Unit - 2

Grammars and Parsing

Grammars and Parsing :-

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

→ By introducing a notation for describing the structure of natural language and describing some naive parsing techniques for that grammar. It describes some characteristics of a good grammar.

→ Consider a simple parsing techniques and introduces the idea of parsing as a search process.

→ It describes a method for building efficient parsers using a structure called chart.

→ It describes an alternative representation of grammars based on transition networks.

→ It describes a top-down chart parser that combines the advantages of top-down and bottom-up approaches.

→ The notion of finite state transducers and discusses their use in morphological processing.

→ It shows how to encode context-free grammars as assertions in PROLOG, introducing the notion of logic grammars.

Grammars and Sentence Structure :-

→ Consider methods of describing the structure of sentences and explores ways of characterizing all the legal structures in a language. The most common way of representing how a sentence is broken into its major subparts and how those subparts are broken up in turn is as a tree. The tree representation for the sentence John ate the cat is shown in Fig 3.1

→ This illustration can be read as follows :

The sentence (S) consists of an initial noun phrase (NP) and a verb phrase (VP). The initial noun phrase is made of

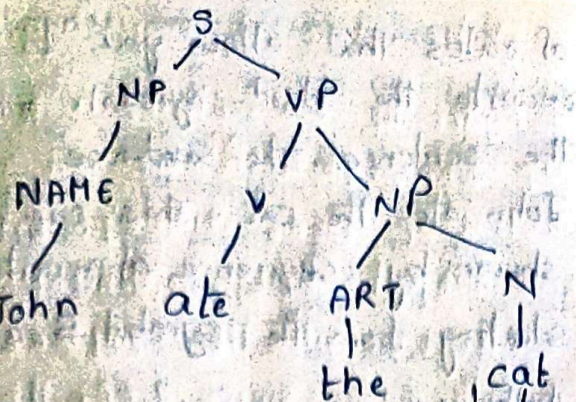


Fig 3.1 A tree representation of John ate the cat

1. S → NP VP
2. VP → V NP
3. NP → NAME
4. NP → ART N
5. NAME → John
6. V → ate
7. ART → the
8. N → cat

Grammar 3.2 A simple grammar

simple NAME John. The verb phrase is composed of a verb (V) ate and an NP, which consists of an article (ART) the and a common noun (N) cat

→ In list notation this same structure could be represented as

(S (NP (NAME John))
(VP (V ate)
(NP (ART the)
(N cat))))

→ 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, and much of the terminology is derived from this analogy with actual trees. The node at the top is called the root of the tree, while nodes at the bottom are called the leaves. We say a link points from a parent node to a child node. The node labeled S in Fig 3.1 is the parent node of the nodes labelled NP and VP, and the node labelled NP is the parent node of the node labelled NAME. While every child node has a unique parent, a parent may point to many child nodes. An ancestor of a node N is defined as N's parent, or the parent of it's parent, and so on. A node is dominated by it's ancestor nodes. The root node dominates all other nodes in the tree.

To construct a tree structure for a sequence sentence, you must know the structures are legal for English. A set of rewrite rules describes what tree structures are allowable. These rules say that a certain symbol may be expanded in the tree by a sequence of other symbols.

Rule 1 says that an S may consist of an NP followed by a VP. Rule 2 says that a VP may consist of a V followed by an NP. Rules 3 and 4 say that an NP may consist of a

NAME or may consist of an ART followed by an N. Rules 5-8 define possible words for the categories. Grammars consisting entirely of rules with a single symbol on the left-hand side, called the mother, are called context-free grammers (CFGs). CFGs are a very important class of grammers for two reasons: The formalism is powerful enough to describe most of the structure in natural languages. Yet it is restricted enough so that efficient parsers can be built to analyze sentences. Symbols that cannot be further decomposed in a grammar, namely the words in the preceding example, are called terminal symbols. The other symbols, such as NP, VP, and S, are called nonterminal symbols. The grammatical symbols such as N and V that describe word categories are called lexical symbols. Of course, many words will be listed under multiple categories. For example, can would be listed under V and N.

Grammars have a special symbol called the start symbol. The start symbol will always be S. A grammar is said to derive a sentence if there is a sequence of rules that allow you to rewrite the start symbol into the sentence. The sentence John ate the cat. This can be shown by a sequence of rewrites starting from the symbol S, as follows:

S
⇒ NP VP (rewriting S)
⇒ NAME VP (rewriting NP)
⇒ John VP (rewriting NAME)
⇒ John V NP (rewriting VP)
⇒ John ate NP (rewriting V)
⇒ John ate ART NP (rewriting NP)
⇒ John ate the N (rewriting ART)
⇒ John ate the cat (rewriting N)

Two important processes are based on derivations. The first is sentence generation, which uses derivations to construct legal sentences. A simple ~~great~~ ~~gender~~ ~~S symbol~~ generator could be implemented by randomly choosing rewrite rules, starting from the S symbol, until you have a sequence of words. The preceeding example shows that the sentence John ate the cat can be generated from the grammar. The second process based on derivations in parsing, which identifies the structure of sentences given a grammar. There are two 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 sentence is generated, or until all possibilities have been explored. The preceeding example demonstrates that John ate the cat is a legal sentence by showing the derivation that could be found by this process.

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. A possible bottom-up parse of the sentence John ate the cat is

⇒ NAME ate the cat	(rewriting John)
⇒ NAME V the cat	(rewriting ate)
⇒ NAME V ART cat	(rewriting the)
⇒ NAME V ART N	(rewriting cat)
⇒ NP V ART N	(rewriting NAME)
⇒ NP V NP	(rewriting ART N)
⇒ NP V VP	(rewriting V NP)
⇒ S	(rewriting NP VP)

A tree representation, such as Fig 3.1, can be viewed as a record of the CFG rules that account for the sentence. In other words, if you kept a record of the parsing process, working either top-down or bottom-up, it would be something similar to the parse tree representation.

What Makes a Good Grammar:-

In constructing a grammar for a language, you are interested in generality, the range of sentences the grammar analyzes correctly; selectivity, the range of non-sentences it identifies as problematic; and understandability, the simplicity of the grammar itself.

One structural analysis of a sentence may appear as understandable as another, and little can be said as to why one is superior to the other. As you attempt to extend a grammar to cover a wide range of sentences, however you often find that one analysis is easily extendable while the other requires complex modification. It is extended is more desirable. In particular, pay close attention to the way the sentence is divided into it's subparts, called constituents. Besides using your ~~attent~~ intution, you can apply a few specific tests, discussed here.

Anytime you decide that a group of words forms a particular constituent, try to construct a new sentence that involves that group of words in a conjunction with another group of words classified as the same type of constituent. This is a good test because for the for the most part only constituents of the same type can be conjoined. The sentences in Fig 3.3, for example, are acceptable, but the following sentences are not:

* I ate a hamburger and on the store.
* I ate a cold hot dog and well

burned

*I ate the hot dog slowly and a hamburger.

To summarize, if the proposed a constituent doesn't conjoin in some sentence with a constituent of the same class, it is probably incorrect. For example if you say that John's hitting of Mary is an NP in John's hitting of Mary alarmed sue, then it should be usable as an NP in other sentences as well. The initial sentence sue was alarmed by John's hitting of Mary. That the proposed constituent appears to behave just like other NPs.

NP-NP: I ate a hamburger and a hot dog.

VP-VP: I will eat the hamburger and throw away the hot dog.

S-S: I ate a hamburger and John ate a hot dog.

PP-PP: I saw a hot dog in the bag and on the stove.

ADJP-ADJP: I ate a cold and well burned hot dog.

ADVP-ADVP: I ate the hot dog slowly and very carefully.

N-N: I ate a hamburger and hot dog.

V-V: I will cook and burn a hamburger.

AUX-AUX: I can and will eat the hot dog.

ADJ-ADJ: I ate the very cold and burned hot dog (that is, very cold and very burned).

Fig 3.3 Various forms of conjunctions.

Consider the two sentences I looked up John's phone number and I looked up John's chimney. Both as subject-verb-complement sentences with the complement in both cases being a PP. This is, up John's phone number would be a PP.

Conjoining up John's phone number with another PP, as in *I looked up John's phone number and in his cupboards is perfectly acceptable.

In particular you should be able to replace the NP John's phone number by the pronoun it. But the resulting sentence, I looked up it, could not be used with the same meaning as I looked up John's phone number. I looked John's phone number up.

The VP could be the complex verb looked up followed by an NP, or it could consist of three components: the V looked, a particle up and an NP. Either of these is a better solution. What types of tests might you do to decide between them?

As you develop a grammar each constituent is used in more and more different ways. You have a growing number of tests that can be performed to see

if a new analysis is reasonable or not. Sometimes the analysis of a new form might force you to back up and modify the existing grammar. This backward step is unavoidable the current state of linguistic knowledge. The important point to remember, though, is when a new rule is proposed for a grammar, you must carefully consider it's interaction with existing rules.

1. S → NP VP
2. NP → ART N
3. NP → ART ADJ N
4. VP → V
5. VP → V NP

3.3 A TOP-DOWN PARSER

→ A parsing algorithm can be described as a proce-dure that searches through various ways of combining grammatical rules to find a combination that generates a tree that could be the structure of the input sentence.

→ To keep this initial formulation simple, we will not explicitly construct the tree.

→ The algorithm will simply return a yes or no answer as to whether such a tree could be build.

→ The algorithm will say whether a certain sentence is accepted by the gram-mar or not.

→ A simple top-down parsing method relates this to work in Artificial Intelligence (AI) on search procedures.

→ A top-down parser starts with the S symbol and attempts to rewrite it into a sequence of terminal symbols that matches the classes of the words in the input 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 so far called the symbol list.

Eg:- The parser starts in the state (S) and after applying the rule S → NP VP the symbol list will be (NP VP). If it then applies the rule NP → ART N the symbol list will be (ART N VP) and so on.

→ The parser could continue in this fashion until the state consisted entirely of terminal symbols, and then it could check the input sentence to see if it matched. But this would be quite wasteful, for a mistake made early on (say, in choosing the rule that rewrites S) is not discovered until much later. A better

algorithm checks the input as soon as it can. In addition, 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. For now the lexicon will be very simple.

→ A very small lexicon for use in the examples is

Eg: cried: V
dogs: N, V
the: ART

→ With a lexicon specified, a grammar, such as that shown as Grammar 3.4, need not contain any lexical rules.

→ Given these changes, a state of the parse is now defined by a pair: a symbol list similar to before and a number indicating the current position in the sentence. positions fall b/w the words, with 1 being the position before the first word.

For example, here is a sentence with its positions indicated:

1 The 2 dogs 3 cried 4

A typical parse state would be (CNVP)2)

→ indicating that the parser needs to find an N followed by a VP, starting at position two.

→ New states are generated from old states depending on whether the first symbol is a lexical symbol or not. If it is a lexical symbol like N in the preceding example, and if the next word can belong to that lexical category, then you can update the state by removing the first symbol and updating the position counter. In this case, since the word dogs is listed as an N in the lexicon, the next parser state would be.

(CVP)3)

which means it needs to find a VP starting at position 3.

