USN □□□□□□□□□□

Internal Assessment Test 1 – JUNE 2024

Sub:	Natural Language Processing				Sub Code:	21AI643	Branch:	AIML
Date:	04/06/2024	Duration:	90 mins	Max Marks:	50	Sem / Sec:	VI / A	OBE

Answer Any of 5 Questions	MARKS	CO	RBT

1

Explain various levels of Natural Language Processing with suitable examples.

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and humans through natural language. NLP involves several levels of language processing, each handling different aspects of language understanding and generation. Here are the various levels of NLP, explained with suitable examples:

1. Phonological Level

The phonological level deals with the analysis of speech sounds and their patterns. It involves understanding and processing the phonemes (the smallest units of sound) in spoken language.

Example:

- Speech Recognition: Converting spoken language into text. For instance, recognizing the spoken phrase "hello" and converting it to the text "hello".

2. Morphological Level

The morphological level involves the study of the structure and formation of words. It focuses on the identification and analysis of morphemes, the smallest meaningful units in a language.

Example:Word Formation: Analyzing the word "unhappiness" into its morphemes: "un-" (prefix), "happy" (root), and "-ness" (suffix).

3. Lexical Level

The lexical level involves understanding and processing words and their meanings. It deals with the analysis of vocabulary and the relationships between words.

Example:

- Part-of-Speech Tagging: Assigning parts of speech to each word in a sentence. For instance, in the sentence "The cat sat on the mat," the words are tagged as: "The/DT cat/NN sat/VBD on/IN the/DT mat/NN."

4. Syntactic Level

The syntactic level focuses on the structure and grammar of sentences. It involves analyzing the arrangement of words and phrases to form grammatically correct sentences.

Example:

- Parsing: Constructing a parse tree for the sentence "The cat sat on the mat" to represent its grammatical structure.

Semantic Level

The semantic level deals with understanding the meaning of words, phrases, and sentences. It involves interpreting the intended meaning and resolving ambiguities.

Example:

- Word Sense Disambiguation: Determining the correct meaning of a word based on context. For example, in the sentence "I went to the bank to deposit money," the word "bank" refers to a financial institution, not a riverbank.

6. Pragmatic Level

The pragmatic level involves understanding the use of language in context and the intentions behind utterances. It considers the context and the speaker's intentions to derive meaning beyond the literal interpretation.

Example:

- Speech Act Recognition: Identifying the intention behind a statement. For example, the sentence "Can you pass the salt?" is interpreted as a request rather than a question about someone's ability.

7. Discourse Level

The discourse level involves the analysis of language beyond individual sentences, considering the context and coherence of multiple sentences in a text or conversation.

Example:

- Coreference Resolution: Identifying when different expressions refer to the same entity. For instance, in the sentences "Alice went to the park. She enjoyed the weather," the pronoun "She" refers to "Alice."

8. Pragmatic Level

The pragmatic level deals with the social aspects of language use. It involves understanding the implications, intentions, and conversational norms in communication.

Example:

- Implicature: Understanding implied meanings. For example, if someone says, "It's getting late," it might imply that they want to leave.

[10]
Def-5
Types with example-5

CO1

L2

2 (a)	Explain Statistical language model and find the probability of the test sentence – P("They play in a big garden") in the following training set using the bi-gram model <S>There is a big garden. Children play in the garden. They play inside beautiful garden. </S>	[05] Def-3 Eg:2	CO1	L3

A statistical language model assigns probabilities to sequences of words, aiming to predict the likelihood of a given word sequence occurring in a language. One common approach is the n-gram model, where the probability of a word depends on the preceding n−1n-1n−1 words.

Bi-Gram Model

A bi-gram model (n=2) predicts the probability of a word based on the previous word. The probability of a sentence is calculated as the product of the conditional probabilities of each word given the previous word.

<S> There is a big garden.

Children play in the garden.

They play inside beautiful garden. </S>

Steps to Calculate Sentence Probability

1. **Tokenize the training sentences and count bi-grams**.
2. **Calculate bi-gram probabilities**.
3. **Compute the probability of the test sentence**.

