Natural Language Processing [NLP]

- Earlier it was also known as Text Analytics, and as the name suggest, we get a lot of text data, we try to make sense of this data, by doing loads of analytics on it.
- Today, the amount of text data being generated is a lot, through social media, websites, blog posts etc, Major outlook is how to make sense of this data.

→ Some applications :

- Social Media [post analysis] and segmentation and labeling for better feeds.
- for [analysis resumes] SOPs for automatic acceptance and rejection.
- also applicable in analyzing requests for loan in banking sector, insurance claiming process etc.

→ The goal of this nlp field is to get computers to perform useful tasks related to [human language], like human-machine and communication.

→ Engaging in complex language behaviour requires various kinds of knowledge of language.

- Phonetics : scientific study of [speech sounds], focusing on how they are studied, transmitted and perceived. (physical and acoustic properties of sound)
- Phonology : studies [the way speech functions within a language], focusing on patterns and system of sounds. (rules & mental representation of sound)
- Morphology : subfield of linguistics that studies the [structure and formation of words]
  - It focuses on how [smaller units of meaning] called morphemes, are combined to create words.
  - smallest unit of meaning in a language.
- Syntax : branch of linguistics that studies the [structure of sentence] and rules governing the arrangement of words & phrases to [form meaningful sentences] eg- Subject Verb Agreement.
- Semantics : It is study of meaning, focuses on how words, phrases and texts [convey meaning] both individually and in combination.

(3)

- **Pragmatics** : Studies how context influences the [interpretation] of meaning in communication. unlike semantics which focuses on [literal meaning] of words & sentences. Pragmatics considers speaker's [intentions and situational context.]

Discourse : Studies how sentences and larger units of language (paragraphs, speeches) are organised to create [meaningful communication].

⇒ Ambiguity in NLP :

- It refers to presence of multiple interpretations for a word, (phrase or sentence.)
- It arises due to inherent complexity of human languages, also that so much of it is (context dependent.)
- This ambiguity leads to (challenges) in accurately interpreting and processing language in NLP applications.
- whole field of speech and language processing is based on (solving these ambiguities at various levels.)

(4)

⇒ Lexical Processing

- It is a fundamental concept in NLP, which involves breaking down and analyzing text to (understand) its structure and meaning

- This is kind of first step in NLP, where text is [segmented into smaller units called lexemes], this can include sentences, phrases and words.

- It also involves [tokenization], which converts raw text into (manageable tokens)

→ Morpheme Identification: Lexical analysis also includes identifying morphemes (smallest unit of language)

- free morphemes: words that can stand alone
  eg: cat

- bounded morphemes: units that cannot stand alone, must attach to a (free morpheme) eg: (-ing) (un-)

→ Lemmatization: reducing words to their base form

→ [Stop word removal]: eliminating common words that do not add significant meaning.

(part of speech)

→ Part of Speech Tagging (PoS Tagging):

- involves categorizing words in their respective parts of speech (nouns, verbs etc), which is crucial for understanding grammatical structure and relationships within sentences.

⇒ Syntactic Processing

- It is focused on analysing the (grammatical) structure of sentences.
- This process helps in understanding how words combine to form meaningful phrases and sentences, which is essential for various NLP applications.

→ Tokenization : The first step involves breaking down text into individual tokens, each can be a word or punctuation punctuation mark.

→ PoS Tagging : as previously stated, each token from above is assigned a grammatical category, such as noun, verb, adjective or adverb.
This tagging helps in understanding the role of each word.

→ Parsing : This step involves constructing a [parse tree or abstract syntax tree, that will represent the grammatical structure of sentences.
Tree is used to illustrate how words are grouped and related to each other according to grammer ~~rule~~ rules.

→ Ambiguity Resolution : Syntatic processing can aid resolving these ambiguities.

✓ • Constituency Parsing : Identifies how groups together to form constituents (phrases)

✓ • Dependency Parsing : focuses on the relationships b/w individual words that depend on each other.

⇒ Semantic Processing :

- It focuses on understanding the meanings, and interpretations of words, phrases and sentences (within their context)
- uncovering the intended meaning behind text.

→ Meaning Extraction : as semantic processing involves in extracting meaning from word & phrases, this includes [recognizing synonyms, autonyms and polysemys] (words with multiple meanings).

→ Contextual Understanding : understanding context is vital in (semantic understanding), here algorithms analyze surrounding text to derive more accurate meaning.

→ Relationship Identification : It involves identifying relationships between different elements in text. This can include relationships such as cause and effect, comparisons or hierarchies.

→ Thematic Role Labeling : it involves identifying roles that different entities play within a phrase or sentence (who did what to whom etc).
This helps to in understanding action and their participants.

⇒ Regular Expressions

- regex for short, is a syntax that allows you to [match strings with specific patterns]
- regex provides a powerful and efficient way to process, extract and manipulate (textual data.)

→ Quantifiers :

- a | b : match either "a" or "b"
- ? : zero or one → a? → a, null
- + : one or more → a+ → a, aaa....
- * : zero or more → a* → null, a, aaa...
- {N} : exactly N no. of times. → a{3} → aaa
- {N,} : N or more no of times → a{3,} → aaa, aaaa....
- {N, M} : Between N and M, no. of times, where N < M.

a{3,5} → aaa, aaaa, aaaaa

- *? : Zero or more, but stop after first match.

→ Examples of Quantifiers usage :

- Hello | Goodbye : matches both strings, hello and goodbye.
- Hey? : "y" can be zero to one time will match 'He' & 'Hey'

- Hello{1,3} : will match "Hello", "Helloo", "Hellooo"
  will look for letter "o" from 1 to 3 times

- He?llo{2} : will match for only "Hlloo" and and "Helloo"

- Hi+ : this match for "Hi" to multiple 'i' behind 'Hi'. so, "Hiiiiii" is also a match.