If the first symbol is a nonterminal, like VP, then it is rewritten using a rule from the grammer. For example, using rule 4 in Grammer 3.4, the new state would be

((V)3)

which means it needs to find a VP starting at position 3. on the other hand, using rule 5, the new state would be

((V NP) 3)

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)3), you generate all possible new states. one of these is picked to be the next state and the rest are saved as backup states. If you ever reach a situation where the current state cannot lead to a solution, you simply pick a new current state from the list of backup states. Here is the alg in a little more detail.

A Simple Top-Down Parsing Algorithm:

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

For example, the possibilities list

(((N) 2) ((NAME) 1) ((ADJ N) 1))

indicates that the current state consists of the symbol list (N) at position 2, and that there are two possible backup states: one consisting of the symbol list (NAME) at position 1 and the other consisting of the symbol list (ADJ N) at position 1.

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

1. Select the current state: Take the first state off the possibilities list and call it C. If the possibilities list is empty, then the alg fails (that is, no successful parse is possible).

2. If C consists of an empty symbol list and the word position is at the end of the sentence, then the alg succeeds

Step	Current state	Backup states	Comment
1.	((S) 1)		Initial position
2.	((NP VP) 1)		rewriting S by rule 1
3.	((ART N VP) 1)		rewriting NP by rules 2 & 3
		((ART ADJ N VP) 1)	
4.	((N VP) 2)		matching ART with the
		((ART ADJ N VP) 1)	
5.	((VP) 3)		matching N with dogs
		((ART ADJ N VP) 1)	
6.	((V) 3)		rewriting VP by rule 5-3
		((V NP) 3)	
		((ART ADJ N VP) 1)	
7.			the parse succeeds as V is matched to cried, leaving an empty grammatical symbol list with an empty sentence

3. otherwise, generate the next possible states.

3.1. 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.

3.2 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.

Consider an example using Grammar 3.4, Figure 3.5 shows a trace of the algr on the sentence The dogs cried. First, the initial S symbol is rewritten using rule 1 to produce a new current state of ((NP VP) 1) in step 2. The NP is then rewritten in turn, but since there are two possible rules for NP in the grammar, two possible states are generated: the new current state involves (ART N VP) at position 1, whereas the backup state involves (ART ADJ N VP) at position 1. In step 4 a word in category ART is found at position 1 of the sentence, and the new current state becomes (N VP). The backup state generated in step 3 remains untouched. The parse continues in this fashion to step 5, where two different rules can rewrite VP. The first rule generates the new current state, while the other rule is pushed onto the stack of backup states. The parse completes successfully in step 7, since the current state is empty and all the words in the input sentence have been accounted for.

Consider the same algorithm and grammar operating on the sentence

$_1$The $_2$old $_3$man $_4$cried $_5$

In this case assume that the word old is ambiguous between an ADJ and an N and that the word man is ambiguous between an N and a V (as in the sentence The sailors man the boats). Specifically, the lexicon is

the : ART
old : ADJ, N
man : N, V
cried : V

The parse proceeds as follows (see figure 3.6). The initial S symbol is rewritten by rule 1 to produce the new current state of ((NP VP) 1). The NP is rewritten in turn, giving the new state of ((ART N VP) 1) with a backup state of ((ART ADJ N VP) 1). The parse continues, finding the as an ART to produce the state ((N VP) 2) and then old as an N to obtain the state ((VP) 3). There are now two ways to rewrite the VP, giving us a current state of ((V) 3) and the backup states of ((V NP) 3) and ((ART ADJ N) 1)

from before. The word man can be parsed as a V, giving the state ((V)4). Unfortunately, while the symbol list is empty, the word position is not at the end of the sentence, so no new state can be generated and a backup state must be used. In the next cycle, step 8, ((V NP)3) is attempted. Again man is taken as a V and the new state ((NP)4) generated. None of the rewrites of NP yield a successful parse. Finally, in step 12, the last backup state, ((ART ADJ N VP)1), is tried and leads to a successful parse.

Parsing as a search procedure

You can think of parsing as a special case of a search problem as defined in AI. In particular, the top-down parser in this section was described in terms of the following generalized search procedure. The possibilities list is initially set to the start state of the parse. Then you repeat the following steps until you have success or failure:

1. Select the first state from the possibilities list (and remove it from the list).
2. Generate the new states by trying every possible option from the selected state (there may be none if we are on a bad path).
3. Add the states generated in step 2 to the possibilities list.

For a depth-first strategy, the possibilities list is a stack. In other words, 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 last-in first-out (LIFO) strategy.

In contrast, in a breadth-first strategy the possibilities list is manipulated as a queue. Step 3 adds the new positions onto the end of the list, rather than the beginning, yielding a first-in first-out (FIFO) strategy.

We can compare these search strategies using a tree format, as in Figure 3.7, which shows the entire space of parser states for the last example. Each node in the tree represents a parser state, and the sons of a node are the possible moves from that state. The number beside each node records when the node was selected to be processed by the alg-. On the left side is the order

Step	current state	Backup states	comment
1.	((S)1)		
2.	((NP VP)1)		S rewritten to NP VP
3.	((ART N VP)1)		NP rewritten producing two new states.
		((ART ADJ N VP)1)	
4.	((N VP)2)		
		((ART ADJ N VP)1)	the backup state remains
5.	((VP)3)		
		((ART ADJ N VP)1)	
6.	((V)3)		
		((V NP)3)	
		((ART ADJ N VP)1)	
7.	((I)4)		
		((V NP)3)	
		((ART ADJ N VP)1)	
8.	((V NP)3)		the first backup is chosen
		((ART ADJ N VP)1)	
9.	((NP)4)		
		((ART ADJ N VP)1)	
10.	((ART N)4)		looking for ART at 4 fails.
		((ART ADJ N)4)	
		((ART ADJ N VP)1)	
11.	((ART ADJ N)4)		fails again
		((ART ADJ N VP)1)	
12.	((ART ADJ N VP)1)		now exploring backup state saved in step3
13.	((ADJ N VP)2)		
14.	((N VP)3)		
15.	((VP)4)		
16.	((V)4)		
		((V NP)4)	
17.	((I)5)		success!

fig 3.6. A top-down parse of The$_1$ old$_2$ man$_3$ cried$_4$ $_5$

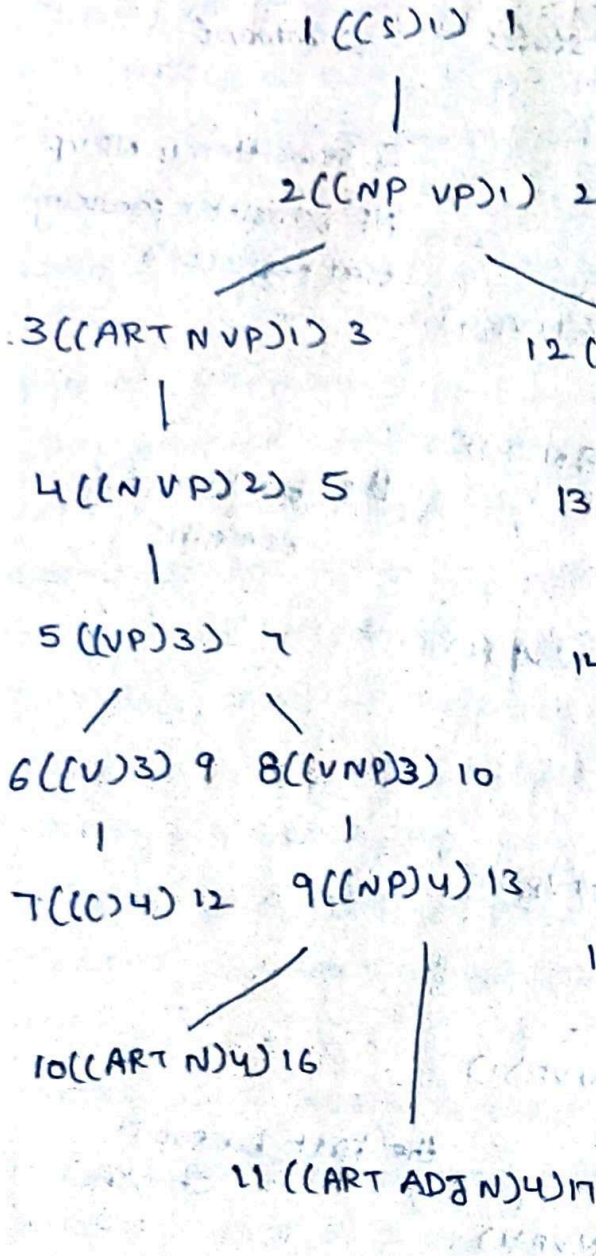


Fig 3.7 Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

produced by the depth-first strategy, and on the right side is the order produced by the breadth-first strategy. Remember, the sentence being parsed is

1 the 2 old 3 man 4 cried 5

the main difference between depth-first and breadth-first searches in this simple example 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. In this example, both depth-first and breadth-

-first searches found the solution but searched the space in a different order. A depth-first search often moves quickly to a solution but in other cases may spend considerable time pursuing futile paths. The breadth-first strategy explores each possible solution to a certain depth before moving on.

In this particular example the depth-first strategy found the solution in one less step than the breadth-first. (The state in the bottom right-hand side of Figure 3.7 was not explored by the depth-first parse).

In certain cases it is possible to put these simple search strategies into an infinite loop.

For example, consider a left-recursive rule that could be a first account of the possessive in English (as in the NP the man's coat):

NP → NP 's N

with a naive depth-first strategy, a state starting with the nonterminal NP would be rewritten to a new state beginning with NP's N. But this state also begins with an NP that could be rewritten in the same way. Unless an explicit check were incorporated into the parser, it would rewrite NPs forever! The breadth-first strategy does better with left-recursive rules, as it tries all other ways to rewrite the original NP before coming to the newly generated state with the new NP. But with an ungrammatical sentence it would not terminate because it would rewrite the NP forever while searching for a solution. For this reason, many systems prohibit left-recursive rules from the grammar.

Many parsers built today use the depth-first strategy because it tends to minimize the number of backup states needed and thus uses less memory and requires less bookkeeping.

Context free Grammar (CFG) :-

NLP - It is the capability of computer software to understand the natural language.

→ There are variety of languages in the world

→ Each language has its own structure called Grammar like SVO (subject verb object), SOV (subject object verb)

→ Each Grammar has set of Rules to determine

→ what is allowable

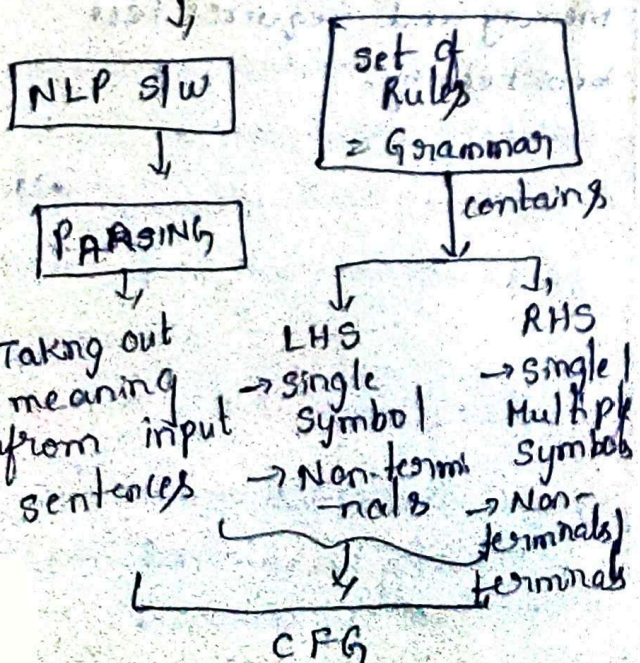
→ what is not allowable.

