Informatique théorique III
Série 4 solution

Prof. Nestmann, 2004

1. Expressions régulières

Soit l'alphabet $\Sigma = \{a, b\}$.

1. Décrire chacun des langages suivants par une expression régulière.
   (a) $\{w \in \Sigma^* \mid |w|_a \text{ est pair } \}$
   (b) $\{w \in \Sigma^* \mid |w|_b \text{ est impair } \}$
   (c) $\{w \in \Sigma^* \mid |w|_a \text{ est pair } \vee |w|_b \text{ est impair } \}$
   (d) $\{w \in \Sigma^* \mid \forall x, y \in \Sigma^* : w \neq xbaby \}$
   (e) $\{w \in \Sigma^* \mid |w|_a = |w|_b \wedge \big(\forall u, v \in \Sigma^* : w = uv \Rightarrow abs(|u|_a - |u|_b) < 2\big)\}$
2. Décrire en langage naturel la propriété des mots des langages décrits par les expressions régulières suivantes.
   (a) $(\epsilon + b)(aa^*b)^*a^*$
   (b) $(a^*b^*)^*aaa(a + b)^*$

Solution.

1. Voici quelques exemples d'expressions régulières valides et une explication informelle,
   (a) $b^*(ab^*ab^*)^*$
   La deuxième partie de cette expression régulière s'assure que les a apparaissent par deux dans les mots. Ces derniers peuvent être séparées par un nombre arbitraire de b. Notons que ϵ est reconnu, de même que les mots constitués de b uniquement car zéro est un nombre pair[1].
   (b) $(a^*ba^*)(ba^*ba^*)^*$
   La première partie de l'expression régulière reconnaît les mots constitués d'un nombre arbitraire de a et d'un unique b. Il reste donc, dans la deuxième partie à reconnaître un nombre pair de a, potentiellement séparés par un nombre arbitraires de b, et le total sera impair. Remarquez que le mot ϵ n'est pas reconnue car $|\epsilon|_b = 0$ qui est pair.
   (c) $\big(b^*(ab^*ab^*)^*\big) + \big((a^*ba^*)(ba^*ba^*)^*\big)$
   Ce langage est l'union des deux précédents. L'opérateur $+$ des expressions regulières représente justement l'union de deux langages.
   (d) $a^*(b + baaa^*)^*a^*$
   Dans cette expression régulière, nous nous assurons que lorsqu'un b est suivi d'un a, soit il existe au moins un autre a, soit nous avons atteint la fin de le mot. Le sous-mot bab n'apparaîtra alors pas.
   (e) $(ab + ba)^*$
   La répétition des ab et ba assure que le nombre de a et de b et égal. Elle assure aussi qu'il n'y aura jamais trois a ou b successifs. C'est cette dernière propriété qui permet de démontrer que la différence de a et de b pour les préfixes d'un mot reconnu est toujours plus petite que 2.
2. Les propriétés des mots reconnus par ces expressions régulières sont :
   (a) $(\epsilon + b)(aa^*b)^*a^*$
   Les mots reconnus par cette expressions n'ont pas de b adjacents. Une formulation équivalente et plus simple de cette propriété est $(a + ba)^*(\epsilon + b)$.

[1] Rappel : m pair $\Leftrightarrow \exists n \in \mathbb{N} : m = 2n$

(b) $(a^*b^*)^*aaa(a+b)^*$
Les expressions $(a^*b^*)^*$ et $(a+b)^*$ ne reconnaissant rien d'autre que tous les mots sur l'alphabet Σ, cette expression régulière reconnaît donc touts les mots qui ont aaa comme sous-mot.

■

2. Preuves sur les expressions régulières

Soit $\Sigma \triangleq \{a, b\}$ et e l'expression régulière que vous avez trouvé dans l'exercice précédent, question 1b. Montrer que $\mathrm{L}(e) = L$ où $L \triangleq \{w \in \Sigma^* \mid |w|_b \text{ est impair }\}$.
Indication : Pour montrer que $L \subseteq \mathrm{L}(e)$, on pourra notamment utiliser le fait que $L = \bigcup_{k\in\mathbb{N}} L_k$ où $L_k = \{w \in \Sigma^* \mid |w|_b = 2*k+1\}$ et montrer par récurrence sur $k \in \mathbb{N}$ que $\forall k : L_k \subseteq \mathrm{L}(e)$.

