LING/C SC/PSYC 438/538

Lecture 12

Sandiway Fong

Today's Topics

- Homework 8 review
- Perl regex
  - backreferences
  - shortest vs. greedy matching
  - exponential time performance

Homework 8 Review

- **Question 1:** in English, names typically begin with an Upper case letter. Other characters may be lower/upper case or include a dash (-), e.g. `Al-Ghad`. Write a regex and find all the matching **words** in the article. How many are there?
- Code:
  - `perl -le 'open $f, "bbc_pandora.txt"; while (<$f>) {while (/\b[A-Z][A-Za-z-]*\b/g) {print $&}}' | wc -l`
  - `404`
- Gets more than named entities:
  - Words at the start of sentence: e.g. The
  - Some words occur many times (*see Question 5*)

Homework 8 Review

- **Question 1 bonus 1:** an earlier slide mentions `use open qw(:std :utf8);` Find a difference in the words reported when running your code with this declaration, i.e. when using:
  - **Hint**: you may want to think about `[A-Za-z-]` vs `[\w-]`
- Code:

  -
    ```
    perl -le 'use open qw(:std :utf8); open $f, "bbc_pandora.txt"; while (<$f>) {while (/\b[A-Z][\w-]*\b/g) {print $&}}' | wc -l
    ```
- **Examples:**
  - Sebastián
  - Piñera
  - País
  - Nación
  - José
  - Ossandón

Homework 8 Review

- **Question 1 bonus 2:** do all name words begin with an Upper case letter? Find one that doesn't.
- **Example:**
  - `Syria's President Bashar al-Assad`

Homework 8 Review

- **Question 2:** abbreviations/acronyms often consist of words, #letters ≥2, containing only Upper case letters, e.g. TV NTV US EPA. Write a regex for this. How many are there?
- Code:

  -
    ```
    perl -le 'open $f, "bbc_pandora.txt"; while (<$f>)
    {while (/\b[A-Z]{2,}+\b/g) {print $&}}' | wc -l
    ```
  - 26
- Examples:

  -
    ```
    II UK US II II TV TV LIVE BACKGROUND TV NTV US EPA
    TV US EPA UK UK UK UK UK US BBC DC BBC UK
    ```

Homework 8 Review

- **Question 3:** many names are *n*-grams, for *n*≥2, a sequence of words each beginning with an Upper case letter, **optionally** beginning with a title, e.g. Mr/Ms/Mrs/Dr, Prime Minister, President or King/Queen, e.g. `Mr Zelensky`, `President Vladimir Putin` or `King Abdullah II`. Write a regex and find all the matching sequences (*#words* ≥2) beginning with a title in the article. Print them. How many are there?
- You should incorporate the utf8 pragma mentioned in Question 1 Bonus 1.
- Code:

  -
    ```
    perl -le 'use open qw(:std :utf8); open $f, "bbc_pandora.txt"; while (<$f>) {while (/\b(M(r|s|rs)|Dr|President|King|Queen)(\s+[A-Z]\w*)+/g) {print $&}}' | wc -l
    ```
  - 28

Homework 8 Review

1. King Abdullah II
2. King Abdullah
3. President Bashar
4. King Abdullah
5. King Abdullah II
6. King Abdullah II
7. President Uhuru Kenyatta
8. Mr Kenyatta
9. Mr Kenyatta
10. President Vladimir Putin
11. President Putin
12. President Volodymyr Zelensky
13. President Volodymyr Zelensky
14. Mr Zelensky
15. President Volodymyr Zelensky
16. Prime Minister Andrej Babis
17. Mr Babis
18. Prime Minister Imran Khan
19. Mr Khan
20. Mr Lasso
21. President Sebastián Piñera
22. Mr Piñera
23. Mr Piñera
24. President Nicos Anastasiades
25. Mr Anastasiades
26. Prime Minister Tony Blair
27. Mrs Blair
28. Mr Amersi

Homework 8 Review

- **Question 4**: write a regex to find all the monetary values quoted in the article. Note currency symbols, comma separators and abbreviations such as m for million.
- Code:
  - `perl -le 'use open qw(:std :utf8); open $f, "bbc_pandora.txt"; while (<$f>) {while (/[\$£][0-9,]+m?\b/g) {print $&}}'`

```
1. £70m
2. $100m
3. £12m
4. $120m
5. $152m
6. £33m
7. £312,000
8. £700m
```