English S V O SOVXX

I eat Mango

other Language S O V } who will decide ↓ Grammar
O S V

Natural Language as input (Sentences)



Example :-

John hit the ball

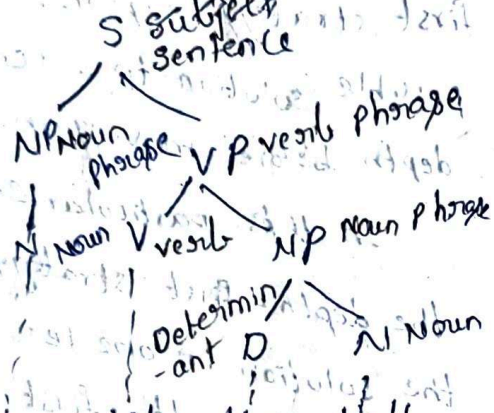
S → NP VP

VP → V NP

NP → N

NP → DN

LHS RHS



Context free Grammar (CFG) :-

Context free Grammar is a formal grammar which is used to generate all possible strings in a given formal language.

Types :-

1. Phrase structure Grammar
2. Backus - Naurs form (BNF)

G = (V, T, P, S)

G = Grammar

V = Non-Terminal set

T = Terminal set

P = Production Rule

S = Start symbol

Rule :-

$\alpha \rightarrow \beta$

$\alpha \rightarrow$ single variable

$\beta \in (V+T)^*$

T = { that, this, a, the, man, book, flight, meal, does, dead, John, ball }

V = { S, NP, Nom, VP, Det, Noun, verb, Aux }

S → Subject Sentence

NP → Noun Phrase

Nom - Nominal (nouns)

VP - verb phrase

Det - Determinant

Noun

verb

Aux - Auxiliary verb

P = {

S → NP VP

S → Aux NP VP

S → VP

NP → Det Noun

VP → verb

VP → verb NP

}

Det → this | that | a | the

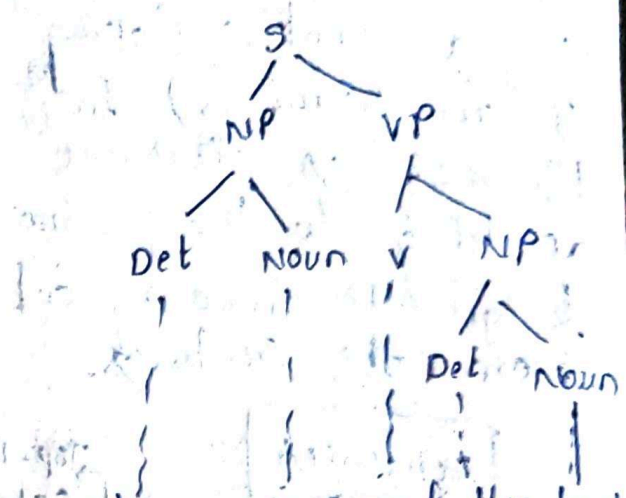
Noun → book | flight | John | ball | meal

verb → book | include | Read

Aux → does | is

Example :-

The man read this book



Properties of CFG :-

→ set of possible derivation

→ string $S \in T^*$

→ Each string in language generated by CFG may have one derivation or more than one derivation (= Ambiguity).

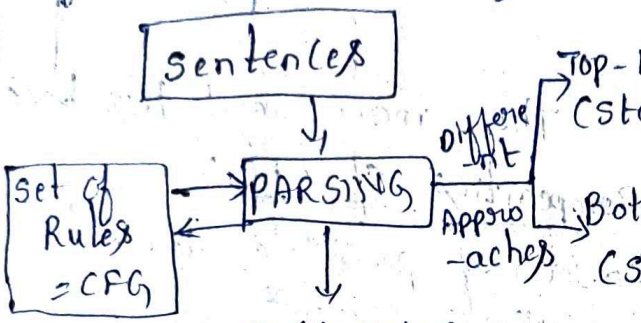
Summary :-

→ CFG is a list of rules that define the set of all well formed sentences in a language.

→ Each rule has a left hand side, which identifies a syntactic category and a right hand side defines its alternative components parts.

Parsing :-

→ It is a method of analy-sing a sentence (string of non-terminals) to determine its structure according to the grammar to get the meaning out from the sentences.



→ Meaning

→ Parse tree for the string

→ It will generate/return all possible parse tree for the string

Police are looking for a thief with a bicycle.

Police with bicycle (or),

Thief with bicycle

Example :-

→ String or sentence given

Book that flight

→ Rules Given

S → NP VP

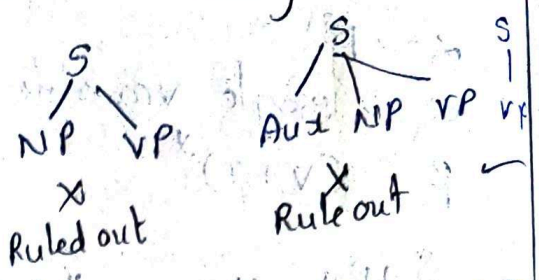
NP → ART N

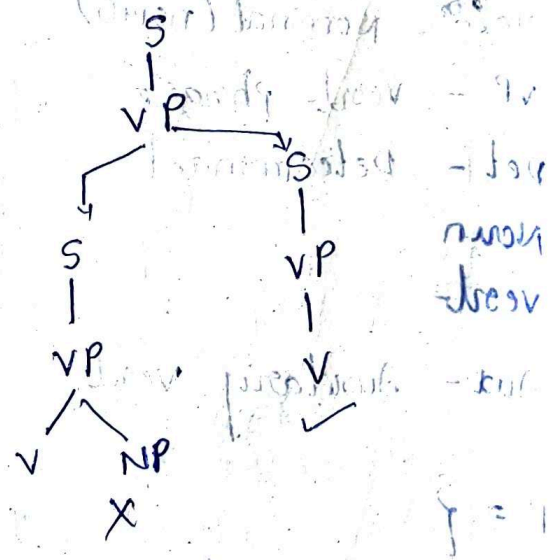
NP → ART ADJ N

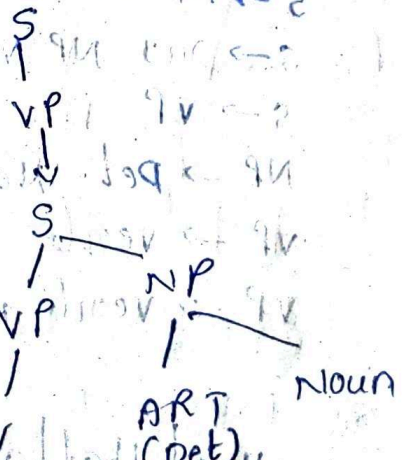
VP → V

VP → V NP

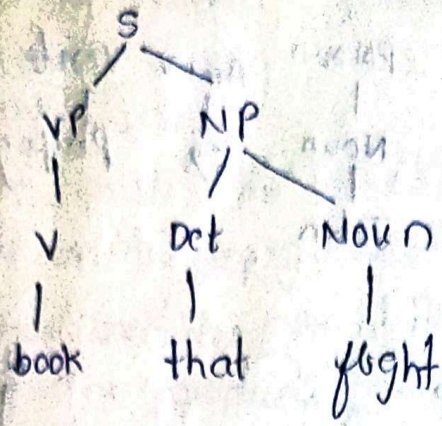
Top Down Parsing (TDP) :-







Finally,

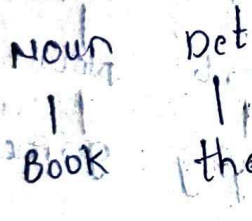


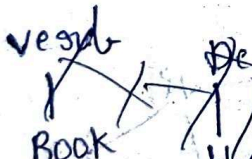
This, is the way top down parsing can be done.

Bottom-up Parsing (BUP) is

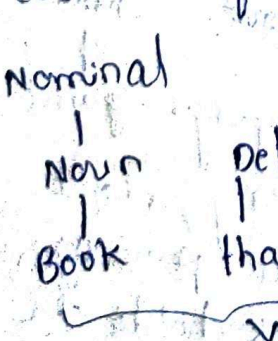
steps is

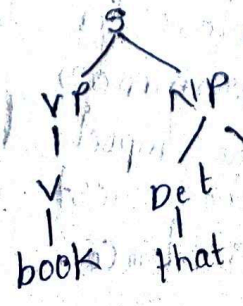
1. Start with the input text
2. ~~Deter~~ Derive the text from Rule.





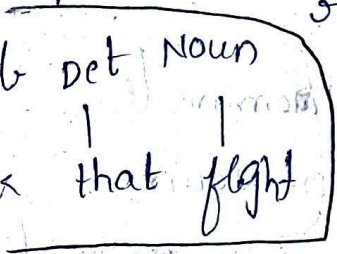
3. Each of these can be derived from non-terminals



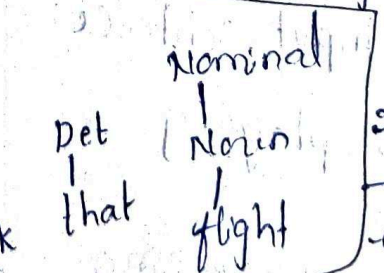


→ Bottom-up parsing starts with the words of input and tries to build trees from the words up, again by applying rules from the grammar one at a time. The parse is successful if the parse succeeds in building a tree rooted in the start symbol S that covers all of the input.

→ Bottom up parsing is a data directed search. It tries to roll back the production process and to reduce the sentence back to the start symbol S.



→ It reduces the string of tokens to the starting symbol by inverting the production, the string is recognized by constructing the rightmost derivation in reverse.



→ The objective of reaching the starting symbol S is achieved by series of reductions, when the Right hand side of some rule matches the

substring of the input string, the substring is replaced with the left hand side of the matched production, the process continues until starting symbol is reached, henceforth Bottom-up parsing can be defined as reduction process.

→. Bottom-up parsing can be viewed as generation of parse tree in post order.

Consider the grammar rules S → NP VP
S → NP Aux V VP
S → VP
NP → Det Nominal
NP → Proper-Noun
Nominal → Noun Nominal
Nominal → Noun
VP → verb
VP → V NP

Grammar Rules

and the input sentence "John is playing game"

The Bottom-up parsing works as follows

1. Noun Aux V verb Noun
   | | | |
   John is playing game

2. PNoun Aux V verb Nom
   | | | |
   Noun is playing Noun
   | | |
   John game

3. NP NP
   | |
   PNoun Nom
   | |
   Noun Aux V verb Noun
   | | | |
   John is playing game

4. VP
   NP / \ NP
   | | |
   PNoun Noun
   | |
   Noun Aux V verb Noun
   | | | |
   John is playing game

5. S
   NP / | \ VP
   | | | / \ NP
   PNoun Noun
   | |
   Noun Aux V verb Noun
   | | | |
   John is playing game

→ Bottom-up parsing adopts the shift-reduce paradigm, where a stack holds the grammar symbol and the input buffer stores the rest of the input sentence.