Solution. Posons $L \triangleq \{w \in \Sigma^* \mid |w|_b \text{ est impair }\}$.
On a trouvé $e = (a^*ba^*)(ba^*ba^*)^*$.
On montre alors les deux inclusions :

- $\mathrm{L}(e) \subseteq L$:
  Par définition $\mathrm{L}(e) = \mathrm{L}(a^*ba^*) \cdot \mathrm{L}(ba^*ba^*)^* = \mathrm{L}(a^*ba^*) \cdot \bigcup_{n\in\mathbb{N}} \mathrm{L}(ba^*ba^*)^n$.
  Soit $w \in \mathrm{L}(e)$. Il existe alors $u \in \mathrm{L}(a^*ba^*)$, $n \in \mathbb{N}$ et $v_1, \ldots, v_n \in \mathrm{L}(ba^*ba^*)$ tels que $w = uv_1 \cdot \ldots \cdot v_n$. Par ailleurs, il est clair que $|w|_b = |u|_b + \sum_{1\leq i\leq n} |v_i|_b$.
  On montre les deux propositions suivantes :
  1. $\forall u \in \mathrm{L}(a^*ba^*) : |u|_b = 1$
     En effet, par définition, $\mathrm{L}(a^*ba^*) = \mathrm{L}(a)^* \, \mathrm{L}(b) \, \mathrm{L}(a)^* = \{a\}^* \cdot \{b\} \cdot \{a\}^*$.
     Soit $u \in \mathrm{L}(a^*ba^*)$. Il existe alors $i, j \in \mathbb{N}$ tel que $u = a^iba^j$.
     On a alors $|u|_b = i * |a|_b + 1 + j * |a|_b = 0 + 1 + 0 = 1$.
     D'où le résultat.
  2. $\forall u \in \mathrm{L}(ba^*ba^*) : |u|_b = 2$
     En effet,par définition, $\mathrm{L}(ba^*ba^*) = \{b\} \, \{a\}^* \, \{b\} \, \{a\}^*$.
     Soit $u \in \mathrm{L}(ba^*ba^*)$. Il existe alors $i, j \in \mathbb{N}$ tel que $u = ba^iba^j$.
     On a alors $|u|_b = 1 + i * |a|_b + 1 + j * |a|_b = 1 + 0 + 1 + 0 = 2$.
     D'où le résultat.

  En revenant au problème qui nous préoccupe, on a donc $|w|_b = 1 + \sum_{1\leq i} 2 = 2*i+1$ et donc $|w|_b$ est impair. On a démontré que $w \in L$.
  D'où l'inclusion $\mathrm{L}(e) \subseteq L$.
- $L \subseteq \mathrm{L}(e)$:
  Remarquons d'abord que $L = \bigcup_{k\in\mathbb{N}} \{w \in \Sigma^* \mid |w|_b = 2*k+1\}$.
  On montre donc par récurrence sur $k \in \mathbb{N}$ que $\forall k \in \mathbb{N} : P(k)$ où
  $$P(k) \triangleq \{w \in \Sigma^* \mid |w|_b = 2*k+1\} \subseteq \mathrm{L}(e)$$
  - cas de base : $P(0)$.
    Soit $w \in \Sigma^*$ tel que $|w|_b = 2*0+1 = 1$ et montrons que $w \in \mathrm{L}(e)$.
    Comme w contient exactement un b et que $w \in \Sigma^*$, il est clair qu'il existe $i, j \in \mathbb{N}$ tel que $w = a^iba^j$.
    Donc $w \in \mathrm{L}(a^*ba^*)$.
    Comme $\epsilon \in \mathrm{L}((ba^*ba^*)^*)$, on a donc $w = w \cdot \epsilon \in \mathrm{L}(a^*ba*) \cdot \mathrm{L}((ba^*ba^*)^*) = \mathrm{L}(e)$.
    D'où $P(0)$.

