

# Corpus Linguistics (L615)

## Regular expressions

Markus Dickinson

Department of Linguistics, Indiana University  
Spring 2009

---

If one wants to be able to describe more complex patterns of words and text, sometimes boolean expressions aren't enough:

- ▶ e.g., you want to find *help to V* constructions in POS-tagged text, and the words and tags are mixed up together
- ▶ You want to find the first verb used in a relative clause and retrieve that.
- ▶ You want to find all Indiana email addresses which occur in a long text.

Anything where you have to match a complex pattern so-called **regular expressions** are useful.

# Regular expressions: What they are

- ▶ A regular expression is a compact description of a set of strings, i.e., a language (in **formal language** theory).
  - ▶ They can be used to search for occurrences of these strings
- ▶ Regular expressions can only describe so-called **regular languages**.
  - ▶ This means that some patterns cannot be specified using regular expressions, e.g., finding a string containing matching left and right parentheses.

Note that just like any other formalism, regular expressions as such have no linguistic contents, but they can be used to refer to strings encoding a **natural language** text.

- ▶ A variety of unix tools (grep, sed, ...), editors (emacs, jEdit, ...), and programming languages (perl, python, Java, ...) incorporate regular expressions.
  - ▶ We'll focus today on grep and use regular expressions in perl later this semester
- ▶ Implementations are very efficient so that large text files can be searched quickly

The various tools differ w.r.t. the exact syntax of the regular expressions they allow, but knowledge of one transfers

# The syntax of regular expressions (I)

Regular expressions consist of

- ▶ strings of literal characters: `c`, `A100`, `natural language`, `30 years!`
- ▶ disjunction:
  - ▶ ordinary disjunction: `devoured|ate`, `famil(y|ies)`
  - ▶ character classes: `[Tt]he`, `bec[oa]me`
  - ▶ ranges: `[A-Z]` (any capital letter)
- ▶ negation:
  - `[^a]` (any symbol but a)
  - `[^A-Z0-9]` (not an uppercase letter or number)

Use **aliases** to designate particular recurrent sets of characters

- ▶ `\d = [0-9]`: digit
- ▶ `\D = [^\d]`: non-digit
- ▶ `\w = [a-zA-Z0-9_]`: alphanumeric
- ▶ `\W = [^\w]`: non-alphanumeric
- ▶ `\s = [\r\t\n\f]`: whitespace character
  - ▶ `\r`: space, `\t`: tab, `\n`: newline, `\f`: formfeed
- ▶ `\S [^\s]`: non-whitespace

# The syntax of regular expressions (II)

- ▶ counters
  - ▶ optionality: ?  
colou?r
  - ▶ any number of occurrences: \* (Kleene star)  
[0-9]\* years
  - ▶ at least one occurrence: +  
[0-9]+ dollars
- ▶ wildcard for any character: .  
beg.n for any character in between beg and n
- ▶ Parentheses to group items together  
ant (farm)?
- ▶ Escaped characters to specify characters with special meanings:  
\\\*, \\+, \\?, \\(, \\), \\|, \\[, \\]

# The syntax of regular expressions (III)

- ▶ Operator precedence, from highest to lowest:

parentheses ()

counters \* + ?

character sequences

disjunction |

- ▶  $\text{fire|ing} = \text{fire or ing}$
- ▶  $\text{fir}(e|\text{ing}) = \text{fir followed by either } e \text{ or } \text{ing}$

# The syntax of regular expressions (IV)

Anchors: anchor expressions to various parts of the string

- ▶ `^` = start of line
  - ▶ do not confuse with `[^ . . . ]` used to express negation
- ▶ `$` = end of line
- ▶ `\b` = non-word character (i.e., word boundary)
  - ▶ word characters are digits, underscores, or letters, i.e., `[0-9A-Za-z_]`

Instead of writing out specific numbers of occurrences, repetition can be represented between `{ }`

- ▶ `a{4}` = 4 a's
- ▶ `a{1,4}` = 1-4 a's

# Some RE practice

- ▶ What does `\$[0-9]+(\.[0-9][0-9])` signify?
- ▶ Write a RE to capture the times on a digital watch (hours and minutes). Think about:
  - ▶ the (im)possible values for the hours
  - ▶ the (im)possible values for the minutes

- ▶ `grep` is a powerful and efficient program for searching in text files using regular expressions.
- ▶ It is standard on Unix, Linux, and Mac OSX, and there also are various ports to Windows (e.g., <http://gnuwin32.sourceforge.net/packages/grep.htm>, <http://www.interlog.com/~tcharron/grep.html> or <http://www.wingrep.com/>).
- ▶ The version of `grep` that supports the full set of operators mentioned above is generally called `egrep` (for extended `grep`).