→ The shift reduce parsing is achieved using four primary actions

1. shift, pushes the next input symbol on the top of the stack.
2. Reduce, reduces Right Hand side of the production to its Left Hand Side
3. Accept, successful completion of parsing and
4. Error, discover the syntax error and, call the error recovery routine.

→ To have an operational shift-reduce parser and to determine the reducing production to be used, it implements LR parsing which uses the LR(K) grammar.

Where
(1) L signifies Left-to-right scanning of input
(2) R indicates rightmost derivation done in reverse and
(3) K, is the number of lookahead symbols used to make parsing decision. The efficiency of Bottom-up parsing lies in the fact that it never explores trees inconsistent with the input.

→ Bottom-up parsing never suggests trees that are not locally grounded in the actual input.

→ However the trees have no hope of leading to an S or filling in with any of its neighbors, are generated in abandon, which adds to the inefficiency of the bottom-up strategy.

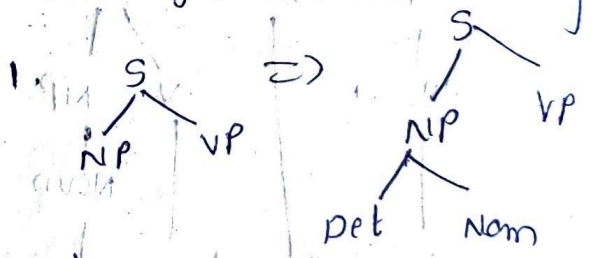
Top-Down Parsing

Grammar Rules

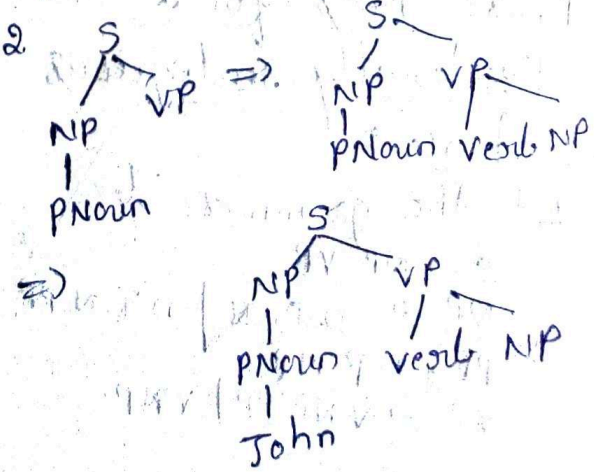
S → NP V
S → NP Aux VP
S → VP
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun Nominal
Nominal → Noun
VP → Verb
VP → V NP

Sentence "John is playing game"

Applying Top-Down Parsing

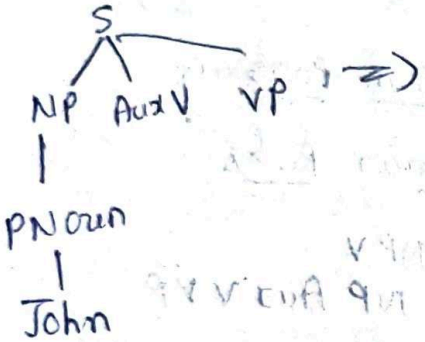
1. 

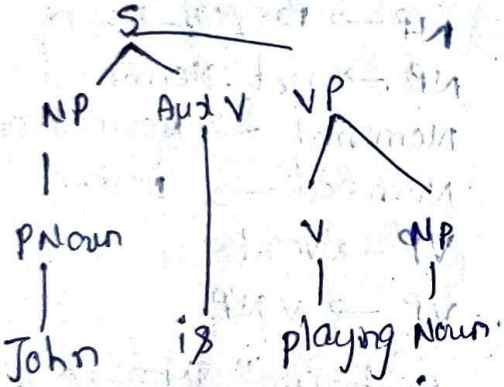
Parts-of-Speech does not match the input string backtrack to the node NP

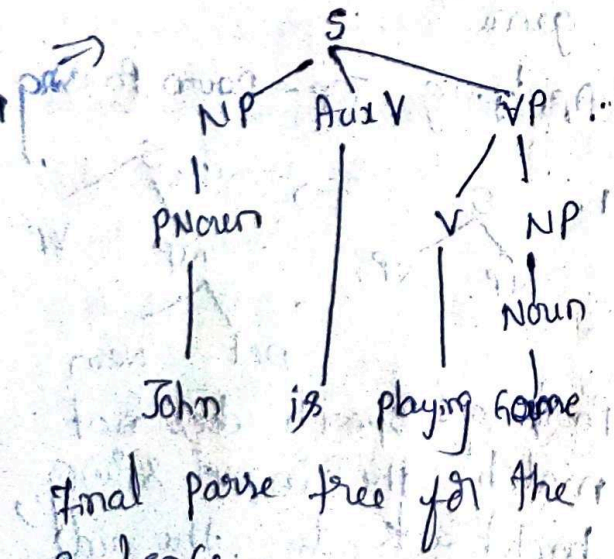
2 

Parts-of-speech verb does not match the input string backtrack to the node S, since PNoun is matched.

3.







Final parse tree for the sentence.

Parse Trees for a Structurally Ambiguous Sentence :-

Let the grammar be

$S \rightarrow NP\ VP$

$NP \rightarrow DT\ N \mid DT\ N\ PP$

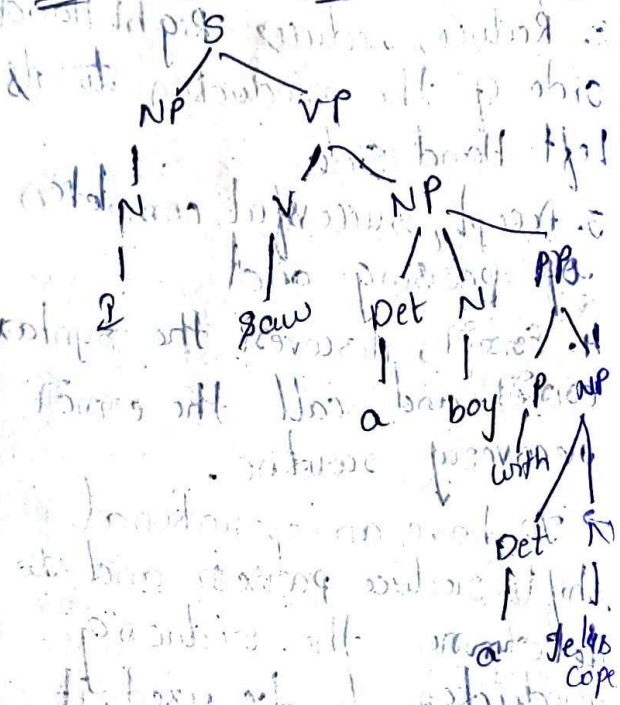
$PP \rightarrow P\ NP$

$VP \rightarrow V\ NP\ PP \mid V\ NP$

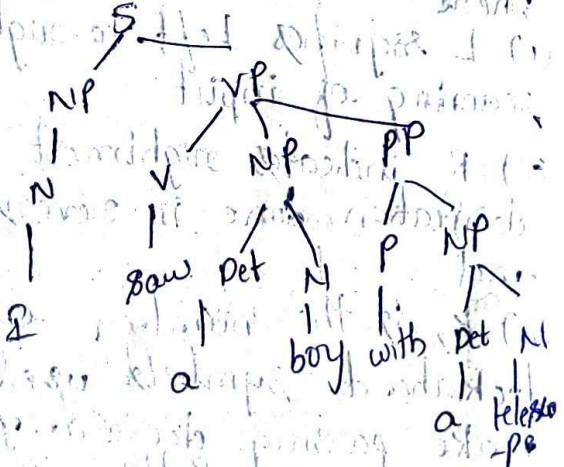
for the sentence

"I saw a boy with a telescope"

Parse Tree - 1



Parse Tree - 2



→ It is a sentence for which the possibility of getting more than one parse tree is called structurally ambiguous sentence.

Sentence "I saw a boy with a telescope"

Grammar:

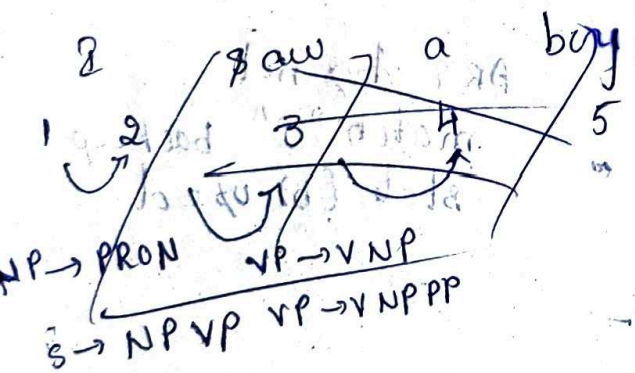
S → NP VP
NP → ART N | ART N PP | PRON
VP → V NP PP | V NP
ART → a | an | the
N → boy | telescope
PRON → I
V → saw

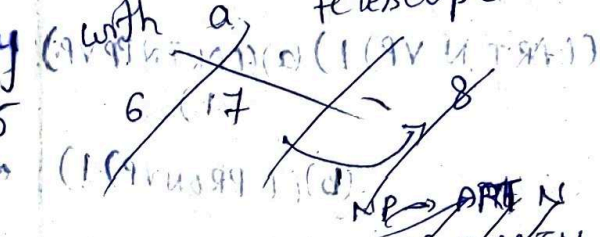
Top Down Parser:

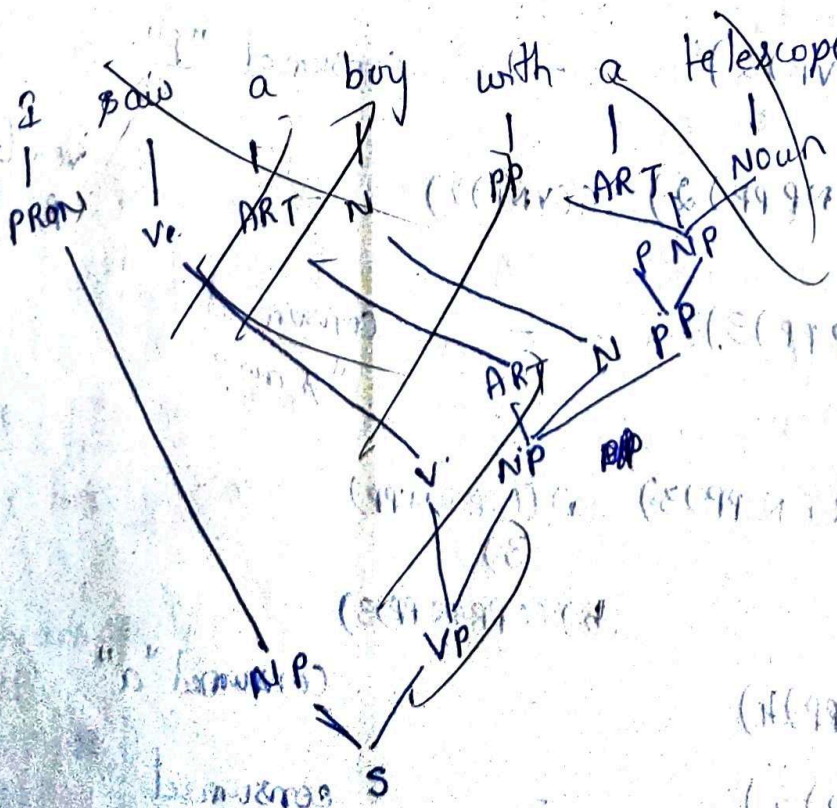
	State	Backup State	Action	Comments
1.	((S) 1)	-	-	Use S → NP VP
2.	((NP VP) 1)	-	-	Use NP → ART N | ART N PP | PRON
3.	((ART N VP) 1)	(a)((ART N PP VP) 1) (b)((PRON VP) 1)	-	ART does not match "I" backup state (b) used
3B	((PRON VP) 1)	-	-	
4.	((VP) 2)	-	Consumed "I"	
5.	((V NP PP) 2)	((V NP) 2)	-	verb Attachment Rule used
6.	((NP PP) 3)	-	Consumed "saw"	
7.	((ART N PP) 3)	(a)((ART N PP) 3) (b)((PRON PP) 3)	Consumed "a"	
8.	((N PP) 4)	-	Consumed "boy"	
9.	((PP) 5)	-		

