Unit-2

Grammars and Parsing

Grammars and Parsing - Top-Down and Botton-Up parsers, Transition Network Grammars, Feature systems and Augmented Grammars, Morphological Analysis and the Lexicon, Parsing with Features, Augmented Transition Networks, Bayes Rule, Shannon game, Entropy and Cross-Entropy.

Grammars and Parsing :-

Grammars and Parsing play a crucial role in NLP. In NLP, grammar is a set of rules that defines the structure of a language. It specifies how sentences in that language should be formed.

Grammars can be formalized using different formalisms like CFG - Context Free Grammars or dependency grammars.

Parsing on the other hand, is the process of analyzing a sequence of symbols according to the rules of a grammar.

It's like breaking down a sentence into its grammatical components to understand its syntactic structure.

There are 2 main types of parsing in NLP:

- Syntactic Parsing or Syntax Parsing.
- Semantic Parsing.

Syntactic Parsing:

This deals with the grammatical structure of

sentences. It involves breaking down sentences into grammatical components such as nouns, verbs, adjectives etc and identifying the relationship blw them.

Semantic parsing:
This goes beyond syntax and aims to understand the meaning of sentences.
It involves mapping sentence structures to a formal representation of their meaning.

One popular tool for parsing in NLP is the use of parsers. These are programs that implement algorithms to analyze the grammatical structure of sentences based on a given grammar.

Some parsers are rule-based, while others use statistical or ML approaches.

Overall, grammars and parsing are fundamental in NLP for tasks like information extraction, question answering, and machine translation, helping machines understand and generate human-like language.

To examine, how the syntactic structure of a sentence can be computed, you must consider two things:

→ The grammar, which is a formal specification of the structures allowable in the language.
→ The parsing technique, which is method of analyzing a sentence to determine its structure according to the grammar.

Grammars and sentence structure:
The most common way of representing how a sentence is broken into its major subparts;

and how those subparts are broken up in turn as a tree.

John ate the cat.

S (NP) VP(V) NP N

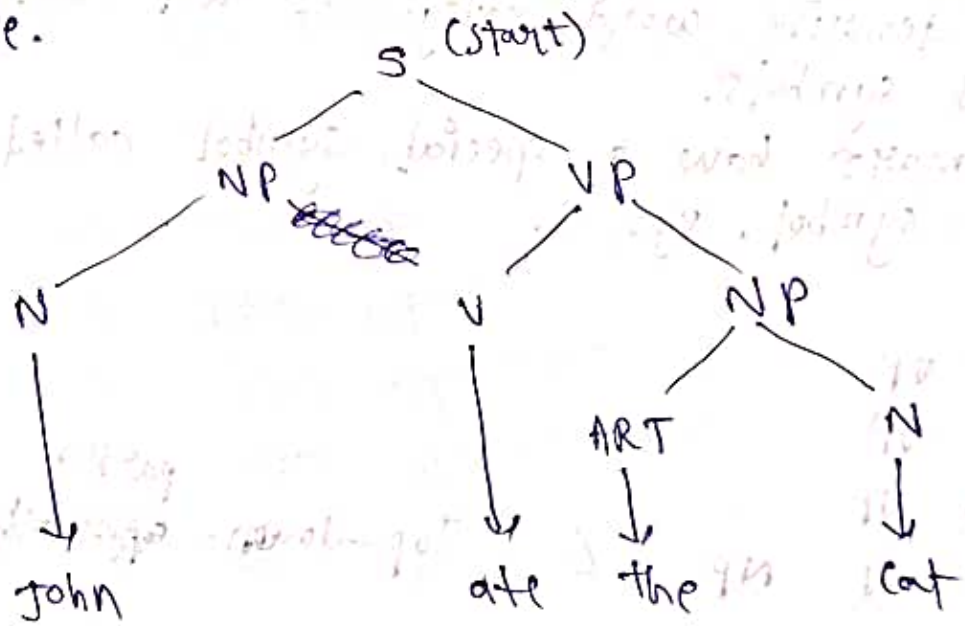
(ART)
article

Trees are a special form of graph; which are structures consisting of labeled nodes connected by links. They are called trees because they resemble upside-down trees.

The node at the top is called the root of the tree, while the nodes at the bottom are called the leaves.

A link points form a parent node to a child node. while every child node has a unique parent, a parent may point to many child nodes.

The root node dominates all other nodes in the tree.



A tree representation.

S → NP VP	N → John
VP → V NP	V → ate
NP → N	ART → the
NP → ART N	N → cat

Simple Grammar

To construct a tree structure for a sentence, you must know what structures are legal for English. A set of rewrite rules describes what tree structures are allowable.

Grammars consisting entirely of rules with a single symbol on the left-hand side, called the mother, are called CFGs.

CFGs are a very important class of grammars for two reasons:

→ The formalism is powerful enough to describe most of the structure in NL.

→ Symbols that can not be further decomposed in a grammar are called terminal symbols. The other symbols such as NP, VP are called non-terminal symbols.

The grammatical symbols such as N and V that describe word categories are called lexical symbols.

