LG 467 Computers in Linguistics

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

Sakol Suethanapornkul

Previously...

- Regular expressions (regex)

^	_____	\d	_____
$	_____	\w	_____
[abc]	_____	\b	_____
[^abc]	_____	\s	_____
a|b	_____	(abc)	_____
.	_____	a{m}	_____
+	_____	a{m, }	_____
*	_____	a{m,n}	_____
?	_____		

Regex in Python

Let's get back to the `re` module. This time, it's `re.split()`

Code 5.1

```
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?

Regex in Python

You can replace one string with another using `re.sub()`

Code 5.1

```
#re.sub(pat, repl, source)

print(re.sub(r"(\w+)?'\w.?", "BAD! ", source))
print(re.sub(r"mm(?!,)", "urgh", source))
```

Regex in Python

Lookarounds help you create custom anchors:

Patterns	Matches	Examples
`(?!pat)`	Anything not followed by pat	`r'\w+(?!\d)'`
`(?<!pat)`	Anything not preceded by pat	`r'(?<!\d)\w+'`
`(?=pat)`	Whatever that is followed by pat	`r'\d+(?=,)'`
`(?<=pat)`	Whatever that is preceded by pat	`r'(?<=-)\d+'`

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

Finite-state Automata (FSAs)

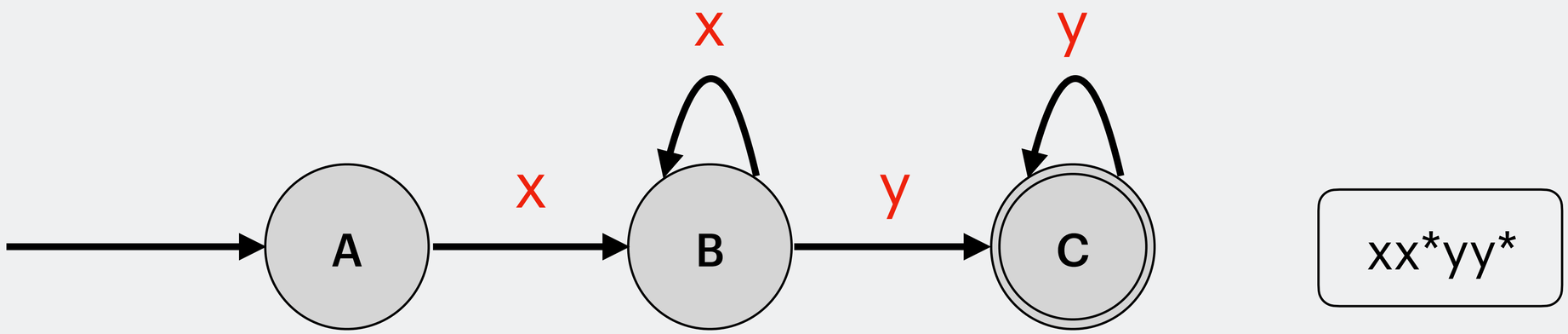
Finite-state automata (FSAs) is a mathematical model:

- 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

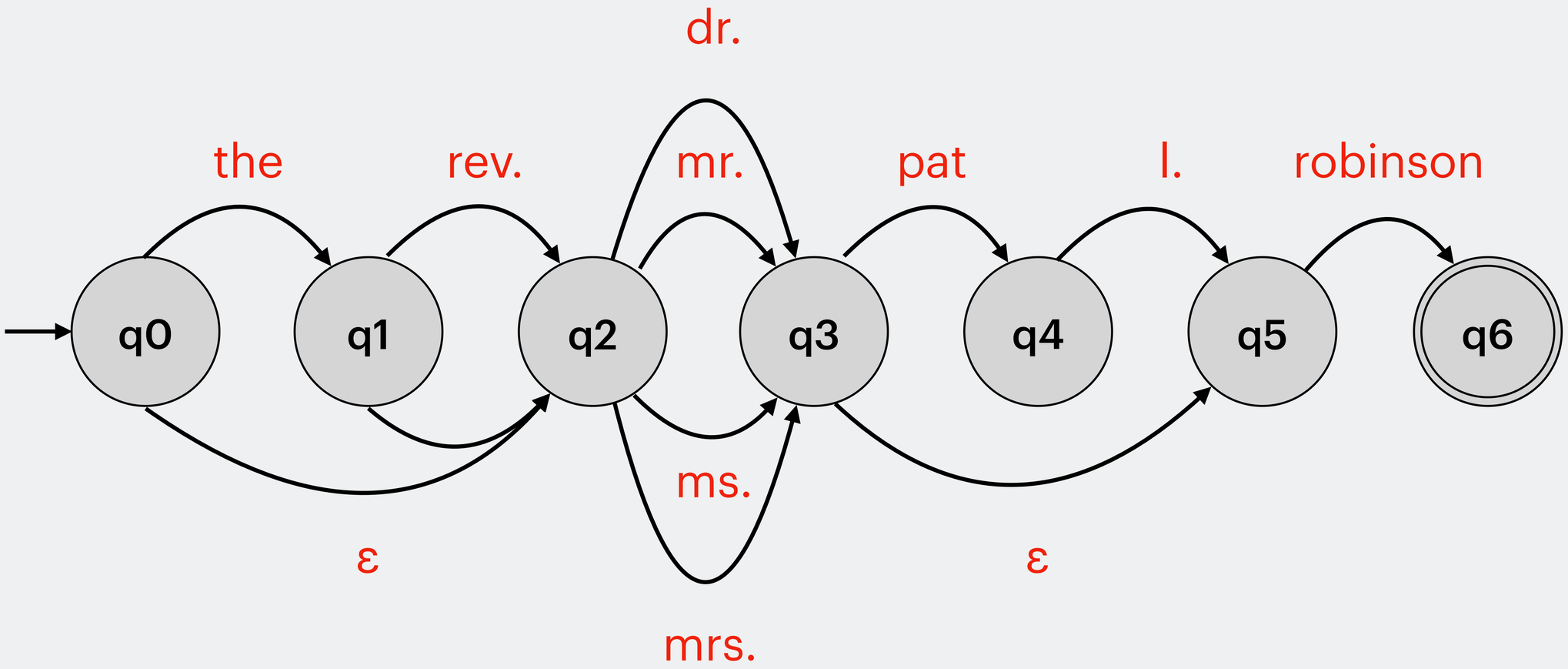
Finite-state Automata (FSAs)

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