Regular Expressions lecture notes

Linguistics 101, Professor Roger Levy

7 January 2015

1. **Trying it on your own:** You can find implemented tools for testing and using regular expressions in many places, including:

   - The `grep` and `egrep` tools on UNIX, including the Mac OS X command line and Cygwin on Windows
   - Most programming languages, including Python, Perl, and Java
   - Many text editors such as Emacs and `vi`
   - Publicly available webpages such as [http://www.regexr.com/](http://www.regexr.com/)

2. Regular expressions are formulas that define **classes of strings**

3. Definition of a **string**, one of the fundamental data types in computational linguistics:

   - Start with a non-empty, finite set of **symbols**—call it $\Sigma$
   - A **string** over $\Sigma$ is a sequence of length zero or greater of symbols from $\Sigma$
     - If the sequence is of length zero then you have the **empty string**, which we denote with $\epsilon$
   - Sometimes we’ll refer to the set of all strings over $\Sigma$ as $\Sigma^*$—read the **Kleene star** * as “any number of ’em”

4. The fundamental building blocks of regular expressions:

   - **the symbols in $\Sigma$** (some may need to be escaped)
   - **Square brackets [], the hyphen -** Used to construct character classes
   - **The caret ~ in brackets** Negating character classes
   - **The period .** A **wildcard:** matches any symbol
   - **Parenthesis ()** Creates groups, establishes precedence
   - **Curly braces {}** Counters
   - **Kleene * and +** at least one/zero of what comes before it
   - **The question mark ?** optionality: either zero or one of what comes before it
   - **The pipe |** anchors to start and end of string
   - **disjunction**
5. **Escaping** special characters: to match ASCII symbols that have special meanings in regular expression definitions (e.g., ^, ., *, (,), precede them with a backslash \. For example, the regex `Do you want \$\$\$\$\$` matches `Do you want $$$?`

6. A few key operations:

   - Using `[ ]` to create character classes:
     - `[ACE]` any of the symbols A, C, or E
     - `[A-E]` any of the symbols A, B, C, D, or E
     - `[A-E_]` any of A–E, or the underscore `_`
     - `[^A-E_]` any character not in the previous class (`^` negates a class)
     - `[-A-Z]` any of A–Z, or the hyphen

   - Using parentheses to change precedence:
     - `ABC?` matches AB or ABC
     - `A(BC)?` matches A or ABC

7. Operator precedence: `()` precedes “suffixes” `* + ?` `{}` precedes concatenation & anchors precedes disjunction `|`

8. The set of strings that a given regular expression matches can be called a **language**!

   Examples:
   - hat
   - `pickles.*`
   - `ε`
   - `ab+`
   - `(ab)+`
   - `[^a]*`
   - `chocolate|vanilla ice cream`
   - `(chocolate|vanilla) ice cream`
   - `(very){0,3} tall trees`

9. Common shorthands—these don’t change the expressive power of regular expressions, but make writing them more convenient!

   - `\w` Match any letter or number, or the underscore character `_`
   - `\W` Match anything not matched by `\w`
   - `\d` Match any non-digit
   - `\d` Match any digit
   - `\s` match any whitespace character
   - `\S` match any non-whitespace character
10. Constructs that you will find in many regular expression implementations but that are not part of “core” regular expressions

\1, \2, etc. **Backreferences**—match whatever was matched in a previous group (?=) and ?!  Positive and negative **lookahead**

Positive and negative lookahead affect the search process for matching regular expressions (wait till Monday!).