Regular Expressions

A regular expression (RE) is a language for specifying text search strings. RE helps us to match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are used to search texts in UNIX as well as in MS WORD in identical way. We have various search engines using a number of RE features.

Properties of Regular Expressions

Followings are some of the important properties of RE –

- American Mathematician Stephen Cole Kleene formalized the Regular Expression language.
- RE is a formula in a special language, which can be used for specifying simple classes of strings, a sequence of symbols. In other words, we can say that RE is an algebraic notation for characterizing a set of strings.
- Regular expression requires two things, one is the pattern that we wish to search and other is a corpus of text from which we need to search.

Mathematically, A Regular Expression can be defined as follows –

- **ε** is a Regular Expression, which indicates that the language is having an empty string.
- **φ** is a Regular Expression which denotes that it is an empty language.
- If **X** and **Y** are Regular Expressions, then
  - **X, Y**
  - **X.Y(Concatenation of XY)**
  - **X+Y (Union of X and Y)**
  - **X*, Y* (Kleen Closure of X and Y)**

  are also regular expressions.

- If a string is derived from above rules then that would also be a regular expression.

Examples of Regular Expressions

The following table shows a few examples of Regular Expressions –

Regular Expressions	Regular Set
(0 + 10*)	{0, 1, 10, 100, 1000, 10000, … }
(0*10*)	{1, 01, 10, 010, 0010, …}
(0 + ε)(1 + ε)	{ε, 0, 1, 01}
(a+b)*	It would be set of strings of a's and b's of any length which also includes the null string i.e. {ε, a, b, aa , ab , bb , ba, aaa…….}
(a+b)*abb	It would be set of strings of a's and b's ending with the string abb i.e. {abb, aabb, babb, aaabb, ababb, ……..}
(11)*	It would be set consisting of even number of 1's which also includes an empty string i.e. {ε, 11, 1111, 111111, …….}
(aa)*(bb)*b	It would be set of strings consisting of even number of a's followed by odd number of b's i.e. {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, ……..}
(aa + ab + ba + bb)*	It would be string of a's and b's of even length that can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null i.e. {aa, ab, ba, bb, aaab, aaba, ……..}

Regular Sets & Their Properties

It may be defined as the set that represents the value of the regular expression and consists specific properties.

Properties of regular sets

- If we do the union of two regular sets then the resulting set would also be regula.
- If we do the intersection of two regular sets then the resulting set would also be regular.
- If we do the complement of regular sets, then the resulting set would also be regular.
- If we do the difference of two regular sets, then the resulting set would also be regular.
- If we do the reversal of regular sets, then the resulting set would also be regular.
- If we take the closure of regular sets, then the resulting set would also be regular.
- If we do the concatenation of two regular sets, then the resulting set would also be regular.

Finite State Automata

The term automata, derived from the Greek word "αὐτόματα" meaning "self-acting", is the plural of automaton which may be defined as an abstract self-propelled computing device that follows a predetermined sequence of operations automatically.

An automaton having a finite number of states is called a Finite Automaton (FA) or Finite State automata (FSA).

Mathematically, an automaton can be represented by a 5-tuple (Q, Σ, δ, q0, F), where −

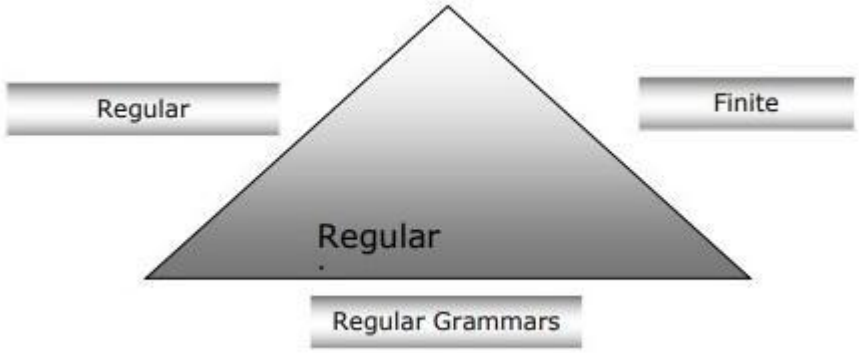
- Q is a finite set of states.
- Σ is a finite set of symbols, called the alphabet of the automaton.
- δ is the transition function
- q0 is the initial state from where any input is processed (q0 ∈ Q).
- F is a set of final state/states of Q (F ⊆ Q).

