Word and Phonetics

- **Regular Expressions**
- **Automata**

Regular Expressions

Regular Expressions

- Each **Regular Expression (RE)** represents a set of strings having certain pattern.
  - In NLP, we can use REs to find strings having certain patterns in a given text.
- Regular Expressions are an algebraic way to describe formal languages.
  - Regular Expressions describe exactly the regular languages.
- A regular expression is built up of simpler regular expressions (using defining rules).
- Simple Definition for Regular Expressions over alphabet Σ
  - ε is a regular expression
  - If $\mathbf{a} \in \Sigma$, $\mathbf{a}$ is a regular expression
  - **or :** If E_1 and E_2 are regular expressions, then $\mathbf{E_1 \mid E_2}$ is a regular expression
  - **concatenation** : If E_1 and E_2 are regular expressions, then $\mathbf{E_1 E_2}$ is a regular expression
  - **Kleene Closure:** If E is a regular expression, then $\mathbf{E^*}$ is a regular expression
  - **Positive Closure:** If E is a regular expression, then $\mathbf{E^+}$ is a regular expression

Searching Strings with Regular Expressions
(*using Python style REs*)

- How can we search for any of following strings?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks
- The **simplest kind of regular expression** is a sequence of simple characters.
  - The regular expression **b** will match with the string "b".
  - The regular expression **bc** will match with the string "bc".
  - The regular expression **woodchuck** will match with the string "woodchuck".
  - The regular expression **woodchucks** will match with the string "woodchucks".
  - The regular expression **woodchuck** will NOT match with the string "Woodchuck".

Regular Expressions: Disjunctions disjunction of characters []

- **Disjunction of Characters:** The **string of characters inside the braces [ ]** specifies a **disjunction** of characters to match.

- The regular expression **[wW]** matches patterns containing either w or W.

Regular Expression	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- **Ranges in []:** If there is a well-defined sequence associated with a set of characters, **dash (-)** in brackets can specify any one character in a **range**.

Regular Expression	Matches
`[A-Z]`	An upper case letter
`[a-z]`	A lower case letter
`[0-9]`	A single digit

Regular Expressions:Disjunctions Negations in []

- **Negations in []:**
  - The square braces can also be used to specify what a single character cannot be, by use of the caret ^.
  - If the caret ^ is the first symbol after the open square brace [, the resulting pattern is negated.

Regular Expression	Matches
`[^A-Z]`	Not an upper case letter
`[^a-z]`	Not a lower case letter
`[^Ss]`	Neither ‘S’ nor ‘s’
`[^e^]`	Neither e nor ^
`a^b`	The pattern **a^b**

Regular Expressions: Disjunctions

or (disjunction) operator | (pipe symbol)

- If E_1 and E_2 are regular expressions, then $\mathbf{E_1 \mid E_2}$ is a regular expression

Regular Expression	Matches
`woodchuck|groundhog`	**woodchuck** or **groundhog**
`a|b|c`	**a** , **b** or **c**
`[gG]roundhog|[Ww]oodchuck`	**woodchuck** , **Woodchuck** , **groundhog** or **Groundhog**
`fl(y|ies)`	**fly** or **flies**

Regular Expressions: Closure Operators Kleene * and Kleene +

- **Kleene * (closure) operator:** The Kleene star means “zero or more occurrences of the immediately previous regular expression.
- **Kleene + (positive closure) operator:** The Kleene plus means “one or more occurrences of the immediately preceding regular expression.

Regular Expression	Matches
`ba*`	**b, ba, baa, baaa, ...**
`ba+`	**ba, baa, baaa, ...**
`(ba)*`	ε, **ba, baba, bababa, …**
`(ba)+`	**ba, baba, bababa, …**
`(b|a)+`	**b, a, bb, ba, aa, ab, …**

Regular Expressions: {} . ?

- **{m,n}** causes the resulting RE to match from m to n repetitions of the preceding RE.
- **{m}** specifies that exactly m copies of the previous RE should be matched
- The question mark **?** marks **optionality of the previous expression**.

Regular Expression	Matches
`woodchucks?`	**woodchuck** or **woodchucks**
`colou?r`	**color** or **colour**
`(a|b)?c`	**ac, bc, c**
`(ba){2,3}`	**baba, bababa**

- A wildcard expression dot . matches any single character (except a carriage return).

Regular Expression	Matches
`beg.n`	**begin, begun, begxn, …**
`a.*b`	any string starts with a and ends with b

Regular Expressions: Anchors ^ $

- **Anchors** are special characters that anchor regular expressions to particular places in a string.
- The **caret ^** matches the start of a string.
  - The regular expression ^The matches the word **The** only at the start of a string.
- The **dollar sign $** matches the end of a line.