Tokenization and Counting

Tokenize the sentences and include start (<S>) and end (</S>) markers:

1. <S> There is a big garden. </S>
2. <S> Children play in the garden. </S>
3. <S> They play inside beautiful garden. </S>

Unigram and Bi-gram Counts

Unigrams:

- <S>: 3
- There: 1
- is: 1
- a: 2
- big: 1
- garden: 3
- Children: 1
- play: 2
- in: 2
- the: 2
- They: 1
- inside: 1
- beautiful: 1
- .</S>: 3

Bi-grams:

- (<S>, There): 1
- (There, is): 1
- (is, a): 1
- (a, big): 1
- (big, garden): 1
- (garden, .</S>): 1
- (<S>, Children): 1
- (Children, play): 1
- (play, in): 1
- (in, the): 1
- (the, garden): 1
- (garden, .</S>): 1
- (<S>, They): 1

- (They, play): 1
- (play, inside): 1
- (inside, beautiful): 1
- (beautiful, garden): 1
- (garden, .</S>): 1

Calculating Bi-Gram Probabilities

The bi-gram probability P(wi|wi−1) is calculated as:

P(wi|wi−1)=Count(wi−1)/ Count(wi−1,wi)

Bi-Gram Probabilities:

- P(There|<S>)=1/3
- P(is|There)=1
- P(a|is)=1
- P(big|a)=1/2
- P(garden|big)=1
- P(.</S>|garden)=1/3
- P(Children|<S>)=1/3
- P(play|Children)=1
- P(in|play)=1/2
- P(the|in)=1/2
- P(garden|the)=1/2
- P(.</S>|garden)=1/3
- P(They|<S>)=1/3
- P(play|They)=1
- P(inside|play)=1/2
- P(beautiful|inside)=1
- P(garden|beautiful)=1
- P(.</S>|garden)=1/3

Probability Calculation:

1. P(They|<S>)=1/3
2. P(play|They)=1
3. P(in|play)=1/2
4. P(a|in)=1/2 (Note: "a" is not directly after "in" in training, let's assume "a" after "in" as seen in unigram counts)
5. P(big|a)=1/2P
6. P(garden|big)=1
7. P(.</S>|garden)=1/3

Sentence Probability:

P("They play in a big garden")=P(They|<S>)×P(play|They)×P(in|play)×P(a|in)×P(big|a)×P(garden|big)×P(.</S>|garden)

=1/3* 1 * 1/2 * 1/2 * 1/2 * 1 * 1/3

=1/3 * 1/8 *1/3

=1/72

2b)	List the problems associated with the n-gram model. Explain how these problems are handled.	[05] Def-3 Eg:-2	CO1	L2

The n-gram model is a widely used statistical language model that predicts the probability of a word given its preceding n−1n-1n−1 words. Despite its simplicity and usefulness, the n-gram model has several inherent problems. Here are the key issues and how they are typically addressed:

Problems with the N-gram Model

1. **Data Sparsity**:
   - **Problem**: As the value of nnn increases, the number of possible n-grams grows exponentially, leading to many n-grams that are not observed in the training data.
   - **Solution**: This issue is addressed using various smoothing techniques such as:
     - **Laplace Smoothing**: Adds a small constant (usually 1) to all counts to ensure no zero probabilities.
     - **Good-Turing Smoothing**: Adjusts the probability of unseen n-grams based on the count of n-grams seen once, twice, etc.

2.Context Limitation:
   - **Problem**: N-grams only consider a fixed-length context, which may not capture long-range dependencies effectively.
   - **Solution**: Using more advanced models such as:
     - **Neural Networks**: Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks can capture long-range dependencies by maintaining a state that evolves over time.
     - **Transformers**: These models use self-attention mechanisms to capture dependencies across the entire sequence, overcoming the fixed context limitation of n-grams.

3.Inability to Handle Out-of-Vocabulary (OOV) Words:

   - **Problem**: N-gram models struggle with words not seen during training, assigning zero
   - **Kneser-Ney Smoothing**: A more sophisticated method that redistributes the probability mass of unseen n-grams more effectively.