Relation between Finite Automata, Regular Grammars and Regular Expressions

Following points will give us a clear view about the relationship between finite automata, regular grammars and regular expressions −

- As we know that finite state automata are the theoretical foundation of computational work and regular expressions is one way of describing them.
- We can say that any regular expression can be implemented as FSA and any FSA can be described with a regular expression.
- On the other hand, regular expression is a way to characterize a kind of language called regular language. Hence, we can say that regular language can be described with the help of both FSA and regular expression.
- Regular grammar, a formal grammar that can be right-regular or left-regular, is another way to characterize regular language.

Following diagram shows that finite automata, regular expressions and regular grammars are the equivalent ways of describing regular languages.



Types of Finite State Automation (FSA)

Finite state automation is of two types. Let us see what the types are.

Deterministic Finite automation (DFA)

It may be defined as the type of finite automation wherein, for every input symbol we can determine the state to which the machine will move. It has a finite number of states that is why the machine is called Deterministic Finite Automaton (DFA).

Mathematically, a DFA can be represented by a 5-tuple (Q, Σ, δ, q0, F), where −

- Q is a finite set of states.
- Σ is a finite set of symbols, called the alphabet of the automaton.
- δ is the transition function where δ: Q × Σ → Q .
- q0 is the initial state from where any input is processed (q0 ∈ Q).
- F is a set of final state/states of Q (F ⊆ Q).

Whereas graphically, a DFA can be represented by diagraphs called state diagrams where −

- The states are represented by **vertices**.
- The transitions are shown by labeled **arcs**.
- The initial state is represented by an **empty incoming arc**.
- The final state is represented by **double circle**.

Example of DFA

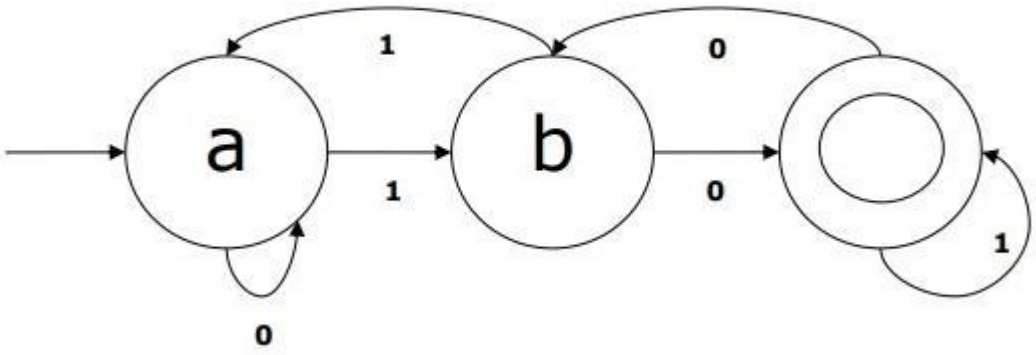
Suppose a DFA be

- Q = {a, b, c},

- $\Sigma = \{0, 1\}$,
- $q_0 = \{a\}$,
- $F = \{c\}$,
- Transition function δ is shown in the table as follows −

Current State	Next State for Input 0	Next State for Input 1
A	a	B
B	b	A
C	c	C

The graphical representation of this DFA would be as follows −



Non-deterministic Finite Automation (NDFA)

It may be defined as the type of finite automation where for every input symbol we cannot determine the state to which the machine will move i.e. the machine can move to any combination of the states. It has a finite number of states that is why the machine is called Non-deterministic Finite Automation (NDFA).

Mathematically, NDFA can be represented by a 5-tuple (Q, Σ, δ, q0, F), where −

- Q is a finite set of states.
- Σ is a finite set of symbols, called the alphabet of the automaton.
- δ :-is the transition function where δ: $Q \times \Sigma \rightarrow 2^{Q}$.
- q0 :-is the initial state from where any input is processed ($q0 \in Q$).
- F :-is a set of final state/states of Q ($F \subseteq Q$).

Whereas graphically (same as DFA), a NDFA can be represented by diagraphs called state diagrams where –

- The states are represented by **vertices**.
- The transitions are shown by labeled **arcs**.
- The initial state is represented by an **empty incoming arc**.
- The final state is represented by double **circle**.