	State	Backup State	Action	Comments
10.	((PNP) 5)	—	—	
11.	((NP) 6)	—	Consumed "with"	
12.	((ARTN) 6) (a)((ARTNPP) 6) (b)((PRON 6)	—	—	
13.			consumed "a"	
			consumed "saw"	success
				telescope

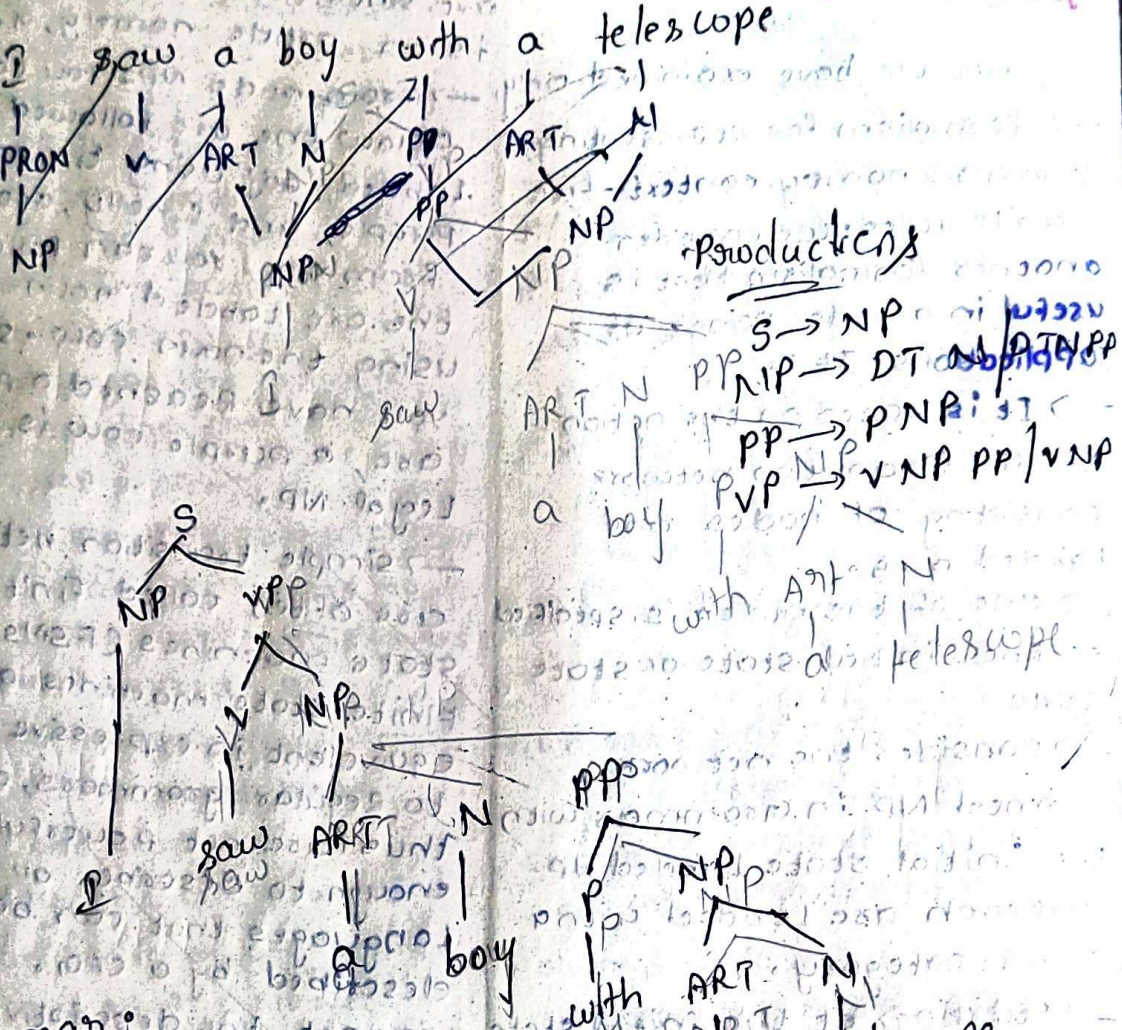
Bottom up Parse







Bottom up Parse :-



Productions

S → NP VP
NP → DT N | DT N PP
PP → P NP
VP → V NP PP | V NP

Grammar :-

S → NP VP
NP → ART N | ART N PP | PRON N
VP → V NP PP | V NP
Prepositional phrase PP → P NP
ART → a | an | the
N → boy | telescope
PRON → I
V → saw
Preposition P → with

porate some additional arc that can always be followed is called a jump arc.

Feature Systems and Augmented Grammars

→ In natural languages there are often agreement Restrictions between words and phrases.

→ For example, the NP a men is not correct English because the article a indicates a single object while the noun men indicates a plural object, the noun phrase does not satisfy the number agreement restriction

of English.

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

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

→ For example, we might define a feature NUMBER that may take a value of either s or p, and we then might write an abgmented CFG Rule such as

NP → ART N only when NUMBER agrees with NUMBER

→ This Rule says that a legal noun phrase consists of an article followed by a noun, but only when the number feature of the second.

→ This one Rule is equivalent to two 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

→ While the two approaches seem similar in ease-of-use in this one example, consider that all Rules in the grammer that use an NP on the Right-hand side would now need to be duplicated to include a rule for NP-SING and a rule for NP-PLURAL, effectively doubling the size of the grammar.

→ And handling additional features, such as person agreement, would double the size of the grammar again and again using features, the size of the augmented grammer Remains the same as the original one yet accounts for agreement constraints.

→ To accomplish this, a constituent is defined as a feature structure a mapping from features to values that defines the Relevant properties of the constituent.

→ In the examples in this book, feature names in formulas will be written in bold face. For example, a feature structure for a constituent ART1 Represents a particular use of the word a might be written as follow.

ART1: (CAT ART
ROOT a
NUMBER s)

→ This says it is a constituent in the category ART that has as its root the word a and is singular. Usually an abbreviation is used that gives the CAT value more prominence and provide an intuitive tie back to simple context-free grammer

In this abbreviated form, constituent ART1 would be written as

ART1: (ART ROOT a NUMBERS)

→ Feature structure 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, as needed. With this, the representation of the NP constituent for the phrase a fish could be

NP1: (NP NUMBER s
1 (ART ROOT a
NUMBERS)
2 (N ROOT fish
NUMBERS))

→ Note that this can also be viewed as a representation of a parse tree shown in where the subconstituent features 1 and 2 correspond to the subconstituent links in the tree.

→ The Rules in an augmented grammer are stated in terms of features structures rather than simple categories. Variables are allowed as feature values so that a Rule can apply to a wide range of situations. For example a rule for simple noun phrases would be as follows.

(NP NUMBER ?n) → (ART NUMBER ?n)
(N NUMBER ?n)

→ This says that an NP constituent can consist of two subconstituent, the first being an ART and the second being an N, in which the NUMBER feature in all three constituents is identical. According to this Rule, constituent NP1 given previously is a legal constituent. on the other hand, the constituent.

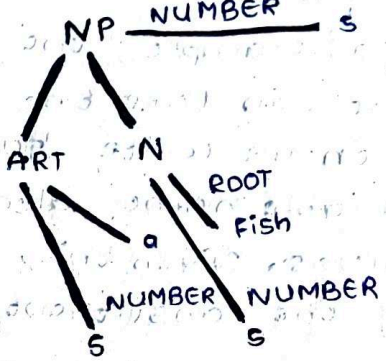


Fig: Viewing a feature structure as an extended parse tree

(NP 1 (ART NUMBERS)
2 (N NUMBERS))

is not allowed by this rule because the NUMBER features on the NP, and the constituent.

(NP NUMBER s
1 (ART NUMBERS)
2 (N NUMBER p))

is not allowed because the NUMBER feature of the N constituent is not identical to the other two NUMBER features.

→ Variables are useful in specifying ambiguity in a constituent. For instance the word fish is ambiguous between a singular and a plural reading.

→ Thus the word might have two entries in the lexicon that differ only by the value of the NUMBER

feature. Alternatively, we could define a single entry that uses a variable as the value of the NUMBER feature that is,

(N ROOT fish NUMBER ?n)

→ This works because any value of the NUMBER feature is allowed for the word fish. In many cases, however, not just any would work, but a range of values is possible. To handle these cases, we introduce constrained variables, which are variables that can only take a value out of a specified list.

→ For example, the variable ?n{s p} would be a variable that can take the value s or the value p. Typically, when we write such variables, we will drop the variable name altogether and just list the possible values. Given this, the word fish might be represented by the constituent.

(N ROOT fish NUMBER ?n{s p})

or more simply as

(N ROOT fish NUMBER{s p})

Formalizing Feature Structures

There is an active area of Research in the formal properties of feature structures. This work views a feature system as a formal Logic. A feature structure is defined as a pratial function from features to feature values. For example, the feature structure.

ART1: (CAT ART
ROOT a
NUMBER s)

is treated as an abbreviation of the following statement in FOPC.

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

Feature structures with disjunctive values map to disjunctions. The structure

THE1: (CAT ART
ROOT the
NUMBER{s p})

would be Represented as

THE1(CAT) = ART ∧ THE1(ROOT) = the
∧ [THE1(NUMBER) = s ∨ THE1(NUMBER) = p]

(Given this, agreement features between values can be defined as equivalent equations)

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.

Because words tend to follow regular morphological patterns, however, many forms of words need not be explicitly included in the lexicon. Most English verbs, for example, use the same set of suffixes to indicate different forms: -s is added for third 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. For the verb want this would require six entries, for want (both in base and present form), wants, wanting, and wanted (both in past and past participle forms).

