

# Natural Language Processing

## CSCI 4152/6509 — Lecture 4

### Basic NLP with Perl

Instructors: Vlado Keselj

Time and date: 14:35 – 15:55, 7-Oct-2025

Location: Studley LSC-Psychology P5260

# Previous Lecture

- Finite state automata (continued)
  - ▶ Deterministic Finite Automaton (DFA)
  - ▶ Non-deterministic Finite Automaton (NFA)
- Review of Deterministic Finite Automata (DFA)
- Non-deterministic Finite Automata (NFA)
- Implementing NFA, NFA-to-DFA translation
- Review of Regular Expressions

# Introduction to Perl

- Created in 1987 by Larry Wall
- Interpreted, but relatively efficient
- Convenient for string processing, system admin, CGI, etc.
- Convenient use of Regular Expressions
- Larry Wall: Natural Language Principles in Perl
- Perl is introduced in lab in more details

# Why Perl?

- We only cover Perl with regular expressions and basic text processing
- Perl is very convenient for these types of tasks
- Perl uses very direct way of using regular expressions
- Perl is still used a lot in text processing, NLP, bioinformatic string processing, etc.
- Perl-style regular expressions are very important in any programming languages for NLP

# Testing Code (as shown in labs)

- Login to timberlea
- Use plain editor, e.g., emacs
- Develop and test program
- Submit assignments
- You can use your own computer, but code must run on timberlea

# Regular Expressions in Perl

- We will learn more about Regular Expressions by using Perl in simple text processing.
- Let us start with a program to count lines in a text

# Perl Program for Counting Lines

```
#!/usr/bin/perl
# program:  lines-count.pl

while (<>) {
    ++$count;
}

print "$count\n";
```

# Regular Expressions in Perl

- Perl provides an easy use of Regular Expressions
- Consider the regular expression: `/pro...ing/`
- Run the following commands on `timberlea`:  
`cp ~prof6509/public/linux.words .`  
`grep proc...ing linux.words`
- Output includes 'processing', and more:  
`co`**processing**  
`food-`**processing**  
`micro`**processing**  
`mis`**proceeding**  
`multi`**processing**  
`...`



# Note About File 'linux.words' and Others

- Some helpful files can be found on timberlea in:  
~prof6509/public/
- or, on the web at:  
<http://web.cs.dal.ca/~vlado/csci6509/misc/>
- For example:  
linux.words  
wordlist.txt  
Natural-Language-Principles-in-Perl-Larry-Wall.pdf  
TomSawyer.txt

# Perl Regular Expressions: 'proc...ing' Example

- Similar functionality as grep:

```
#!/usr/bin/perl
# run as: ./re-proc-ing.pl linux.words

while ($r = <>) {
    if ($r =~ /proc...ing/) {
        print $r;
    }
}
```

## Shorter 'proc...ing' Code

- There are several ways how this program can be made shorter: first, let us use the default variable '\$\_':

```
while ($_ = <>) {  
    if ($_ =~ /proc...ing/) {  
        print $_;  
    }  
}
```

- Shorter version:

```
while (<>) {  
    if (/proc...ing/) {  
        print;  
    }  
}
```

## Even Shorter 'proc...ing' Code

- and shorter:

```
while (<>) {  
    print if (/proc...ing/);  
}
```

- and shorter:

```
#!/usr/bin/perl -n  
print if (/proc...ing/);
```

- or as a one-line command:

```
perl -ne 'print if /proc...ing/'
```

# More Special Character Classes

`\d` — any digit

`\D` — any non-digit

`\w` — any word character

`\W` — any non-word character

`\s` — any space character

`\S` — any non-space character

# A More Complete List of Iterators

\* example: `\s*`

+ example: `\d+`

? example: `\d?\d`

{n} example: `B\d{8}`

{n,m} example: `\w{3,5}`

{n,} example: `-\{5,72\}`

{,m} example: `.\{,20\}`

# A More Complete List of Iterators

\* — zero or more occurrence

+ — one or more occurrences

? — zero or one occurrence

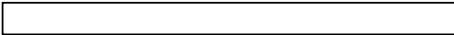
{ $n$ } — exactly  $n$  occurrences

{ $n, m$ } — between  $n$  and  $m$  occurrences

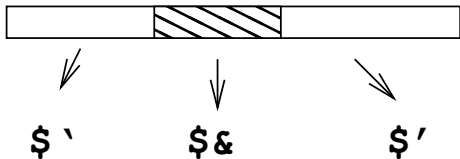
{ $n, \}$  — at least  $n$  occurrences

{ $, m$ } — at most  $m$  occurrences

# Some Special Variables Assigned After a Match in Perl

`$var =` 

regular expression match: `$var =~ /re/`





# Example: Counting Simple Words

```
#!/usr/bin/perl
```

```
my $wc = 0;
while (<>) {
    while (/\\w+/) { ++$wc; $_ = $'; }
}
print "$wc\\n";
```

## Example: Counting Simple Words (2)

- Consider the following variation:

```
#!/usr/bin/perl
```

```
my $wc = 0;
while (<>) {
    while (/\\w+/g) { ++$wc }
}
print "$wc\\n";
```

# Counting Words and Sentences

```
#!/usr/bin/perl
# simplified sentence end detection

my ($wc, $sc) = (0, 0);
while (<>) {
    while (/\\w+|\\[.!?]+/) {
        my $w = $&; $_ = $';
        if ($w =~ /^\\[.!?]+$/ ) { ++$sc }
        else { ++$wc }
    }
}

print "Words: $wc Sentences: $sc\\n";
```

# More on Perl RegEx'es

<code>\G</code>	anchor, end of the previous match
<code>(?=re)</code>	look-ahead
<code>(?!re)</code>	negative look-ahead
<code>(?&lt;=re)</code>	look-behind
<code>(?&lt;!re)</code>	negative look-behind

- Some examples:

`/foo(?!.*foo)/` — finding last occurrence of 'foo'

`s/(?<=\be)(?=mail)/-/g` — inserting hyphen

`/\b\w+(?<!s)\b/` — a word not ending with 's'

# An Example with \G

```
while (<>) {  
  while (1) {  
    if      (/\\G\\w+/gc) { print "WORD: $&\\n" }  
    elsif (/\\G\\s+/gc) { print "SPACE\\n" }  
    elsif (/\\G[.,;?!]/gc)  
                        { print "PUNC: $&\\n" }  
    else { last }  
  }  
}
```

- Option g must be used with \\G for global matching
- Option c prevents position reset after mismatch

# Back References

- `\1 \2 \3 ...` match parenthesized sub-expressions
- for example: `/(a*)b\1/` matches  $a^nba^n$ ; such as `b`, `aba`, `aabaa`, ...
- Sub-expressions are captured in `(...)`
- Aside, in `grep`: `\(...\)`
- `(?:...)` is grouping without capturing

# Back Reference Examples

Consider examples:

$\text{/(a+(b+))(c+(d+))\4/}$  and  $\text{/(a+(b+))(c+(d+))\3/}$