1. Expressions régulières

Étant donné l'alphabet $\Sigma = \{0, 1\}$.

1. Décrire chacun des langages suivants par une expression régulière.
   (a) $\{w \in \Sigma^* \mid \#0(w)$ est pair $\}$
   (b) $\{w \in \Sigma^* \mid \#1(w)$ est impair $\}$
   (c) $\{w \in \Sigma^* \mid \#0(w)$ est pair $\vee \#1(w)$ est impair $\}$
   (d) $\{w \in \Sigma^* \mid \forall x, y \in \Sigma^* : w \neq x101y\}$
   (e) $\{w \in \Sigma^* \mid \#0(w) = \#1(w) \wedge \forall w'$ si $w' \ll_p w$ alors $abs(\#0(w') - \#1(w')) < 2\}$
2. Décrire en langage naturel la propriété des chaînes des langages décrits par les expressions régulières suivantes.
   (a) $(\epsilon + 1)(00^*1)^*0^*$
   (b) $(0^*1^*)^*000(0 + 1)^*$

2. Preuves sur les expressions régulières

Afin de vous convaincre que l'expression regulière que vous avez trouvée à l'exercice précédent au point 1.1(a) est correcte, démontrez qu'elle dénote exactement le langage qu'elle est supposée décrire. Vous aurez besoin de cette définition.

Définition 2.1 *Soit $a \in \Sigma$, la fonction $\#a : \Sigma^* \to \mathbb{N}$ est définie par*

$$\begin{aligned} \#a(\epsilon) &= 0 \\ \#a(a) &= 1 \\ \#a(b) &= 0 && \text{if } a \neq b \\ \#a(w \cdot w') &= \#a(w) + \#a(w') \end{aligned}$$

□

Pour démontrer que votre expression régulière est correcte, vous devez d'abord montrer que toute chaîne w appartenant au langage dénoté par votre expression statisfait la propriété $\#0(w)$ pair. Puis que toute chaîne w telle que $\#0(w)$ est pair appartient au langage dénoté par votre expression.

La première partie se fait en raisonnant sur la structure des chaînes du langage. La seconde par induction sur les entiers et sans doute avec l'aide des lemmes suivant.

Lemme 2.2 *Pour tout $a \in \Sigma$, et tout $w \in \Sigma^*$ tel que $\#a(w) = 0$ nous avons $w \in (\Sigma \setminus \{a\})^*$.* □

Lemme 2.3 *Pour tout $a \in \Sigma$, et tout $w \in \Sigma^*$ tel que $\#a(w) > 1$ nous avons*

$$\exists x, y \in \Sigma^* : w = xay \wedge \#a(x) = \#a(w) - 1$$

□

3. Langages à fermeture finie

Démontrer la proposition suivante.

Proposition 3.4 *Pour tout alphabet Σ, il n'existe que deux langages différents L_1 et L_2 sur Σ dont la fermeture L_1^* et L_2^* est finie.* □

Trouvez d'abord ces deux langages (indice : ils sont indépendants de l'alphabet) et démontrez que leur fermeture est finie. Ensuite démontrez par l'absurde que ce sont les seuls dont la fermeture est finie. Pour ce dernier point vous aurez peut être besoin du lemme suivant,

Lemme 3.5 *Soit deux ensemble A et B finis tels que $A \subseteq B$, alors $\|A\| \leq \|B\|$.* □

4. Traitement de données textuelles avec les expressions régulières

Ceci n'est pas à proprement parler un exercice. Nous souhaitons juste vous montrer que les théories que nous développons dans ce cours ont des applications bien concrètes. En effet les expressions régulières offrent un outil très puissant et efficace (via traduction en AFD) pour traiter des données textuelles. De nombreux outils UNIX et bibliothèques de programmation utilisent les expressions regulières pour reconnaître, extraire ou manipuler du texte.