4.High Memory Consumption:
   - **Problem**: Storing large n-gram models requires significant memory, especially for large values of nnn and extensive vocabularies.
   - **Solution**: Several techniques can be employed to reduce memory usage:
     - **Pruning**: Removing low-frequency n-grams from the model.
     - **Backoff and Interpolation**: Combining lower-order n-grams with higher-order ones to reduce the model size while maintaining performance.
   - probability to any sequence containing an OOV word.
   - **Solution**: Various strategies can mitigate this issue:

     - **UNK Token**: Replacing rare or unseen words with a special token (e.g., UNK) to handle unknown words.
     - **Subword Models**: Breaking words into smaller units such as characters or subwords, which can generalize better to unseen words (e.g., Byte Pair Encoding (BPE), WordPiece).

5..Lack of Semantic Understanding:
   - **Problem**: N-grams rely purely on surface statistics and do not understand the meaning or context beyond the immediate words.
   - **Solution**: Incorporating models that capture semantic information:
     - **Word Embeddings**: Representing words as dense vectors that capture semantic similarities (e.g., Word2Vec, GloVe).
     - **Contextualized Embeddings**: Using models like BERT or GPT that provide context-sensitive word representations.

3	Describe C-Structure and F-Structure in LFG. Write C-Structure and F-Structure for the sentence - ‘She saw stars’ using CFG rules S -> NP VP VP -> V {NP} {NP} PP* {S’} PP -> P NP NP -> Det N {PP} S’ -> Comp S	[10] Desc-3 Cstructure-3 Fstructure-4	CO2	L3

The term ‘lexical functional’ is composed of two parts:

The lexical part- is derived from the fact that the lexical rules can be formulated to help define the **structure of a sentence**

The functional part - is derived from **grammatical functions such as subject and object or roles played by various arguments in a sentence.**

LFG represents sentences at two syntactic levels –

1. Constituent structure (c-structure)
2. Functional structure (f-structure).

C-Structure and F-structure in LFG

C-Structure

The c-structure is derived from the phrase and sentence structure syntax.. C-Structure is used for encoding linear order constituency and hierarchical relations. This represents the hierarchical organization of words into phrases, typically visualized as a tree diagram. It shows how words combine to form phrases and sentences based on context-free grammar (CFG) rules.

F-structure

f-structure encodes the information obtained from phrase and sentence structure rules and functional specifications. As the grammatical functional role cannot be derived directly from phrase and sentence structure, functional specifications are annotated as the nodes of c-structure, which when applied to sentences, results in f-structure. This represents the syntactic functions and grammatical relations (like subject, object, etc.) of the sentence. It is usually depicted as a feature structure (attribute-value matrix) that captures the grammatical functions and their relationships.

Example: She saw stars in the sky

CFG rules to handle this sentence are:

S -> NP VP
VP -> V {NP} {NP} PP* {S’}
PP -> P NP
NP -> Det N {PP}
S’ -> Comp S

S -> NP VP
NP (She)
VP (saw stars)
VP -> V {NP}
V (saw)
NP (stars)
NP -> N
N (She, stars)

```
     S
    / \
   NP   VP
   |     / \
  N   V   NP
  |    |      |
 She saw stars
```

C-Structure of the sentence - She saw stars

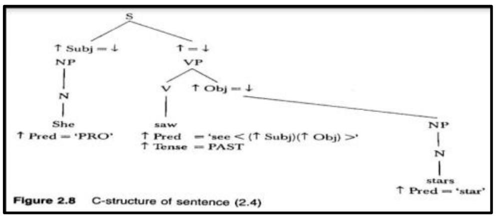


Figure 2.8 C-structure of sentence (2.4)

Finally, the f-structure is the set of attribute-value pairs, represented as

□ She" is the subject.
□ "saw" is the verb.
□ "stars" is the object.

{
PRED 'see<SUBJ, OBJ>'
SUBJ [PRED 'pro']
OBJ [PRED 'star']
TENSE PAST
}

□ **C-Structure**: Shows the hierarchical arrangement of words into phrases using CFG rules.
□ **F-Structure**: Represents the grammatical functions and relationships of the sentence components.

4 (a)

Describe DFA and NFA. Mention the properties of finite automation.

Finite automata are mathematical models used to represent and analyze the behavior of discrete systems. They come in two primary types: Deterministic Finite Automaton (DFA) and Non-deterministic Finite Automaton (NFA).

