# Les expressions régulières : techniques d'utilisation

Christophe Cérin
Université de Paris XIII
IUT, Villetaneuse, France

christophe.cerin@iutv.univ-paris13.fr

(version au 05/01/2011)

# Objectifs du cours

- Quelles sont les applications, les déclinaisons du concept dans une toute petite palette d'outils (sed, awk, grep) mais c'est suffisant pour passer ensuite à Java, Python, C#, Perl…
- Familiariser l'étudiant avec la recherche de motifs ;
- Quelles sont les normes pour les décrire ?
- Nombreux exemples inclus dans le cours ;

# Introduction

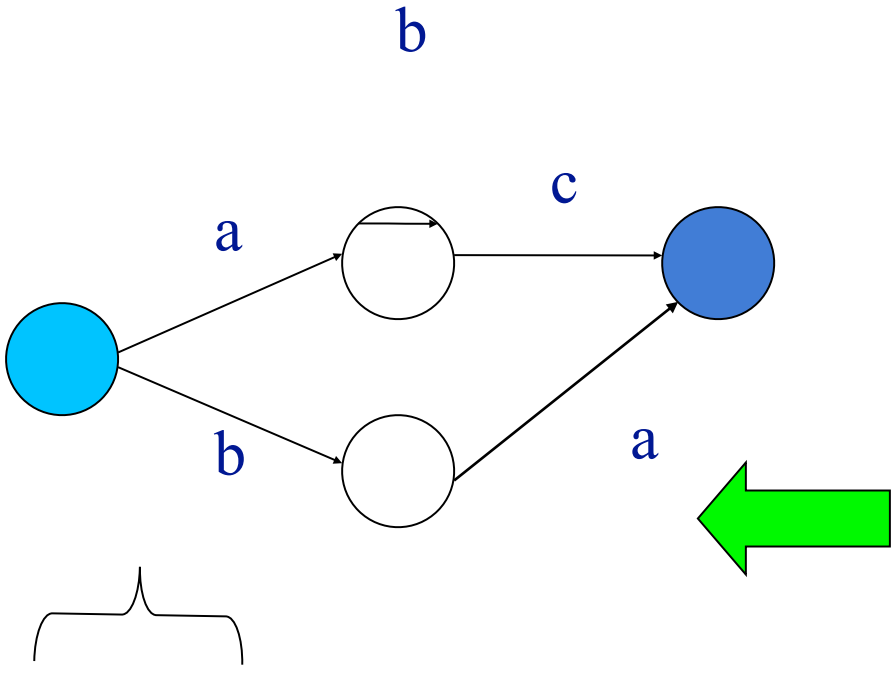
- Les applications veulent rechercher des mots dans un texte, valider si une adresse email est bien formée, tester l'intégrité d'un fichier XML…
- On a besoin de tester si un motif apparaît
- Un mot(if) c'est une suite de lettres sur un alphabet donné
- Un langage c'est une union ensembliste de mots

# Introduction (un peu de théorie)

- Les expressions régulières (regex) dénotent des langages
- Il y a 3 opérations sur les langages que les opérateurs sur les regex représentent ;
- La concaténation, l'union, la fermeture (ou l'étoile);
- Les expressions régulières sont construites par induction. Ex : si E et F sont des regex alors E+F est une regex qui dénote l'union de L(E) et L(F)

# Automate = Regex

Notations :
b*: b, zéro fois ou plus. * n'est pas interprété ici comme « n'importe quoi »
+: « où »




Ceci est un Automate d'état fini (DFA) dont l'expression régulière est ab*c + ba

Motif = ab*c + ba

Pour découvrir le motif avec un programme :

```
If(carLu = 'a') then …
Else
    If (carLu = 'b') then …
```

# Normalisation des Regex

- Norme = se fixer une représentation, des raccourcis admis par tous ! Il y a 2 normes : POSIX et UNICODE
- Problème avec l'internationalisation des caractères
- UNICODE : définition de classes de caractères (les majuscules...) qui ont des propriétés (jeu de caractères avec un id unique quelque soit la langue)
- POSIX
- Et on en veut encore plus : il nous faut signifier du groupage, de la capture, de la condition, du contrôle.