Grammars have a special symbol called the start symbol. eg: 'S'.

S
⇒ NP VP
⇒ N VP
⇒ John VP
⇒ John V NP ← Top-down parser
⇒ John ate NP
⇒ John ate ART N
⇒ John ate the N
⇒ John ate the cat.

Two important processes are based on derivations. The first is sentence generation, which uses derivations to construct legal sentences.

A simple generator could be implemented by randomly choosing rewrite rules, starting from the 'S' symbol, until you have a sequence of words.

The second process based on derivations is parsing, which identifies the structure of sentences given a grammar.

There are 2 basic methods of searching.

→ A top-down strategy starts with the S symbol and then searches through different ways to rewrite the symbols until the input sequence is generated.

→ In a bottom-up strategy, you start with the words in the sentence and use the rewrite rules backward to reduce the sequence of symbols until it consists solely of S.

The left-hand side of each rule is used to rewrite the symbol on the right-hand side.

⇒ John ate the cat
⇒ N ate the cat
⇒ N V the cat
⇒ N V ART cat
⇒ N V ART N
⇒ NP V ART N
⇒ NP V NP
⇒ NP VP
⇒ S

← Button-up parser.

S → NP VP
NP → ART N
NP → ART ADJ N
VP → V
VP → V NP

A Top-Down Parser :-

In top-down parsing, the analysis begins with the start symbol of the grammar and tries to transform it into the input sentence by applying production rules.

It starts from the top of the parse tree (root) and proceeds towards the leaves.

A popular top-down parsing technique is RDP - Recursive Descent Parsing, where each non-terminal in the grammar is associated with a parsing function.

Pros:

→ Intuitive : Resembles how humans might construct sentences from high-level structures.

→ Efficient for LL(k) grammars. (left-to-right scan, leftmost derivation with 'k' tokens of lookahead)

Cons:

→ Can get stuck in backtracking if the chosen production rule leads to an incorrect path.

→ Not suitable for all types of grammars.

A top-down parser starts with the S symbol and attempts to rewrite it into a sequence of terminals symbols that matches the classes of the words in the I/p sentence.

The state of the parse at any given time can be represented as a list of symbols that are the result of operations applied called the symbol list.

Rather than having a separate rule to indicate the possible syntactic categories for each word, a structure called the lexicon is used to efficiently

store the possible categories for each word.
eg: cried : V
dogs : N/N
the : ART

A parsing algorithm that is guaranteed to find a parse if there is one must systematically explore every possible new state. One simple technique for this is called backtracking.

Using this approach, rather than generating a single new state from the state (VP) you generate all possible new states.

A simple Top-Down Parsing Algorithm:

The algorithm manipulates a list of possible states called the possibilities list.
The first element of this list is the current state, which consists of a symbol list and a word position in the sentence.
The remaining elements of the search state are the backup states, each indicating an alternate symbol list - word position pair.

eg: (((N) 2 ((NAME) 1) ((ADJ N) 1))

↓ current state

↓ Backup states

The algorithm starts with the initial state (S) 1 and no backup states.

(i) Select the current state: Take the first state off the possibilities list and call it C. If the possibilities list is empty, then the algorithm fails.

(ii) If 'C' consists of an empty symbol list and the word position is at the end of the sentence, then the algorithm succeeds.

(iii) Otherwise, generate the next possible states.

→ If the first symbol on the symbol list of 'C' is a lexical symbol, and the next word in the sentence can be in that class, then create a new state by removing the first symbol from the symbol list and updating the word position, and add it to the possibilities list.

→ otherwise, if the first symbol on the symbol list of 'C' is a non-terminal, generate a new state for each rule in the grammar that can rewrite that non-terminal symbol and add them all to the possibilities list.

Step	Current State	Backup State
1.	((S) 1)	
2.	((NP VP) 1)	
3.	((ART N VP) 1)	
4.		((ART ADJ N VP) 1)
4.	((N VP) 2)	
		((ART ADJ N VP) 1)
5.	((VP) 3)	
		((ART ADJ N VP) 1)
6.	((V) 3)	
		((V NP) 3)
		((ART ADJ N VP) 1)

Top-Down depth first parse of "The dogs cried".

Parsing as a search Procedure:

The possibilities list is initially set to the start state of the parse.

