Regular expressions

An introduction to a powerful tool
to process text

(Perl syntax for Microsoft .NET)

2006-04-21

Risø, Alexandre Alapetite

Search and replace

- Search
  - `*.doc`
- Replace
  - `Risoe`
  - `Risø`
- Limitations of traditional tools

Background of regular expressions

- Theoretical computer science
  - Pattern matching: automata theory, formal language theory, complexity theory, computability theory
- Should be named regex or regexp (regexen)
  - Little to do with real formal regular expressions
- Unix
  - QED: first editor using regex
    - Ken Thompson (~1966)
- Perl
  - Practical Extraction and Report Language
    - Larry Wall (1987)
  - PCRE (Perl compatible) by Philip Hazel (~1997)

Regular expression engines

- Original (Unix)
  - Deterministic Finite Automaton: faster but very limited: no back references, cannot capture sub expressions. (awk, grep, lex, …)
- Traditional
  - Nondeterministic Finite Automaton: possible misses of longer matches
- POSIX (Institute of Electrical and Electronics Engineers)
  - NFA + backtrack: slower, guarantee longest match possible but always “greedy”.
- **PCRE** (Perl compatible)
  - Backtracking + NFA (Perl, Python, Apache, Emacs, Tcl, …): the most expressive. Needs careful syntax to limit backtracking.
    - We present here the Microsoft .NET 2.0 version (System.Text.RegularExpressions)

About regular expressions

- More or less standard syntax
  - Text editors (Emacs, …)
  - System (Unix, Apache, …)
  - Programming (Perl, .NET, Java, PHP, JavaScript, …)
- Covers most needs for searching and replacing; boundaries are far away
- Regular expressions are *easier to write than to read*!
- Databases of common regexes

Character escapes and classes (1/2)

- Any character except new line `\n`: `.`
- Anti slash to escape special characters: `\`
  - The dot itself: `\.` and the back slash: `\\`
- Set of characters: `[aeiouW-Z0-579]`
  - Minus itself must be escaped or placed at the end: `[ae-]`
  - Only `\ [ ] ^ -` have special meanings
    - Example to match `[ ] ^ or -` themselves: `[\[\]^-]`
    - Example with no need to escape: `[.$+?]`
- Negative set: `[^aeiouW-Z0-579]`
  - Circumflex itself must be escaped or not placed at the beginning: `[\^]`

Character escapes and classes (2/2)

- Word character: `\w` , negative: `\W`
- Digit: `\d`, negative: `\D`
- Space: `\s`, negative: `\S`
  - Tabulation: `\t`, vertical tabulation: `\v`
  - New line: `\n`, carriage return: `\r` (Windows: `\r\n`)
- Binary character: ASCII `\x20`, Unicode `\u0020`
- POSIX classes: `[:xxx:]`, negative: `[:^xxx:]]`
  - Example: any control character: `[:cntrl:]`
  - Not recognised by Microsoft .NET 2.0
- Unicode classes (next slides)

Unicode

- Internationalisation of regular expressions
  - Example: `\w` matches also `[aæå]` in Danish
  - Microsoft .NET “culture invariant” modifier
    - Otherwise sensitive to the regional settings of the PC
  - UTF-8 modifier in PCRE, native in .NET
- Unicode classes: `\p{xx}`
  - Negative classes: `\P{xx}`
  - Example: any mathematical symbol: `\p{Sm}`
- Class of classes: `[MW-Z\d\p{Ll}\p{Sm}]`

Unicode classes

C	Other
Cc	Control
Cf	Format
Cn	Unassigned
Co	Private use
Cs	Surrogate
L	Letter
Ll	Lower case letter
Lm	Modifier letter
Lo	Other letter
Lt	Title case letter
Lu	Upper case letter
M	Mark
Mc	Spacing mark
Me	Enclosing mark
Mn	Non-spacing mark
N	Number
Nd	Decimal number
Nl	Letter number
No	Other number

