LG 467 Computers in Linguistics

[1-2021] Topic 3: Regular Expressions (and Search)

Sakol Suethanapornkul

Previously...

- Tokenization with NLTK

Code 3.7

```
from nltk.tokenize import word_tokenize, TweetTokenizer
```

Code 3.8

```
sent = '''If you’re happy and you know it, clap your
hands.'''
sent2 = '''I didn't wanna go into detail how these
things'd been done!'''

word_tokenize(sent)
word_tokenize(sent2)
```

Previously...

- Tokenizing tweets with TweetTokenizer() class

Code 3.8

```
tweet = '''@MayorBowser @DOEE_DC @DCDPW @capitalweather
@washingtonpost Been #flooding like this for years no
help from the #DCgovernment #cafritz
#connecticutavenue'''

twt = TweetTokenizer()
twt.tokenize(tweet)
```

How does tokenization work?

- In part:
  - lists of abbreviations (don't split *U.S.A.*, *U.K.*, or *e.g.*)
  - Heuristics (capital → previous period was a sentence ending)
- But most importantly:
  - **Patterns**:
    - anything with a *XXXX*'s: split the genitive *s*
    - split/don't split hyphenated words
    - and etc.

Regular Expressions

Regular expression

A **regular expression** (or **regex**):

- is a sequence of characters that specifies a search pattern
- describes a regular language (in formal language theory)
- has no linguistic content

Regex is widely used:

- in search engines, text processors, etc.
- in tokenization, pattern matching (→ chatbots), etc.

Regular expression

A **regular expression** (or **regex**):

- is very powerful but quite cryptic
- is fun, once you understand them

Regex is particularly useful for searching in texts:

- when we have a **pattern** to search for; and
- a **corpus** of texts to search through

By convention, regex is often given between slashes → `/hello/`

Regex in Python

We can use regex with the standard module `re`:

Code 4.1

```
import re

# In Python, a regex search is written as
# x = re.match(pat, str)

result = re.match('this', 'this is a cat’)

result
result.group()

# or
re.match('this', 'this is a cat').group()
```

Regex in Python

`match()` checks whether a string begins with the pattern

Code 4.2

```
# match() finds exact beginning match
str1 = "you're okay?"
str2 = "How are you?"

a = re.match('you', str1)
b = re.match('you', str2)

# To extract the match use .group()
a.group()
b.group()
```

Regex in Python

`search()` finds first match anywhere in the source string

Code 4.3

```
str1 = "You're about to find out how powerful regex is"

re.search('find')
re.match('find')    # compare
```

Question: How can we extract multiple matches from a multiline file using `search()`?

Regex in Python

Quiz: Extract "happy" from the following lyric [5 mins]

Code 4.4

```
song = '''If you’re happy and you know it,
Clap your hands.
If you're happy and you know it,
Clap your hands.

If you're happy and you know it,
And you really want to show it,
If you're happy and you know it,
Clap your hands.'''
```

Hint: split the string by a newline and use a for-loop

Regex in Python

`findall()` finds all matches in the source string

Code 4.5

```
str1 = "Where are we? Who are we? Why are we here? We
aren't sure!"

re.findall('we', str1)  # obtain a list
```

Question: How many "we" in the string? How many did we get? Anything missing?

Tutorial

Thus far, what we have done is find an exact pattern from strings

- This is like `.count('sh')` or `.replace('love', 'like')`

In the previous examples, we provided **literal strings**, e.g., :

- /woodchuck/ matches "woodchuck"
- /Woodchuck/ matches "Woodchuck"
- /Woodchucks/ matches "Woodchucks"

Image source: Stick PNG

Regex: Disjunctions

Braces [ ] specify a disjunction.