# Grep: Examples for using regular expressions

## (I)

In the following, we assume a text file `f.txt` containing, among others, the strings that we mention as matching.

- ▶ Strings of literal characters:  
`egrep 'and' f.txt` matches and, Ayn Rand, Candy and so on
- ▶ Character classes:  
`egrep 'the year [0-9][0-9][0-9][0-9]' f.txt` matches the year 1776, the year 1812, the year 2001, and so on

# Grep: Examples for using regular expressions (II)

- ▶ disjunction (|): `egrep 'couch|sofa' f.txt` matches couch or sofa
- ▶ grouping with parentheses:  
`egrep 'un(interest|excit)ing' f.txt` matches uninteresting or unexciting.
- ▶ Any character (.):  
`egrep 'o.e' f.txt` matches ore, one, ole

# Grep: Examples for using regular expressions (III)

- ▶ Kleene star (\*):  
egrep 'a\*rgh' f.txt matches argh, aargh, aaargh
- ▶ One or more (+):  
egrep 'john+y' f.txt matches johny, johnny, ...,  
but not johy
- ▶ Optionality (?):  
egrep 'joh?n' f.txt matches jon and john

## Revisiting *help/help to*

Remember our Perl program that searched for variants of *help*?

- ▶ Let's (re)-unpack that a bit

We compared *help V* & *help to V*

- ▶ and *help NP V* & *help NP to V*
- ▶ For these latter cases, we simplified the NP to be a single noun (tag starts with n) or pronoun (p)

The patterns we used:

- ▶ *help V*: `\b(help\w*?/v\w*?\s+\w+/v\w*?)\b`
- ▶ *help to V*:  
`\b(help\w*?/v\w*?\s+to/to\s+\w+/v\w*?)\b`
- ▶ *help NP V*:  
`\b(help\w*?/v\w*?\s+\w+/[np]\w*?\s+\w+/v\w*?)\b`
- ▶ *help NP to V*:  
`\b(help\w*?/v\w*?\s+\w+/[np]\w*?\s+to/to\s+\w+/v\w*?)\b`

# Breaking down the regular expression

```
\b(help\w*?/v\w*?\s+\w+/v\w*?)\b
```

So, what do we see here?

- ▶ Word boundaries before *help* and at the end of the expression
- ▶ *help* followed by a sequence of 0 or more (\*) word characters (\w)
  - ▶ This matches *help*, *helps*, *helpful*, etc.
  - ▶ We'll talk about \*? momentarily
- ▶ Next up is /v\w\*?: this matches a string starting with /v and followed by any word characters
  - ▶ Taking these 2 together, we could match, e.g., *helping/vbg*
- ▶ Then we have \s+: 1 or more whitespace characters
- ▶ Then \w+/v\w\*?: the end of this is the same as before, i.e., matches any verb.
  - ▶ But with \w+, we match *any* word, not just *help*

# Greediness & Capturing parentheses

## ▶ Greediness

- ▶ In Perl, *\** is *greedy*: it tries to match as much text as possible

- ▶ Consider a text *John goes to the store* and an RE `t.*s`
- ▶ With the normal, greedy *\**, this matches *to the s*
- ▶ With the non-greedy *\*?* (i.e., `t.*?s`), this matches *the s*

## ▶ Capturing parentheses: parentheses do more than just distinguish subparts of an RE

- ▶ They “capture” the part(s) of the RE you may want further access to
- ▶ `\b(help\w*?/v\w*?\s+\w+/v\w*?)\b`
  - ▶ We can use `$1` to refer to the captured part of the RE (and `$2` if there were a second capture, etc.)
  - ▶ e.g., `<word>(\w+)</word>` will match the whole string, but only capture the part in-between the XML tags

# Online web searching with REs

- ▶ Both the BNC and the European Parliament corpus can be searched using on-line web-forms.
- ▶ Both of the web forms allow **regular expressions** for advanced searching.
- ▶ To provide efficient searching in large corpora, in these search engines regular expressions over characters are limited to single tokens (i.e. generally words).
- ▶ BNC:
  - ▶ web form: <http://sara.natcorp.ox.ac.uk/lookup.html>
  - ▶ regular expressions are enclosed in { }
- ▶ European Parliament Corpus:
  - ▶ web form: <http://logos.uio.no/cgi-bin/opus/opuscqp.pl?corpus=EUROPARL;lang=en>
  - ▶ in the simplest case, regular expressions are enclosed in " "