P	Punctuation
Pc	Connector punctuation
Pd	Dash punctuation
Pe	Close punctuation
Pf	Final punctuation
Pi	Initial punctuation
Po	Other punctuation
Ps	Open punctuation
S	Symbol
Sc	Currency symbol
Sk	Modifier symbol
Sm	Mathematical symbol
So	Other symbol
Z	Separator
Zl	Line separator
Zp	Paragraph separator
Zs	Space separator

- Positive: `\p{xx}`
- Negative: `\P{xx}`

Unicode 4.0 classes (1/2)

- IsAlphabeticPresentationForms
- IsArabic
- IsArabicPresentationForms-A
- IsArabicPresentationForms-B
- IsArmenian
- IsArrows
- IsBasicLatin
- IsBengali
- IsBlockElements
- IsBopomofo
- IsBopomofoExtended
- IsBoxDrawing
- IsBraillePatterns
- IsBuhid
- IsCJKCompatibility
- IsCJKCompatibilityForms
- IsCJKCompatibilityIdeographs
- IsCJKRadicalsSupplement
- IsCJKSymbolsandPunctuation
- IsCJKUnifiedIdeographs
- IsCJKUnifiedIdeographsExtensionA
- IsCherokee
- IsCombiningDiacriticalMarks
- IsCombiningDiacriticalMarksforSymbols
- IsCombiningHalfMarks
- IsCombiningMarksforSymbols
- IsControlPictures
- IsCurrencySymbols
- IsCyrillic
- IsCyrillicSupplement
- IsDevanagari
- IsDingbats
- IsEnclosedAlphanumerics
- IsEnclosedCJKLettersandMonths
- IsEthiopic
- IsGeneralPunctuation
- IsGeometricShapes
- IsGeorgian
- IsGreek
- IsGreekExtended
- IsGreekandCoptic
- IsGujarati
- IsGurmukhi
- IsHalfwidthandFullwidthForms
- IsHangulCompatibilityJamo
- IsHangulJamo
- IsHangulSyllables
- IsHanunoo
- IsHebrew
- IsHighPrivateUseSurrogates
- IsHighSurrogates
- IsHiragana
- IsIPAExtensions
- IsIdeographicDescriptionCharacters

Unicode 4.0 classes (2/2)

- `IsKanbun`
- `IsKangxiRadicals`
- `IsKannada`
- `IsKatakana`
- `IsKatakanaPhoneticExtensions`
- `IsKhmer`
- `IsKhmerSymbols`
- `IsLao`
- `IsLatin-1Supplement`
- `IsLatinExtended-A`
- `IsLatinExtended-B`
- `IsLatinExtendedAdditional`
- `IsLetterlikeSymbols`
- `IsLimbu`
- `IsLowSurrogates`
- `IsMalayalam`
- `IsMathematicalOperators`
- `IsMiscellaneousMathematicalSymbols-A`
- `IsMiscellaneousMathematicalSymbols-B`
- `IsMiscellaneousSymbols`
- `IsMiscellaneousSymbolsandArrows`
- `IsMiscellaneousTechnical`
- `IsMongolian`
- `IsMyanmar`
- `IsNumberForms`
- `IsOgham`
- `IsOpticalCharacterRecognition`
- `IsOriya`
- `IsPhoneticExtensions`
- `IsPrivateUse`
- `IsPrivateUseArea`
- `IsRunic`
- `IsSinhala`
- `IsSmallFormVariants`
- `IsSpacingModifierLetters`
- `IsSpecials`
- `IsSuperscriptsandSubscripts`
- `IsSupplementalArrows-A`
- `IsSupplementalArrows-B`
- `IsSupplementalMathematicalOperators`
- `IsSyriac`
- `IsTagalog`
- `IsTagbanwa`
- `IsTaiLe`
- `IsTamil`
- `IsTelugu`
- `IsThaana`
- `IsThai`
- `IsTibetan`
- `IsUnifiedCanadianAboriginalSyllabics`
- `IsVariationSelectors`
- `IsYiRadicals`
- `IsYiSyllables`
- `IsYijingHexagramSymbols`