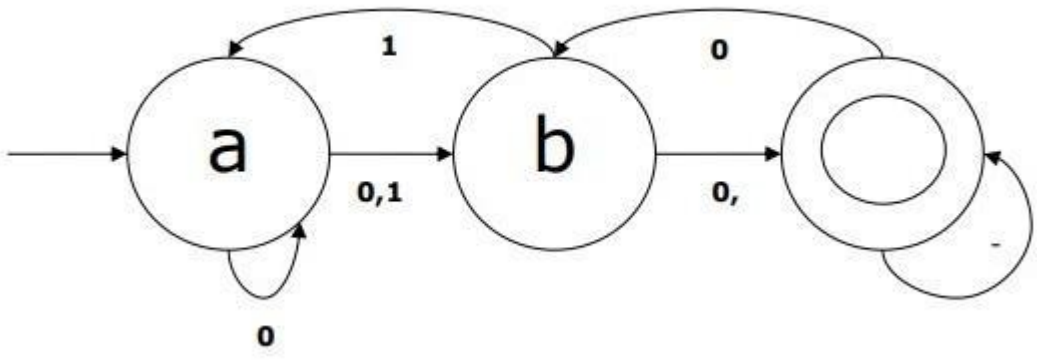
Example of NDFA

Suppose a NDFA be

- Q = {a, b, c},
- Σ = {0, 1},
- q_0 = {a},
- F = {c},
- Transition function δ is shown in the table as follows –

Current State	Next State for Input 0	Next State for Input 1
A	a, b	B
B	C	a, c
C	b, c	C

The graphical representation of this NDFA would be as follows –



Morphological Parsing

The term morphological parsing is related to the parsing of morphemes. We can define morphological parsing as the problem of recognizing that a word breaks down into smaller meaningful units called morphemes

producing some sort of linguistic structure for it. For example, we can break the word *foxes* into two, *fox* and *-es*. We can see that the word *foxes*, is made up of two morphemes, one is *fox* and other is *-es*.

In other sense, we can say that morphology is the study of –

- The formation of words.
- The origin of the words.
- Grammatical forms of the words.
- Use of prefixes and suffixes in the formation of words.
- How parts-of-speech (PoS) of a language are formed.

Types of Morphemes

Morphemes, the smallest meaning-bearing units, can be divided into two types –

- Stems
- Word Order

Stems

It is the core meaningful unit of a word. We can also say that it is the root of the word. For example, in the word foxes, the stem is fox.

- **Affixes** – As the name suggests, they add some additional meaning and grammatical functions to the words. For example, in the word foxes, the affix is – es.

  Further, affixes can also be divided into following four types –

  - **Prefixes** – As the name suggests, prefixes precede the stem. For example, in the word unbuckle, un is the prefix.
  - **Suffixes** – As the name suggests, suffixes follow the stem. For example, in the word cats, -s is the suffix.
  - **Infixes** – As the name suggests, infixes are inserted inside the stem. For example, the word cupful, can be pluralized as cupsful by using -s as the infix.
  - **Circumfixes** – They precede and follow the stem. There are very less examples of circumfixes in English language. A very common example is 'A-ing' where we can use -A precede and -ing follows the stem.

Word Order

The order of the words would be decided by morphological parsing. Let us now see the requirements for building a morphological parser –

Lexicon

The very first requirement for building a morphological parser is lexicon, which includes the list of stems and affixes along with the basic information about them. For example, the information like whether the stem is Noun stem or Verb stem, etc.

Morphotactics

It is basically the model of morpheme ordering. In other sense, the model explaining which classes of morphemes can follow other classes of morphemes inside a word. For example, the morphotactic fact is that the English plural morpheme always follows the noun rather than preceding it.

Orthographic rules

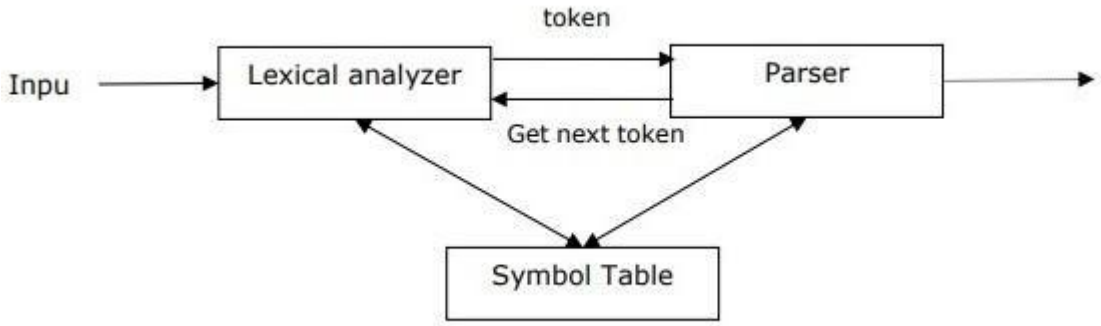
These spelling rules are used to model the changes occurring in a word. For example, the rule of converting y to ie in word like city+s = cities not citys.

