Université de Lille 1 SCIENCES ET TECHNOLOGIES

TDM 1 - Automates et Langages

Manipulation d'expressions régulières

septembre 2015

1 Introduction

Les expressions rationnelles (encore appelées régulières) peuvent être utilisées dans des outils (comme dans la fonction rechercher et remplacer de nombreux éditeurs de texte) et sont proposées (nativement ou via un bibliohèque) dans la plupart des langages de programmation. En voici quelques exemples (liste loin d'être exhaustive)

Java	API standard	paquetage java.util.regex ainsi, entre autres, que plusieurs méthodes de la classe String
Python	API standard	module re
Javascript	natif	
PHP	API standard	2 bibliothèques : PCRE (syntaxe Perl) ou POSIX
Perl	natif	

La syntaxe des expressions régulières pour les langages et applications diffère quelque peu de la notation « mathématique » vue en cours tout en en reprenant les principes. Par ailleurs elle varie aussi selon les outils ou langages mais les différentes notations possèdent quand même un large socle commun.
Précisons aussi que l'alphabet cible est composé de tous les caractères. C'est donc un alphabet fini mais de cardinalité importante.

2 La syntaxe des expressions régulières

La concaténation est obtenue par juxtaposition des expressions régulières, l'étoile de Kleene utilise bien le caractère `*` et on peut utiliser des parenthèses pour isoler des sous-expressions. Par contre, l'union (le `+` binaire des expressions rationnelles vues en cours) dans les expressions régulières utilise le symbole `|`. Le symbole `+` est consacré à l'opérateur unaire, qui signifie *« une ou plusieurs occurrences de »* (rappelons nous $L^+ = L.L^*$). On dispose surtout d'un certains nombre d'opérateurs supplémentaires non indispensables en théorie mais très utiles en pratique.

Certains caractères ont une signification particulières dans les expressions, on les appelle des *méta-caractères*. En voici la liste :

```
\  {  }  [  ]  ^  $  ?  .  *  +  |  (  )
```

Il permettent de définir des opérateurs dont la signification est donnée dans le tableau ci-dessous.

Expression	Signification
`x`	une occurrence du **caractère** x
`\x`	le **caractère** x en tant que caractère ordinaire (utilisé pour les méta-caractères)
`.`	n'importe quel **caractère**
`e+`	une ou plusieurs occurrences de e
`e*`	zéro ou une ou plusieurs occurences de e
`e?`	une occurrence optionnelle de e
`e{n}`	exactement n occurrences de e
`e{n,m}`	au moins n occurrences et au plus m occurrences de e
`(e)`	parenthésage de e
`e|f`	e ou f
`[xyz]`	n'importe quel caractère figurant entre les crochets (donc le caractère x ou le caractère y ou le caractère y)
`[x-z]`	n'importe quel caractère dans l'intervalle allant de x à z (donc x, y ou z)
`[a-dtux-z]`	n'importe quel caractère dans l'intervalle allant de a à d ou le caractère t ou le caractère u ou n'importe quel caractère dans l'intervalle allant de x à z
`[^x]`	n'importe quel caractere sauf x
`[^x-z]`	n'importe quel caractere sauf x, y ou z
`^x`	un x au début d'une ligne
`x$`	un x à la fin d'une ligne
`\t$`	tabulation
`\n$`	newline
`\s$`	un espace (espace usuel, ou tabulation etc)
`\d$`	un chiffre décimal
(Liste non exhaustive)	

Les opérateurs de plus forte priorité sont `[]` `()` puis viennent `+` `*` `?` puis la concaténation et enfin l'opérateur `|` qui a la plus faible priorité.
Pour cette séance, vous vous familiariserez avec les expressions régulières. C'est la syntaxe des RegExp javascript que vous utiliserez.