Grouping

- Grouping: `(  )`
  - Grouping (and capturing): `a(bc|de)f`
  - Grouping only (non capturing): `(?:  )`

Alternation (1/2)

- Alternation: `|`
  - `(gray|grey)` or `gr(a|e)y`
  - Multiple: `a|b|cd|def`
  - But `[aeiou]` is better than `a|e|i|o|u`

Quantifiers

- One: default
- N repetitions: `{n}`
  - N or more: `{n,}`
- N to M repetitions: `{n,m}`
  - Zero or more: `*` or `{0,}`
  - One or more: + or `{1,}`
  - Zero or one: ? or `{0,1}`

Assertions (1/2)

- Beginning of a line: `^`
  - Beginning of a string: `\A`
- End of a line: `$`
  - End of a string: `\Z`
- Word boundary: `\b` , negative: `\B`

Example 1

- Search for the word “test” followed by any digits, if any.
  - Corpus:
    This is a sentence with many antitests and tests such as test, test123 and test4.
  - Search:
    `(\btest\d*\b)`

Exercise 1

- Search for numbers that have two digits, a dot, three digits or more, and which are strictly smaller than 80:
  - Corpus:
    Log file with 1.1234, 12.234, 12.2345, 12.34, 45.6789, 95.123.
  - Search:

Captures

- Keep some precise pieces of what has been found
  - Capture: `(   )`
  - Named: `(?<myMask>     )`
  - Non capturing (grouping only): `(?:  )`

Back references

- Numbered back reference
  - `( )( )  \1 \k<2>`
  - HTML example: `<([a-z]+)>.*?</\1>`
- Named back reference
  - `(?<myMask> )  \k<myMask>`
  - Example searching two repeated characters: `(?<char>\w)\k<char>`

Substitutions

- Use what has been found to build the replacement:
  - Numbered capture: `$2`
  - Named capture: `$<myPattern>`
  - Copy of the match: `$&`
  - All the text before: `` $` `` or after: `$'` the match
  - Last captured group: `$+`
  - Entire input string: `$_`
  - Dollar sign: `$$`

Example 2

- Changing European date formats to (pseudo) ISO
  - Corpus:
    ```
    13/10/32, 14/7/1789, aa/bb/cc,
    123456/78/911234.
    ```
  - Search:
    ```
    \b(?<day>\d{1,2})/
    (?<month>\d{1,2})/
    (?<year>\d{2,4})\b
    ```
  - Replace:
    ```
    ${year}-${month}-${day}
    ```

Exercise 2

- Changing European date formats to (pseudo) ISO
  - Corpus:
    `14/7/1789, 14-7-1789, aa/bb/cc, 123456/78/911234.`
  - Search:
  - Replace:

Assertions (2/2)

- Look ahead
  - Positive: `(?=  )`
  - Negative: `(?!  )`
- Look behind
  - Positive: `(?<=  )`
  - Negative: `(?<!  )`
- All are non backtracking (deterministic)
  - Non backtracking sub expression (optimisation): `(?>  )`

Alternation (2/2)

- Back reference alternation:
  - By name: `(?(myMask)yes|no)`
  - By number: `(?(1)yes|no)`
- Look ahead alternation:
  - `(?(?=expression)yes|no)`

Matching behaviour

- Default “greedy” matching (longest)
  - `?`, `*`, `+`, `{n,m}`
- “Lazy” quantifiers (shortest)
  - `??`, `*?`, `+?`, `{n,m}?`

Matching options

- Ignore case: `(?i: )`
- Multi line: `(?m: )`
  - The `^` and `$` match the beginning and the end of any line
  - See also assertions `\A` and `\Z`
- Single line: `(?s: )`
  - The `.` matches every character including the new line
- Explicit capture: `(?n: )`
  - Inverse the meaning of the `(?: )` grouping construct
- Combining options: `(?imnsx-imnsx: )`
- Other PCRE options: partial matching, explicit white space, compiled patterns, etc.