- cas inductif : $P(k) \Rightarrow P(k+1)$.
  Soit $k \in \mathbb{N}$ et supposons que $P(k)$ soit vrai. Montrons $P(k+1)$.
  Soit $w \in \Sigma^*$ tel que $|w|_b = 2 * (k+1) + 1 = 2 * k + 1 + 2$ et montrons que $w \in \mathrm{L}(e)$.
  Comme w contient au moins deux b, prenons v le plus petit suffixe de w contenant exactement deux b. Il existe alors $u \in \Sigma^*$ tel que $w = u \cdot v$.
  Comme $|w|_b = |u|_b + |v|_b = 2 * k + 3 = |u|_b + 2$, on a $|u|_b = 2 * k + 1$. Par hypothèse de récurrence, on conclut que $u \in \mathrm{L}(e)$.
  De plus, comme v est le plus petit suffixe de w contenant exactement deux b, il est clair que v commence par un b (sinon v s'écrit $a \cdot v'$ et v' est un suffixe de w plus petit que v contenant exactement deux b). Il est donc clair qu'il existe $i, j \in \mathbb{N}$ tel que $v = ba^iba^j$. Donc $v \in \mathrm{L}(ba^*ba^*)$.
  Ainsi $w = u \cdot v \in \mathrm{L}(e) \cdot \mathrm{L}(ba^*ba^*)$.
  Comme $\mathrm{L}(e) = \mathrm{L}(a^*ba^*)\,\mathrm{L}(ba^*ba^*)^*$,
  on a $\mathrm{L}(e) \cdot \mathrm{L}(ba^*ba^*) = \mathrm{L}(a^*ba^*)\,\mathrm{L}(ba^*ba^*)^*\,\mathrm{L}(ba^*ba^*) \subset \mathrm{L}(a^*ba^*)\,\mathrm{L}(ba^*ba^*)^*$
  (car si x est une expression régulière, $\mathrm{L}(x)^*\,\mathrm{L}(x) \subseteq \mathrm{L}(x)^*$).
  C'est-à-dire, $\mathrm{L}(e) \cdot \mathrm{L}(ba^*ba^*) \subseteq \mathrm{L}(e)$.
  Ainsi $w \in \mathrm{L}(e)$.
  D'où $P(k+1)$.

Par théorème de récurrence, on conclut que $\forall k \in \mathbb{N}^* : P(k)$.

Soit maintenant $u \in L$. Par définition de L, il existe $k \in \mathbb{N}$ tel que $|u|_b = 2*k+1$. On a donc $u \in \{w \in \Sigma^* \mid |w|_b = 2 * k + 1\}$. Comme $P(k)$ est vrai, on en conclut que $u \in \mathrm{L}(e)$.
D'où l'inclusion $L \subseteq \mathrm{L}(e)$.

On en conclut que $L = \mathrm{L}(e)$. ■

3. Langages à fermeture finie

Démontrer la proposition suivante.

Proposition 3.1 *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.* □

Pour cela,

1. Trouvez ces deux langages L_1 et L_2 (indice : ils sont indépendants de l'alphabet).
2. Démontrez les deux proposition suivantes.

$$\forall L : L \subseteq \Sigma^* \Rightarrow L\emptyset = \emptyset L = \emptyset$$
$$\forall L : L \subseteq \Sigma^* \Rightarrow L\,\{\epsilon\} = \{\epsilon\}\,L = L$$

3. Démontrez, à l'aide des deux propositions précédentes, que les fermetures L_1^*, L_2^* de L_1 et L_2 sont finies.
4. Démontrez par l'absurde que ce sont les seuls langages dont la fermeture est finie.

Preuve. Soit Σ un alphabet quelconque.

1. Ces deux langages sont $L_1 = \emptyset$ et $L_2 = \{\epsilon\}$. Par définition de l'ensemble vide nous avons $L_1 \neq L_2$.

2. Ces deux propositions découlent directement de la définition de la concaténation de langages.

   Pour (1) nous avons $L\emptyset = \{uv \mid u \in L \wedge v \in \emptyset\}$. Or par définition de l'ensemble vide, il n'existe pas de v tel $v \in \emptyset$. L'ensemble décrit par la proposition est donc vide (et vice-versa pour $\emptyset L = \emptyset$).

   Pour (2) nous avons $L\{\epsilon\} = \{uv \mid u \in L \wedge v \in \{\epsilon\}\}$. Donc v ne peut que être égal à ϵ et $uv = u$ avec $u \in L$ on a donc $L\{\epsilon\} = L$ (et vice-versa pour $\{\epsilon\} L = L$).

3. (a) $L_1^* = \emptyset^*$ est fini. Nous montrons d'abord par induction naturelle que

   $$\forall n \in \mathbb{N}^* : L_1^n = \emptyset$$

   i. Cas de base. $L_1^1 = L_1 L_1^0 = \emptyset L_1^0 = \emptyset$.

   ii. Induction. $L_1^{n+1} = L_1 L_1^n = \emptyset L_1^n = \emptyset$.