In contrast, by using the methods described in section 3.4 to strip suffixes there needs to be only one entry for want. The idea is to store the base form of the verb in the lexicon and use context-free rules to combine verbs with suffixes to derive the other entries. Consider the following rule for present tense verbs:

(V ROOT ?r SUBCAT ?s VFORM pres AGR 3s) → (V ROOT ?r SUBCAT ?s VFORM base) (ts)

where ts is a new lexical category that contains only the suffix morpheme -s. This rule, coupled with the lexicon entry

want: (V ROOT want SUBCAT {_np_vp:inf_np_vp:inf} VFORM base)

would produce the following constituent given the input string want -s.

want: (V ROOT want SUBCAT {*_np_vp:inf_np_vp:inf} VFORM pres AGR 3s)

Another rule would generate the constituents for the present tense form not in third person singular, which for most verbs is identical to the root form:

(V ROOT ?r SUBCAT ?s VFORM pres AGR {1s 2s 1p 2p 3p}) → (V ROOT ?r SUBCAT ?s VFORM base IRREG-PRES -)

Transition Network

But this rule needs to be modified in order to avoid generating erroneous interpretations. Currently, it can transform any base form verb into a present tense form, which is clearly wrong for some irregular verbs. For instance, the base form be cannot be used as a present form (for example, *We be at the store). To cover these cases, a feature is introduced to identify irregular forms. Specifically, verbs with the binary feature +IRREG-PRES have irregular present tense forms. Now the rule above can be stated correctly:

(V ROOT ?r SUBCAT ?s VFORM pres AGR {1s 2s 1p 2p 3p}) → (V ROOT ?r SUBCAT ?s VFORM base IRREG-PRES -)

Because of the default mechanism, the IRREG-PRES feature need only be specified on the irregular verbs. The regular verbs default to -, as desired. Similar binary features would be needed to flag irregular past forms (IRREG-PAST), such as saw), and to distinguish -en past participles from -ed past participles (EN-PASTPRT). These features restrict the application of the standard lexical rules, and the irregular forms are added explicitly to the lexicon. Grammar 4.5 gives a set of rules for deriving different verb and noun forms using these features.

Given a large set of features, the task of writing lexical entries appears very difficult. Most frameworks allow some mechanisms that help alleviate these problems. The first technique — allowing default values for features — has already been mentioned with this capability, if an entry takes a default value for a given feature, then it need not be explicitly stated. Another commonly used technique is

present tense

1. (V ROOT ?r SUBCAT ?s VFORM pres AGR 3s) → (V ROOT ?r SUBCAT ?s VFORM base IRREG-PRES -) +s

2. (V ROOT ?r SUBCAT ?s VFORM pres AGR {1s 2s 1p 2p 3p}) → (V ROOT ?r SUBCAT ?s VFORM base IRREG-PRES +)

past tense

3. (V ROOT ?r SUBCAT ?s VFORM past AGR {1s 2s 3s 1p 2p 3p}) → (V ROOT ?r SUBCAT ?s VFORM base IRREG-PAST +) +ED

past participle

4. (V ROOT ?r SUBCAT ?s VFORM pastprt) → (V ROOT ?r SUBCAT ?s VFORM base EN-PASTPRT +) +ED

5. (V ROOT ?r SUBCAT ?s VFORM pastprt) → (V ROOT ?r SUBCAT ?s VFORM base EN-PASTPRT +) +EN

present participle

6. (V ROOT ?r SUBCAT ?s VFORM ing) → (V ROOT ?r SUBCAT ?s VFORM base) +ING

plural nouns

7. (N ROOT ?r AGR 3p) → (N ROOT ?r AGR 3s IRREG-PL -) +S

Grammar 4.5 Some lexical rules for common suffixes on verbs and nouns

to allow the lexicon writing to c
then indicate a cluster with a si
listing them all. Later, additional te
allow the inheritance of features in a

Figure 4.6 contains a small l
words to be used in the examples t
for the word saw as a noun, as
past tense form of the verb see -

The saw was broken
Jack wanted me to sa
I saw Jack eat the p

with an algorithm for stripping
spelling, as described in section 3.7,
using any of the basic parsing alg
lexicon in figure 4.6 and Grammar
words can be derived: been, being
(two interpretations), sawed, sawing,
and wanted, for example, the w
into the sequence cry +s, and the
tense entry from the base form in

Figure 4.6 A lexicon

saw: (CAT N ROOT SAW1 AGR 3s)
saw: (CAT V ROOT SAW2 VFORM base SUBCAT -NP)
saw: (CAT V ROOT SEE1 VFORM past SUBCAT -NP)
see: (CAT V ROOT SEE1 VFORM base SUBCAT -NP IRREG-PAST + EN-PASTPRT +)
seed: (CAT N
be: (CAT V ROOT BE1 VFORM base PRES + IRREG-PAST + IRREG-PRES + SUBCAT -adjp -NP)
cry: (CAT V ROOT CRY1 VFORM base SUBCAT -none)
dog: (CAT N ROOT DOG1
a: (CAT ART ROOT A1 AGR 3s)

to allow the lexicon writing to define clusters of features, and then indicate a cluster with a single symbol rather than listening listing them all. Later, additional techniques will be discussed that allow the inheritance of features in a feature hierarchy.

Figure 4.6 contains a small lexicon. It contains many of the words to be used in the examples that follow. It contains three entries for the word saw, as a noun, as a regular verb, and as the irregular past tense form of the verb see, as illustrated in the sentences

The saw was broken
Jack wanted me to saw the board in half
I saw Jack eat the pizza.

With an algorithm for stripping the suffixes and regularizing the spelling, as described in section 3.7, the derived entries can be generated using any of the basic parsing algorithms on Grammar 4.5, with the lexicon in figure 4.6 and Grammar 4.5, correct constituents for the following words can be derived: been, being, cries, cried, crying, dogs, saws (two interpretations), sawed, sawing, seen, seeing, seeds, wants, wanting, and wanted. For example, the word cries would be transformed into the sequence cry +s, and then rule I would produce the present tense entry from the base form in the lexicon.

Figure 4.6 A lexicon

```
saw: (CAT N
      ROOT SAW1
      AGR 3s)

saw: (CAT V
      ROOT SAW2
      VFORM base
      SUBCAT -np)

saw: (CAT V
      ROOT SEE1
      VFORM past
      SUBCAT -np)

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

seed: (CAT N
       ROOT SEED1
       AGR 3s)

the: (CAT ART
      ROOT THE1
      AGR {3s 3p})

to: (CAT TO)

want: (CAT V
       ROOT WANT1
       VFORM base
       SUBCAT -np -vp:inf -np-vp:inf)

was: (CAT V
      ROOT BE1
      VFORM past
      AGR {1s 3s}
      SUBCAT -adjp -np)

be: (CAT V
     ROOT BE
     VFORM base
     AGR {1s 2s 1p 2p 3p}
     SUBCAT {-adjp -np})

or: (CAT ART
     ROOT A1
     AGR 3s)

be: (CAT V
     ROOT BE1
     VFORM base
     IRREG-PRES +
     IRREG-PAST +
     SUBCAT {-adjp -np})

cry: (CAT V
      ROOT CRY1
      VFORM base
      SUBCAT -none)

dog: (CAT N
      ROOT DOG1
      AGR 3s)

fish: (CAT N
       ROOT FISH1
       AGR {3s 3p}
       IRREG-PLT)

happy: (CAT ADJ
        SUBCAT -vp:inf)

he: (CAT PRO
     ROOT HE1
     AGR 3s)

is: (CAT V
     ROOT BE1
     VFORM pres
     SUBCAT -adjp -np)

Jack: (CAT NAME
       AGR 3s)

man: (CAT N
      ROOT MAN1
      AGR 3s)

men: (CAT N
      ROOT MAN1
      AGR 3p)
```

Often a word will have multiple interpretations that use different entries and different lexical rules. The word saws, for instance, transformed into the sequence saw +S, can be a plural noun (via rule 7 and the first entry for saw), or the third person present form of the verb saw (via rule 1 and the second entry for saw). Note that rule 1 cannot apply to the third entry, as its VFORM is not base.

The success of this approach depends on being able to prohibit erroneous derivations, such as analyzing seed as the past tense of the verb see. This analysis will never be considered if the FST that strips suffixes is correctly designed specifically, the word see will not allow a transition to the state that allows the -ed suffix. But even if this were produced for some reason, the IRREG-PAST value in the entry for see would prohibit rule 3 from applying.

Parsing with features

The parsing algorithms developed for context-free grammars can be extended to handle augmented context-free grammars. This involves generalizing the algorithm for matching rules to constituents. For instance, the chart-parsing algorithms developed in the Chapter 3 all used an operation for extending active arcs with a new constituent. A constituent X could extend an arc of the form:

$C \rightarrow C_1 \ldots C_i \circ X \ldots C_n$

to produce a new arc of the form

$C \rightarrow C_1 \ldots C_i X \circ \ldots C_n$.

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. The key to defining this matching operation precisely is to remember the definition of grammar rules with features. Rules such as

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

says that an NP can be constructed out of an ART and an N if all three agree on the AGR feature. It does not place any restrictions on any other features that the NP, ART, or N may have. Thus, when matching constituents against this rule, the only thing that matters is the AGR feature. All other features in the constituent can be ignored. For instance, consider extending extending arc 1 with the constituent

2. (ART ROOT A AGR 3S)

To make arc 1 applicable, the variable ?a must be instantiated to 3S, producing

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

This arc can now be extended because every feature in the rule is in constituent 2.

4. (NP AGR 3s) → (ART AGR 3s) o (N AGR 3s)

Now, consider extending this arc with the constituent for the word dog:

5. (N ROOT DOG 1 AGR 3s)

This can be done because the AGR feature agree. This completes the arc.

6. (NP AGR 3s) → (ART AGR 3s) (N AGR 3s) ;

This means the parser has found a constituent of the form (NP AGR 3s).

This algorithm can be specified more precisely as follows: Given an arc A, where the constituent following the dot is called NEXT, and a new constituent X, which is being used to extend the arc,

a. Find an instantiation of the variables such that all the features specified in NEXT are found in X.

b. Create a new arc A', which is a copy of A except for the instantaneous of the variables determine in step a)

c. Update A' as usual in a chart parser.

For instance let A be arc 1, and X be the ART constituent 2. Then NEXT will be (ART AGR ?a). In step a, NEXT is matched against X, and you find that ?a must be instantiated to 3s. In step b, a new copy of A is made, which is shown as arc 3. In step c, the arc is updated to produce the new arc shown as arc 4:

when considered variables such as ?a {3s 3p}, are involved, the matching proceeds in the same manner, but the variables binding must be one of the listed values. If a variable, ?a is used in a constituent, then one of its possible values must match the requirement in the rule. If both the rule and the constituent contain variables, the result is a variable ranging over the intersection of their allowed values. For instance, consider extending arc 1 with the constituent (ART ROOT the AGR ?v {3s 3p}), that is, the word the. To apply, the variable ?a would have to be instantiated to ?v{3s, 3p}, producing the rule.

(NP AGR ?v {3s 3p}) → (ART AGR ?v {3s 3p}) o (N AGR ?v {3s 3p})

This arc could be extended by (N ROOT dog AGR 3s), because ?v {3s 3p} could be instantiated by the value 3s. The resulting arc would be identical to arc 6. The entry in the chart for the is not changed by this operation. It still has the value ?v {3s 3p}. The AGR feature is restricted to 3s only in the arc.

Another extension is useful for regarding the structure of the parse. Subconstituent features (1, 2, and so on, depending on which subconstituent is being added) are automatically inserted by the parser each time an arc is extended. The values of these features name subconstituents already in the chart.