L'utilitaire UNIX `grep`[1] est un programme qui permet de trouver les lignes d'un fichier dont une *partie* est filtrée par une expression régulière donnée et d'imprimer ces lignes sur le flot de sortie standard. A l'aide de ce programme nous allons extraire des informations sur les entêtes de courriers électroniques non sollicités. De tels messages, stockés sequentiellement dans des fichiers textes, sont disponibles sur le site `www.spamarchive.org`. Téléchargez l'un de ces fichiers[2] et décompressez-le en utilisant l'utilitaire `gunzip`.

Il existe une kyrielle de syntaxes différentes pour les expressions régulières selon l'outil ou la bibliothèque utilisée. Dans ce qui suit, nous allons utiliser la syntaxe connue sous le nom de « extended grep » associée à l'outil `egrep`. Un sous-ensemble de cette syntaxe est donnée ci-dessous, r dénote une expression régulière.

`a`	filtre le caractère donné.
`.`	filtre n'importe quel caractère.
`[ ]`	filtre un des caractères donné entre les crochet.
`r|r'`	filtre soit l'expression r, soit r'.
r`?`	filtre l'expression r zéro ou une fois exactement.
r`*`	filtre l'expression r zéro ou un nombre arbitraire de fois.
r`{n,m}`	filtre l'expression r au moins `n` fois mais pas plus de `m` fois.

La concatenation est implicite, l'expression "`ab`" filtre la chaîne "`ab`". Pour les ensembles de caractères il est possible de spécifier des intervalles, ainsi par exemple "`[0-9a-z]`" filtre un chiffre ou un caractère minuscule. Les parenthèses peuvent être utilisée pour délimiter les expressions. Pour filtrer l'un des meta-caractères donnés ci-dessus tel que "`.`" ou "`*`" il faut « l'échapper » c'est à dire le faire précéder d'un "`\`". Par exemple "`\.`" filtrera un point et non pas n'importe quel caractère.

Ouvrez un terminal sur votre station UNIX et rendez-vous dans le répertoire qui contient le fichier téléchargé et décompressé `105.r2`. Tapez :

```
> egrep "Subject:" 105.r2
```

[1] Au cas ou `grep` n'est pas installé sur votre station UNIX (c'est peu probable), il existe une implémentation libre disponible à l'adresse `http://www.gnu.org/software/grep/grep.html`.

[2] Par exemple le fichier `ftp://spamarchive.org/pub/archives/submit/105.r2.gz`

Ceci va nous renvoyer sur le flot de sortie toutes les lignes qui contiennent la chaîne "`Subject:`", c'est à dire tous les sujets des messages (sous l'hypothèse que cette chaîne particulière n'apparait pas ailleurs, comme par exemple dans le corps d'un message). Cherchons maintenant les messages subliminaux du capitalisme outrancier,

```
> egrep "Subject:.*( rich| money ).*" 105.r2
```

Notez l'utilisation des espaces, "richer", "richest" ou "richerblabla" seront aussi filtrés, mais pas "moneyblabla".

Essayez de filtrer :

1. Le nom des programmes utilisés pour envoyer les messages (entête "`X-Mailer:`").
2. Le sujet des messages *et* l'expéditeur des messages (entête "`From:`").
3. Les lignes avec des adresses électroniques qui ne contiennent que des caractères alphanumériques (caractères de l'alphabet et chiffres) et qui se terminent en "`.com`".
4. Les lignes contenant des adresses IP sous forme textuelle (ex. 128.0.0.1).
5. Les lignes contenant une heure au format `HH:MM:SS`.

Si vous rencontrez des difficultés consultez le manuel de `egrep` (`man egrep`). Pour plus d'informations sur la structure des données des messages contenus dans le fichier texte, consultez la RFC internet 2822[3].

Notez que grep est très limité dans ses possibilités de transformation, il ne fait que sélectionner les lignes d'un fichier. Pour des transformations plus complexe sur les données filtrées — par exemple pour filtrer les adresses IP uniquement et non pas les lignes contenant de telles adresses — il faut utiliser des utilitaires tels que `sed` ou `awk` ou encore développer son propre programme dans un langage de programmation en utilisant une bibliothèque d'expression régulières.

[3] `http://www.ietf.org/rfc/rfc2822.txt`