Patterns	Matches	Examples
`[bkmrs]`	Characters in set	`re.search(r'[bkmrs]ite', ..)`
`[123]`	1, 2, or 3	`re.search(r'[123]', ..)`

In cases where there is a well-defined sequence, use – inside [ ]

Patterns	Matches	Examples
`[A-Z]`	An uppercase letter	`re.search(r'[A-Z]', ..)`
`[a-z]`	A lowercase letter	`re.search(r'[a-z]', ..)`
`[0-9]`	A single digit	`re.search(r'[0-9]', ..)`

Regex: Counters

To go beyond a single character, we need a few counters

Stephen Cole Kleene

Patterns	Matches	Examples	Possibilities
.	Any character but \n	`r'[Tt].'`	To to
+	Previous char. 1+ times	`r'[Bb]a+'`	Ba Baa Baaa ba baa
*	Previous char. 0+ times	`r'man*'`	ma man mann
?	Previous char. 0/1 time	`r'[Cc]olou?r'`	Color color Colour Colour

Question: Are `/ba+/` and `/baa*/` same or different?

Image source: Wikipedia

Regex: Special characters

In many instances, we'd like to match certain types of characters

Patterns	Matches	Examples	Possibilities
`\d`	A single digit	`r'\d+'`	1 12 123 1234 .....
`\w`	An alphanumeric char.	`r'\w+'`	man 120 The .....
`\s`	A whitespace char.	`r'\s'`	Hello_world
`\b`	A word boundary	`r'\b'`	|Green| |idea| |sleeps|

Note: The opposites are: `\D` `\W` `\S` `\B`

Regex: Anchors

Anchor our search patterns to particular places in a string

Patterns	Matches	Examples	Test case: Email
^	Start of source string	`r'^\w+'`	suesakol@staff.tu.ac.th
$	End of source string	`r'\w+$'`	suesakol@staff.tu.ac.th

How about these emails?

Stephen@yahoo.com

123Peter@gmail.com

SS89@georgetown.edu

Lo.31_Cha@staff.tu.ac.th

Regex: Escape characters

If you want a special regular expression character to just behave normally, prefix it with \

Patterns	Matches	Examples
`\$`	A dollar sign	`r'\$[0-9]+'`
`\^`	A carat	`r'a\^b'`
`\+`	A plus	`r'^\+\d+'`
`\.`	A period	`r'\.$'`
`\*`	An asterisk	`r'\*+'`
`\[ \] \( \)`	Braces or parentheses	`r'\(\d+\)'`

Regex: Disjunction # 2

Let's get back to disjunction...

Patterns	Matches	Examples
`[A-Z]`	Any uppercase letter	`r'^[A-Z].+'`
`[^abc]`	Not a, b, or c	`r'^[^abc].+'`
`ab|cd`	ab or cd	`r'cat|dog'`

What will you get with these?

puppy|ies

city|ies

walks|ed

Regex: Parentheses

Parenthesis are not part of the match but establish "groups" inside of the match

Code 4.6

```
str = '''I'd like to go for a walk every day.
I walked 3 kilos yesterday. My friend loves
walking too. She walks a lot.'''
```

Regex: Parentheses

Parenthesis are not part of the match but establish "groups" inside of the match

```
str = '''I'd like to go for a walk every day.
I walked 3 kilos yesterday. My friend loves
walking too. She walks a lot.'''
```

Code 4.6

[Continued]

We want to capture these instances of "walk"

Regex: Parentheses

Let's see an even cooler example of parentheses

Code 4.7

```
head = '''From sakol.suethana@staff.tu.ac.th Sat Dec
12 09:14:16 2015'''
```

Regex: Parentheses

Let's see an even cooler example of parentheses

```
head = '''From sakol.suethana@staff.tu.ac.th Sat Dec
12 09:14:16 2015'''
```

Code 4.7
[Continued]

How can we capture this group "@....."?

Regex: More fun!

Specify a range of characters inside {}

Patterns	Matches	Examples
`\w{m}`	A char. Exactly m times	`r'\w{5}'`
`\w{m,}`	A char. m or more times	`r'\w{3,}'`
`\w{m,n}`	A char. b'w m & n times	`r'$\d{1,4}(\.\d{1,2})?'`

There are a lot more (greedy vs. non-greedy match and positive/ negative lookaround)

Practice

Find all the occurrences of "the" in the following sentence:

The boy said, "Look mom. Behind the glass! The pandas are sleeping in their house!"