Deterministic Finite Automaton (DFA)

A DFA is a finite state machine where for each state and input symbol, there is exactly one transition to a next state. In other words, given the current state and an input symbol, the next state is uniquely determined.

Formal Definition: A DFA is defined by a 5-tuple (Q,Σ,δ,q0,F)(Q, \Sigma, \delta, q_0, F)(Q,Σ,δ,q0,F), where:

- QQQ is a finite set of states.
- Σ\SigmaΣ is a finite set of input symbols (alphabet).
- δ:Q×Σ→Q\delta: Q \times \Sigma \rightarrow Qδ:Q×Σ→Q is the transition function.
- q0∈Qq_0 \in Qq0∈Q is the start state.
- F⊆QF \subseteq QF⊆Q is the set of accept states (final states).

Example: Consider a DFA that accepts binary strings ending in "01":

- Q={q0,q1,q2}Q = \{q_0, q_1, q_2\}Q={q0,q1,q2}
- Σ={0,1}\Sigma = \{0, 1\}Σ={0,1}
- δ\deltaδ is defined as:
  - δ(q0,0)=q1\delta(q_0, 0) = q_1δ(q0,0)=q1
  - δ(q0,1)=q0\delta(q_0, 1) = q_0δ(q0,1)=q0
  - δ(q1,0)=q1\delta(q_1, 0) = q_1δ(q1,0)=q1
  - δ(q1,1)=q2\delta(q_1, 1) = q_2δ(q1,1)=q2
  - δ(q2,0)=q1\delta(q_2, 0) = q_1δ(q2,0)=q1
  - δ(q2,1)=q0\delta(q_2, 1) = q_0δ(q2,1)=q0
- q0q_0q0 is the start state.
- F={q2}F = \{q_2\}F={q2}

Non-deterministic Finite Automaton (NFA)

An NFA is a finite state machine where for each state and input symbol, there can be zero, one, or multiple transitions to the next state. This allows the NFA to have multiple possible next states for a given state and input symbol, including the possibility of transitioning without consuming any input symbol (ε-transitions).

Formal Definition: An NFA is defined by a 5-tuple (Q,Σ,δ,q0,F)(Q, \Sigma, \delta, q_0, F)(Q,Σ,δ,q0,F), where:

- QQQ is a finite set of states.
- Σ\SigmaΣ is a finite set of input symbols (alphabet).
- δ:Q×Σ→2Q\delta: Q \times \Sigma \rightarrow 2^Qδ:Q×Σ→2Q is the transition function, where 2Q2^Q2Q denotes the power set of QQQ (i.e., the set of all subsets of QQQ).
- q0∈Qq_0 \in Qq0∈Q is the start state.
- F⊆QF \subseteq QF⊆Q is the set of accept states (final states).

Example: Consider an NFA that accepts the language of binary strings containing "01":

- Q={q0,q1,q2}Q = \{q_0, q_1, q_2\}Q={q0,q1,q2}
- Σ={0,1}\Sigma = \{0, 1\}Σ={0,1}
- • δ\deltaδ is defined as:
- o δ(q0,0)={q0,q1}\delta(q_0, 0) = \{q_0, q_1\}δ(q0,0)={q0,q1}
- o δ(q0,1)={q0}\delta(q_0, 1) = \{q_0\}δ(q0,1)={q0}
  - δ(q1,1)={q2}\delta(q_1, 1) = \{q_2\}δ(q1,1)={q2}
  - δ(q2,0)={q2}\delta(q_2, 0) = \{q_2\}δ(q2,0)={q2}
  - δ(q2,1)={q2}\delta(q_2, 1) = \{q_2\}δ(q2,1)={q2}
- q0q_0q0 is the start state.
- F={q2}F = \{q_2\}F={q2}

Properties of Finite Automata

1. **Determinism**:
   - DFA: For each state and input symbol, there is exactly one transition.
   - NFA: For each state and input symbol, there can be multiple transitions, including none.
2. **Acceptance of Languages**:
   - Both DFAs and NFAs accept the same class of languages, known as regular languages.
   - For every NFA, there is an equivalent DFA that accepts the same language.
3. **Transition Function**:
   - DFA: δ:Q×Σ→Q\delta: Q \times \Sigma \rightarrow Qδ:Q×Σ→Q
   - NFA: δ:Q×Σ→2Q\delta: Q \times \Sigma \rightarrow 2^Qδ:Q×Σ→2Q

[06]
Des-3
Eg:3

CO2

L2

1. **Computational Power**:
   - o Both DFAs and NFAs have the same computational power, meaning they can recognize the same set of regular languages.