- H.*llo :
  [. is used for any character.] (filler)
  eg: Hillo, Hello, Hellohello
  H .* llo

- H.*?llo : only gets Hillo & Hello.

→ Common Pattern Collections.

[A-Z] : matches any uppercase letters from "A" to "Z"

[a-z] : "

[0-9] : "

[abc] : match only character that is a, b or c

[^abc] : opposite of above.

(ba+)* ba (ba+)*

→ General Tokens :

. → any character

\n → newline character

\s → any whitespace character including \n & \t

\t → tab character

\S → any non whitespace character.

\w → any word character
↳ uppercase, lowercase, 0-9 & _

\W → any non word character.

\ → literal character "\"

^ → start of line

$ → end of line

\d → any digit characters

\D → non digit characters.

→ Examples :

^I → will return true if it starts with I

I$ → will return true if it ends with I.

Note : To extract ? from text, use \? to extract

To extract \ from text, use \\

→ Greedy & Non greedy approach

- (Greedy Quantifiers) attempts to match as ~~much~~ many characters as possible while still allowing overall regex pattern to succeed.

  string : abbbbbb

  pattern : ab{2,5}

  returns : abbbbb

- [Non greedy Quantifiers] or lazy quantifiers match as few characters as possible. This is useful when you want to capture smallest possible substring that still satisfies the regex pattern.

  returns : abb

⇒ Finite State Automata

- These are mathematical models of computation that are used to represent and control execution flow in ~~statements~~ systems.
- They consist of finite number of states, transitions between those states, and input symbols that trigger these transitions

→ Key Characteristics:

- States: An FSA can be in exactly one state at any given time. Here the no. of states is finite, which is a (defining characteristic) of finite state automata.
- Transitions: The movement from one state to another is determined by a [transition function], which is based on the current state and the input symbol being processed.
- Input Symbols: FSAs process input [sequence symbol by symbol] transitioning between states according to the (defined rules.)

→ FSAs form the basis for understanding (regular languages), which are essential in text processing and [compiler design]

→ (Protocol Design): used to design [communication protocols] and [network protocols], ensuring systems (respond correctly to (various inputs))

⇒ Deterministic FSA

- Specific type of FSA, which has more strict rules, ensuring that for each state and input symbol, there is exactly (one possible transition to next state.)
- For every state and input symbol, there is a single defined transition to another state.

$x \to y$
$y \to z$ ×2

- A DFA can be formally defined by as a:

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q : finite set of states

Σ : finite set of input symbols

δ : It is the transition function mapping $Q \times \Sigma$ to Q

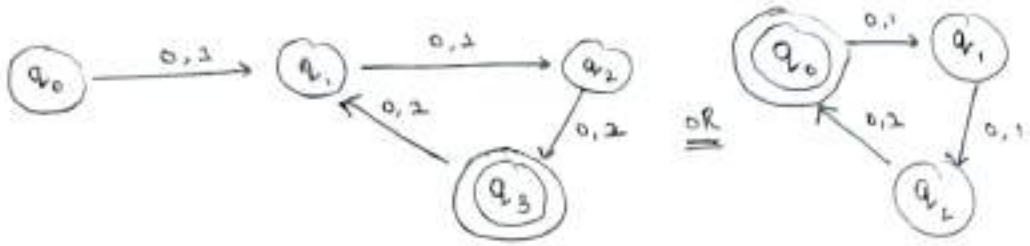
q_0 : initial state

F : set of accepting states.

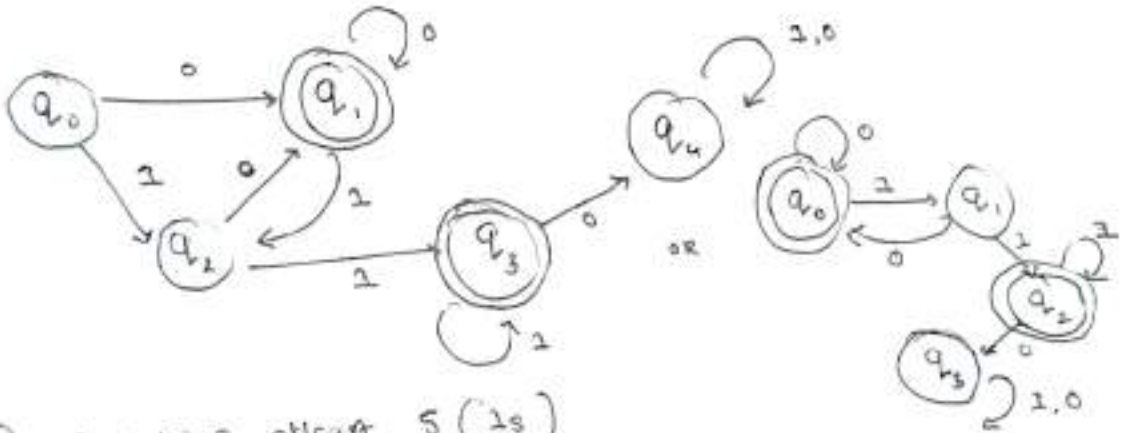
(13)

- A DFA processes input strings symbol by symbol, transitioning between states according to the defined rules.