Exercice 1 :
Dans un navigateur web, ouvrez l'outil de manipulation d'expressions que vous trouverez sur le portail pédagogique (S5info/AEL)
Cet outil permet de composer une expression puis de tester si des «mots» appartiennent au langage dénoté. La zone de saisie du mot s'encadre en vert ou en rouge selon que le mot correspond à l'expression ou pas.
Pour chacun des langages décrits ci-dessous, vous devrez trouver une expression rationnelle.
Votre travail sera à rendre sous forme d'un fichier reprenant toutes les expressions demandées (numérotées). Pour chacune vous donnerez des exemples de mots appartenant au langage et des exemples de mots n'y appartenant pas.
Les modalités de rendu sont précisées par l'enseignant responsable de votre groupe.

1. les mots composés de lettres (ASCII, donc non accentuées) majuscules ou minuscule, et commençant nécessairement par une majuscule.
2. **les numéros de téléphone en France**, au format international : 9 chiffres précédés de `+33` Le premier des 9 chiffres est différent de 0.
3. **les identifiants de département** français. Un identifiant est soit numéro compris entre 01 et 95 à l'exception de 20, soit un numéro compris entre 971 et 976, soit 2A, soit 2B.
4. **les identificateurs** commençant par une lettre, et pouvant comporter lettres, chiffres et underscore. Le signe 'underscore' ne peut être utilisé que comme **séparateur** (donc ni en début, ni en fin, et on ne peut en mettre deux successifs).
5. **les nombres entiers** en Java. Un entier peut commencer par un `0` : il est alors en octal et ne peut contenir ni `8` ni `9`. S'il commence par un autre chiffre, il est en décimal. S'il commence par le préfixe `0x` c'est qu'il est écrit en hexadécimal et peut donc comporter des chiffres ou des lettres entre a et f ou entre A et F. Dans tous les cas le «underscore» peut être utilisé comme séparateur, avec la même contrainte que ci-dessus.

6. **les listes d'identificateurs** séparés par une virgule. Les espaces sont autorisés avant comme après la virgule ainsi qu'en début ou fin de liste.
7. **les listes d'identificateurs**. Cette fois chaque identificateur peut être suivi d'une initialisation par un entier. Par exemple `x, y=3, red=0xFF0000`
8. **les littéraux chaînes de caractères (version 1)** : ils commencent et terminent par `"`. **Entre** ces deux délimiteurs, on peut trouver tout caractère **sauf** le `"`. Par exemple, `"a"b"` n'est pas valide.
9. l**es littéraux chaînes de caractères (version 2)** : ils commencent et terminent par `"`. **Entre** ces deux délimiteurs, le caractère `"` doit impérativement être doublé. Exemples de chaînes valides : `"ab""c"`, `""""`
   Exemples de chaînes invalides : `"a"b"`, `"""`
10. **les littéraux chaînes de caractères (version 3)** : ils commencent et terminent par `"`. **Entre** ces deux délimiteurs, le caractère `\` est une caractère spécial («d'échappement»). Il doit impérativement soit être doublé, soit suivi immédiatement du caractère `"` .
    Exemples de chaînes valides : `"a\\b\"c"`, `"\\\""`
    Exemples de chaînes invalides : `"a"b"`, `"a\c"`, `"a\"`, `"\\\"`
11. **Noms XML.** Un nom XML doit commencer par une lettre ou par «deux-points» ou par «underscore». Ensuite, il peut comporter ces mêmes caractères mais également des chiffres ou des points ou des tirets (signe «moins»).
12. **Réferences d'entité XML.** Une référence d'entité XML est composée d'un nom précédé de `&` et suivi de `;`
13. **Valeurs d'attribut en XML** (version simplifiée). Une valeur d'attribut est délimitée par des signes `"`. Entre ces délimiteurs on ne peut trouver ni <, ni `"` et, s'il y figure, un caractère `&` est nécessairement le début d'une référence d'entité.
14. **Balises ouvrantes XML** (version simplifiée). Une balise ouvrante est située entre < et > Elle commence par un nom puis contient une liste (éventuellement vide) d'attributs. Chaque attribut est précédé d'au moins un espace. Il est composé d'un nom, puis du signe égal, puis d'une valeur d'attribut. Les espaces sont autorisés de part et d'autre du signe égal, ainsi qu'avant le > final.
    Exemple : `<button type="button" name="changer">`