Regular Expression	Matches
`.$`	any character at the end of a string
`\.$`	dot character at the end of a string
`^[A-Z]`	any uppercase character at the beginning of a string
`^The dog\.$`	a string that contains only the phrase **The dog.**

Regular Expressions: Precedence of Operators

- The order precedence of RE operator precedence, from highest precedence to lowest precedence is as follows

  – Parenthesis ()
  – Counters * + ? {}
  – Sequences and anchors ^ $
  – Disjunction |

- The regular expression **the*** matches **theeeee** but not **thethe**
- The regular expression **(the)*** matches **thethe** but not **theeeee**

Regular Expressions: backslashed characters

- Aliases for common sets of characters

RE	Expansion	Match
\d	[0-9]	any digit
\D	[^0-9]	any non-digit
\w	[a-zA-Z0-9_]	any alphanumeric/underscore
\W	[^\w]	a non-alphanumeric
\s	[␣\r\t\n\f]	whitespace (space, tab)
\S	[^\s]	Non-whitespace

- Special characters need to be backslashed.

RE	Match
*	an asterisk "*"
\\.	a period "."
\?	a question mark
\n	a newline
\t	a tab

Regular Expressions: Example

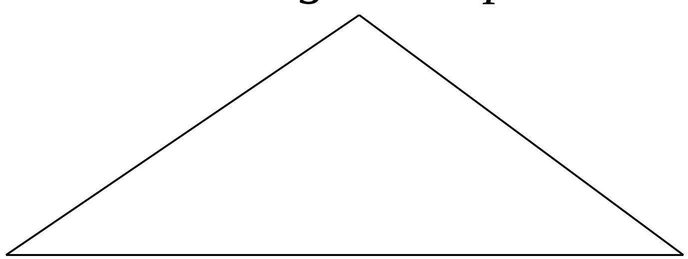
- We want to write a RE to find cases of the English article **the**

Regular Expression	Matches
`the`	this pattern will miss the word **The**
`[tT]he`	this pattern will still incorrectly return texts with the embedded in other words (e.g.,**other** or **theology**).
`[^a-zA-Z][tT]he[^a-zA-Z]`	But there is still one more problem with this pattern: it won't find the word the when it begins a line.
`(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)`	We can avoid this problem by specifying that before **the** we require either the beginning-of-line or a non-alphabetic character, and the same at the end of the line:

Regular Expressions & FSAs

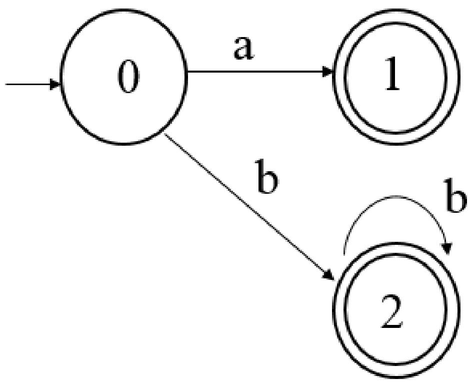
- Any regular expression can be realized as a **finite state automaton (FSA)**
- There are two kinds of FSAs
  - Deterministic Finite state Automatons (DFAs)
  - Non-deterministic Finite state Automatons (NFAs)
- Any NFA can be converted into a corresponding DFA.
- A FSA (a regular expression) represents a **regular language**.

Regular Expressions

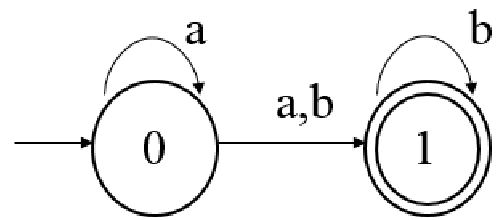


Finite Automata *Regular Languages*

Regular Expressions: A DFA and A NFA



A DFA : $a \mid b^+$



A NFA: $a^*(a|b)b^*$

Regular Expressions: Regular Languages

- Operations on regular languages and FSAs:
  - concatenation, closure, union
- Properties of regular languages
  - closed under concatenation, union, disjunction, intersection, difference, complementation, reversal, closure.
- Equivalent to finite-state automata.

Formal Definition of Finite-State Automaton

- FSA is $Q \times \Sigma \times q_0 \times F \times \delta$
- Q: a finite set of N states $q_0, q_1, \ldots q_N$
- Σ: a finite input alphabet of symbols
- q_0: the start state
- F: the set of final states -- F is a subset of Q for NFAs
- $\delta(q,i)$: transition function
  - DFA : There is exactly one arc leaving a state q with a symbol a. There is no arc with the empty string.
  - NFA : There can be more than one arc leaving a state q with a symbol a. There can be arcs with empty string.