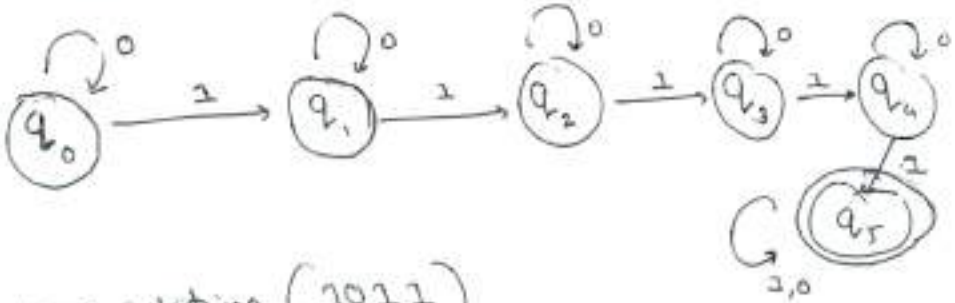
(1) → divisible by 3 (ω) {0,1}



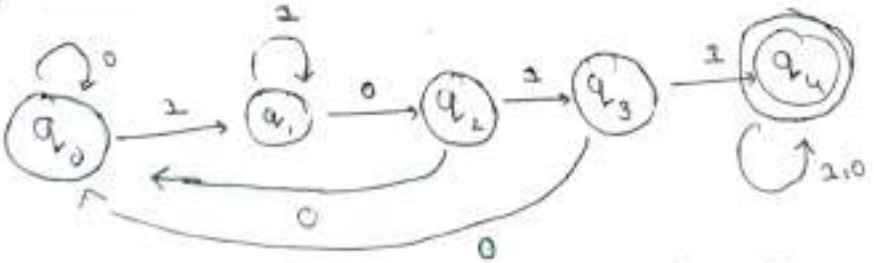
(2) 110 is not a substring of (ω)



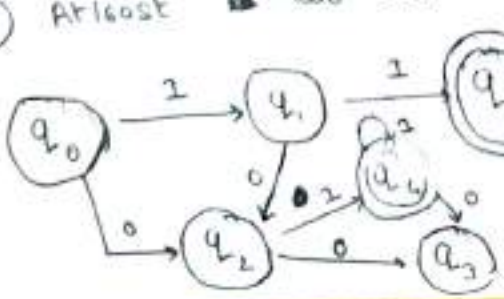
(3) ω contains atleast 5 (1s)



(4) contains substring (1011)



(5) Atleast ● Two 1s and atmost two 0s



- In an attempt to reduce ambiguities, we try to resolve each word down to its [smallest form]
- Here, we will discuss, how can be differenciate plural from singular, or like ◆ break down plural into its singular form.
- The problem of recognizing that a word (like foxes) break down into [component morphemes] (fox and -es) building a structured representation of this fact is called [morphological parsing.]

→ Morphology : it is study of [internal structure of words] and how they are formed from smaller and meaningful units called [morphemes.]

- It involves understanding the relationships, between grammatical information, meaning and the form of words.

→ Key Aspects of Morphology

Lexicon : A dictionary of root words, their categories (noun, adjective etc) and subcategories (single, plural) and affixes that can be attached to them.

affixes → [an addition to the base form or stem of a word in order to modify its meaning or create a new word.]

- Orthographic rules : rules that govern correct spelling and writing of words.
  like rules telling to change words ending in "-y" to "ies" in plural.

- Morphotactics : rules for [placing morphenes with stems to form meaningful words]
  A stem is any morphene or collection of morphenes to which an affix could be added.

→ Stemming : Technique in NLP to reduce words to its base form or to its root by (removing suffixes)
running, runs, ran → run

→ Lemmatization : it reduces the word to its root form (lemma) by considering the meaning and grammer of word
- It uses kind of a dictionary based approach and requires (PoS) tagging (to convert word correctly.)
  Part of Speech

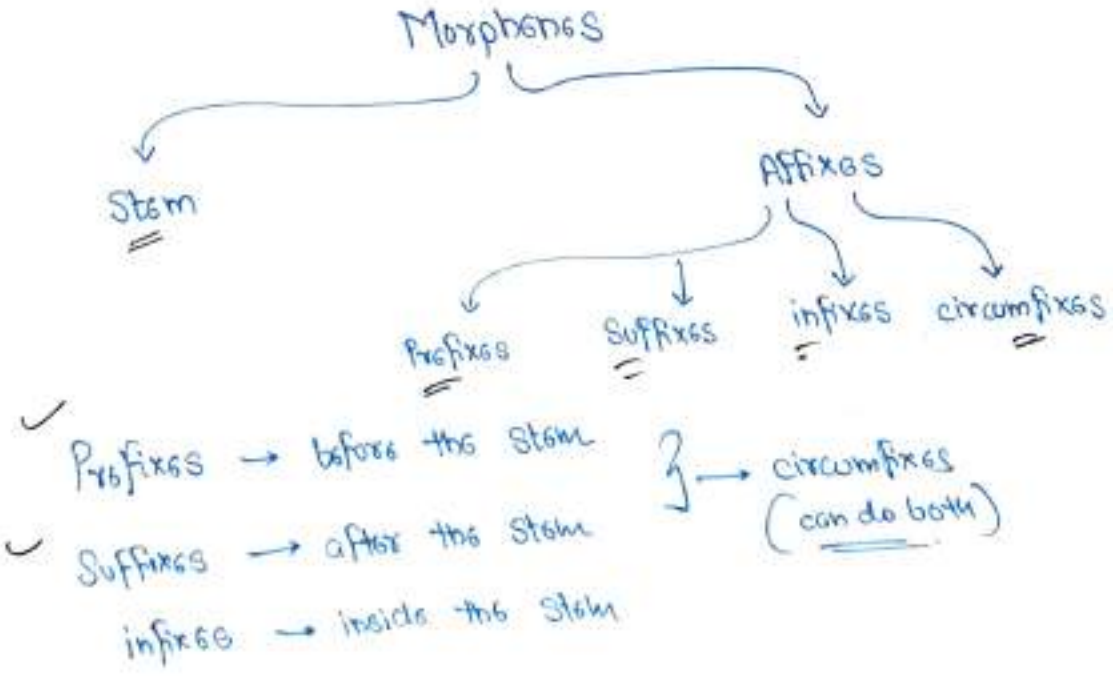
word	Stemming	lemmatization
running	run	run
flies	fli → non real word	fly
better	better	good

→ Stemming, as it just removes suffixes, it may also provide sometimes [non real words].