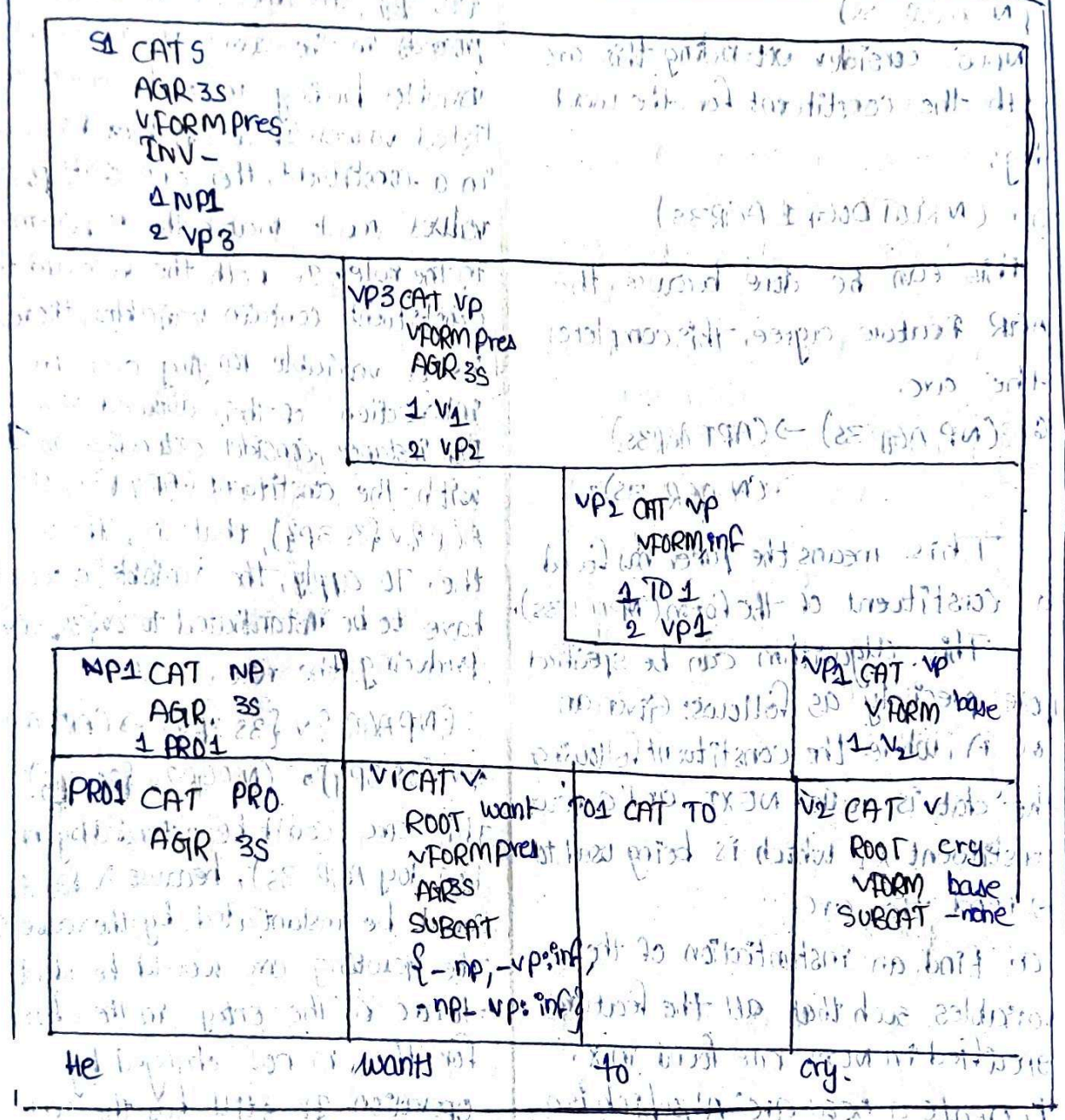


Figure : The chart for he wants to cry.

with this treatment, and assuming that the chart already contains two constituents, ART 1 and N1, for the words the and dog, the constituent added to the chart for the phrase the dog would be

(NP AGR 3S
1 ART1
2 N1)

where ART1 = (ART ROOT the AGR {3S 3P}) and N1 = (N ROOT dog AGR {3S 3P}). Note that the AGR feature of ART 1 was not changed. This it could be used with other interpretations that require the value 3P if they are possible. Any of the chart-parsing algorithms described in chapter 3 can now be used with an augmented grammar by using these extensions to extend arcs and build constituents. Consider an example! Figure contains the final chart produced from parsing the sentence, He wants to cry, using Grammar 4.8. The rest of this section considers how

some of the nonterminal symbols were constructed for the chart.

Constituent NP1 was constructed by rule 3, repeated here for convinience

3. (NP AGR ?a) → (PRO AGR ?a)

To match the constituent PRO1, the variable ?a must be instantiated to 3S. Thus the new constituent built is

NP1: (CAT NP
AGR 3S
1 PRO1)

Next consider constructing constituent VP1 using rule 4, namely

4. (VP AGR ?a VFORM ?v) → (V SUBCAT -none AGR ?a VFORM ?v)

For the right-hand side to match constituent V_2, the variable ?v must be instantiated to base. The AGR feature of V_2 is not defined, so it defaults to —. The new constituent is

VP1: (CAT VP
AGR —
VFORM base
1 V_2)

Generally, default values are not shown in the chart. In a similar way, constituent VP_2 is built from PO_1 and VP1 using rule 9, VP_3 is built from V_1 and VP_2 using rule 6, and S_1 is built from NP1 and VP_3 using rule 1.

Augmented Transition Networks

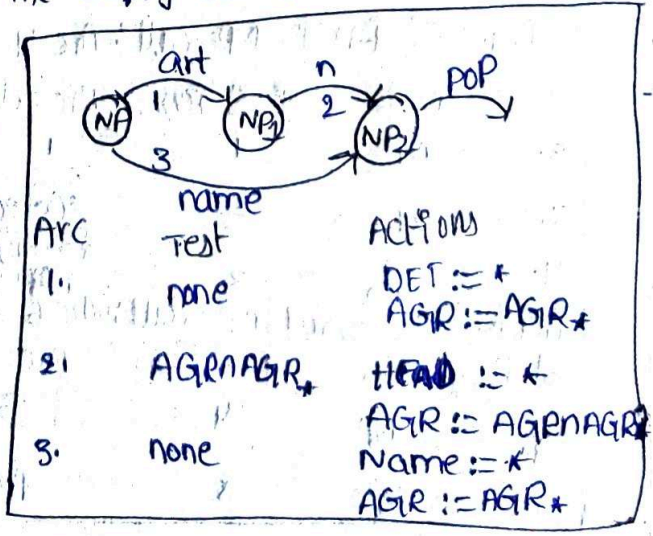
Features can also be added to a recursive transition network to produce an augmented transition network (ATN). Features in an ATN are traditionally called registers. constituent structures are created by allowing each network to have a set of registers. Each time a new network is pushed, a new set of registers are is created. As the network is traversed, these registers are set to values by actions associated with each arc. when the network is popped, the registers are assembled to form a constituent structure, with the CAT slot being the network name.

Grammar 4.1 is a simple NP network. The actions are listed in the table below the network. ATN use a special mechanism to extract the result of following an arc. when a lexical arc, such as arc 1, is followed, the constituent built from the word in the input is put into a special variable named *. The action DET := * then assigns this constituent to the DET register. The second action on this arc,

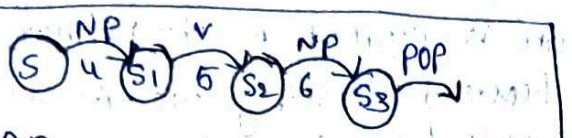
AGR := AGR*

assigns the AGR register of the network to the value of the AGR register of the new word (the constituent in *)

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.



Grammar 1: A simple NP network.



ARC	Test	Actions
4:	none	SUBJ := *
5:	none	AGR := AGR_{SUBJ} MAIN-V := * AGR := AGR_{SUBJ} AGR*
6:	none	OBJ := *

Grammar A simple S network.

If a test fails, this arc is not traversed. The test on arc 2 indicates that the arc can be followed only if the AGR feature of the network has a non-null intersection with the AGR register of the new word (the noun constituent in *).

Features on push arcs are treated similarly. The constituent built by traversing the NP network is returned as the value * this in Grammar 2. The action on the arc from S to S1,

SUBJ := *

would assign the constituent returned by the NP network to the register SUBJ. The test on arc 2 will succeed only if the AGR register of the constituent in the SUBJ register has a non-null intersection with the AGR register of the new constituent (the verb). This test enforced subject-verb agreement.

Trace of S Network

step	Node	position	Arc followed	Registers set.
1.	S	1	arc 4 succeeds (for recursive call see trace below)	SUBJ ← (NP DET the HEAD dog AGR 3s)
5	S1	3	arc 5 (check if 3p ∩ 3p)	MAIN-V ← saw AGR ← 3p
6.	S2	4	arc 6 (for recursive call trace, see below)	OBJ ← (NP NAME Jack AGR 3s)
9.	S3	5	POP arc succeeds	returns (S SUBJ (NP DET the HEAD dog AGR 3s) MAIN-V saw AGR 3p OBJ (NP NAME Jack AGR 3s))

Trace of First NP call: Arc 4

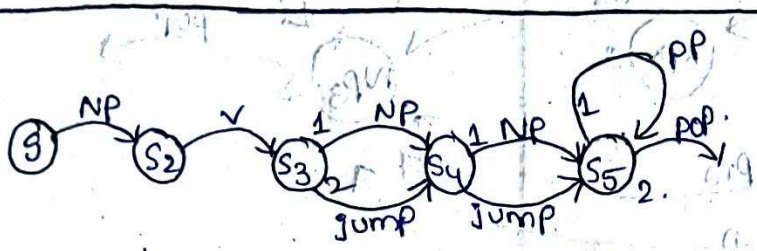
step	Node	position	Arc followed	Registers set.
2.	NP	1	1	DET ← the
3.	NP1	2	2 (check if {3s 3p} ∩ 3p)	AGR ← {3s 3p}
4.	NP2	3	POP	HEAD ← dog returns (NP DET the HEAD dog AGR 3s)

Trace of second NP call: Arc 6

step	Node	position	Arc followed	Registers set.
7.	NP	4	3	NAME ← John AGR ← 3s
8.	NP2	5	POP	returns (NP NAME John AGR 3s)

Figures Trace tests and actions used with 1 The 2 dog 3 saw 4 Jack 5

With the lexicon in section 4.3, the ATN accepts the following sentences:
The dog cried.
The dogs saw Jack.
Jack saw the dogs.



