Regular Expressions

Regular Expression

- Represent Regular Language.
- Regular language is a language which is accepted by Finite Automata (FA)
- Regular expression is method to represent a language
- Let L = { ε, a, aa, aaa, …..} a language
- Then a* is a Regular Expression representing the above language.

Regular Expression

- Regular expressions can be used to perform all types of **text search** and **text replace** operations.
- A regular expression is a sequence of characters that forms a **search pattern**.
- Regular expressions helps to represent finite rules of a language from which infinite sentences can be created.

Regular Expression

- Regular Expression rules

- Let ‘R’ be a Regular Expression over alphabets Σ if R is
  1) ‘ε’ is a Regular expression denoting the set {ε} => Null string
  2) ‘Φ’ is Regular expression denoting the empty set {} **=>** Null set
  3) For each symbol a ∈ Σ, a is a regular expression denoting set {a}
  4) Union of Regular expression is also Regular expression
  5) Concatenation of two Regular expression is also Regular expression
  6) Kleene closure * of Regular expression is also regular expression
  7) The regular expression over Σ are precisely those obtained recursively by the above rules once or several times.

Basic Regex Patterns

- By default, all major regex engines match in case-sensitive mode.
- **Anyone from list []:**
- Matches any of a set of characters inside square brackets
- At least one character from the list given in []
- For Example
- **‘[abc]d’** – any character from **‘abc’** and then ‘**d’**
- **‘[cC]ake’** – any character from **‘cC’** and then ‘**ake’**
- **‘[0123456789].00’** – any digit and then .00

Shortened Sequence (-):

- When all the characters are in a sequence use -, as **[a-z], [A-Z],[0-9]**

- **Caret (^):**
- [^apt] i.e., any character other than a, p or t
- [^a-z] i.e., any character other than small case character

- Has to be the first character otherwise taken as caret
- [e^] i.e., either e or ^
- a^b i.e., sequence a^b

Optional characters (?):

- colou?r i.e., color with or without u
- Books? i.e., book or books

- **Kleene *:**
- We may generally need more than one character of a character
- Means zero or one or many number of the immediately previous character
- E.g., baa* i.e., can have any number of a’s at the end

a*:

- **Pakistan**
- **baaaaaaaa**
- **Cricket**

- **[ab]*:**
- We may need to repeat complex combinations • E.g., ababab or abbbbb or bbbbbb or aaaaa etc.
- [1-9][0-9]* i.e., price for something that may be 1 or more

Kleene +:

- Any number of characters but at least 1 e.g., price [1-9]+

- **Wildcard (.):**
- beg.n i.e., having any single character after g and before n

- **aardvark.*aardvark:**
- **Looking for two occurrences of the same word (aardvark)**

Anchors:

- They anchor regular expressions to a particular places in a string
- Caret (^) suggests the start of a line e.g., ^The indicates words starting with “The”

- Dollar ($) suggest the end of a line e.g., The dog\.$ indicates the line ending with period
- Boundary (b) limits the before or after characters e.g., \bthe\b means the and not other or their etc.

It helps you define your language with a set of rules for example \b99\b means that 99 is a word but not 299!

Disjunction |:

- Sometimes we would want one piece of string or another, and is specified as cat | dog i.e., having cat or dog