Syntactic analysis or parsing or syntax analysis is the third phase of NLP. The purpose of this phase is to draw exact meaning, or you can say dictionary meaning from the text. Syntax analysis checks the text for meaningfulness comparing to the rules of formal grammar. For example, the sentence like “hot ice-cream” would be rejected by semantic analyzer.

In this sense, syntactic analysis or parsing may be defined as the process of analyzing the strings of symbols in natural language conforming to the rules of formal grammar. The origin of the word ***‘parsing’*** is from Latin word ***‘pars’*** which means ***‘part’***.

Concept of Parser

It is used to implement the task of parsing. It may be defined as the software component designed for taking input data (text) and giving structural representation of the input after checking for correct syntax as per formal grammar. It also builds a data structure generally in the form of parse tree or abstract syntax tree or other hierarchical structure.



The main roles of the parse include –

- To report any syntax error.
- To recover from commonly occurring error so that the processing of the remainder of program can be continued.
- To create parse tree.
- To create symbol table.
- To produce intermediate representations (IR).

Types of Parsing

Derivation divides parsing into the followings two types –

- Top-down Parsing
- Bottom-up Parsing

Top-down Parsing

In this kind of parsing, the parser starts constructing the parse tree from the start symbol and then tries to transform the start symbol to the input. The most common form of topdown parsing uses recursive procedure to process the input. The main disadvantage of recursive descent parsing is backtracking.

Bottom-up Parsing

In this kind of parsing, the parser starts with the input symbol and tries to construct the parser tree up to the start symbol.

Concept of Derivation

In order to get the input string, we need a sequence of production rules. Derivation is a set of production rules. During parsing, we need to decide the non-terminal, which is to be replaced along with deciding the production rule with the help of which the non-terminal will be replaced.

Types of Derivation

In this section, we will learn about the two types of derivations, which can be used to decide which non-terminal to be replaced with production rule –

Left-most Derivation

In the left-most derivation, the sentential form of an input is scanned and replaced from the left to the right. The sentential form in this case is called the left-sentential form.

Right-most Derivation

In the left-most derivation, the sentential form of an input is scanned and replaced from right to left. The sentential form in this case is called the right-sentential form.

Concept of Parse Tree

It may be defined as the graphical depiction of a derivation. The start symbol of derivation serves as the root of the parse tree. In every parse tree, the leaf nodes are terminals and interior nodes are non-terminals. A property of parse tree is that in-order traversal will produce the original input string.

Concept of Grammar

Grammar is very essential and important to describe the syntactic structure of well-formed programs. In the literary sense, they denote syntactical rules for conversation in natural languages. Linguistics have attempted to define grammars since the inception of natural languages like English, Hindi, etc.

The theory of formal languages is also applicable in the fields of Computer Science mainly in programming languages and data structure. For example, in 'C' language, the precise grammar rules state how functions are made from lists and statements.

A mathematical model of grammar was given by **Noam Chomsky** in 1956, which is effective for writing computer languages.

Mathematically, a grammar G can be formally written as a 4-tuple (N, T, S, P) where −

- **N** or $\mathbf{V_N}$ = set of non-terminal symbols, i.e., variables.
- **T** or ∑ = set of terminal symbols.
- **S** = Start symbol where S ∈ N
- **P** denotes the Production rules for Terminals as well as Non-terminals. It has the form α → β, where α and β are strings on $V_N \cup \sum$ and least one symbol of α belongs to V_N