   Nous avons donc, $L_1^* = \bigcup_{n\in\mathbb{N}} L_1^n = L_1^0 \cup \bigcup_{n\geq 1} \emptyset = \{\epsilon\}$ qui est un ensemble fini.

   (b) $L_2^* = \{\epsilon\}^*$ est fini. Nous montrons d'abord par induction naturelle que

   $$\forall n \in \mathbb{N} : L_2^n = \{\epsilon\}$$

   i. Cas de base. $L_2^0 = \{\epsilon\}$.

   ii. Induction. $L_2^{n+1} = L_2 L_2^n = \{\epsilon\} L_2^n = L_2^n \overset{\text{(induct.)}}{=} \{\epsilon\}$.

   Nous avons donc $L_2^* = \bigcup_{n\in\mathbb{N}} L_2^n = \bigcup_{n\in\mathbb{N}} \{\epsilon\} = \{\epsilon\}$ qui est un ensemble fini.

4. Pour tout Σ, si $L \subseteq \Sigma^*$ est différent de L_1 et L_2, alors L^* est infini. Nous considérons les deux cas suivants.

   (a) $\Sigma = \emptyset$. Par définition nous avons $\Sigma^* = \{\epsilon\}$. Dès lors les langages possibles $L \subseteq \Sigma^*$ sur Σ sont exactement L_1 et L_2 et la proposition est trivialement satisfaite.

   (b) $\Sigma \neq \emptyset$. Prenons un langage quelconque $L \subseteq \Sigma^*$ tel que $L \neq L_i$ pour i=1,2. De ce fait il existe au moins un mot $w \in L$ tel que $w \neq \epsilon$. Montrons que L^* est infini.

   Pour cela construisons les ensembles de mots W_m avec $m \in \mathbb{N}$, définis par

   $$W_m = \{w^k \mid 1 \leq k \leq m\}$$

   par construction nous avons $\#(W_m) = m$, W_m est donc fini, et $W_m \subseteq L^*$ (chaque $w^k \in L^k \subseteq L^*$).

   Nous procédons par l'absurde. Supposons$^{(*)}$ que la fermeture de L est finie, cela signifie $\exists n \in \mathbb{N} : \#(L^*) = n$. Soit n' ce nombre, nous avons :

   $$\#(W_{n'+1}) = n' + 1 > n' = \#(L^*)$$

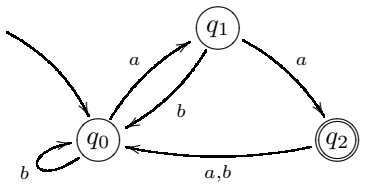
   Cependant étant donné que $W_{n'+1} \subseteq L^*$, nous avons $\#(W_{n'+1}) \leq \#(L^*)$, ce qui contredit l'équation ci-dessus. L'hypothèse $(*)$ est donc fausse, et donc L^* est infini.

   De (a) et (b) nous déduisons 4.

De 1., 2. et 4. nous déduisons la proposition . ■

4. Transformation d'un automate en expression régulière

Soit $\Sigma = \{a, b\}$ et M l'AFD suivant :



On souhaite calculer une expression régulière x telle que $\mathrm{L}(x) = \mathrm{L}(M)$.

1. Résoudre ce problème par la méthode du système d'équations et le lemme d'Arden.
2. Résoudre ce problème en utilisant les AFNG.

Solution.

1. D'après le cours, on a le système d'équations suivant :

$$\begin{cases} q_0 & \eqsim & bq_0 + aq_1 \\ q_1 & \eqsim & bq_0 + aq_2 \\ q_2 & \eqsim & (a+b)q_0 + \boldsymbol{\epsilon} \end{cases}$$

En remplaçant q_1 dans q_0, on a

$$\begin{aligned} q_0 &\eqsim bq_0 + aq_1 \\ &\eqsim bq_0 + a(bq_0 + aq_2) \\ &\eqsim (b+ab)q_0 + aaq_2 \end{aligned}$$

Par le lemme d'Arden, on a donc

$$q_0 \eqsim (b+ab)^* aaq_2$$

En remplaçant dans q_2, on obtient

$$\begin{aligned} q_2 &\eqsim (a+b)q_0 + \boldsymbol{\epsilon} \\ &\eqsim (a+b)(b+ab)^* aaq_2 + \boldsymbol{\epsilon} \end{aligned}$$