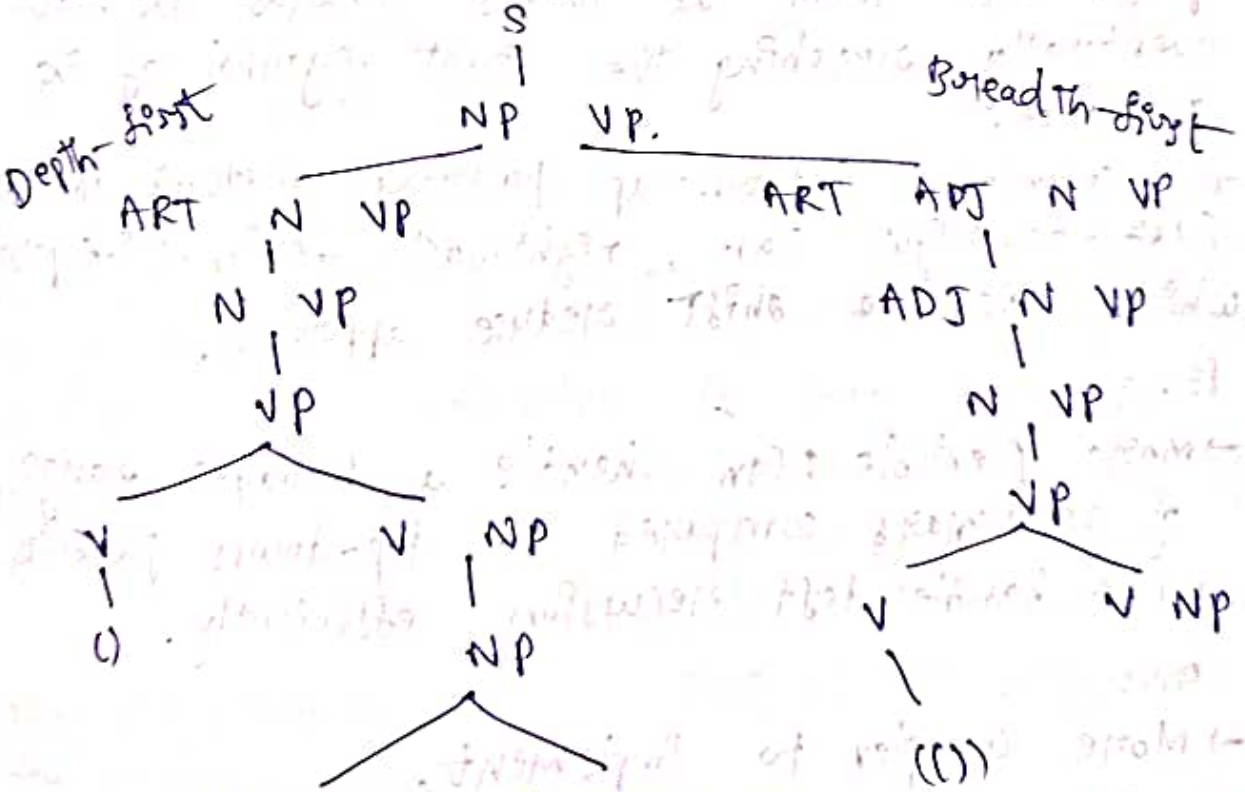
Repeat the following steps until you have success or failure.

→ select the first state from the possibilities list (and remove it from the list).

→ Generate the new states by trying every possible option from the selected state.

→ Add the states generated in step 2 to the possibilities list.

For a depth-first strategy, the possibilities list is a stack. Step-1 always takes the first element off the list, and step-3 always puts the new states on the front of the list, yielding a LIFO strategy.



Search Tree

Each node in the tree represents a parser state, and the sons of a node are the possible moves from the state.

The main difference between depth-first and breadth-first searches is the order in which the two possible interpretations of the first NP are examined.

With the depth-first strategy, one interpretation is considered and expanded until it fails, only then is the second one considered.

With the breadth-first strategy, both interpretations are considered alternately, each being expanded one step at a time.

A Bottom-Up Chart Parser:-

In bottom-up parsing, the analysis starts with the input sentence and tries to build the parse tree from the leaves towards the root, eventually reaching the start symbol of the grammar.

A common bottom-up parsing method is LR (left-to-right scan, rightmost derivation) parsing which uses a shift-reduce approach.

Pros:

→ More flexible: Can handle a broader range of grammars compared to top-down parsers.

→ Can handle left recursion efficiently.

Cons:

→ More complex to implement.

→ Might not be as intuitive as top-down parsing.

The main difference b/w top-down and bottom-up parsers is the way the grammar rules are used.

eg: NP → ART · ADJ N

The basic operation in bottom-up parsing then is to take a sequence of symbols and match it to the right-hand side of the rules.

→ Rewrite a word by its possible lexical categories.

→ Replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side symbol.

S → N PVP

NP → ART ADJ N

NP → ART N

NP → ADJ N

VP → AUX VP

VP → V NP

Simple CFG.

A data structure called a chart is introduced that allows the parser to store the partial results of the matching it has done.

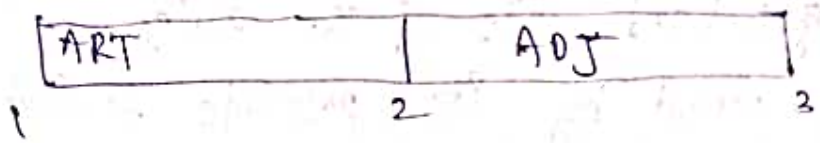
Matches are always considered from the point of view of one constituent, called the key.

To find rules that match a string involving the key, look for rules that start with the key.

The chart maintains the record of all the constituents derived from the sentence.

It also maintains the record of rules that have matched partially but are not complete. These are called the active arcs.