Phrase Structure or Constituency Grammar

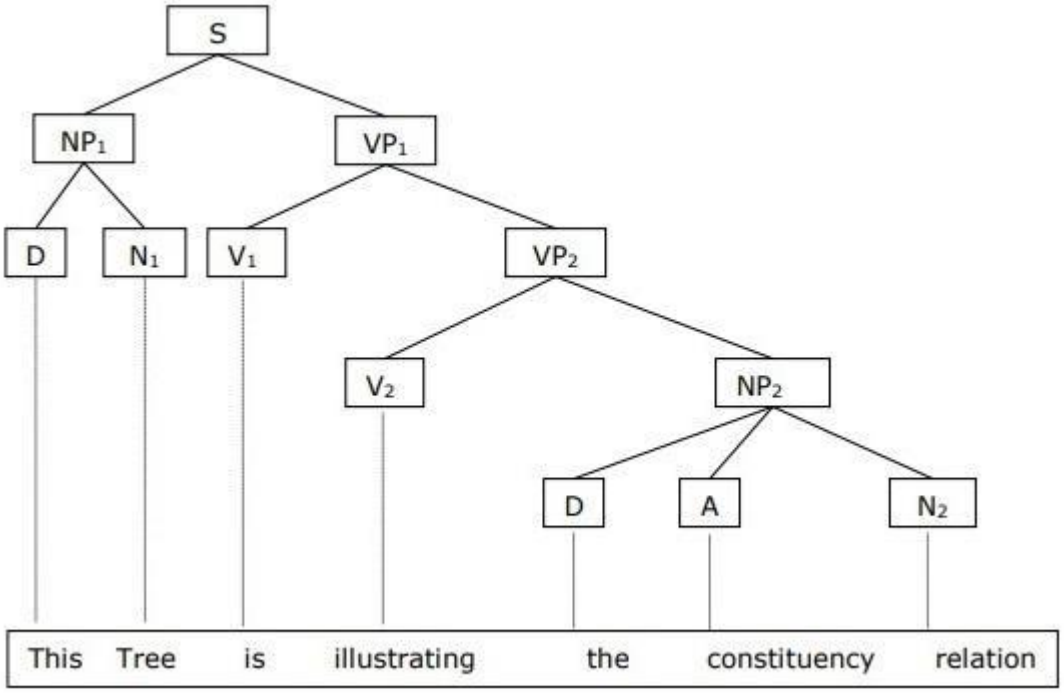
Phrase structure grammar, introduced by Noam Chomsky, is based on the constituency relation. That is why it is also called constituency grammar. It is opposite to dependency grammar.

Example

Before giving an example of constituency grammar, we need to know the fundamental points about constituency grammar and constituency relation.

- All the related frameworks view the sentence structure in terms of constituency relation.
- The constituency relation is derived from the subject-predicate division of Latin as well as Greek grammar.
- The basic clause structure is understood in terms of **noun phrase NP** and **verb phrase VP**.

We can write the sentence **"This tree is illustrating the constituency relation"** as follows −



Dependency Grammar

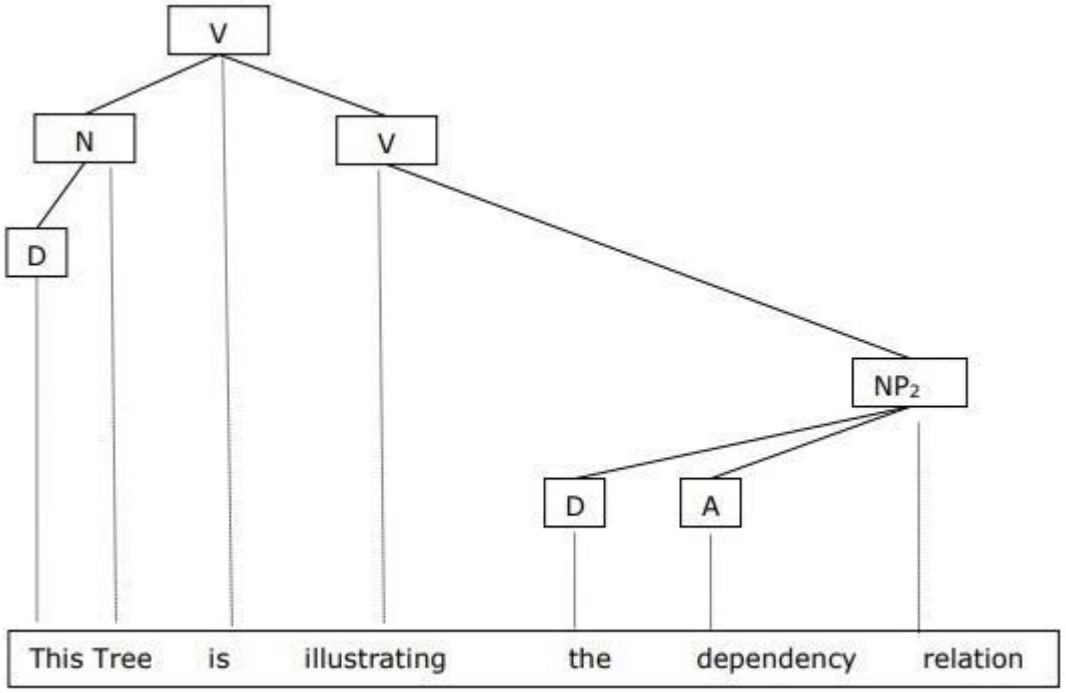
It is opposite to the constituency grammar and based on dependency relation. It was introduced by Lucien Tesniere. Dependency grammar (DG) is opposite to the constituency grammar because it lacks phrasal nodes.