4. **Epsilon Transitions (ε-transitions)**:

   - o DFA: Does not allow ε-transitions.
   - o NFA: Allows ε-transitions, where the automaton can move to a new statewithout consuming any input symbol.

2. **Determ inistic vs. Non-deterministic Behavior**:
   - o DFA: Deterministic in nature, meaning the next state is uniquely determined.
   - o NFA: Non-deterministic, meaning there can be multiple possible next states for a given state and input.

3. **Convers ion**:
   - o An NFA can be converted to an equivalent DFA using the subset construction (or powerset construction) method.

4b) Define morphology. Explain Stem and Affix classes of morphemes with examples. Desc-2 Exam-2 CO2 L3

Morphology is the branch of linguistics that studies the structure and form of words in a language. It focuses on the way words are constructed from smaller units called morphemes. A morpheme is the smallest grammatical unit in a language that carries meaning. Morphology examines how these morphemes combine to form words and how they interact within a language's grammatical rules.

Classes of Morphemes
Morphemes can be broadly categorized into two main classes: stems and affixes.
1. Stem (Root)
A stem, or root, is the base part of a word that carries the core meaning. It is the part of the word to which affixes (prefixes, suffixes, infixes, or circumfixes) can be attached to modify its meaning or grammatical function. Stems can often stand alone as words themselves.
Examples:

- In the word "unhappiness":
  - o "happy" is the stem.
- In the w ord "running":
  - o "run" is the stem.

2. Affix

o An affix is a morpheme that is attached to a stem to form a new word or alter its meaning. Affixes cannot stand alone and are always bound morphemes. Affixes can be further divided into different types based on their position relative to the stem:

- **Prefix**: An affix that is attached to the beginning of a stem.
  - o **Examples**:
    - ▪ "un-" in "unhappy" (where "un-" means "not")
    - ▪ "pre-" in "preview" (where "pre-" means "before")
- **Suffix**: An affix that is attached to the end of a stem.
    - ▪ **Examples**: "-ness" in "happiness" (where "-ness" turns an adjective into a noun)
    - ▪ "-ing" in "running" (where "-ing" indicates a present participle or gerund)
- **Infix**: An affix that is inserted within a stem. Infixes are less common in English but are found in other languages.
  - o **Examples**:
  - o In Tagalog (a language spoken in the Philippines), the infix "**-um-**" can be inserted within a root word. For instance, "sulat" (to write)
    - ▪ becomes "sumulat" (wrote).
- **Circumfix**: An affix that surrounds a stem, attaching to both the beginning and the end. Circumfixes are also rare in English but can be found in other languages.
  - o **Examples**:
    - ▪ In German, the word "gearbeitet" (worked) uses the circumfix "ge-...-t" with the stem "arbeit" (work).
- **Morphology** studies the structure of words and how they are formed from morphemes.
- **Stem** (root) is the core part of a word carrying its main meaning.
- Examples: "happy" in "unhappy", "run" in "running"
- Affix is a morpheme attached to a stem to modify its meaning or function.
- o Prefix: "un-" in "unhappy"

- Suffix: "-ness" in "happiness"
    - ○ **Infix**: "-um-" in Tagalog "sumulat"
- o Circumfix: "ge-...-t" in German "gearbeitet"
- Understanding morphology helps in analyzing word formation, language structure, and the way meaning is constructed in different languages.

5a) What is Poss Tagging?List & Explain Different taggers with suitable examples? [05] CO2 L2

Part-of-Speech (POS) tagging is the process of assigning a part of speech to each word in a given text. The parts of speech include categories like nouns, verbs, adjectives, adverbs, pronouns, conjunctions, prepositions, and more. POS tagging helps in understanding the syntactic structure of a sentence and is a crucial step in many natural language processing (NLP) tasks.

Different Types of POS Taggers

There are several types of POS taggers, each employing different methods and algorithms to perform tagging. Here are some of the common types:

1. Rule-Based Taggers

Rule-based taggers use a set of hand-crafted linguistic rules to assign POS tags to words. These rules are based on the morphological, syntactic, and sometimes semantic properties of the words.