ELIZA

ELIZA is an early NLP system that could carry out a conversation!

User: Men are all alike.

ELIZA: **IN WHAT WAY?**

User: They're always bugging us about something or other.

ELIZA: **CAN YOU THINK OF A SPECIFIC EXAMPLE?**

User: Well, my boyfriend made me come here.

ELIZA: **YOUR BOYFRIEND MADE YOU COME HERE**

User: He says I'm depressed much of the time.

ELIZA: **I AM SORRY TO HEAR YOU ARE DEPRESSED**

User: It's true. I am unhappy

ElIZA: **DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY**

User: I need some help, that much seems certain.

ELIZA: **WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP**

User: Perhaps I could learn to get along with my mother.

ELIZA: **TELL ME MORE ABOUT YOUR FAMILY**

User: My mother takes care of me.

ELIZA: **WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU**

User: My father.

ELIZA: **YOUR FATHER**

User: You are like my father in some ways.

Adapted from: Columbia's CS class

ELIZA-style regular expressions

Step 1: replace first person with second person references

```
s/\bI('m| am)\b /YOU ARE/g
s/\bmy\b /YOUR/g
S/\bmine\b /YOURS/g
```

Step 2: use additional regular expressions to generate replies

```
s/.* YOU ARE (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* YOU ARE (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Step 3: use scores to rank possible transformations

Adapted from: Columbia's CS class

ELIZA

Let's see how good ELIZA is: [LINK]

- Read more about ELIZA in L & C Chapter 6 (section 6.7)

Regex in Python

Let's get back to `re` module. This time, we'll look at `split()`

Code 4.8

```
source = '''I go on too many dates
But I can't make 'em stay
At least that's what people say, mm, mm
That's what people say, mm, mm'''