→ while, lemmatization always produces actual words and is also [context aware] with help of (PoS)

→ Stemming is used when (speed is more important than accuracy, like search engine, quick text filtering etc.

lemmatization is opposite, it is used when [accuracy is crucial] in applications like chatbot, sentiment analysis.



Prefixes → before the stem
Suffixes → after the stem
infixes → inside the stem
} → circumfixes (can do both)

Eg: The word "unbelievably"

Stem: believe

affixes: un, able, -ly → suffix.
prefix infix

- English does not stack more than 4 to 5 affixes together, while some languages like Turkish can have words with more than 10 affixes

- Languages that string multiple affixes together are called agglutinative languages.

→ The process of combining morphemes to form words is crucial, among various methods, four key processes are:

1. Inflection
2. Derivation
3. Compounding
4. Cliticization

1. Inflection Morphology:

- Inflection modifies a word to express [grammatical relationships] without changing core meaning of the word.

- does not create a new word, but alters the form of existing one, category of original word also remains same: noun stay noun, verb stays verb.

- often used in tense (Tense changes) no., case etc

Base word	Inflected form	Function
walk	walked	past tense
dog	dogs	plural form
happy	happier	comparitive

2. Derivation Morphology

- Derivation forms a new word with a new meaning by adding [prefixes and suffixes (affixes)]
- It often changes [word's grammatical category]
  eg : noun to verb, adjective to noun.
- creates a new lexeme (a word with distinct meaning) and will also (change PoS form of word.)

Base word	Derived Form	Word Class Change.
Happy	Happiness	adjective → noun
Compute	computer	verb → noun
act	active	verb → adjective

3. Compounding Morphology

- Compounding combines two or more independent words to form a single new word with a unique meaning.
- The meaning of compound cannot be predicted from its parts
- can be written as one word, hyphenated or seperate words

First word	Second word	Compound word
Sun.	Flower .	sunflower -
black .	board .	blackboard .
life ·	cycle ·	lifecycle .

4. Cliticization

- It occurs when clitics, which are grammatical elements that cannot stand alone, attach to another word
- phonologically dependent, grammatically independent.
- common in spoken language and informal writing

Base Form	Clitic ized Form
I am	I'm
they have	they've

Part - of - Speech (PoS)

- PoS Tagging is a fundamental task in NLP that involves assigning a grammatical category to each word in a text.
- These categories include :

→ Noun : a word that names a person, place, thing or idea

- They can function as [subject or object] in a sentence
- They can be further classified into (proper & common) nouns

→ Verb : a word which describes an [action or state] (run, is, think)

- verbs are central to predicates of a sentence, as they determine what the (subject) is doing.
- Verbs can be transitive (requiring an object) or intransitive (not requiring an object)

→ Pronoun : A pronoun replaces a noun

- Pronouns help to [avoid repetition] and can refer to specific nouns in a sentence
- can be classified into personal, demonstrative, interrogative and relative pronouns.

- Pronouns are a significant challenge in NLP, as it involves in determining which noun a pronoun refers. This is crucial in maintaining (coherence)
  (Quality of logical or consistency)
  in text understanding.

→ Preposition : it shows relationships between words
  Eg : on, under, with
  - provides [additional information] on about time, location or direction.
  - They connect nouns or pronouns to other words in a sentence.

→ Adverb : it provides (additional information) about a verb or adjective.
  Eg. Quickly, very
  - Adverbs provide context about how, when, where, or to what extent an (action) occurs.
  - They can enhance meaning of verbs & adjectives.
    (a word describing nouns & pronouns.)

→ Conjunction : It connects words, phrases or clauses

eg- and, but, because.

- can be classified into coordinating (joining equal elements) and subordinating (joining dependent clauses to independent ones)

- conjunctions are important to understand sentence structure and relationship between ideas.

→ Participle : It is a verb that functions as an adjective (modifies noun or pronoun)

eg: running water (running describing water)

eg: [broken window]

→ Articles : an article specifies whether a noun is definite or indefinite (eg: the, an, a)

definite → specific entity

indefinite → any member of a group.

→ Gerund : Verb that function as a ~~noun~~ noun

- It always ~~ends~~ ends in -ing.

eg: (Swimming) is my fav activity

Swimming is a verb acting as noun. (subject of the sentence).

⇒ Syntax Parsing

- It is the process of analyzing a (sequence of symbols) (such as words in a sentence) to determine its (grammatical structure) according to (formal rules of grammar.)

- In NLP, parsing helps in understanding the [relationships between words and phrases.]

→ Parse Trees :

- These trees represent the (syntactic structure of a sentence), showing how [words group together in phrases] and how these phrases relate to each other.

- Each node in the tree corresponds to [grammatical construct (unit)] and each node represents a grammatical unit or element, like noun, verb or phrase.

(VP, NP)

→ Abstract Syntax Tree : → used in compilers

- ASTs are parse trees, but it omits certain syntactic details, focusing hierarchial structure of sentences.

- [(mostly used in coding), like for eg it [does not include parenthesses and syntactic sugar] etc.

- This simplification allows compiler to focus on essential details

→ Context Free Grammer (CFGs)

- A CFG is a (system) used to describe the [syntax of languages], especially programming languages and natural languages.

- Grammer : nothing but [set of rules] used to [construct valid sentences in a language.]