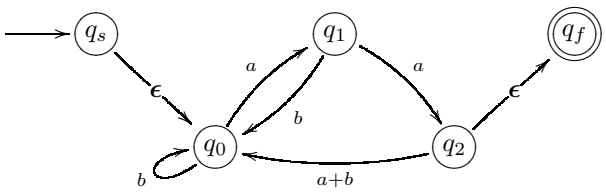
Par le lemme d'Arden, on a donc

$$q_2 \eqsim ((a+b)(b+ab)^* aa)^* \boldsymbol{\epsilon} \eqsim ((a+b)(b+ab)^* aa)^*$$

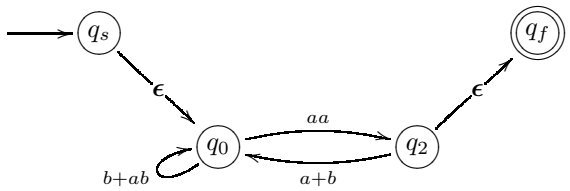
La solution cherchée est donc (par exemple) :

$$x \eqsim (b+ab)^* aa((a+b)(b+ab)^* aa)^*$$

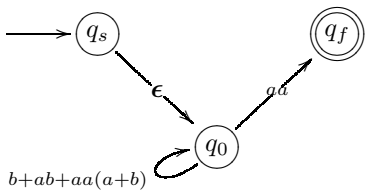
2. On calcule d'abord $\mathrm{G}(M)$ (notez que l'on a omis toutes les transitions étiquetées par $\boldsymbol{\emptyset}$) :



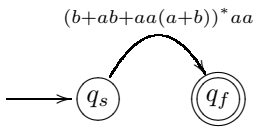
On réduit d'abord par q_1 et on obtient (après simplifications) :



On réduit ensuite par q_2 :



Finalement, en réduisant par q_0 :



La solution cherchée est donc (par exemple) :

$$x \approx (b + ab + aa(a + b))^*aa$$

■

5. Équivalence d'expressions régulières

On considère l'ensemble $\mathbf{RE}_\Sigma$ des expressions régulières sur un alphabet Σ. Montrer que

$$\forall x \in \mathbf{RE}_\Sigma : x^* \approx (xx)^*(\epsilon + x)$$

On pourra utiliser le résultat suivant :

$$\forall x, y \in \mathbf{RE}_\Sigma : x \lesssim y \wedge y \lesssim x \Rightarrow x \approx y$$

Solution. On utilise comme suggéré le résultat donné en montrant d'abord que $x^* \lesssim (xx)^*(\epsilon + x)$ puis que $(xx)^*(\epsilon + x) \lesssim x^*$.

– $x^* \lesssim (xx)^*(\epsilon + x)$:
  On utilise l'implication donnée en cours :

$$\forall x, y, z : y + zx \leq z \Rightarrow yx^* \leq z$$

  en prenant $x = x$, $y = \epsilon$ et $z = (xx)^*(\epsilon + x)$.
  Il nous faut donc montrer que :

$$\epsilon + (xx)^*(\epsilon + x)x \leq (xx)^*(\epsilon + x)$$

On a

$$\begin{aligned}
\epsilon + (xx)^*(\epsilon + x)x &\approx \epsilon + (xx)^*(x + xx) && \text{distributivité de } \cdot \text{ par rapport à } + \\
&\approx \epsilon + (xx)^*x + (xx)^*xx && \text{distributivité de } \cdot \text{ par rapport à } + \\
&\approx \epsilon + (xx)^*xx + (xx)^*x && \text{commutativité de } + \\
&\approx (\epsilon + (xx)^*xx) + (xx)^*x && \text{associativité de } + \\
&\approx (xx)^* + (xx)^*x && \text{car } \forall x : \boldsymbol{\epsilon} + x^*x \approx x^* \text{ et en prenant } x = xx \\
&\approx (xx)^*(\epsilon + x) && \text{distributivité de } \cdot \text{ par rapport à } + \\
&\lesssim (xx)^*(\epsilon + x) && \text{car } \lesssim \text{ est réflexive}
\end{aligned}$$

On en conclut donc que $x^* \lesssim (xx)^*(\epsilon + x)$ en utilisant l'implication ci-dessus.