Example:

- **Brill Tagger**: The Brill tagger is a well-known rule-based tagger that starts with an initial assignment of POS tags and then applies a series of transformational rules to correct the tags. For instance, it might use rules like "if a word ending in 'ed' is preceded by 'was,' then tag it as a past participle."

Example Sentence:

- "The cat sat on the mat."
- Initial tagging: The/DT cat/NN sat/VBD on/IN the/DT mat/NN.
- After applying rules: (No changes needed in this simple example.)

2. Statistical Taggers

Statistical taggers use probabilistic models to assign POS tags based on the likelihood of a particular sequence of tags. These models are trained on annotated corpora.

Example:

- **Hidden Markov Model (HMM) Tagger**: HMM taggers use the probabilities of tag sequences (state transitions) and the probabilities of words given tags (emissions) to determine the most likely tag sequence for a sentence.

Example Sentence:

- "The cat sat on the mat."
- Tagging: The/DT cat/NN sat/VBD on/IN the/DT mat/NN.

3. Machine Learning-Based Taggers

Machine learning-based taggers use various machine learning algorithms to learn from annotated training data. These taggers can capture more complex patterns in the data.

Examples:

- **Maximum Entropy (MaxEnt) Tagger**: This tagger uses the maximum entropy principle to model the probabilities of different tags.

- **Conditional Random Fields (CRF) Tagger**: CRF taggers model the conditional probabilities of the tags given the input sequence, allowing for the incorporation of various contextual features.

Example Sentence:

- "The cat sat on the mat."
- Tagging with a CRF Tagger: The/DT cat/NN sat/VBD on/IN the/DT mat/NN.

4. Neural Network-Based Taggers

Neural network-based taggers, particularly those using deep learning techniques, have become popular due to their ability to capture complex patterns and dependencies in the data.

Examples:

- **Recurrent Neural Networks (RNNs)**: RNNs, including Long Short-Term Memory (LSTM) networks, can handle sequential data effectively and are used for POS tagging.
- **Bidirectional LSTMs (BiLSTMs)**: BiLSTMs process the sequence in both directions (forward and backward), capturing context from both sides of a word.
- **Transformer-Based Models**: Models like BERT (Bidirectional Encoder Representations from Transformers) have set new benchmarks in POS tagging by using self-attention mechanisms to understand context.

- **Exampl e Sentence:** "The cat sat on the mat."
- Tagging with BERT: The/DT cat/NN sat/VBD on/IN the/DT mat/NN.

5b) Calculate the Minimum Edit Distance Algorithm by given string “abcdef” to “adcgef ”. [5] C02 L3

The Minimum Edit Distance (MED) algorithm, also known as the Levenshtein distance algorithm, calculates the minimum number of operations required to transform one string into another. The operations considered are:

Insertion of a character.
Deletion of a character.
Substitution of a character.

Initialization:

- Create a 2D table where the cell at row iii and column jjj represents the minimum edit distance between the first iii characters of the first string and the first jjj characters of the second string.
- Initialize the first row and first column of the table. The first row represents transforming an empty string into the second string, which requires jjj insertions. The first column represents transforming the first string into an empty string, which requires iii deletions.

1. **Filling the Table**:
   - For each cell (i,j)(i, j)(i,j), check the following conditions:
     - If the characters of both strings match (i.e., str1[i−1]==str2[j−1]str1[i-1] == str2[j-1]str1[i−1]==str2[j−1]), then the value is the same as the diagonal value (i.e., no additional cost is required).
     - If the characters do not match, consider the minimum of the three operations (insertion, deletion, substitution) plus one.

	null	a	d	c	g	e	f
null	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
b	2	1	1	2	3	4	5
c	3	2	2	1	2	3	4
d	4	3	2	2	2	3	4
e	5	4	3	3	3	2	3
f	6	5	4	4	4	3	2

- For cell (1,1): Characters 'a' and 'a' match, so cost = 0 (same as diagonal value).
- For cell (1,2): Characters 'a' and 'd' do not match. Min operations = min(insert 1+1, delete 1+1, substitute 1+1) = 1.
- Continue filling the table using the same logic.