print(re.split(r"[', ]", source))
```

Question: Some things went wrong. What were they?

nltk_regexp_tokenize()

You can use regex to tokenize texts with NLTK:

NLTK Ch.3

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)     # set flag to allow verbose regexps
...     (?:[A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(?:-\w+)*        # words with optional internal hyphens
...   | \$?\d+(?:\.\d+)?%? # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.              # ellipsis
...   | [][.,;"'?():-_`]   # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

See this for even more complex regex patterns!

Finite-state Automata (FSAs)

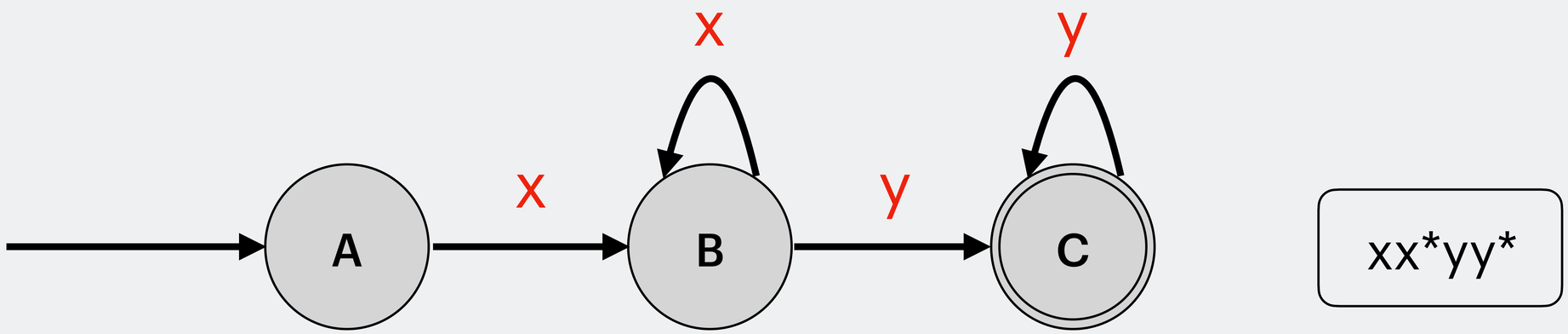
Finite-state automata (FSAs) is a mathematical model:

- that describes one type of language (**regular language**)
- the product of which can be converted to regex (& vice versa)

Finite-state Automata (FSAs)

An automaton comprises four elements:

1. States
2. Initial state
3. Transition rules
4. 1+ final states



Main idea: FSA generates language corresponding to the paths

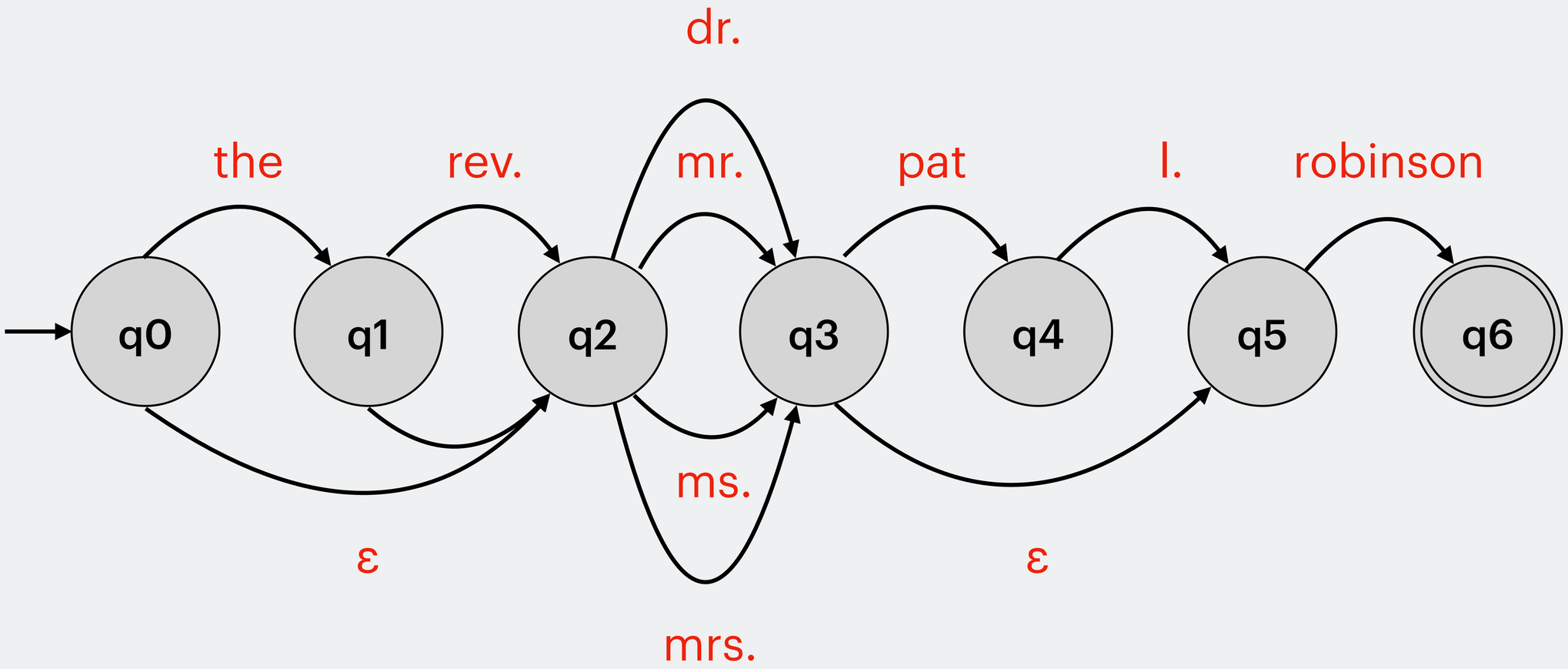
Finite-state Automata (FSAs)

An automaton can be associated with a set of strings it accepts/ generates

- FSAs can be useful tools for recognizing – and generating – subsets of natural language
- But they cannot represent all natural language phenomena

Adapted from: Columbia's CS class

Finite-state Automata (FSAs)



Adapted from: Columbia's CS class

Our plan next week...

- Corpora and Search!
  - List comprehension `[w for w in token if len(w) < 12]`
  - File input and output
- Readings:
  - No reading! 🥳👏