- A Context Free Grammer is usually defined as a 4-tuples :

$$G = (V, \Sigma, R, S)$$

(V = Non, Σ = T, R = P)

V → it is a finite set of non terminal symbols (or variables) that represent [abstract syntactic categories.] (theoretical)

T/Σ → it is a finite set of terminal symbols of language

$$(V \cap T = \phi)$$

$V \cap T = \phi$ (null) — ∩ = intersection

P/R → finite set of [production rules] each production rule has the form :

$A \rightarrow \alpha$ (A = non terminal; α = terminal/non terminal such that α is a part of $V \cup T$ (union))

$A \in V$ (non terminal)

$\alpha \in (V \cup T)^*$ (a string of terminals or non terminals, it could also be empty string, denoted by ε)

S ∈ V → it is the start symbol, representing the whole sentence or program.

(non-terminal)

→ A CFG generates a language by starting with the start symbol (S) and repeatedly replacing non-terminals using the production rules [until a string composed entirely of terminal symbols is produced. This process is called (derivation).

→ EXAMPLE:

V = {S}

T = {(,)}

Start Symbol = S

Production rules:

S ⟶ (S)

S ⟶ SS

S ⟶ ε (ε denotes empty string)

Derivation: for "(()())"

1. Start with S
2. Applying rule S ⟶ SS, now we have SS
3. for first S, using rule S ⟶ (S), now, we have (S)S

2. Applying rule, $S \longrightarrow (S)$

now, we have (S)

3. Now applying $S \longrightarrow SS$,

now, we have (SS)

4. Now applying again $S \longrightarrow (S)$

we have : $((S)(S))$

5. here we apply $S \longrightarrow \varepsilon$ (empty string)

to finally derive : $(()())$

→ A Parse Tree or a Derivation Tree visually represents the [structure of a derivation according to CFG.]

→ Each node in the tree represents a symbol (either terminal or non terminal), and the children of the node represents the symbols on the right / left hand side of a production rule applied to that node.

→ Parse Trees are particularly useful because they reveal the [hierarchical structure of sentences], which is essential for tasks like [syntax analysis] in compilers.

⇒ [Leftmost and Rightmost Derivations]

- In CFGs, derivations describe how to generate strings from the start symbol by applying production rules successively.
- Two common strategies for organizing these derivations are [leftmost & rightmost derivations], both will lead to same parse tree for (unambiguous grammar).
- They both can (differ) in the order of non terminals expanded.
- These are helpful in [detecting] and [reducing] ambiguities in grammar.

→ LEFTMOST DERIVATIONS :

- Here, in the parse tree, we replace the leftmost nonterminal in the current string
- it is used in (top-down parsing) (recursive descent parsers)

example : rule : $S \rightarrow aSb \mid \varepsilon$

To derive : aabb

1. apply rule to get : aSb
2. reapply the rule to get : aaSbb
3. now, use $S \rightarrow \varepsilon$, to get aabb.

Example 2: CFGs rules :

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow (E)$$

$$E \rightarrow id$$

To derive : id + id * id

→ Stepwise leftmost derivation :

1. To start off using rule $E \rightarrow E + E$

   we get $E + E$

2. Now, we will take leftmost E

   using rule : $E \rightarrow id$

   we get, id + E

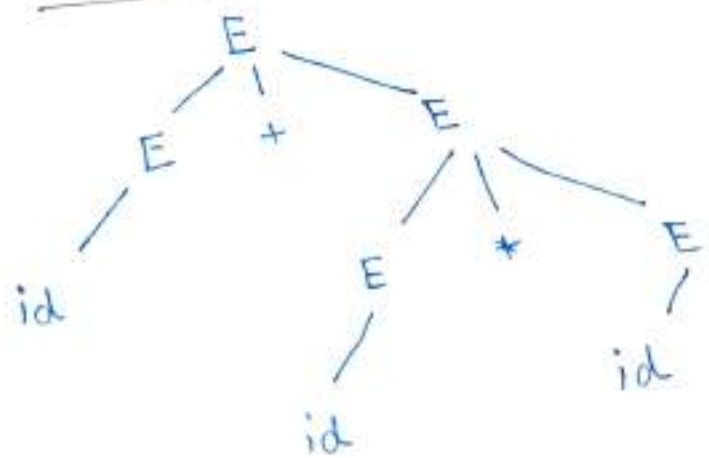
3. Now, expanding the other E, using rule :

   $E \rightarrow E * E$

4. Now, starting with leftmost E, we will convert both Es to ids using rule : $E \rightarrow id$

Finally we get = id + id * id.

Tree :



→ RIGHTMOST DERIVATIONS

- Here, at each step you replace the (rightmost non terminal) in the string
- common in bottom-up parsing (LR parsers) where the parser effectively a rightmost derivation in reverse.

Examples : $E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow id$

To derive : $(id + id) * id + id * id$

Steps wise : 1. Starting with just E, using rule: $E \rightarrow E+E$ to get $\boxed{E+E}$

2. Now, selecting the rightmost E, and then using rule: $E \rightarrow E * E$

we get : $E + (E * E)$

3. Now, replacing the rightmost E with $E \rightarrow id$

we get : $E + E * id$

4. Now, expanding the rightmost E with $E \rightarrow E * E$

We get $E + E * E * id$

5. Now expanding right most E to (E)

$E + E * (E) * id$

6. using rule $E \rightarrow E + E$ on right most E

we get : $E + E * (E + E) * id$

7. replacing both right most E with id.

