CS 784: Computational Linguistics
Lecture 2.1: Regular Expressions

Freda Shi

School of Computer Science, University of Waterloo
fhs@uwaterloo.ca

January 9, 2025

[Some slides adapted from Joyce Chai.]

Why Regular Expressions?

Regular expressions are a powerful tool for string manipulation, with **simpler implementation** and **higher execution efficiency** than general-purpose programming languages.

They are integrated into all modern programming languages.

The Python code below finds all words that contain at least two "*a*"s in the given string.

```
import re

result = re.findall(
    r'\b\w*a\w*a\w*\b',
    'anaconda banana cantaloupe durian elephant'
)
print(result)
```

Implementing the Same Function in Python

```
def find_words_with_more_than_2_a(text):
    words = text.split()
    result = []
    for word in words:
        count_a = word.count('a')
        if count_a > 1:
            result.append(word)
    return result

result = find_words_with_more_than_2_a(
    'anaconda banana cantaloupe durian elephant'
)
print(result)
```

Key Idea of Regular Expressions

A string is a sequence of symbols.

For text-based search, a string is a sequence of alphanumeric characters (letters, numbers, spaces, tabs, and punctuation marks).

Regular expression is a pattern that describes **a set of strings**.

This set of strings can be also referred to as a (formal) **language**.

The language that `r'\b\w*a\w*a\w*\b'` defines is the set of all words that contain at least two "*a*"s.

Inductively Defined Regular Expressions

Regular expressions can be defined inductively.

- The empty string ϵ is a regular expression.
- Any symbol in the alphabet is a regular expression.
- If A and B are both regular expressions, then so are:
  - The concatenation of A and B (A followed by B).
  - The disjunction of A and B (either A or B).
  - The Kleene (pronounced as clay-knee) star of A (repeat A zero or more times).

Special Symbols

In normal regular expressions, the following symbols have special meanings:

Symbol	Meaning/Matches
.	Any character
+ ca+t	One or more of the preceding character 'cat', 'caat', 'caaat', etc.
* ca*t	Zero or more of the preceding character 'ct', 'cat', 'caat', 'caaat', etc.
? ca?t	Zero or one of the preceding character 'ct', 'cat'

More Counting Operators

Numbers in curly braces specify the number of repetitions.

Symbol	Meaning/Matches
{n} ca{2}t	Exactly *n* of the preceding character 'caat'
{n,} ca{2,}t	At least *n* of the preceding character 'caat', 'caaat', 'caaaat', etc.
{,m} ca{,2}t	At most *m* of the preceding character 'ct', 'cat', 'caat'
{n,m} ca{2,4}t	Between *n* and *m* of the preceding character 'caat', 'caaat', 'caaaat'

All counting operators can be used to repeat a regular expression with multiple characters.

Example: (ab)+ matches 'ab', 'abab', 'ababab', etc.

Anchors

Anchors are special characters that match the beginning or end of a string.

Symbol	Meaning/Matches
^	Beginning of a string
^cat	'cat' at the beginning of a string
$	End of a string
cat$	'cat' at the end of a string
\b	Word boundary
\bcat\b	'cat' as a whole word, not like in concatenation

Concatenation

String concatenation between regular expressions A and B is simply denoted by AB.

Regular Expression	**Matches**
Cat	'Cat'
CatDog	'CatDog'

Disjunctions

Brackets [ ] are used to specify a set of symbols.

Any character inside the brackets is a match.

Also supports range matching: any symbol with an ASCII code between the specified range is a match.

Some characters (., +, *, ?, |) escape their special meanings when inside brackets.

Regular Expression	**Matches**
[Cc]at	'Cat', 'cat'
[1234567890]	Any digit
[0–9]	Any digit
[a–zA–Z]	Any English letter
Cat [.]	'Cat.'

See ASCII table at `https://www.asciitable.com/`.

Negations in Disjunctions

$\hat{}$ is the negation symbol in regular expressions.

The caret symbol only means negation when it is the **first character** in **brackets**, otherwise it is the character itself.

Regular Expression	**Matches**
[$\hat{}$Cc]	Neither 'C' nor 'c'
[$\hat{}$0–9]	Any non-digit
[$\hat{}$a–z]	Any non-lowercase letter
[$\hat{}$C$\hat{}$]	Neither 'C' nor '$\hat{}$'
a$\hat{}$b	'a$\hat{}$b'

Common Symbol Sets

There are some predefined common sets of symbols that are often used in regular expressions.

Symbol	Meaning
\d	[0–9], any digit
\D	[^0–9], any non-digit
\w	[a–zA–Z0–9_], any alphanumeric/underscore
\W	[^a–zA–Z0–9_], any non-alphanumeric/underscore
\s	[\t\n\r\f], any whitespace
\S	[^ \t\n\r\f], any non-whitespace

String Disjunctions

Anything inside a single pair of brackets matches a single character, no matter how long the pattern is.

To enable disjunctions of strings, use the pipe symbol `|`.

Regular Expression	**Matches**
`Cat|Dog`	'Cat' or 'Dog'
`a|b|c`	`[abc]`
`[Cc]at|[Dd]og`	'Cat', 'cat', 'Dog', or 'dog'

Expressivity of Regular Expressions

Regular expressions have the same expressive power as **nondeterministic finite-state automata**, which is less powerful than context-free grammars.

For example, there's no regular expression recognizing $\mathcal{L} = \{a^n b^n \mid n \geq 0\}$.

See more examples and discussions in Notre Dame CSE 30151 Unit 1 and CS 360.

Useful Python Functions

```
import re
```

- `re.search(pattern, string)`
  Search for the pattern anywhere in the string.
- `re.match(pattern, string)`
  Match the pattern at the beginning of the string.
- `re.split(pattern, string)`
  Split the string by the pattern.
- `re.sub(pattern, replacement, string)`
  Replace the pattern with the replacement in the string.
- `re.findall(pattern, string)`
  Find all occurrences of the pattern in the string.
  Caveat: once a pattern is found, the next search starts from the end of the previous match—for example,
  `re.findall(r'(ab)+', 'abab')` will return `['ab']`.

Complexity of Regular Expressions

The complexity of regular expressions depends on

1. The complexity of the regular expression itself;
2. The algorithm used to match the regular expression.

- Deterministic Finite Automata (DFA) can match regular expressions in $\mathcal{O}(n)$ time, where n is the length of the string;
  Constructing a DFA from a regular expression is $\mathcal{O}(2^m)$, where m is the number of symbols in the pattern.
- Non-deterministic Finite Automata (NFA) can match regular expressions in $\mathcal{O}(nm)$ time.
- Backtracking algorithms can match many simple patterns in $\mathcal{O}(n)$, and complex regular expressions (like (a|aa)*b) in $\mathcal{O}(2^n)$ time.

See more in Notre Dame CSE 30151 Unit 1 and CS 360.

Revisiting the Example on Slide 2

```
import re

result = re.findall(
    r'\b\w*a\w*a\w*\b',
    'anaconda banana cantaloupe durian elephant'
)
print(result)
```

Regular Expression	Meaning
\b	Word boundary
\w*	Zero or more alphanumeric characters
a	The letter ‘a’

Next

Probability and Basic Information Theory