Example

Before giving an example of Dependency grammar, we need to know the fundamental points about Dependency grammar and Dependency relation.

- In DG, the linguistic units, i.e., words are connected to each other by directed links.
- The verb becomes the center of the clause structure.
- Every other syntactic units are connected to the verb in terms of directed link. These syntactic units are called ***dependencies***.

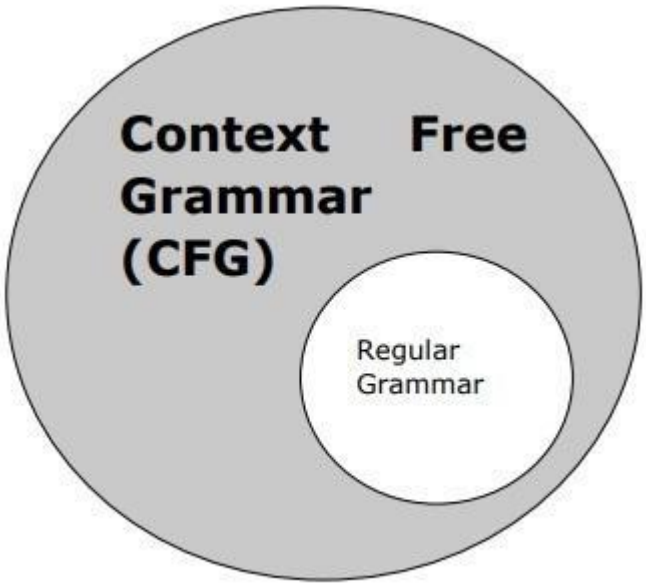
We can write the sentence **"This tree is illustrating the dependency relation"** as follows;



Parse tree that uses Constituency grammar is called constituency-based parse tree; and the parse trees that uses dependency grammar is called dependency-based parse tree.

Context Free Grammar

Context free grammar, also called CFG, is a notation for describing languages and a superset of Regular grammar. It can be seen in the following diagram –



Definition of CFG

CFG consists of finite set of grammar rules with the following four components –

Set of Non-terminals

It is denoted by V. The non-terminals are syntactic variables that denote the sets of strings, which further help defining the language, generated by the grammar.

Set of Terminals

It is also called tokens and defined by Σ. Strings are formed with the basic symbols of terminals.

Set of Productions

It is denoted by P. The set defines how the terminals and non-terminals can be combined. Every production(P) consists of non-terminals, an arrow, and terminals (the sequence of terminals). Non-terminals are called the left side of the production and terminals are called the right side of the production.

Start Symbol

The production begins from the start symbol. It is denoted by symbol S. Non-terminal symbol is always designated as start symbol.

The purpose of semantic analysis is to draw exact meaning, or you can say dictionary meaning from the text. The work of semantic analyzer is to check the text for meaningfulness.

We already know that lexical analysis also deals with the meaning of the words, then how is semantic analysis different from lexical analysis? Lexical analysis is based on smaller token but on the other side semantic analysis focuses on larger chunks. That is why semantic analysis can be divided into the following two parts –

Studying meaning of individual word

It is the first part of the semantic analysis in which the study of the meaning of individual words is performed. This part is called lexical semantics.

Studying the combination of individual words

In the second part, the individual words will be combined to provide meaning in sentences.

The most important task of semantic analysis is to get the proper meaning of the sentence. For example, analyze the sentence **"Ram is great."** In this sentence, the speaker is talking either about Lord Ram or about a person whose name is Ram. That is why the job, to get the proper meaning of the sentence, of semantic analyzer is important.

Elements of Semantic Analysis

Followings are some important elements of semantic analysis –

Hyponymy

It may be defined as the relationship between a generic term and instances of that generic term. Here the generic term is called hypernym and its instances are called hyponyms. For example, the word color is hypernym and the color blue, yellow etc. are hyponyms.

Homonymy

It may be defined as the words having same spelling or same form but having different and unrelated meaning. For example, the word "Bat" is a

homonymy word because bat can be an implement to hit a ball or bat is a nocturnal flying mammal also.