$E \rightarrow id$

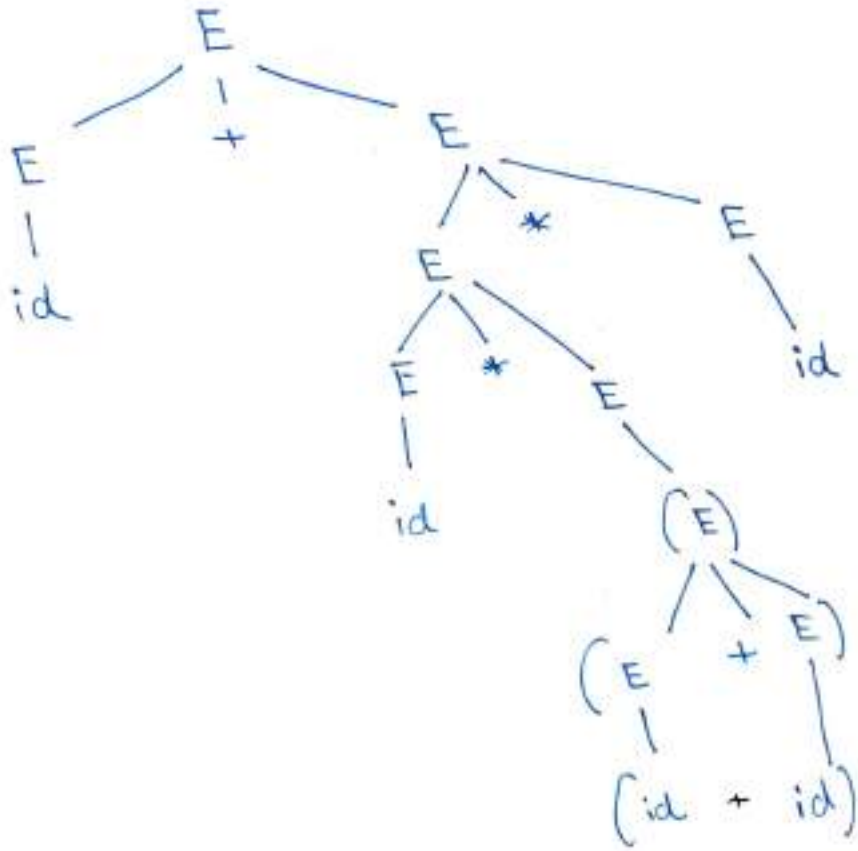
we get $E + E * (id + id) * id$

8. still doing above step we get :

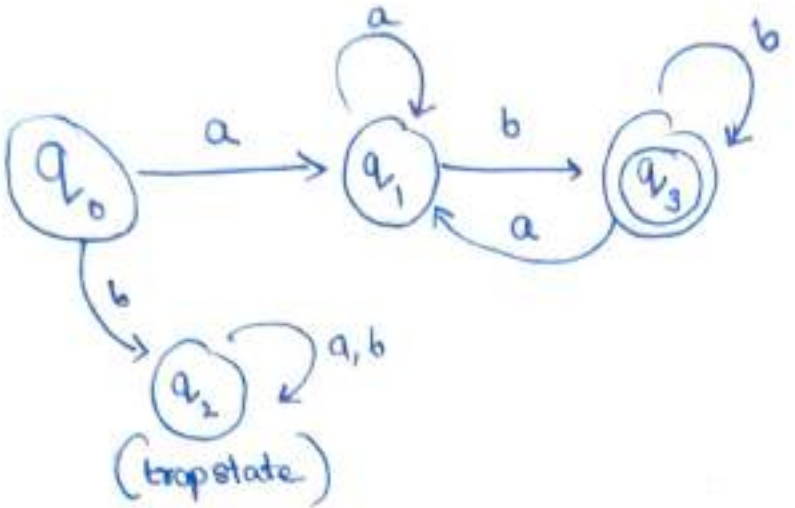
$id + id * (id + id) * id$

rearranging it we get : $(id + id) * id + id * id$

Tree :



→ Designing DFA for strings starting with 'a' and ending with 'b'



all strings not starting with 'a' or not ending

→ Finite State Morphology

- It is used to do morphological analysis (identify and explore relationships b/w words) and generation using FSA and Finite State Transducer (FST).

→ Key components of FS Morphology

- Lexicon : a collection of root words and affixes
- Morphotactic Rules : Defines how morphemes (prefixes, root and suffixes combine. (walk + ing = walking)
- Phonological Rules : Handle spelling changes after addition of ~~new~~ affixes.
- FSA : used to recognize valid words in a language.
- Finite State Transducers (FST):
  - used to transform, one linguistic form to another (lemma ⟷ word form)
  - it can be said to be an extension of FSA that maps input symbols to output symbols.
  - instead of just accepting or rejecting a string, an FST translates one representation into another.

→ The Lexicon :

- It is a structured collection of morphemes (roots, affixes and stems) in a language. It acts as a database for word building elements.

→ components of the lexicon :

- root morphemes (base word) : the core meaning of a word.
  examples : "run", "cat", "teach"
- Affixes (prefixes, suffixes and infixes) :
  modify the meaning of the root
  - prefixes : un- , pre-, dis- (unhappy, prepaid)
  - suffixes : -ing, -ed, -s (running, jumped, cats)
  - infixes : very rare in english.
- Irregular forms : some words do not follow standard affixtion rules
  eg go → went (instead of goed)
- Function words : articles, pronouns and conjunction.
  examples : the, and, be.

→ Role in NLP :

- provides word look ups for parsing
- helps in lemmatization (finding the base form of words)
- supports morphological generation (forming words from root).

→ Morphotactics :

- It refers to the rules that dictate how morphenes combine to form valid words.

→ Rules ensure :

1. Correct Morphene Order : Some morphenes must appear in a certain order

   - re-organize ✓ (prefix + root)
   - organize-re ✗ (invalid order)

2. Obligatory and optional morphenes :

   at some places, morphenes can be compulsory, while at other place, they might be optional.

   eg- plural words (optional)

3. Morphene Compatibility : certain morphenes cannot co-occur.

   eg. un-believe-able (unbelievable) ✓

   un-happy-s ✗

   (un- & plural -s cannot appear together).