The active arcs indicating possible constituents. These are indicated by the arrows and are interpreted from top to bottom.



NP → ART ∘ ADJ N

NP → ART ∘ N

NP → ADJ ∘ N

NP → ART ADJ ∘ N

The chart after seeing an ADJ in posi-2.

The arc extension algorithm:

To add a constituent C from position P_1 to P_2:

→ Insert C into the chart from position P_1 to P_2.

→ For any active arc of the form $X \rightarrow X_1 \ldots \circ C \ldots X_n$ from position P_0 to P_1, add a new active arc $X \rightarrow X_1 \ldots C \circ \ldots X_n$ from position P_0 to P_2

→ For any active arc of the form $X \rightarrow X_1 \ldots X_n \circ C$ from position P_0 to P_1, then add a new constituent of type X from P_0 to P_2.

The basic operation of a chart-based parser involves combining an active arc with a completed constituent.

The result is either a new completed constituent or a new active arc that is an extension of the original active arc.

New completed constituents are maintained on a list called the agenda until they themselves are added to the chart.

As with the top-down parsers, you may use a depth-first or breadth-first search strategy, depending on whether the agenda is implemented as a stack or a queue.

A bottom-up chart parsing algm:

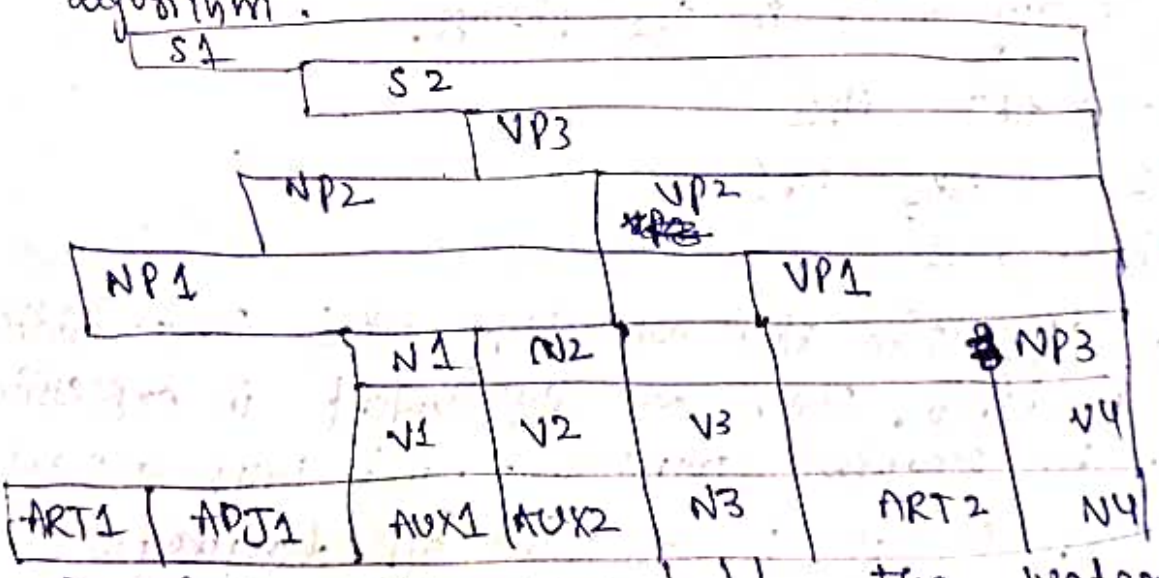
Do until there is no input left:

(i) If the agenda is empty, look up the interpretations for the next word in the input and add them to the agenda.

(ii) Select a constituent from the agenda.

(iii) For each rule in the grammar of form $X \rightarrow C\ X_1 \cdots X_n$, add an active arc of form $X \rightarrow C \circ C \circ X_1 \cdots X_n$ from position p1 to p2.

(iv) Add 'C' to the chart using arc extension algorithm.



The large can can hold the waters. The final chart

Efficiency Considerations:

Chart-based parsers can be considerably more efficient than parsers.

A pure top-down or bottom-up search strategy could require up to c^n operations to parse a sentence of length n, where c is a constant that depends on the specific algo you use.

The worst-case complexity of $k^* n^3$, where n is the length of the sentence, and k is constant depending on the algo.

Transition Network Grammars:

It is based on the notion of a transition n/w consisting of nodes and labeled arcs.

One of the nodes is specified as the initial state (or start state).

Starting at the initial state, you can traverse an arc if the current word in the sentence is in the category on the arc.

If the arc is followed, the current word is updated to the next word.

A phrase is a legal NP if there is a path from the node NP to a pop arc that accounts for every word in the phrase.