Example 3

- Add a new line after full stops (not after abbreviations), if there is not one already, and remove the possible space:
  - Corpus:

    ```
    Ventrikelflimmer.  Må køre til stamafdeling.
    Jr. overført fra operationsstue 8.¶
    Pt. afleveret fra opvågningen.
    ```

  - Search:

    ```
    (?<!(?i:jr|pt))(\.)(?!\n) ?
    ```

  - Replace:

    ```
    $1\n
    ```

Subroutines

- Different from back references:
  - They try again all the possibilities
  - Useful to make expressions shorter and for recursive patterns
  - Not supported by Microsoft .NET 2.0
- Call a previous mask by number: `(?2)`
- Call a previous mask by name: `(?P>myMask)`
  - `(?P<myMask>xxx) xxx (?P>myMask)`

Recursive patterns

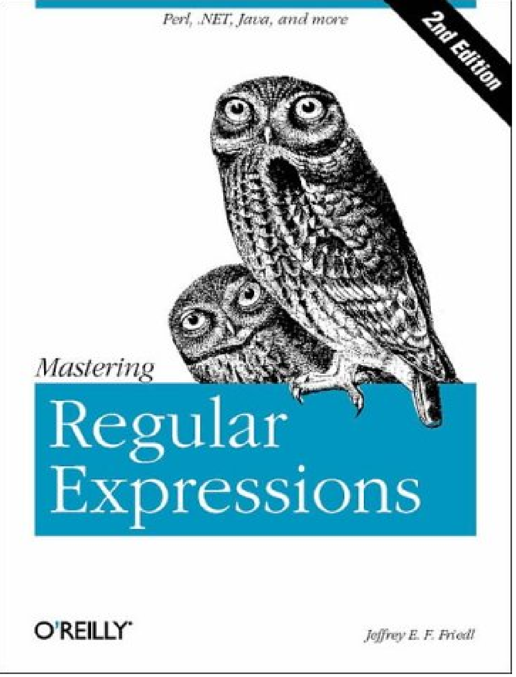
- Call a subroutine from itself
  - Call a mask by number: `(?2)`
  - Call a mask by name: `(?P>myMask)`
- Call the entire expression: `(?R)`
  - Example of the parenthesis language: `\(((?>[^()]+)|(?R))*\)`
- Test for recursion: `(R)`
  - Do not match on first call

Final advises

- Avoid using the `.`
  - Use (negative) classes instead
- Avoid captures for grouping `(  )`
  - Grouping only: `(?:  )`
  - Explicit capture option: `(?n:  )`
- Use anchors such as `^` and `$` whenever possible

Need more power?

- “Call back” system to program some of the tests and replacements
  - PCRE “call out”: `(?C)`
    or by explicit number: `(?C2)`
- Lex & Yacc
  - Lexical analyser: Lex
    - Eric Schmidt, Mike Lesk
    - GNU Flex (Fast lexical analyzer, Vern Paxson, ~1987)
  - Grammatical analyser: Yacc
    - Yet Another Compiler Compiler
    - Backus-Naur form
    - GNU Bison
- Programming with regex: Perl



References

- O’reilly: http://www.oreilly.com/catalog/regex/
- PCRE: http://www.pcre.org/pcre.txt
- Microsoft .NET: http://msdn2.microsoft.com/en-us/library/hs600312(VS.80).aspx
- PHP: http://www.php.net/manual/reference.pcre.pattern.syntax.php

Credits

- Author:
  - Alexandre Alapetite (2006-04-21)
  - http://alexandre.alapetite.net
- Work done for Risø National Laboratory
  - http://www.risoe.dk/sys/spm/
- Licence:
  - Creative Commons BY-SA (FR)
  - Attribution-ShareAlike 2.0 France
  - http://creativecommons.org/licenses/by-sa/2.0/fr/