⇒ Ambiguity in English

- Natural language, is often ambigious, meaning that a single sentence can be parsed with different ways resulting in different interpretations. This is known as (syntactic ambiguity)

- EXAMPLE : I saw the man with the telescope.

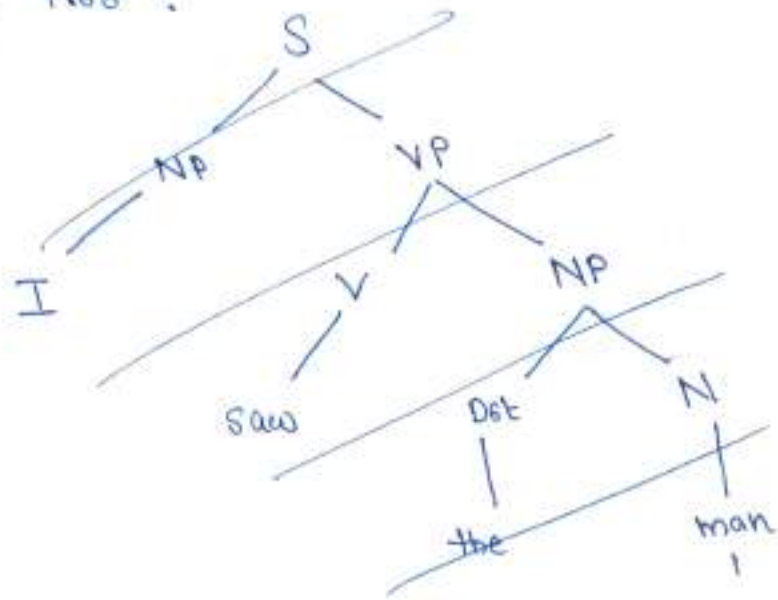
- The above sentence can be interpreted in atleast two ways depending on how you attach the prepositional phrase "with the telescope" in the parse tree.

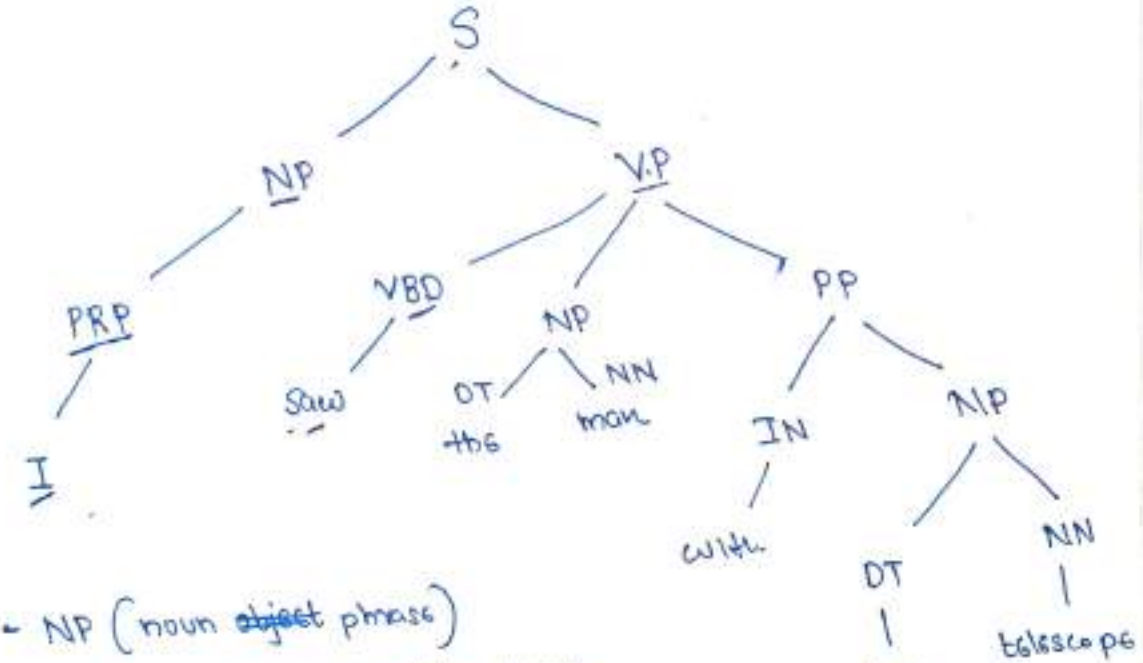
→ Interpretation 1: "I used the telescope to see the man"

→ Here "with the telescope" is counted in the (verb phrase (VP)). It tells us how I saw the man.

- S: The root node, representing the whole sentence.
- NP (Noun Phrase) → Subject
  PRP: ~~Proper Nouns~~ Personal Pronoun. → I (personal/subject pronoun)
- VP (Verb Phrase) → Predicate (everything after subject)
  - VBD (past tense verb) → saw

Parse Tree :





- NP (noun ~~object~~ phrase)

  Direct Object → the man
  - the → Determiner (DT)
  - man → NN (noun)

- PP (Prepositional Phrase, Modifier)