– $(xx)^*(\epsilon + x) \lesssim x^*$
  On utilise cette fois-ci l'autre implication donnée en cours :

$$\forall x, y, z : y + xz \lesssim z \Rightarrow x^*y \lesssim z$$

  en prenant $x = xx$, $y = \epsilon + x$ et $z = x^*$.
  Il nous faut donc montrer que :

$$\epsilon + x + xxx^* \lesssim x^*$$

  On a

$$\begin{aligned}
\epsilon + x + xxx^* &\approx \epsilon + x(\epsilon + xx^*) && \text{distributivité de } \cdot \text{ par rapport à } + \\
&\approx \epsilon + xx^* && \text{car } \forall x : \boldsymbol{\epsilon} + xx^* \approx x^* \text{ et en prenant } x = x \\
&\approx x^* && \text{car } \forall x : \boldsymbol{\epsilon} + xx^* \approx x^* \text{ et en prenant } x = x \\
&\lesssim x^* && \text{car } \leq \text{ est réflexive}
\end{aligned}$$

  On en conclut que $(xx)^*(\epsilon + x) \lesssim x^*$ en utilisant l'implication ci-dessus.

■

6. 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. 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`[2] est un programme qui permet de trouver les lignes d'un fichier dont une *partie* est filtrée par une expression régulière donnée et des les imprimer 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[3] et décompressez-le en utilisant l'utilitaire `gunzip`.

La syntaxe qui permet de spécifier des expressions régulières change selon la bibliothèque ou l'outil utilisé. Dans ce qui suit, nous allons utiliser la syntaxe connue

[2] 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`.

[3] Par exemple le fichier `ftp://spamarchive.org/pub/archives/submit/401.r2.gz`

sous le nom de « extended grep » associée à l'outil `egrep` (ou `egrep -E`). 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.
$r \mid r'$	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 le mot "`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 `401.r2`. Tapez :

```
> egrep "Subject:" 401.r2
```

L'exécution du programme renvoye sur le flot de sortie toutes les lignes qui contiennent le mot "`Subject:`", c'est à dire tous les sujets des messages (sous l'hypothèse que ce mot particulier n'apparaît pas ailleurs, par exemple dans le corps d'un message). Cherchons maintenant les sirènes du capitalisme outrancier,

```
> egrep "Subject:.*( rich| money ).*" 401.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. Les publicités pour les tubes de dentrifice.
3. Le sujet des messages *et* l'expéditeur des messages (entête "`From:`").
4. 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`".
5. Les lignes contenant des adresses IP sous forme textuelle (ex. 128.0.0.1).
6. 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[4].

Notez que `grep` est très limité dans ses possibilités de transformation, il ne fait que sélectionner les lignes d'un fichier (et ne permet pas, par exemple, de filtrer des motifs qui s'étendent sur plusieurs lignes). Pour des transformations plus complexe sur les données filtrées ou pour un accès plus fin au données — 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. □

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

Solution.

1. `"X-Mailer:"`
2. `"dents blanches"`
3. `"(Subject|From):"`
4. `"[a-zA-Z0-9]*@[a-zA-Z0-9]*\.com"`
5. Une première approximation est donnée par

   `"([0-9]{1,3}\.){3,3}[0-9]{1,3}"`

   Cependant, cette expression regulière reconnaît un sur-ensemble du langage des adresses IP. En effet, les adresses IP valides notées textuellement vont de `0.0.0.0` à `255.255.255.255` tout en incluant — généralement — les mots tels que `000.00.0.0`. Ainsi l'adresse IP invalide `1.999.0.258` est, par exemple, reconnue par l'expression ci-dessus.
   Pour filtrer un nombre de `0` à `255`, tout en acceptant les mots `01`, `041`,... il faut utiliser l'expression régulière suivante,

   $num =$ `([01]?[0-9]?[0-9])|(2[0-4]?[0-9])|(25[0-5])`

   Les adresses IP valides sont donc filtrées en remplaçant num par sa définition dans l'expression suivante :

   `"(`num`)\.){3,3}(`num`)"`

6. Une première approximation est donnée par

   `"([0-9][0-9]:){2,2}[0-9][0-9]"`

   Cependant, cette expression régulière reconnaît un sur-ensemble des heures au format `HH:MM:SS`. En effet, les heures vont de `00` à `23`, les minutes et les secondes de `00` à `59`. En posant,

   $$\begin{aligned} hh &= \texttt{([0-1][0-9])|(2[0-3])} \\ mm &= \texttt{[0-5][0-9]} \end{aligned}$$

   Les heures valides sont filtrées en remplaçant hh et mm par leur définition dans l'expression suivante :

   `"(`hh`):(`mm`):(`mm`)"`

■