The minimum edit distance (MED) is found in the cell (6,6), which is 2. This means the minimum number of operations required to transform "abcdef" to "adcgef" is 2.

6a)

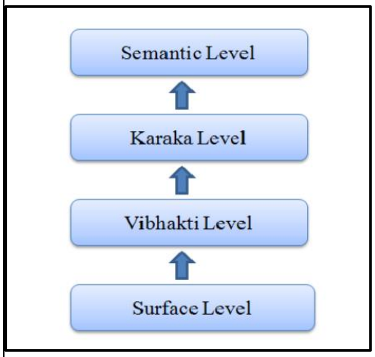
Describe Paninian framework for Indian languages.

[05] CO1 L1

Paninian grammar (PG) was written by Panini in 500 BC in Sanskrit. PG framework can be used for other Indian languages and even for some Asian languages. Unlike English, Asian languages are SOV (Subject-Object-Verb) ordered and inflectionally rich. The inflections provide important syntactic and semantic for language analysis and understanding. The classical Paninian Grammar facilitates the task of obtaining the semantics through syntactical framework. In PG, an extensive and perfect interpretation of Phonology, Morphology, Syntax, and Semantics is available.

Layered representation in panini grammar:

Paninian Grammar (PG) framework is said to be **syntactico–semantic that is one can go from the surface layer to deep semantics by passing through immediate layers.** PG works on various levels of language analysis to achieve the meaning of the sentence from the hearer's perspective. To achieve the desired meaning, the grammar analysis is divided itself internally into various levels as shown in the figure below.



Semantic Level:

- Represents the speaker's actual intention, that is, his real thought for the sentence.

Surface L evel:

- Surface level is the actual string or the sentence. It captures the written or the spoken sentences as it is.
- **Vibhakti Level:** words. At the Vibhakti level, a noun is formed containing a noun, which contains the instances of noun or pronoun, etc.
- Vibhakti is the word suffix, which helps to find out the participants, gender as well as form of the word.
- Vibhakti level is purely syntactic. At this level, the case endings are used to form the local groups of the Vibhakti for verbs includes the verb form and the auxiliary verbs.
- Vibhakti gives Tense Aspect Modality details of the word which is popularly known as TAM.

Karaka Level:

- At the Karaka level, the relation of the participant noun, in the action, to the verb is determined.
- Karaka relations are Syntactico-semantic.
- These relations are established in between the verb and other constituent nouns that are present in the sentences. Through, these relations, the Karakas try to capture the information from the semantics of the texts.
- Kakara level processes the semantics of the language but represents it at the syntactic level. Hence itacts as a bridge between semantic and syntactic analysis of a language.

Karaka T heory:

- The etymological meaning of the word Karaka is 'one who does something', i.e. one who performs an action.
- The Karaka and the Kriya, i.e. the cases and verb are bounded with the sense of mutual requirement.
- The one who performs an action, accepts an action, or otherwise helps to perform an action is known as a Karaka.
- There is a mutual expectancy in between the action i.e. Kriya and the adjuncts i.e. Karaka.
- The presence of one calls for the existence of the other. In other wordsKirya and Karaka are mutually exclusive.

bounded by space (place) or by time.

- Various K **arakas -**

1. Karta - subject
2. Karma - object

1. Karana - instrument
2. Sampradhana - beneficiary
3. Apadana - separation

4. Adhara or Adhikarana - locus
5. Sambandh - relation
6. Tadarthya - purpose

The Karta, Karma, and Karana are considered the foremost Karakas while Sampradana, Apadana, and Adhikarana Karakas are known as the influenced Karakas.

Karta Karaka:

The Karta Karaka is the premier one according to action and it is used to perform an action independently of its own. An action indicated in a sentence is entirely dependent upon the Karta-Karaka. Activity either resides in or rises from the Karta only.
The man cut the wood **with** an axe
Ram cuts the apple **with** knife.

karana - axe and knife

Sampradhana Karaka:

The word Sampradana can be interpreted as 'He to whom something is given properly'. Sampradana Karaka
receives or gets benefited from the action. It can also be said that, the person/object for which the Karma is intentional, is known as Sampradana. In this regard, the Sampradana is the final destination of the action.