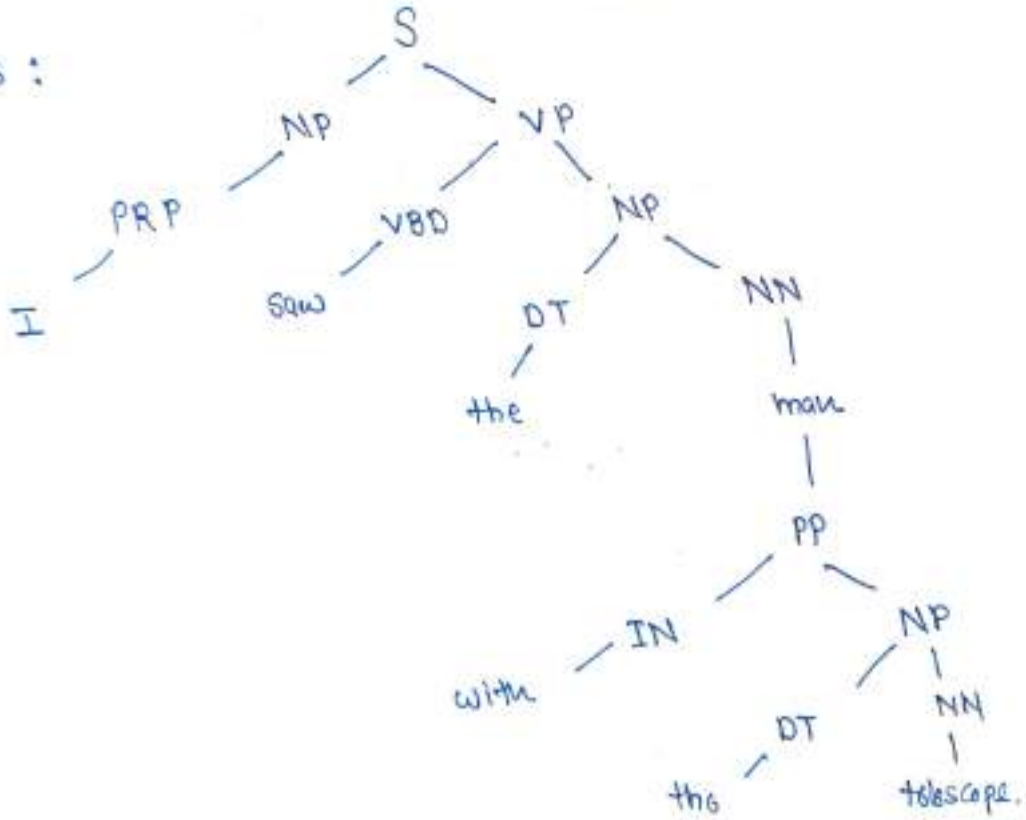
  → with the telescope.
  - with → Preposition (IN)
  - the → DT (Determiner)
  - telescope → NN (noun).

→ NOTE: As the PP is inside VP, it means "with the telescope" is modifying "saw" rather the "the man".

→ Alternative Meaning with Different ~~Meaning~~ Tree.

- here we attach the PP ("with the telescope") to the NP ("the man"), now meaning changes to:
  - now, "~~the~~ with the telescope" describes "the man" not "saw"
  - This now, means "I saw the man who had a telescope"

Tree:



Above is an example of

(Syntactic Ambiguity)

- The [Penn Treebank] POS (Parts of Speech) Tag set categories words into different syntactic components.

⇒ Noun Phrase Subcategories

- A (NP) Noun Phrase is a phrase centered around a noun.

→ **NNP : Proper Noun, Singular**

- used for single popular pronouns (names, places, brands)
- eg - Microsoft, Elon Musk
- very useful in parsing named entities. (people, organization, location)

→ **NNPS : Proper Noun, Plural**

- used for plural proper nouns (group names, countries, groups)
- example : beatles, Indians.

→ **NN : Noun, Singular**

- refers to single noun, also it should not be proper.
- eg : cat, book

→ **NNS : Noun, Plural**

- eg - cats, books

→ **PRP : Personal Pronouns**

- used for subject & object pronouns.
- eg - He, They.
- PRP replaces NN is sentence
- very useful in dependency parsing and coreference resolution.

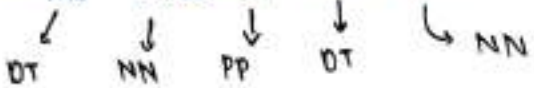
→ NP with a Possessive (POS)

- used for possessive noun phrases.
- eg- John's book, The company's policy

- POS is always a separate node in parse tree.

→ Prepositional Phrase (PP)

- Prepositional phrases modifies the noun and often indicate location
- Eg: The book on the table.



⇒ Verb Phrase subcategories

- A Verb Phrase contains a verb and its object, complements or modifiers

→ VB : Base verb

- used for dictionary form of verbs
- Eg- go, eat

→ VBD : Verb, Past Tense

- used for past tense actions
- Eg - played, ~~ate~~ ate.

→ VBG : Verb ~~Past~~ Participle
Present

- used for ~~words~~ verbs ending (-ing).
- eg : playing, running

→ VBN : Verb, Past Participle

- used for perfect tenses and passive voices
- eg : eaten, broken
- commonly used with "have / has / had" in perfect tenses.

→ VBZ ; Verb, 3rd Person singular present

- used for verbs ending in -s in third person singular.
- eg - runs, eats
- used with singular subjects like : he, she, it

→ VP with an Infinitive (VP → TO VP)

- eg : to love, to play.

⇒ Some additional Important categories for parse trees.

→ Determiner (DT) ✓

- mainly used for articles (a, an, the) and others such as (this, some, many)

→ IN : Preposition or Subordinating conjunction.

(in, on, at, by, with) ✓ (because, although, while)

→ CC : Coordinating Conjunctions : → joins two grammatical elements (NPs / VPs)

(and, or, but, nor, yet, so)

→ JJ : Adjective

- used for descriptive words
- eg - big, beautiful, expensive

→ RB : Adverb

- used to modify verbs and adjectives
- eg - she quickly ran to the mall

↳ modifying ran

→ the test was very difficult

↳ adjective

↳ adverb

very modifying difficult

Example : The chicken is ready to eat