# Normes (1)

- Pour de l'aide sur les jeux de caractères : man unicode, man charsets, man utf-8
- iconv est un outil de conversion
- Les caractères d'échappement :

```
\a          alarm (= BEL)
\b          backspace
\e          escape
\f          formfeed
\n          newline
\r          carriage return
\t          tab
\v          vertical tab
\nnn        octal character (up to 3 octal digits)
\xhh        hexadecimal character (up to 2 hex digits)
\x{hh...}   hexadecimal character, any number of digits in
UTF-8 mode
```

# Normes (2) : méta caractères

There are two different sets of meta-characters: those that are recognized anywhere in the pattern except within square brackets, and those that are recognized in square brackets. Outside square brackets, the meta-characters are as follows:

| | |
|---|---|
| \ | general escape character with several uses |
| ^ | assert start of string (or line, in multiline mode) |
| $ | assert end of string (or line, in multiline mode) |
| . | (dot) match any character except newline (by default) |
| [ | start character class definition |
| \| | start of alternative branch |
| ( | start subpattern |
| ) | end subpattern |

# Normes (3)

| | |
|---|---|
| ? | extends the meaning of (<br>also 0 or 1 quantifier<br>also quantifier minimizer |
| * | 0 or more quantifier |
| + | 1 or more quantifier<br>also "possessive quantifier" |
| { | start min/max quantifier |

Part of a pattern that is in square brackets is called a "character class". In a character class the only meta-characters are:

| | |
|---|---|
| \ | general escape character |
| ^ | negate the class, but only if the first character |
| - | indicates character range |
| [ | POSIX character class (only if followed by POSIX syntax) |
| ] | terminates the character class |

# Norme (4)

Use of backslash is for specifying generic character types:

| | |
|---|---|
| \d | any decimal digit |
| \D | any character that is not a decimal digit |
| \s | any whitespace character |
| \S | any character that is not a whitespace character |
| \w | any "word" character |
| \W | any "non-word" character |
| \b | matches at a word boundary |
| \B | matches when not at a word boundary |
| \A | matches at start of subject |
| \Z | matches at end of subject or before newline at end |
| \z | matches at end of subject |
| \G | matches at first matching position in subject |

# Norme (5)

POSIX CHARACTER CLASSES

Perl supports the POSIX notation for character classes,
which uses names enclosed by [: and :] within the enclosing square brackets. PCRE also supports this notation.

For example,

**[01[:alpha:]%]**

matches "0", "1", any alphabetic character, or "%".

# Norme (6)

Pour de l'aide : man regex, man re_format ou encore man isdigit, man isalpha... (cf librairie C) pour vérifier les contenus des classes. Exemple ispunct :

041 ``!" 042 ``"" 043 ``#" 044 ``$" 045 ``%"
046 ``&" 047 ``'" 050 ``(" 051 ``)" 052 ``*"
053 ``+" 054 ``," 055 ``-" 056 ``." 057 ``/"
072 ``:" 073 ``;" 074 ``<" 075 ``=" 076 ``>"
077 ``?" 100 ``@" 133 ``[" 134 ``\" 135 ``]"
136 ``^" 137 ``_" 140 ``` 173 ``{" 174 ``|"
175 ``}" 176 ``~"

Supported class names:

- alnum letters and digits
- alpha letters
- ascii character codes 0 - 127
- blank space or tab only
- cntrl control characters
- digit decimal digits (same as \d)
- graph printing characters, excluding space
- lower lower case letters
- print printing characters, including space
- punct printing characters, excluding letters and digits
- space white space (not quite the same as \s)
- upper upper case letters
- word "word" characters (same as \w)
- xdigit hexadecimal digits

# POSIX character classes

[:alnum:] matches alphabetic or numeric characters. This is equivalent to [A-Za-z0-9].

[:alpha:] matches alphabetic characters. This is equivalent to [A-Za-z].

[:blank:] matches a space or a tab.

[:cntrl:] matches control characters.

[:digit:] matches (decimal) digits. This is equivalent to [0-9].

[:graph:] (graphic printable characters). Matches characters in the range of ASCII 33 - 126. This is the same as [:print:], below, but excluding the space character.

[:lower:] matches lowercase alphabetic characters. This is equivalent to [a-z].

[:print:] (printable characters). Matches characters in the range of ASCII 32 - 126. This is the same as [:graph:], above, but adding the space character.

[:space:] matches whitespace characters (space and horizontal tab).

[:upper:] matches uppercase alphabetic characters. This is equivalent to [A-Z].

[:xdigit:] matches hexadecimal digits. This is equivalent to [0-9A-Fa-f].