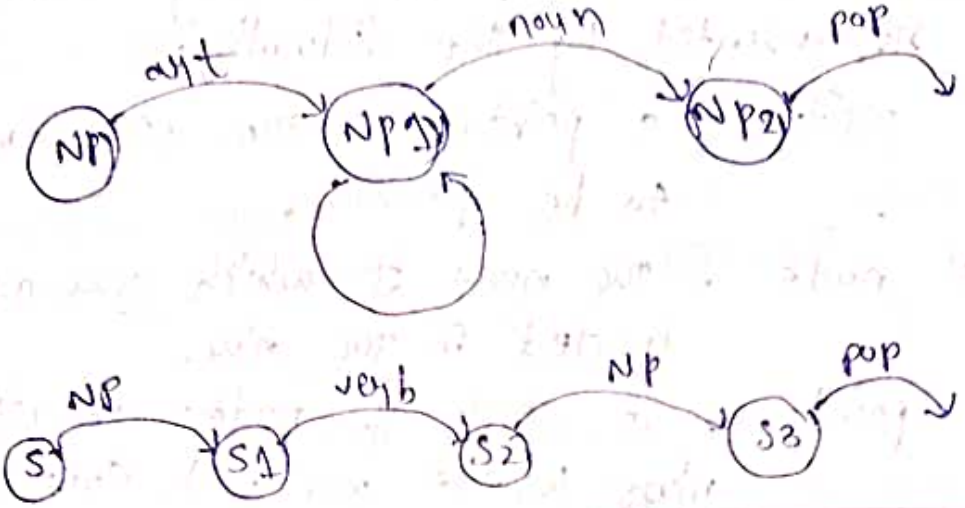
NP ⟶ ART NP1

NP1 ⟶ ADJ NP1

NP1 ⟶ N

Simple transition nets are often called FSMs - Finite State Machines. FSMs are equivalent in expressive power to regular grammars, and thus are not powerful enough to describe all languages that can be described by a CFG.

A recursive transition n/w (RTN) is like a simple transition n/w, except that it allows arc labels to refer to other n/ws as well as word categories. Uppercase labels refer to n/ws.



Arc Type	Example	Arc Type Ex How Used
CAT	noun	"succeeds only if" current word is of the named category
WRD	of	↳ current word is identical to the label
PUSH	NP	↳ named n/w can be successfully traversed.
JUMP	jump	always succeeds
POP	pop	succeeds and signals the successful end of the n/w.

The arc labels for RTNs.

In practice, RTN systems incorporate some additional arc types that are useful but not formally necessary.

Arcs that are labeled with n/ws are called push arcs, and arcs labeled with word categories are called cat arcs. An arc that can always be followed is called a jump arc.

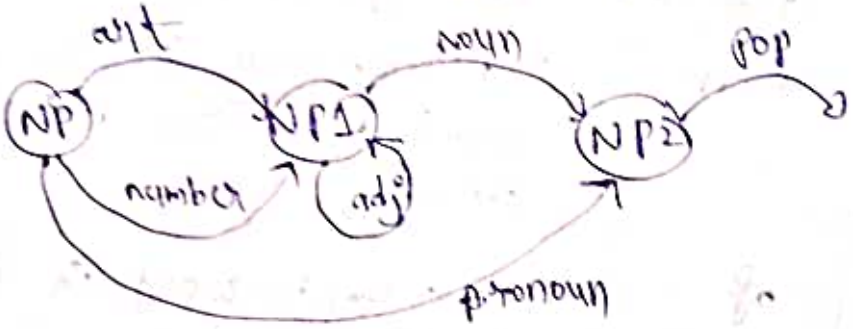
Top Down Parsing with RTN:

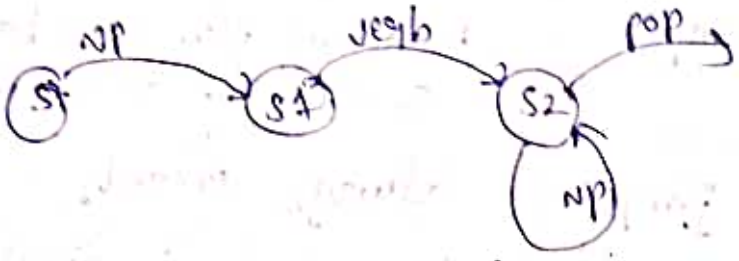
An algm for parsing with RTNs can be developed along the same lines as the algm for parsing CFGs. The state of the parser at any moment can be represented by the following:

current position - a pointer to the next word to be parsed.

current node - the node at which you are located in the n/w.

return points - a stack of nodes in other n/ws where you will continue if you pop from the current n/w.





An RTN parser can be constructed to use a chart-like structure to gain the advantage of chart parsing.

In RTN systems, the chart is often called the well-formed substring table (WFST).

An RTN using a WFST has the complexity $k * n^3$ where, n is the length of the sentence.

Transition n/ws are a graphical representation of the structure of a language and how it can be parsed. They are used in computational linguistics for NLP.

Nodes: Represent states (or conditions.
Arcs / Edges: Represent transitions b/w states.

Transition n/ws are often used to model the process of recognizing or generating sentences.
There are 2 main types.

Recognition Transition N/ws:
Used for recognizing whether a given input sentence belongs to a particular language.
Each node corresponds to a state of the parsing process, and arcs represent transitions based on i/p symbols.

Generation Transition N/ws:
Used for generating sentences in a particular language.
Each node represents a state in generating a sentence and arcs represent possible transitions or choices in constructing the sentence.

