Mastering Regular Expressions

Wing Ning Li
Basic Concepts

• A Regular Expression is used to describe a set of strings.
• In many programming languages and software tools, A Regular Expression is used for finding strings in the regular expression from a target text. This process is referred to as matching.
• A regular expression consists of two types of symbols: alphabet symbols and meta symbols.
Basic Concepts

• A typical alphabet consists of both lower and upper cases of the 26 letters (a, b,…, z, A, B,…, Z), decimal digits (0, 1, 2,…,9), special characters ($,#,_ etc.), “invisible characters” (CR for carriage return, LF for line feed, TAB, start of line, end of line, etc.)

• Meta characters or symbols are symbols having special meaning in a regular expression, such as (, ), |, *, etc,
Basic Concepts

• When a character or symbol can be used both as a meta character and normal character, an escape sequence is needed to clarify the ambiguity. In this case, the character by itself represents the meta character and the character with an escape sequence represents the normal character. The escape character is backslash \.

• * is the meta character for Kleene Closure and \* is the character *.
Examples of Meta Characters

• (, ), used as a pair for grouping.
• * for Kleene Closure.
• | for union or alternation same as + used in our main textbook.
• [, ], used as a pair for character set or character class, which is a shorthand notation for a regular expression denoting a set of characters.
Examples of Regular Expressions

- The set of binary strings ending with 0:
  
  \((0 \mid 1)^*0\)
  
  This expression has 4 meta characters and 3 normal characters.

- Any decimal digit:

  \([0-9]\)
  
  character class is used here.

  \([0123456789], [21339875640], 0|1|2|3|4|5|6|7|8|9\)
  
  are three other possible solutions.
Examples of Regular Expressions

- Set of decimal strings of length one or more:
  \[0-9]*[0-9]\]
- The lower and upper case 26 letters of English:
  \[a-zA-Z]\]
- Set of hexadecimal strings of length one or more
  \[0-9a-fA-F]*[0-9a-fA-F]\]
- Alpha-numeric strings that start with a letter
  \[a-zA-Z][a-zA-Z0-9]*\]
Examples of Meta Characters

- Negated character class [^ ]
- Any character that is not a decimal digit [^0-9]
- Set of strings containing no decimal digit [^0-9]*
- . Dot or point is a meta character denoting the alphabet.
Examples of Meta Characters

- Set of strings starts with a letter and ends with a digit
  \[a-zA-Z].*[0-9] \text{ or } [a-zA-Z](.)*[0-9]
- ^, \A start of the line
- $, \Z, \z end of the line
- \<, \b, \B start of a word
- \>, \b, \B end of a word
Examples of Regular Expressions

• Lines in a file that have an even number of 0s and an even number of 1’s:
  \^((01|10)(00|11)^*(01|10)|00|11)^*$

• Something of money amount:
  \$(0|[1-9][0-9]*)\.[0-9][0-9]$

• Any problems with the answers above?
Examples of Meta Characters

- ? To qualify the expression before it as optional items.
- July? Denoting \{July, Jul\}. Using the main textbook notation, it is Jul(y+epsilon)
- Something of money amount:
  \$(0|[1-9][0-9]*)(\.[0-9][0-9])?
Examples of Meta Characters

• + To repeat the expression before it one or more times. Note (exp)+ is equivalent to (exp)*(exp).

• {min, max} to repeat the expression before it from min to max number of times.

• Time of day such as 11:43 pm or 1:13 am can be described as a regular expression

Your answer?
More Meta Characters

• \w same as [a-zA-Z0-9_]  
• \W same as[^a-zA-Z0-9_]  
• \d same as [0-9]  
• \D same as[^0-9]  
• \s white-space (space, tab, newline, etc.)  
• \S non white-space  
• \t, \011 (octal escape), as tab character.
More Meta Characters and expressions

- Character representations
- Character class representations
- Anchors and “zero-width assertions”
- Quantifiers and alternation
  - Greedy *, +, ?, {min, max}
  - Lazy *?, +?, ??
- Capturing (for text matching and processing tasks)
Framework of application

- Pattern strings are described using regular expressions
- A target string could be a line of text, an array of lines, or a file.
- Matching: find the pattern strings as sub strings in the target string.
- Inspecting matching strings, modifying matching strings, making decision based on matching results.
Matching Engines

• **DFA:**
  – Features such as parentheses, back references, or lazy quantifiers are not allowed

• **NFA:**
  – Anything or everything is fine

• **POSIX NFA**
  – Standardizing the semantics of all NFA
Matching By Regular Expressions

• How to design or craft regular expressions to make the matching you want exactly and efficiently.

• Where in the target text the regular expression matches.

• How quickly the tool finds the match or reports failure.
Matching By Regular Expressions

• Two all-encompassing rules:
  – The match that begins earliest (leftmost) wins.
  – The standard quantifiers (*, +, ?, and {m,n}) are greedy.

• Both rules address situations where multiple matches are possible for the target text.
Matching Examples

• Regular expression:
  – cat

• Target string or text
  – *The dragging belly indicates your cat is too fast*

• Where is the match?
Matching Examples

- Regular expression: 
  - fat|cat|belly|your
- Target string or text
  - The dragging belly indicates your cat is too fat
- Where is the match?
Matching Examples

• Regular expression:
  – [0-9]+  

• Target string or text
  – March 1998

• Where is the Match?
Matching Examples

• Regular expression:
  – ^.*[0-9]+  
• Target string or text
  – Copyright 2003
• What does [0-9]+ match? 2003 or 3 or others
Matching Examples

• Regular expression:
  – ^.*[0-9][0-9]

• Target string or text
  – About 24 characters long
Matching Examples

• Regular expression:
  – to(nite|knight|night)

• Target string or text
  – After and before tonight
Matching Examples

• DFA:
  – Text-Directed

• NFA
  – Regex-Directed