TDM 1 - Automates et Langages

septembre 2018

Manipulation d'expressions régulières (1ère partie)

1 Introduction

Les expressions rationnelles (encore appelées régulières ou RegExp) 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	

Selon l'API ou l'outil utilisé, la syntaxe des RegExp varie légèrement mais autour d'une large socle commun. Dans le cadre de cette première séance nous utiliserons les RegExp javascript, elles-même très proches de celles de Perl.

2 La syntaxe des expressions régulières

La syntaxe des expressions régulières pour les langages et applications (notation textuelle) diffère de la notation « mathématique » vue en cours tout en en reprenant les principes. Voici la correspondance entre la notation vue en cours et la notation textuelle.

- **En dehors** des *méta-caractères* suivants :
  `\ { } [ ] ^ $ ? . * + | ( )`
  chaque caractère désigne une (seule) occurrence de lui-même.
  Par exemple, `a` (notation textuelle) équivaut à l'expression a (notation mathématique)..
- la concaténation est obtenue par simple juxtaposition (sans signe opérateur)
  (ex : `ab` équivaut à $a.b$)
- l'étoile de Kleene (signifiant 0,1 ou plusieurs occurrences) utilise le caractère `*`, mais il n'est évidemment pas mis "en exposant"
  (ex : `a*` équivaut à a^*)
- l'union (le « ou ») est notée par le signe `|` au lieu du signe + de la notation mathématique.
  (ex : `a*|b*` équivaut à $a^* + b^*$)
- le symbole + est consacré à l'opérateur unaire, qui signifie *« une ou plusieurs occurrences de »*
  (ex : `a+` équivaut à $a^+ = a.a^*$)
- on peut utiliser des parenthèses pour encadrer une sous-expression (notez que les parenthèses ont également un autre rôle que l'on évoquera plus tard).
  (ex `(a|b)*` équivaut à $(a + b)^*$)

Encore quelques exemples :

Notation mathématique	notation textuelle
$a^* + b$	`a*|b`
$aab + \varepsilon$	`aab|`
$(ab + b)^*$	`(ab|b)*`
$(ab)^+ + (ba)^+$	`(ab)+|(ba)+`

On dispose d'opérateurs supplémentaires, non indispensables en théorie mais très utiles en pratique. Voici un tableau récapitulatif de la syntaxe :

Expression	Signification
x	une occurrence du **caractère** x (sauf si x est un méta caractère)
`.`	n'importe quel **caractère**, sauf la fin de ligne
`\t`	tabulation
`\n`	newline
`\s`	un espace (espace usuel, ou tabulation etc)
`\d`	un chiffre décimal
`\` x	le **caractère** x en tant que caractère ordinaire (quand x est un méta-caractère)

`(`e`)`	parenthésage de e (où e est une expression quelconque)
e`|`f	e ou f

Opérateurs de répétition

	nombre d'occurrences de e autorisées (e désigne une expression quelconque)
e`*`	≥ 0
e`+`	≥ 1
e`?`	0 ou 1
e`{n}`	n exactement (n est un entier naturel)
e`{n,m}`	$n \leq \quad \leq m$
e`{n,}`	$n \leq$

Classes de caractères

Une classe désigne UNE occurrence d'un caractère qui doit respecter la spécification indiquée dans les crochets

`[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 compris dans l'intervalle allant de x à z (donc x, y ou z) au sens de l'ordre des caractères
`[a-dtux-z]`	n'importe quel caractère compris dans l'intervalle allant de a à d ou le caractère t ou le caractère u ou n'importe quel caractère compris dans l'intervalle allant de x à z
`[^x]`	n'importe quel caractère **sauf** x
`[^x-z]`	n'importe quel caractère **sauf** ceux allant de x à z

NB : à l'intérieur des crochets les méta caractères sont traités comme des caractères ordinaires, sauf \ et]

Assertions

`^`x	le caractère x à condition qu'il soit au début d'une ligne
x`$`	le caractère x à condition qu'il soit à la fin d'une ligne
(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.

Exercice 1 :

> *Vous conserverez dans un fichier de texte une copie de l'ensemble de vos résultats aux questions de l'exercice.*
> *La séance prochaine sera encore consacrée aux RegExp, mais avec un autre outil (egrep).*
> *La plupart des réponses aux questions de cet exercice seront réutilisées la semaine prochaine.*
> *Il n'y a pas de rendu de TDM cette semaine.*

Dans un navigateur web, ouvrez l'outil de manipulation d'expressions que vous trouverez sur le portail pédagogique (S5info/AEL/tdm)
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. Afin de tester l'expression, vous constituerez dans chaque cas une liste de mots du langage et une liste de mots n'y appartenant pas (conservez ces listes).

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` **non** suivi d'un `x` : il est alors en octal et ne peut contenir ni 8 ni 9. 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. S'il commence par un autre chiffre que `0`, il est en décimal. Dans tous les cas le « underscore » peut être utilisé comme séparateur, avec la même contrainte qu'à la question précédente.
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 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.
8. 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"`, `"""`
9. **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\"`, `"\\\"`
10. **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 »).
11. **Réferences d'entité XML.** Une référence d'entité XML est composée d'un nom précédé de `&` et suivi de `;`
12. **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.
13. **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">`