Transition n/ws are particularly flexible and can be adapted for various tasks in NLP, including parsing, language generation, and understanding.

Features systems and Augmented Grammars:

In NL, there are often agreement restrictions b/w words and phrases.
eg: "a men" is not correct English.
(NP) a → single obj
men → plural obj.

There are many other forms of agreement, including subject-verb agreement, gender agreement for pronouns, restrictions btw the head of a phrase and the form of its complement and so on.

To handle such phenomena conveniently, the grammatical formalism is extended to allow constituents to have features.

NP → ART N only when $Number_1$ agrees with $Number_2$.

This rule says that a legal noun phrase consists of an article followed by a noun, but only when the number feature of the first word agrees with the number feature of the second.

This one rule is equivalent to 2 CFG rules, that would use different terminal symbols for encoding singular and plural forms of all noun phrases such as

NP-SING → ART-SING N-SING
NP-PLURAL → ART-PLURAL N-PLURAL

A feature structure – a mapping from features to values that defines the relevant properties of the constituent.

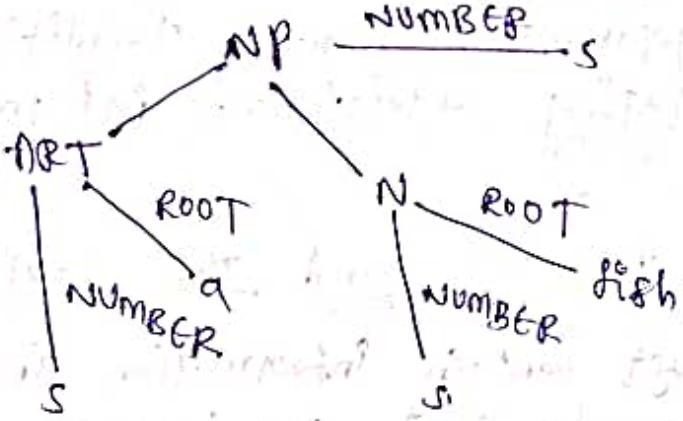
```
ART1 :
        (CAT ART
         ROOT a
         NUMBER s)

ART1 : (ART ROOT a NUMBER s)
```

Feature structures can be used to represent larger constituents as well. To do this, feature structures themselves can occur as values.

Special features based on the integers -1, 2, 3 and so on - will stand for the first subconstituent, second subconstituent and so on.

NP1: (NP NUMBER S)
1 (ART ROOT a NUMBER S) $\in$ {"a fish"}
2 (N ROOT fish NUMBER S))

The rules in an augmented grammar are stated in terms of feature structures rather than simple categories. Variables are allowed as feature values so that a rule can apply to a wide range of situations.



Viewing a feature structure as an extended parse tree

Variables are also useful in specifying ambiguity in a constituent.

Formalizing Feature Structures:

There is an active area of research in the formal properties of feature structures.

A feature structure is defined as a partial function from features to feature values.

ART1 : (CAT ART
ROOT a
NUMBER s)

ART1 (CAT) = ART ∧ ART1 (ROOT) = a ∧ ART1 (NUMBER) =

There is an interesting issue of whether an augmented context-free grammar can describe languages that can not be described by a simple CFG

Feature Systems:

In linguistics and NLP, features refer to distinctive characteristics (or properties of linguistic elements (like words or phrases).

Feature systems are used to represent and describe these features in a structured way.

Augmented Grammars:

Augmented grammars expand traditional grammars by incorporating additional information (in features.

Morphological Analysis and the Lexicon :-

The lexicon must contain information about all the different words that can be used, including all the relevant feature value restrictions.

When a word is ambiguous, it may be described by multiple entries in the lexicon, one for each different use.

Most English verbs use the same set of suffixes to indicate different forms :- -s is added for 3rd person singular present tense, -ed for past

tense, -ing for the present participle and so on.

Without any morphological analysis, the lexicon would have to contain every one of these forms.

eg: want ⇒ want, wants, wanting, wanted

The idea is to store the base form of the verb in the lexicon and use ~~context~~ context-free rules to combine verbs with suffixes, to derive the other entries.

Binary features would be needed to flag irregular past forms and to distinguish -en past participles from -ed past participles.

These features restrict the appn of the standard lexical rules, and the irregular forms are added explicitly to the lexicon.

Given a larger set of features, the task of writing lexical entries appears very difficult.

The first technique allowing default values for features has already been mentioned.

Another commonly used technique is to allow the lexicon writing to define clusters of features and then indicate a cluster with a single symbol rather than listing them all.

With an algm for stripping the suffixes and regularizing the spelling, the derived entries can be generated using any of the basic parsing algms.

With the lexicon correct constituents for the following words can be derived: been, being, cries, cried, crying, dogs, saws, sawed, sawing, seen, sees, feeds, wants, wanting and wanted.