Homework 8 Review

- **Question 5**: using the Perl hash table described in a previous lecture, re-do question 3 and collect together mentions of names, e.g. `King Abdullah` occurs multiple times. Then print names and number of occurrences in tabular form.
- Code:

  -
    ```
    perl -le 'use open qw(:std :u; while (<$f>) {while
    (/\b(M(r|s|rs)|Dr|Prime
    Minister|President|King|Queen)(\s+[A-Z]\w*)+/g)
    {$name{$&}++}}; for (keys %name){printf "%-30s
    %s\n", $_, $name{$_}}'
    ```

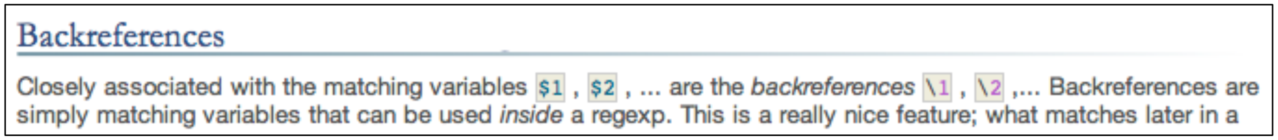
Homework 8 Review

Mr Piñera	2
President Sebastián Piñera	1
President Putin	1
Prime Minister Tony Blair	1
Mr Anastasiades	1
President Bashar	1
President Volodymyr Zelensky	3
Mr Zelensky	1
Mr Babis	1
King Abdullah	2
Mr Kenyatta	2
Mrs Blair	1
President Vladimir Putin	1
President Uhuru Kenyatta	1
Mr Khan	1
Prime Minister Imran Khan	1
Mr Lasso	1
Mr Amersi	1
King Abdullah II	3
Prime Minister Andrej Babis	1
President Nicos Anastasiades	1

Homework 8 Review

Mr Piñera	2	President Vladimir Putin	1
President Sebastián Piñera	1	President Uhuru Kenyatta	1
President Putin	1	Mr Khan	1
Prime Minister Tony Blair	1	Prime Minister Imran Khan	1
Mr Anastasiades	1	Mr Lasso	1
President Bashar	1	Mr Amersi	1
President Volodymyr Zelensky	3	King Abdullah II	3
Mr Zelensky	1	Prime Minister Andrej Babis	1
Mr Babis	1	President Nicos Anastasiades	1
King Abdullah	2		
Mr Kenyatta	2		
Mrs Blair	1		

Chapter 2: JM

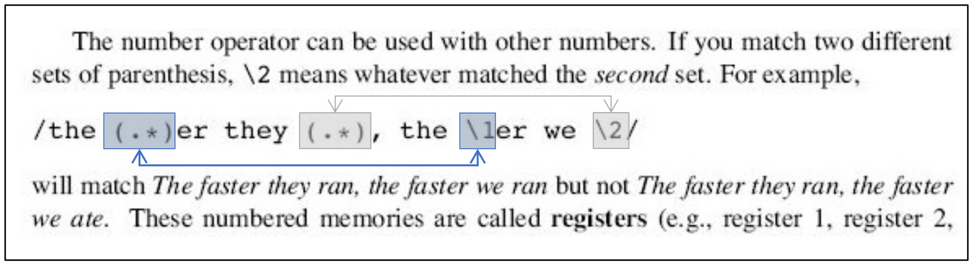

Backreferences

Closely associated with the matching variables $1 , $2 , ... are the *backreferences* \1 , \2 ,... Backreferences are simply matching variables that can be used *inside* a regexp. This is a really nice feature; what matches later in a

- s/([0-9]+)/<\1>/

what does this do?

Backreferences give Perl regexs more expressive power than **Finite State Automata** (FSA)


The number operator can be used with other numbers. If you match two different sets of parenthesis, \2 means whatever matched the *second* set. For example,