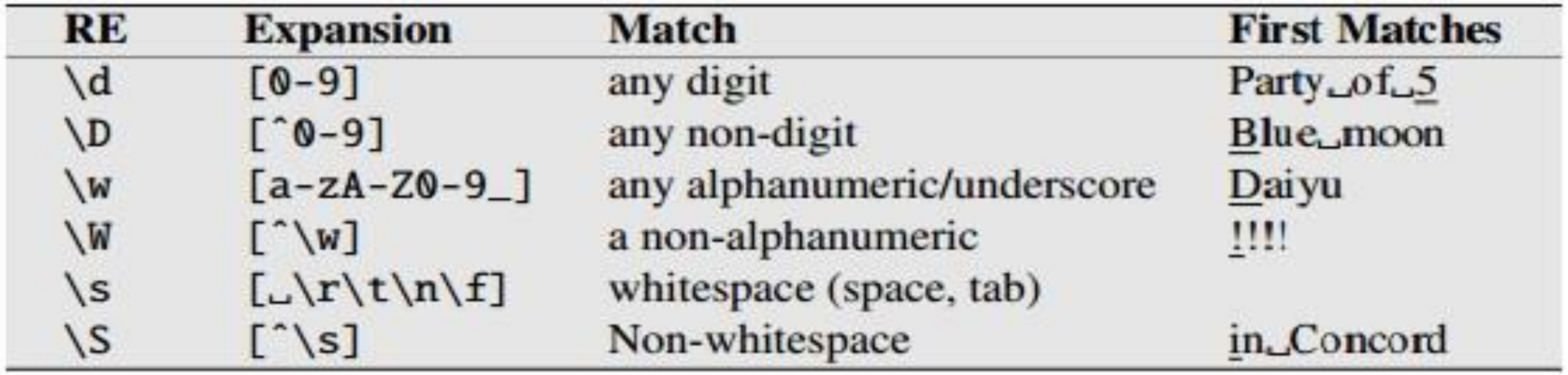
- **Grouping and precedence:**
- Using disjunction along with other strings we may want one or other form of it. It can be grouped with different character sequences using () e.g., if we
  want either guppy or guppies we can write gupp(y | ies)

Referring to an earlier occurrence of instance

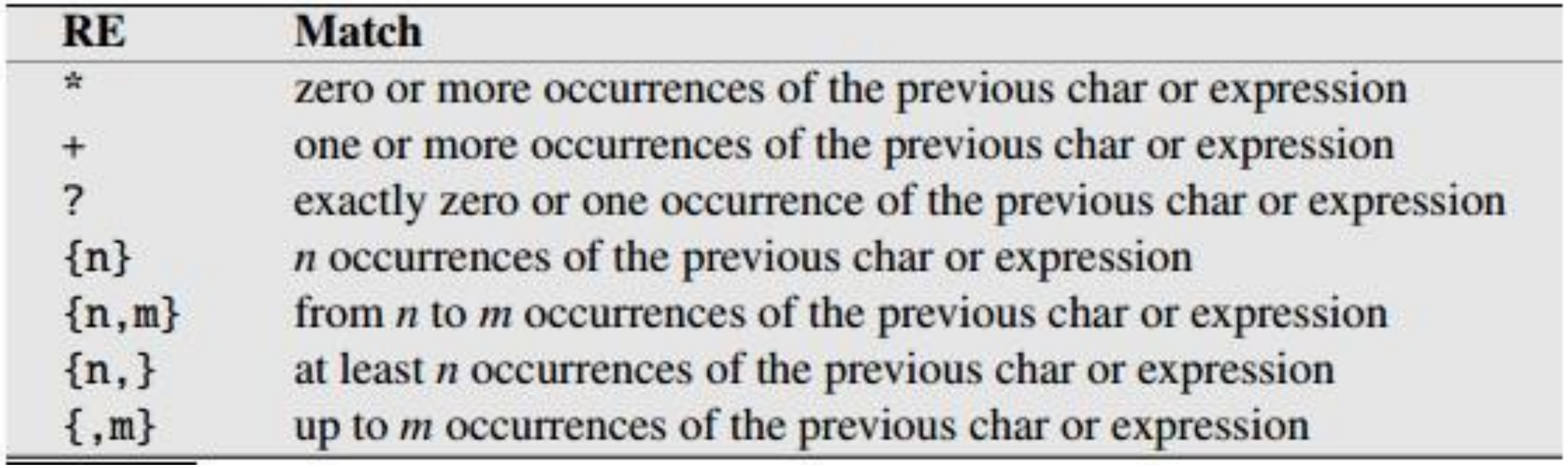
- the (.*)er they were, the \1er they will be
- \1 referring to what is used in the parenthesis
- **Multiple capture groups**
- /the (.*)er they (.*), the \1er we \2/

- \1 refers to what has occurred in the first group, while \2 refers to what has occurred in the second group