Polysemy

Polysemy is a Greek word, which means "many signs". It is a word or phrase with different but related sense. In other words, we can say that polysemy has the same spelling but different and related meaning. For example, the word "bank" is a polysemy word having the following meanings –

- A financial institution.
- The building in which such an institution is located.
- A synonym for "to rely on".

Difference between Polysemy and Homonymy

Both polysemy and homonymy words have the same syntax or spelling. The main difference between them is that in polysemy, the meanings of the words are related but in homonymy, the meanings of the words are not related. For example, if we talk about the same word "Bank", we can write the meaning 'a financial institution' or 'a river bank'. In that case it would be the example of homonym because the meanings are unrelated to each other.

Synonymy

It is the relation between two lexical items having different forms but expressing the same or a close meaning. Examples are 'author/writer', 'fate/destiny'.

Antonymy

It is the relation between two lexical items having symmetry between their semantic components relative to an axis. The scope of antonymy is as follows –

- **Application of property or not** – Example is 'life/death', 'certitude/incertitude'
- **Application of scalable property** – Example is 'rich/poor', 'hot/cold'
- **Application of a usage** – Example is 'father/son', 'moon/sun'.

Meaning Representation

Semantic analysis creates a representation of the meaning of a sentence. But before getting into the concept and approaches related to meaning representation, we need to understand the building blocks of semantic system.

Building Blocks of Semantic System

In word representation or representation of the meaning of the words, the following building blocks play an important role –

- **Entities** – It represents the individual such as a particular person, location etc. For example, Haryana. India, Ram all are entities.
- **Concepts** – It represents the general category of the individuals such as a person, city, etc.
- **Relations** – It represents the relationship between entities and concept. For example, Ram is a person.
- **Predicates** – It represents the verb structures. For example, semantic roles and case grammar are the examples of predicates.

Now, we can understand that meaning representation shows how to put together the building blocks of semantic systems. In other words, it shows how to put together entities, concepts, relation and predicates to describe a situation. It also enables the reasoning about the semantic world.

Approaches to Meaning Representations

Semantic analysis uses the following approaches for the representation of meaning –

- First order predicate logic (FOPL)
- Semantic Nets
- Frames
- Conceptual dependency (CD)
- Rule-based architecture
- Case Grammar
- Conceptual Graphs

Need of Meaning Representations

A question that arises here is why do we need meaning representation? Followings are the reasons for the same –

Linking of linguistic elements to non-linguistic elements

The very first reason is that with the help of meaning representation the linking of linguistic elements to the non-linguistic elements can be done.

Representing variety at lexical level

With the help of meaning representation, unambiguous, canonical forms can be represented at the lexical level.

Can be used for reasoning

Meaning representation can be used to reason for verifying what is true in the world as well as to infer the knowledge from the semantic representation.

Lexical Semantics

The first part of semantic analysis, studying the meaning of individual words is called lexical semantics. It includes words, sub-words, affixes (sub-units), compound words and phrases also. All the words, sub-words, etc. are collectively called lexical items. In other words, we can say that lexical semantics is the relationship between lexical items, meaning of sentences and syntax of sentence.

Following are the steps involved in lexical semantics –

- Classification of lexical items like words, sub-words, affixes, etc. is performed in lexical semantics.
- Decomposition of lexical items like words, sub-words, affixes, etc. is performed in lexical semantics.
- Differences as well as similarities between various lexical semantic structures is also analyzed.

NLP - Word Sense Disambiguation

We understand that words have different meanings based on the context of its usage in the sentence. If we talk about human languages, then they are ambiguous too because many words can be interpreted in multiple ways depending upon the context of their occurrence.

Word sense disambiguation, in natural language processing (NLP), may be defined as the ability to determine which meaning of word is activated by

the use of word in a particular context. Lexical ambiguity, syntactic or semantic, is one of the very first problem that any NLP system faces. Part-of-speech (POS) taggers with high level of accuracy can solve Word's syntactic ambiguity. On the other hand, the problem of resolving semantic ambiguity is called WSD (word sense disambiguation). Resolving semantic ambiguity is harder than resolving syntactic ambiguity.

For example, consider the two examples of the distinct sense that exist for the word ***"bass"*** –

- I can hear bass sound.
- He likes to eat grilled bass.

The occurrence of the word **bass** clearly denotes the distinct meaning. In first sentence, it means **frequency** and in second, it means **fish**. Hence, if it would be disambiguated by WSD then the correct meaning to the above sentences can be assigned as follows –

- I can hear bass/frequency sound.
- He likes to eat grilled bass/fish.