```
/the (.*)er they (.*), the \1er we \2/
```

will match *The faster they ran, the faster we ran* but not *The faster they ran, the faster we ate*. These numbered memories are called **registers** (e.g., register 1, register 2,

Shortest vs. Greedy Matching

- default behavior
  - in Perl regex matching:
    - *take the longest possible matching string*
    - *and see if it works*
    - *if so, ok*
    - *if not, backtrack (take a shorter match and try again)*
  - aka **greedy matching**
    - *This behavior can be changed, see next slide*

Shortest vs. Greedy Matching

from http://www.perl.com/doc/manual/html/pod/perlre.html

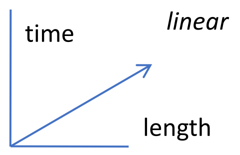
- Example:

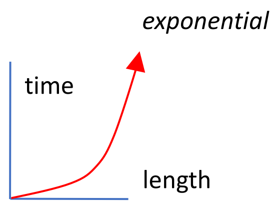
```
                      (.*?)
                                 (.*)
$_ =  "The food is under the bar in the barn.";
if ( /foo(.*?)bar/ ) {
        print ”matched <$1>\n";
}
```

- Output:
  - greedy (.*): `matched <d is under the bar in the >`
  - shortest (.*?): `matched <d is under the >`
- Notes:
  - ? immediately following a repetition operator like * (or +) makes the operator work in non-greedy mode
  - + immediately following a repetition operator makes it **non-backtracking** greedy

Regex: exponential time

- Regex search is supposed to be fast
    - but searching is not necessarily proportional to the length of the string (or corpus) being searched
    - in fact, Perl RE matching can can take exponential time (in length)





- **non-deterministic**
    - *may need to backtrack (revisit last choice point) if it matches incorrectly part of the way through*
    - Let's consider a?a?a?aaa matching against the string *aaa*

Regex: exponential time

- Consider a?a?a?aaa matching against the string *aaa*
  - For expository purposes: $a_1\ a_2\ a_3$
  - red *a* = failure to match (causes backtracking)

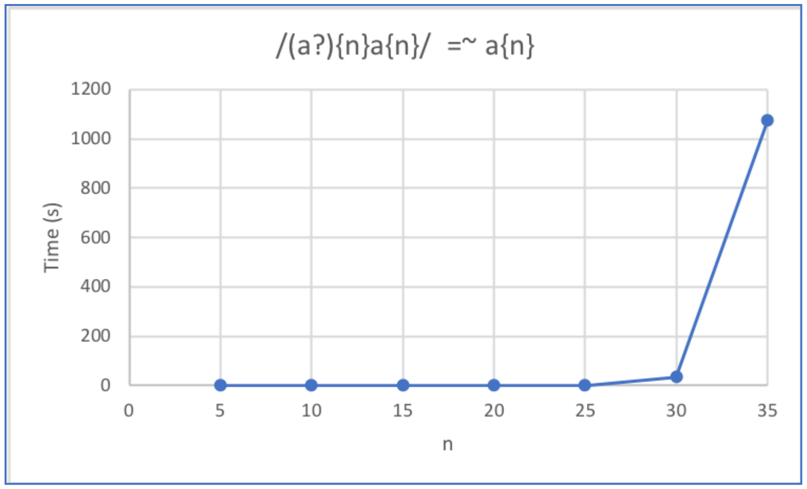
Tries:

1. a_1? a_2? a_3?aaa
2. a_1? a_2? ? $a_3$aa
3. a_1? ? a_2 ? $a_3$aa
4. a_1? ? ? $a_2 a_3$a
5. ? a_1 ? a_2 ? $a_3$aa
6. ? a_1 ? ? $a_2 a_3$a
7. ? ? a_1 ? $a_2 a_3$a
8. ? ? ? $a_1 a_2 a_3$ ← success!

Regex: exponential time

- Now consider scaling up `a?a?a?aaa`, i.e. $(a?)^n a^n$ matching against a^n

n in a?nan	Time (s)
5	0.008
10	0.006
15	0.01
20	0.052
25	0.083
30	34.48
35	1077



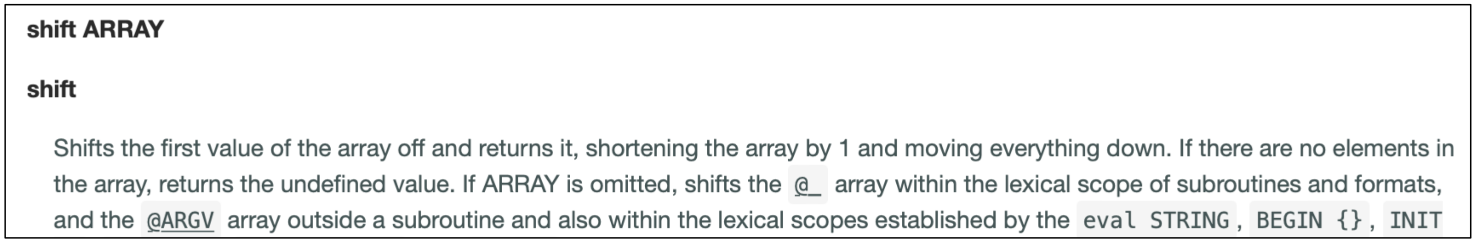
Reference: https://swtch.com/~rsc/regexp/regexp1.html

Regex: exponential time

- regex `(a?)`^{n}a^n matching against a^n for a range of values for *n*

```
time perl -e '$n = shift; $na = "a" x $n; print $na =~ /(a?){$n}a{$n}/' 25
real 0m3.201s
user 0m3.190s
sys 0m0.007s
```

- Note:
  - `shift` defaults to working on @ARGV, that's how $n gets 25 above.



shift ARRAY

shift

Shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down. If there are no elements in the array, returns the undefined value. If ARRAY is omitted, shifts the `@_` array within the lexical scope of subroutines and formats, and the `@ARGV` array outside a subroutine and also within the lexical scopes established by the `eval STRING`, `BEGIN {}`, `INIT`