Other operators

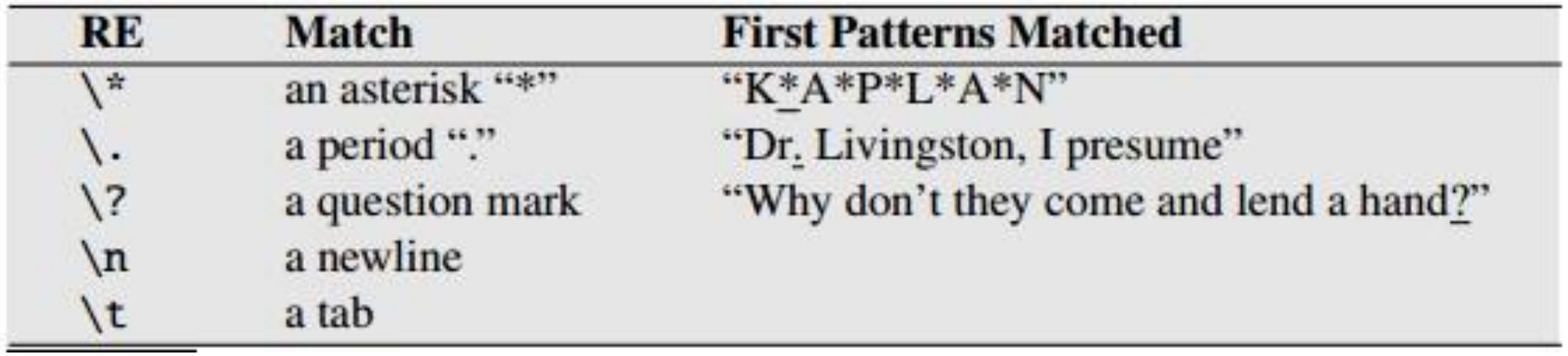


RE	Expansion	Match	First Matches
\d	[0-9]	any digit	Party␣of␣5
\D	[^0-9]	any non-digit	Blue␣moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[␣\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in␣Concord



RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression
{,m}	up to *m* occurrences of the previous char or expression

Regex to Match Symbols



RE	Match	First Patterns Matched
`\*`	an asterisk "*"	"K*A*P*L*A*N"
`\.`	a period "."	"Dr. Livingston, I presume"
`\?`	a question mark	"Why don't they come and lend a hand?"
`\n`	a newline	
`\t`	a tab	

Regular Expressions	**Regular Set**
(0 + 10*)	L = { 0, 1, 10, 100, 1000, 10000, … }
(0*10*)	L = {1, 01, 10, 010, 0010, …}
(0 + ε)(1 + ε)	L = {ε, 0, 1, 01}
(a+b)*	Set of strings of a’s and b’s of any length including the null string. So L = { ε, a, b, aa , ab , bb , ba, aaa…….}
(a+b)*abb	Set of strings of a’s and b’s ending with the string abb. So L = {abb, aabb, babb, aaabb, ababb, …………..}
(11)*	Set consisting of even number of 1’s including empty string, So L= {ε, 11, 1111, 111111, ……….}

(aa)*(bb)*b	Set of strings consisting of even number of a’s followed by odd number of b’s , so L = {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, …………..}
(aa + ab + ba + bb)*	String of a’s and b’s of even length can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null, so L = {aa, ab, ba, bb, aaaa, abab…………..}

Practice

It certainly was ***the*** thing I was looking for. It has aes***the***tic appearance and is very subtle at controlling different channels. ***The*** o***the***r thing I appreciate about it is its ***the***mes ***the*** app and ***the***255.

- the
- [tT]he
- \b[tT]he\b
- [ˆa-zA-Z][tT]he[ˆa-zA-Z]

Practice

- $[0-9]+
- $[0-9]+\.[0-9][0-9]
- (ˆ|\W)$[0-9]+(\.[0-9][0-9])?\b
- (ˆ|\W)$[0-9]{0,3}(\.[0-9][0-9])?\b
- \b[6-9]+ *(GHz|[Gg]igahertz)\b
- \b[0-9]+(\.[0-9]+)? *(GB|[Gg]igabytes?)\b

C++ Regex

- Import the regex module `#include<regex>`
- https://en.cppreference.com/w/cpp/regex

Regular Expressions (Applications)

- Chatbots
- Text normalization
- Tokenization
- Stemming
- Lemmatization
- Grammar correction
- Sentences generation
- Text/Content verification