Evaluation of WSD

The evaluation of WSD requires the following two inputs –

A Dictionary

The very first input for evaluation of WSD is dictionary, which is used to specify the senses to be disambiguated.

Test Corpus

Another input required by WSD is the high-annotated test corpus that has the target or correct-senses. The test corpora can be of two types &minsu;

- **Lexical sample** – This kind of corpora is used in the system, where it is required to disambiguate a small sample of words.
- **All-words** – This kind of corpora is used in the system, where it is expected to disambiguate all the words in a piece of running text.

Approaches and Methods to Word Sense Disambiguation (WSD)

Approaches and methods to WSD are classified according to the source of knowledge used in word disambiguation.

Let us now see the four conventional methods to WSD –

Dictionary-based or Knowledge-based Methods

As the name suggests, for disambiguation, these methods primarily rely on dictionaries, treasures and lexical knowledge base. They do not use corpora evidences for disambiguation. The Lesk method is the seminal dictionary-based method introduced by Michael Lesk in 1986. The Lesk definition, on which the Lesk algorithm is based is **"measure overlap between sense definitions for all words in context"**. However, in 2000, Kilgarriff and Rosensweig gave the simplified Lesk definition as **"measure overlap between sense definitions of word and current context"**, which further means identify the correct sense for one word at a time. Here the current context is the set of words in surrounding sentence or paragraph.

Supervised Methods

For disambiguation, machine learning methods make use of sense-annotated corpora to train. These methods assume that the context can provide enough evidence on its own to disambiguate the sense. In these methods, the words knowledge and reasoning are deemed unnecessary. The context is represented as a set of "features" of the words. It includes the information about the surrounding words also. Support vector machine and memory-based learning are the most successful supervised learning approaches to WSD. These methods rely on substantial amount of manually sense-tagged corpora, which is very expensive to create.

Semi-supervised Methods

Due to the lack of training corpus, most of the word sense disambiguation algorithms use semi-supervised learning methods. It is because semi-supervised methods use both labelled as well as unlabeled data. These methods require very small amount of annotated text and large amount of plain unannotated text. The technique that is used by semisupervised methods is bootstrapping from seed data.

Unsupervised Methods

These methods assume that similar senses occur in similar context. That is why the senses can be induced from text by clustering word occurrences by using some measure of similarity of the context. This task

is called word sense induction or discrimination. Unsupervised methods have great potential to overcome the knowledge acquisition bottleneck due to non-dependency on manual efforts.

Applications of Word Sense Disambiguation (WSD)

Word sense disambiguation (WSD) is applied in almost every application of language technology.

Let us now see the scope of WSD –

Machine Translation

Machine translation or MT is the most obvious application of WSD. In MT, Lexical choice for the words that have distinct translations for different senses, is done by WSD. The senses in MT are represented as words in the target language. Most of the machine translation systems do not use explicit WSD module.

Information Retrieval (IR)

Information retrieval (IR) may be defined as a software program that deals with the organization, storage, retrieval and evaluation of information from document repositories particularly textual information. The system basically assists users in finding the information they required but it does not explicitly return the answers of the questions. WSD is used to resolve the ambiguities of the queries provided to IR system. As like MT, current IR systems do not explicitly use WSD module and they rely on the concept that user would type enough context in the query to only retrieve relevant documents.

Text Mining and Information Extraction (IE)

In most of the applications, WSD is necessary to do accurate analysis of text. For example, WSD helps intelligent gathering system to do flagging of the correct words. For example, medical intelligent system might need flagging of "illegal drugs" rather than "medical drugs"

Lexicography

WSD and lexicography can work together in loop because modern lexicography is corpusbased. With lexicography, WSD provides rough empirical sense groupings as well as statistically significant contextual indicators of sense.

Difficulties in Word Sense Disambiguation (WSD)

Followings are some difficulties faced by word sense disambiguation (WSD) –

Differences between dictionaries

The major problem of WSD is to decide the sense of the word because different senses can be very closely related. Even different dictionaries and thesauruses can provide different divisions of words into senses.

Different algorithms for different applications

Another problem of WSD is that completely different algorithm might be needed for different applications. For example, in machine translation, it takes the form of target word selection; and in information retrieval, a sense inventory is not required.

Inter-judge variance

Another problem of WSD is that WSD systems are generally tested by having their results on a task compared against the task of human beings. This is called the problem of interjudge variance.

Word-sense discreteness

Another difficulty in WSD is that words cannot be easily divided into discrete submeanings.