Example:
Dipti gave chocolates to Shambhavi
Shambhavi is sampradhana.

Ram gave me a book.
me is sampradhana

He gave flowers for Shanbhavi
Shambhavi is sampradhana

Apadana Karaka:

About Apadana Karaka Panini stated that, as when separation is affected by a verbal action, the point of separation is called Apadana. During the execution of the action whenever the task of separation from a certain entity is executed then **whatever remains unmoved or constant is known as Apadana.** Thus, an Apadana denotes the starting point of an action of separation. The entity from which something gets separated or is separated out is known as Apadana.
Example:
Shambhavi tore the page from the book with a scissor.
From the book is apadana

Adhikarana Karaka:

'Adhikarana' is the place or thing, which is the location of the action existing in the agent or the object. Adhikarana is assigned to the locus of the action i.e. Kriya. Adhikarana may indicate the place at which the Kriya (the action) is taking place or the time at which the Kriya is carried out. Any action i.e. the Kriya is either

Example:
'Yesterday Shambhavi hit the dog with the stick in front of the shop.'
The Karaka annotation of the above sentence can be given as

hit	: verb (root)
Yesterday	: Kala-Adhikarana (time)
Shambhavi	: Agent i.e. Karta
Dog	: Karma
Stick	: Karana
Shop	:Desh-Adhikarana (location i.e. Place)

6b) Explain Transformational Grammar with Examples 5marks Co1 L1

Explanation-3
Steps & Examples -2

Transformational Grammar has 3 Components

Transformational Grammar, introduced by Noam Chomsky in the 1950s, is a theory of grammar that focuses on how different sentence structures can be generated from a common underlying structure through a series of transformations. The core idea is that sentences have a deep structure (which captures the core semantic relations) and a surface structure (which is the actual spoken or written form). Transformations are the rules that convert deep structures into surface structures.

Key Concepts

1. **Deep Structure**: This represents the abstract syntactic representation of a sentence, capturing the basic syntactic relations and meaning.
2. **Surface Structure**: This is the actual sentence as spoken or written, which results from applying transformational rules to the deep structure.
3. **Transformational Rules**: These are rules that transform the deep structure into the surface structure. Examples include movement (e.g., moving a word or phrase to a different position in the sentence), insertion, deletion, and substitution.

Examples of Transformational Grammar

Example 1: Question Formation

Consider the declarative sentence:

- **Deep Structure**: "The cat is on the mat."

To transform this into a question, we apply a movement transformation known as "Subject-Auxiliary Inversion":

- **Surface Structure**: "Is the cat on the mat?"

Transformational Rule:

- Move the auxiliary verb ("is") to the front of the sentence.

Example 2: Passive Transformation

Consider the active sentence:

- **Deep Structure**: "The dog chased the cat."

To transform this into a passive sentence, we apply the passive transformation:

- **Surface Structure**: "The cat was chased by the dog."

Transformational Rules:

1. Move the object ("the cat") to the subject position.
2. Insert the auxiliary verb "was."
3. Add the preposition "by" before the original subject ("the dog").

Example 3: Relative Clause Formation

Consider the sentence with a relative clause:

- **Deep Structure**: "I met the man. The man is a doctor."

To combine these sentences into one with a relative clause:

- **Surface Structure**: "I met the man who is a doctor."

Transformational Rules:

1. Identify the noun phrase common to both sentences ("the man").
2. Replace the repeated noun phrase in the second sentence with a relative pronoun ("who").
3. Embed the second sentence within the first as a relative clause.

Tree Diagrams in Transformational Grammar

Tree diagrams (or syntax trees) are often used to represent the structures of sentences. They visually illustrate the relationships between different parts of a sentence.

Tree Diagram for "The cat is on the mat."

Deep Structure Tree:

```
       S
      / \
    NP   VP
    |    / \
   Det  V   PP
   |    |   / \
  The  is  P   NP
           |   |
          on  Det N
              |   |
             the mat
```

Surface Structure Tree for "Is the cat on the mat?":

```
         S
        / \
      Aux  NP VP
       |    |   / \
      Is  Det  N   PP
           |    |   / \
          the  cat P   NP
                   |   |
                  on  Det N
                      |   |
                     the mat
```