$

Ainsi :

$$a^n b^m \equiv (a^2)^p ab (b^2)^q$$

Ce résultat étant vrai pour tous les couples (n, m) formés d'entiers tous les deux impairs, l'expression régulière $(a^2)^* ab (b^2)^*$ appartient au langage L.

L est finalement le langage dénoté par une expression régulière de la forme $(a^2)^*(b^2)^*|(a^2)^* ab (b^2)^*$, expression qui peut se mettre sous la forme $(a^2)^*(\varepsilon|ab)(b^2)^*$. L est donc un langage régulier.

Exercice 8

La preuve se fait en deux temps.

1. Dans un premier temps montrons que pour tout mot $w = a_1 \dots a_n$ de n lettres le langage singleton $\{w\}$ est régulier. On montre cela par récurrence sur $n = |w|$.

   cas de base $n = 0$: $w = \varepsilon$, $\{\varepsilon\}$ est régulier, car $\{\varepsilon\} = \varnothing^*$.

   cas de base $n = 1$: $w = a_1$, $\{a_1\}$ est régulier par définition.

   cas $n > 1$: $w = a_1 \dots a_{n-1} a_n$. Par hypothèse de récurrence, le langage $\{a_1 \dots a_{n-1}\}$ est régulier. De plus le langage singleton $\{a_n\}$ est aussi régulier par définition. Leur concaténation $\{a_1 \dots a_{n-1}\} \cdot \{a_n\} = \{w\}$ est donc aussi un langage régulier.

2. Dans un second temps, nous pouvons maintenant montrer que tout langage L fini est régulier, par récurrence sur $n = |L|$.

   cas de base $n = 0$: $L = \varnothing$ est régulier par définition.

   cas de base $n = 1$: $L = \{w\}$ est régulier par la propriété 1 précedemment montrée.

   cas $n > 1$: $L = \{w_1, \dots, w_n\} = \{w_1\} \cup \{w_2, \dots, w_n\}$. Par hypothèse de récurence, $\{w_2, \dots, w_n\}$ est régulier et $\{w_1\}$ est régulier par la propriété 1. Donc leur union L est un langage régulier.

Exercice 9

Question 1. La fonction `derivative : re -> char -> re` ne pose pas de difficulté.

```
let rec derivative re c =
  match re with
  | Empty -> Empty
  | Epsilon -> Empty
  | Char c0 -> if c = c0 then Epsilon else Empty
  | Alt (re1, re2) -> Alt (derivative re1 c, derivative re2 c)
  | Concat (re1, re2) ->
      let cont = Concat (derivative re1 c, re2) in
      if has_epsilon re1
      then Alt (cont, derivative re2 c)
      else cont
  | Star re0 -> Concat (derivative re0 c, re)
```

Question 2. La fonction `bmatch : re -> string -> bool` ne fait qu'itérer la fonction `derivative` sur chaque caractère de la chaîne.

```
let bmatch re s =
  let rec loop i n re =
    if i >= n
    then has_epsilon re
    else loop (i+1) n (derivative re s.[i])
  in
  loop 0 (String.length s) re
```

Question 3. L'implémentation naïve de notre fonction crée des dérivées contenant plein de sous-expressions redondantes. Par exemple, pour l'expression $(a|a^*)^* b$, si on lit un a, l'expression dérivée est :

$$(\varepsilon|\varepsilon a^*)(a|a^*)^* b|\varnothing$$

De façon générale les cas `Concat` et `Star` peuvent dupliquer une sous-expression, menant à un comportement exponentiel de l'algorithme.

Question 4. Il existe plusieurs façon de rendre l'algorithme plus efficace en pratique. On peut utiliser des identités algébriques (par exemple $r|\varnothing = r$ ou $\varepsilon r = r$) pour éviter de construire des expressions inutiles. Utiliser une table de mémoïsation est également une optimisation qui ne crée qu'une seule copie de chaque sous-expression. Cela permet ensuite de tester en temps constant l'égalité de deux expressions régulières et ouvre ainsi la porte à d'autres optimisations. Les deux optimisations peuvent êtres implémentées par des *smart constructors* qui évitent au programmeur de construire des expressions redondantes.

```
(* définition des smart constructors *)
let memo_table = Hashtbl.create 16

let memo re =
  try Hashtbl.find memo_table re
  with Not_found ->
    Hashtbl.add memo_table re re;
    re

let empty = Empty
let epsilon = Epsilon

let char c = memo (Char c)

let concat re1 re2 =
  match (re1, re2) with
  | Empty, _ | _, Empty -> Empty
  | Epsilon, e | e, Epsilon -> e
  | Star rre1, Star rre2 ->
      if rre1 == rre2 then re1 else memo (Concat (re1, re2))
  | _ -> memo (Concat (re1, re2))

let star = function
  | Empty | Epsilon -> Epsilon
  | Star _ as e -> e
  | e -> memo (Star e)

let alt re1 re2 =
  match (re1, re2) with
  | Empty, e | e, Empty -> e
  | Epsilon, e | e, Epsilon ->
      if has_epsilon e then e else memo (Alt (e, Epsilon))
  | _ when re1 == re2 -> re1
  | _ -> memo (Alt (re1, re2))

(* nouvelle version de derivate avec smart constructors *)
let rec derivative re c =
  match re with
  | Empty -> empty
  | Epsilon -> empty
  | Char c0 -> if c = c0 then epsilon else empty
  | Alt (re1, re2) -> alt (derivative re1 c) (derivative re2 c)
  | Concat (re1, re2) ->
      let cont = concat (derivative re1 c) re2 in
      if has_epsilon re1
      then alt cont (derivative re2 c)
      else cont
  | Star re0 -> concat (derivative re0 c) re
```