# Norme (7) : exemple

```
 $ egrep  "[^[:digit:]]7" Calendrier.txt
  - 7 semaines de cours
  - 7 semaines de cours
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   4  5  6  7
8  9 10 => semaine 0
 2  3  4  5  6  7  8 => semaine 4
23 24 25 26 27 28 29 => semaine 7
 6  7  8  9 10 11 12 => semaine 8
 4  5  6  7  8  9 10 => semaine 12
 1  2  3  4  5  6  7 => Vacances de Noel
 5  6  7  8  9 10 11 => semaine 20
 5  6  7  8  9 10 11 => semaine 23
 2  3  4  5  6  7  8 => semaine 27
 7  8  9 10 11 12 13 => semaine 30 avec mardi 8 Mai
 4  5  6  7  8  9 10 => semaine 34
```

# Norme (8) : exemple

```
$ egrep  "[[:blank:]]7|[[:blank:]]6" Calendrier.txt
  - 7 semaines de cours
  - 7 semaines de cours
                  1  2      1  2  3  4  5  6                 1  2  3
 3  4  5  6  7  8  9   7  8  9 10 11 12 13   4  5  6  7  8  9 10 =>
semaine 0
 2  3  4  5  6  7  8 => semaine 4
16 17 18 19 20 21 22 => semaine 6
23 24 25 26 27 28 29 => semaine 7
 6  7  8  9 10 11 12 => semaine 8
 4  5  6  7  8  9 10 => semaine 12
 1  2  3  4  5  6  7 => Vacances de Noel
 5  6  7  8  9 10 11 => semaine 20
 5  6  7  8  9 10 11 => semaine 23
 2  3  4  5  6  7  8 => semaine 27
    1  2  3  4  5  6 => suite semaine 29 avec mardi 1er Mai
 7  8  9 10 11 12 13 => semaine 30 avec mardi 8 Mai
 4  5  6  7  8  9 10 => semaine 34
```

# Norme (9)

- Difficile : on ne veut pas 'matcher' dans un groupe.

if the string "the white queen" is matched against the pattern

**the ((?:red|white) (king|queen))**

the captured substrings are "white queen" and "queen", and are numbered 1 and 2. The maximum number of capturing sub-patterns is 65535, and the maximum depth of nesting of all subpatterns, both capturing and non-capturing, is 200.

# Norme (10)

Le quantificateur spécifiant un nombre minimal et un nombre maximal d'occurences

**z{2,4}**

est mis en correspondance avec "zz", "zzz", or "zzzz". donc

**[aeiou]{3,}**

est mis en correspondance avec au moins 3 voyelles alors que

**\d{8}**

est mis en correspondance avec exactement 8 digits.

# Norme (11)

For convenience (and historical compatibility) the three most common quantifiers have single-character abbreviations:

* is equivalent to {0,}
\+ is equivalent to {1,}
? is equivalent to {0,1}

It is possible to construct infinite loops by following a subpattern that can match no characters with a quantifier
that has no upper limit, for example:

**(a?)***

Earlier versions of Perl and PCRE used to give an error at compile time for such patterns.

# Norme (12)

CONDITIONAL SUBPATTERNS

It is possible to cause the matching process to obey a sub-pattern conditionally or to choose between two alternative subpatterns, depending on the result of an assertion, or whether a previous capturing subpattern matched or not.

The two possible forms of conditional subpattern are

**(?(condition)yes-pattern)**
**(?(condition)yes-pattern|no-pattern)**

# Standard Unicode Properties
# http://www.unicode.org/

Voir le document unicode.html/ dans le répertoire courant… mais on ne fera pas d'exercices avec cette norme (c'est pour information)

# Exercice à programmer avec tous les outils vus et à venir !

- Un fichier contient :

| CPU rank | P Avail rank |
|---|---|
| chunk 1 | 1 | 6 |
| chunk 2 | 3 | 3 |
| chunk 3 | 4 | 1 |
| chunk 4 | 6 | 2 |
| chunk 5 | 5 | 4 |
| chunk 6 | 8 | 5 |
| chunk 7 | 9 | 6 |
| chunk 8 | 7 | 6 |
| chunk 9 | 10 | 6 |
| chunk 10 | 2 | 6 |

- Vérifier que les valeurs sont des int puis calculer le Spearman rank Correlation factor:

$$1-(6*\sum di^2)/(n*(n-1)*(n+1)$$

d_i = différences des valeurs sur la même ligne

# Bilan, synthèse

- Informellement, une expression régulière est une expression construire à partir de l'alphabet de base (a,b,...,Z) et des classes ([[:digit:]]...) en les composant avec les opérations de choix (|), d'itéré (*,+,{n,m}) et concaténation (opération implicite)

- Les outils du monde unix comme sed, awk, grep les intègrent ;
- Les language Python, Java, Perl, C# les intègrent sous forme de librairie
- http://www.pcre.org/ est une librairie pour programmer avec les REGEX en C.