Present Tense :

(V ROOT ?r SUBCAT ?s VFORM pres NUM 3s)

V ROOT ?r SUBCAT ?s VFORM base IRREG-PRES

etc)

eg: saw : (CAT V
ROOT SEE1
VFORM past
SUBCAT _np)

see : (CAT V
ROOT SEE1
VFORM base
SUBCAT _np
IRREG-PAST +
EN-PASTPRT +)

Often a word will have multiple interpretations that use different entries and different lexical rules.

Morphological Analysis :

Morphology is the study of the structure of words and how they are formed from smaller meaningful units called "morphemes."

Morphological analysis involves breaking down words into morphemes and understanding their grammatical and semantic functions.

eg: unhappiness.

un → morpheme indicating negation.

happy → root morpheme (base meaning)

ness → morpheme denoting a state or quality.

In NLP, morphological analysis is crucial for tasks like stemming and lemmatisation aiding in text processing and information retrieval.

Lexicon :

The lexicon is like the vocabulary bank of a language. It contains information about words,

including their meanings, and pronunciations, and grammatical properties.

Lexical entries often include morphological information, allowing NLP systems to understand the structure and usage of words in a language.

In NLP apps, the lexicon is crucial for tasks like NER, sentiment analysis, and machine translation.

Parsing with Features:-

The parsing algms developed for CFGs can be extended to handle augmented CFGs.

This involves generalizing the algm for matching rules to constituents.

The chart parsing algms developed all used an operation for extending active arcs with a new constituent.

eg: $C \rightarrow C1 \ldots Ci \circ X \ldots Cn.$

A similar operation can be used for grammars with features, but the parser may have to instantiate variables in the original arc before it can be extended by X.

(NP AGR ?a) → o (ART AGR ?a) (N AGR ?a)

(ART ROOT A AGR 3S)

(NP AGR 3S) → o (ART AGR 3S) (N AGR 3S)

When constrained variables such as ?a {3S 3P}, are involved, the matching proceeds in the same manner, but the variable binding must be one of the listed values.

Another extension is useful for recording the structure of the parse.

Subconstituent features are automatically inserted by the parser each time an arc is extended.

Parsing with features adds an extra layer of sophistication to the analysis by incorporating additional information about the linguistic elements involved.

Features can represent attributes such as gender, number, tense or any other relevant characteristics of words or phrases.

eg: S → NP VP

S → NP [Gender = ?g] VP [Tense = ?t]

NP [Gender = feminine] VP [Tense = present]

Parsing with features is commonly used in computational linguistics and NLP for tasks such as syntactic and semantic parsing.

Features help capture not just the structure but also the nuanced details and relationships b/w different linguistic elements, making the parsing process more context-aware and semantically informed.

Augmented Transition Networks :-

Features can also be added to a RTN to produce an ATN.

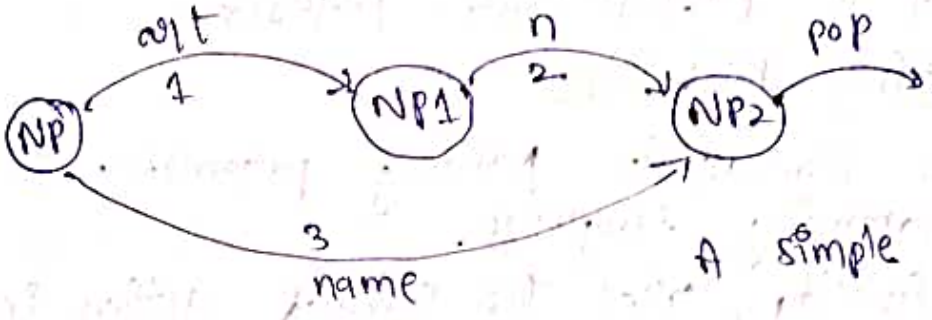
Features in an ATN are traditionally called registers. Constituent structures are created by allowing each n/w to have a set of registers.

ATN use a special mechanism to extract the result of an arc.

eg: DET := *

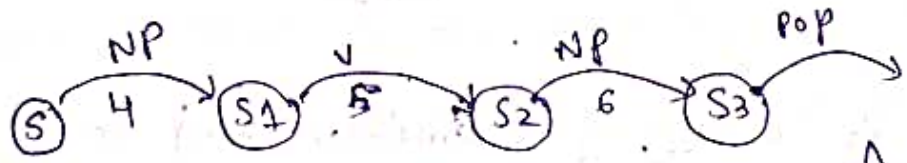
AGR := AGR*

Agreement checks are specified in the tests. A test is an expression that succeeds if it returns a non-empty value and fails if it returns the empty set (or) nil.



A simple NP n/w.

ARC	Test	Action
1	none	DET := * AGR := AGR*
2	AGR ∩ AGR*	HEAD := * AGR := AGR ∩ AGR*
3	none	NAME := * AGR := AGR*



A simple 'S' n/w.

ARC	Test	Action
4	none	SUBJ := *
5	AGR_{subj} ∩ AGR*	MAIN-V := * $AGR := AGR_{subj}$ ∩ AGR
6	none	OBJ := *

If a test fails, its arc is not traversed. The constituent built by traversing the NP n/w is returned as the value *.

