Informatique théorique III
Série 4

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) $(a^*b^*)^*aaa(a + b)^*$

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 **??**. 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 $\forall k : L_k \subseteq \mathrm{L}(e)$.

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,

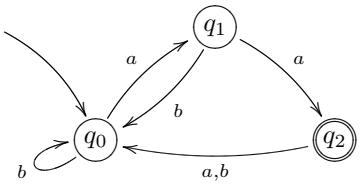
1. Trouvez ces deux langages L_1 et L_2 (indice : ils sont indépendants de l'alphabet).
2. Démontrez les deux proposition suivantes.

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

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.

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.

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 :

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

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`[1] 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[2] 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 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.

[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/401.r2.gz`

`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 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[3].

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. □

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