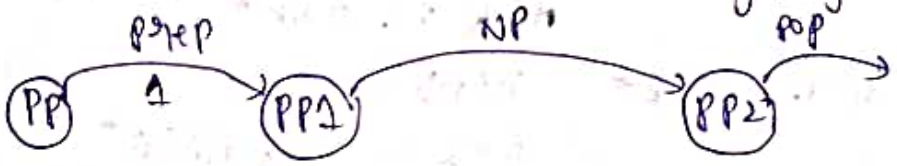
An ATN Grammar for Simple Declarative Sentences:

The allowed sentence structure is an initial NP

followed by a main verb, which may then be followed by a max of two NPs and many PPs depending on the verb.
Allowable noun complements include an optional number of prepositional phrases.

Presetting Registers:

It is similar to passing parameter in a programming language.
This facility, called the SENDER action in the original ATN systems, is useful to pass information to the n/w that aids in analyzing the new constituent



The PP n/w.

Arc	Test	Action
PP/1	---	P := *
PP1/1	---	POBJ := *

ATN:

An ATN is a type of formalism used in computational linguistics for parsing natural language.
It extends the concept of transition n/w by incorporating procedures and conditions at transitions, making it more expressive and adaptable.

Components:

States: represent linguistic structures or conditions.
Transitions: Describe allowable moves from one state to another.
Procedures: specify actions or operations associated with transitions.

Conditions: Represent constraints or checks that must be satisfied for a transition to occur.

Benefits:
- → Flexibility
- → Modularity.
- → Adaptability.

Challenges:
- → Complexity.
- → Efficiency.

Bayes Rule:

Bayes rule (or Bayes theorem) plays a crucial role in NLP, particularly in probabilistic models of text and speech.
It provides a way to update the probability estimate for a hypothesis as more evidence or information becomes available.

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

where, $P(H|E)$ is the posterior probability of H given E

$P(E|H)$ is the likelihood of E given H.

$P(H)$ is the prior probability of H.

$P(E)$ is the marginal probability of E.

Applications in NLP:

In NLP, Bayes theorem is often used in text classification, spam filtering, sentiment analysis, and other tasks where the goal is to infer some hidden information from observed data.

Naive Bayes classifier:

One of the most common apps of Bayes theorem in NLP is the Naive Bayes classifier.
This classifier is very effective for text classification tasks.

The "naive" part assumes that the features (words in the text) are independent given the class label.

The Naive Bayes classification process involves:

(i) Training Phase: estimating the probabilities from the training data.

(ii) Prediction Phase: using Bayes theorem to predict the class of a new document.

(iii) Choosing the class: selecting the class with the highest posterior probability.

Shannon Game:-

The Shannon Game, named after Claude Shannon, a pioneer in information theory, is an interesting concept in the field of NLP.

The game involves predicting the next letter (or word) in a sequence based on the preceding text.

Shannon used this idea to estimate the entropy of the English language, which refers to the amount of information and predictability within a language.

Procedure:

(i) Text Selection: Choose a text passage from a language corpus.

(ii) Prediction Task: Given a sequence of characters (or words), predict the next character (or word) in the sequence.

(iii) Evaluation: Compare the predictions with the actual text to evaluate the model's performance.

Applications in NLP:

In NLP, the Shannon Game can be applied to evaluate language models, such as n-gram models, HMMs - Hidden Markov Models.

N-gram Models:

Unigram Model: Predicts the next word based on the frequency of words in the corpus.

Bigram Model: Predicts the next word based on the previous word.

Trigram Model: Predicts the next word based on the two preceding words, and so forth.

NL Models:

Modern NLP approaches use NN, such as RNN, and LSTMs, and transformers like GPT-3 & BERT to predict the next word (or char in a seq).

Entropy and Redundancy:

Shannon used the game to measure the entropy of English, which quantifies the uncertainty or unpredictability in the language.

lower entropy means higher predictability in the language.

Entropy and Cross-Entropy:-

In the context of NLP, entropy and cross-entropy that measure the unpredictability and the performance of probabilistic models.

Entropy:

Entropy quantifies the uncertainty (or unpredictability of a random variable.

$$H(X) = \sum_{i=1}^{n} P(x_i) \log P(x_i).$$

For a language model, entropy can be thought of as the avg amount of information needed to predict the next element in a sequence.

Cross-Entropy:

It measures the difference b/w two probability distributions: The true distribution of the data and the predicted distribution Q.

$$H(P, Q) = \sum_{i=1}^{n} P(x_i) \log Q(x_i).$$

In NLP, cross entropy is used to evaluate language models. A lower cross-entropy indicates that the model's predictions are closer to the true distribution of the data.

Kullback-Leibler Divergence:

The difference b/w cross-entropy and entropy is captured by the KL divergence.

$$D_{KL}(P \| Q) = H(P, Q) - H(P).$$

KL divergence is always non-negative and is zero if and only if $P = Q$.

Applications in NLP:

→ Language modeling
→ Text generation.
→ Machine translation
→ Speech Recognition.
→ Text classification.