Arc	Test	Actions
S/1		SUBJ := * MOOD := DECL
S2/1	AGR SUBJ AGR	MAIN-V := *
S3/1	SUBCAT MAIN-V ∩ {-NP-NP-NP}	OBJ := * SUBCAT := SUBCAT MAIN-V ∩ {-NP-NP-NP}
S3/2	SUBCAT MAIN-V ∩ -NONE	SUBCAT := -NONE
S4/1	SUBCAT MAIN-V ∩ -NP-NP	IOBJ := OBJ OBJ := *
S5/1	-----	MODS := APPEND (MODS, *)

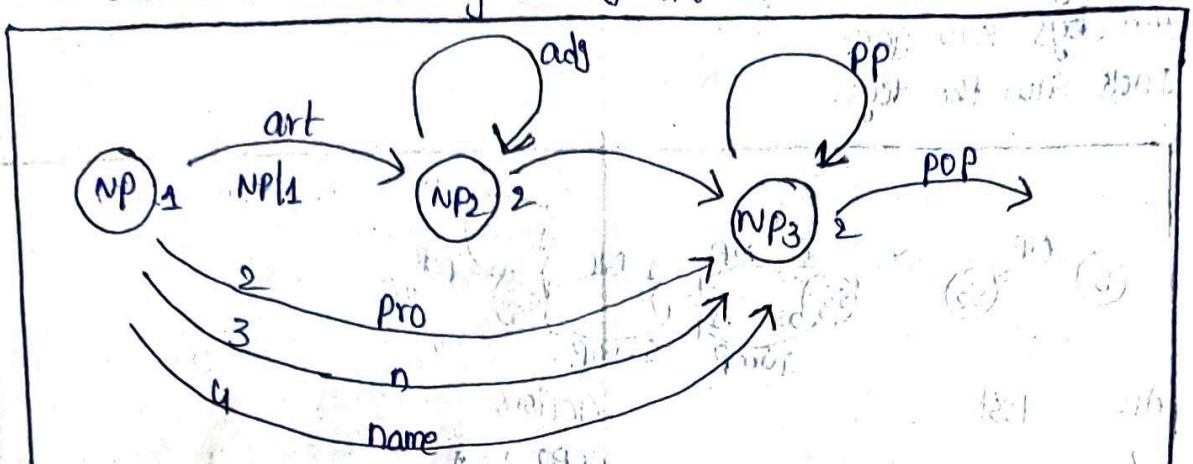
Grammar: An S network for assertions

Consider an example. A trace of a parse of the sentence The dog saw Jack is shown in Figure 3. It indicates the current node in the network, the current word position, the arc that is followed from the node, and the register manipulations that are performed for the succesful parse. It starts in the S network but moves immediately to the NP network from the call on arc 4. The NP network checks for number agreement as it accepts the word sequence the dog. It contains constructs a noun phrase with the AGR feature plural. when the POP arc is followed, it completes arc 4 in the S network. The NP is assigned to the SUBJ register and then checked for agreement with the verb when arc 3 is followed. The NP Jack is accepted in another call to the NP network.

An ATN Grammar for Simple Declarative Sentences.

Here is a more comprehensive example of the use of an ATN, describe some declarative sentences. The allowed sentence structure is an initial NP followed by a main verb, which may then be followed by a maximum of two NPs and many PPs, depending on the verb. using the feature system extensively, you can create a grammar that accepts any of the preceding complement forms, leaving the actual verb-complement agreement to the feature restrictions. Grammar 4 shows the S network. Arcs are numbered using the conventions discussed in class. For instance, the arc S3/1 is the arc labeled 1 leaving node S3. The NP network in grammars

allows simple names, bare plural nouns, pronouns, and a simple sequence of a determiner followed by an adjective and a head noun.



Arc	Test	Actions
NP/1	------	DET := 0
NP/2	------	PRO := 0
NP/3	AGR* ∩ 3p	HEAD := *
NP/4	------	AGR := AGR*
NP/1	------	NAME := *
NP2/2	AGR ∩ AGR*	ADJS := Append(ADJS, *)
NP3/1	------	HEAD := *
		AGR := AGR ∩ AGR*
		MODS := Append (MODS, *)

Grammar 5: The NP network.

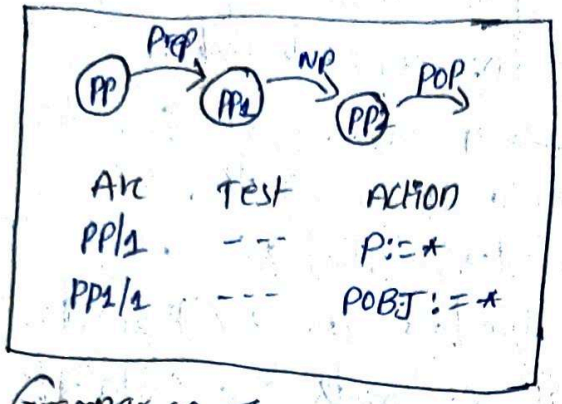
allowable noun complements include an optional number of prepositional phrases. The prepositional phrase network in Grammar 6 is straightforward. Examples of parsing sentences with this grammar are left for the exercises.

Presetting Registers

One further extension to the feature manipulation facilities in ATNs involves the ability to preset registers in a network as that network is being called, much like parameter passing in a programming language. This facility, called the SENDER action in the original ATN systems, is useful to pass information to the network that aids in analyzing the new constituent.

Consider the class of verbs, including want and pray, that accept complements using the infinitive forms of verbs, which are introduced by the word to. According to the classification in section 4.2 this includes the following

- VP: inf — mary wants to have a party.
- NP-VP: inf — mary wants John to have a party.

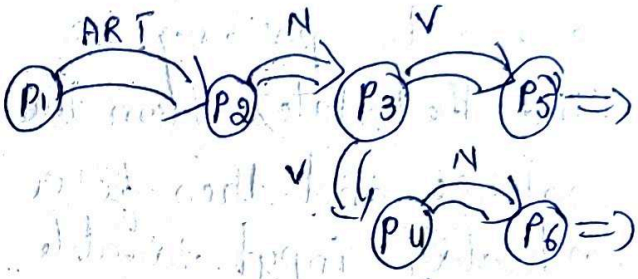


Grammar :6 The PP network

In the context-free grammar developed earlier, such complements were treated as VPs with the VFORM value inf. To capture this same analysis in an ATN, you would need to be able to call a network corresponding to VPs but preset the VFORM register in that network to inf. Another common analysis of these constructs is to view the complements as a special form of sentence with an understood subject. In the first case it is Mary who would be the understood subject (that is, the host), while in the other case it is John. To capture this analysis, many ATN grammars preset the SUBJ register in the new S network when it is called.

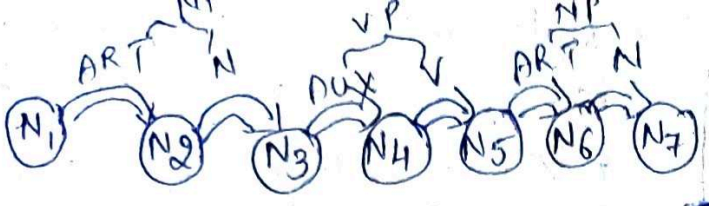
Transition Network Grammars

→ It is a method to represent the natural languages. It is based on applications of directed graphs and finite state automata. The transition network can be constructed by the help of some inputs, state and outputs. A transition network may consist of some states or nodes, some labeled arcs from one state to the next state through which it will move. The arc represents the rule or some conditions upon which the transition is made from one state to another state.



Non-Deterministic Parsing Network.

Eg:- A transition network is used to recognize a sentence consisting of an article, a noun, an auxiliary, a verb, an article, a noun would be represented by the transition network as follows.



Transition Network

→ Transition Network is also a deterministic & non-deterministic parsing. We find the states from the sentence and then for a particular input variable that is the semantic rule from the English Grammar. For a particular semantic rule we found another state like first "* have" a boy age 18"
ART N

Types of Transition Networks

1. Recursive Transition Networks (RTN)
2. Augmented Transition Networks (ATN)

1. Recursive Transition Networks (RTN)

→ RTNs are considered as development for finite state automata with some essential conditions to take the recursive completion for some definitions in consideration.

→ A recursive transition network consists of nodes (states) and labeled arcs (transitions). It permits arc labels to refer to other networks and they in turn may refer back to the referring network rather than just permitting word categories.

→ It is a modified version of transition network.

→ It allows arc labels that refer to other networks rather than word category. A recursive transition network can have 5 types of arcs (Allen's, JM's) like

1. CAT: current word must belong to category.
2. WORD: current word must match label exactly. Collection of words available for sentence
3. PUSH: Named network must be successfully traversed (when we are going from one state to another state which we

4. JUMP: can always be traversed. (labeled arc going to one state to another state)

5. POP: can always be traversed and indicates that input string has been accepted by the network. In RTN, one state is specified as a start state. A string is accepted by an RTN if a POP arc is reached and all the input has been consumed. Let us consider a sentence:

"The stone was dark black"

Here The : ART

stone : ADJ NOUN

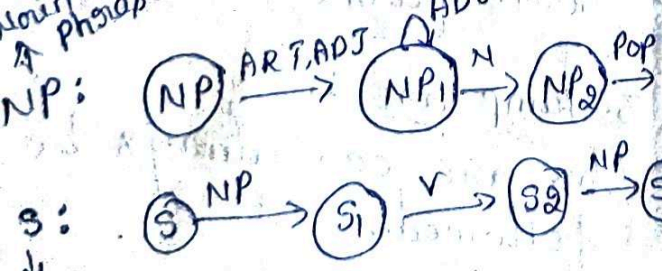
was : VERB

Dark : ADJ

Black : ADJ NOUN

The RTN structure is given in figure

The RTN structure is given in figure



RTN structure

The 1 stone 2 was 3 dark 4 5
ART ADJ/NOUN V ADJ

black 6
ADJ/NOUN

The stone, NP

The stone was 2

The stone was dark 3

The stone was, dark black 4

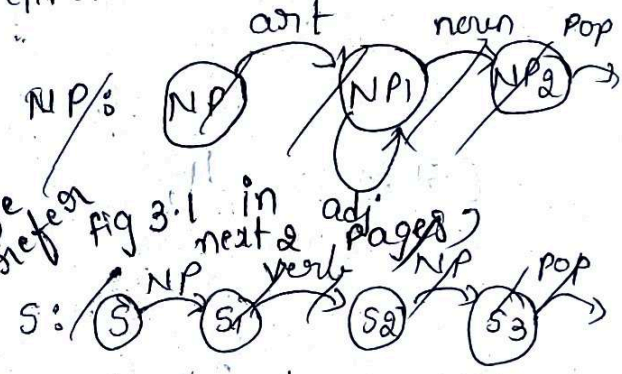
1. consider the grammar

NP → ART NP1

NP1 → ADJ NP1

NP1 → N

Given below is the RTN



(please refer fig 3.1 in next 2 pages)

Derive top down parse for the sentence "The old man cried"

Sol:-

Given the sentence

"The 1 old 2 man 3 cried 4 5

The : ART

old : ADJ, N

man : AN, V

cried : V

Step	current Node	Current position	Return points	Arc to be followed	comments
1.	(S,	1,	NIL)	S/1	initial position
2.	(NP,	1,	(S1))	NP/1	followed push arc to NP network to return ultimately to S1
3.	(NP1,	2,	(S1))	NP1/1	followed arc NP/1 (the)
4.	(NP1,	3,	(S1))	NP1/2	followed arc NP1/1 (old)
5.	(NP2,	4,	(S1))	NP2/2	followed arc NP1/2 (man) since NP1/1 is not applicable
6.	(S1,	4,	NIL)	S1/1	the pop arc gets us back to S1
7.	(S2,	5,	NIL)	S2/1	followed arc S2/1 (cried)

parse succeeds on pop arc from S2.

A trace of a top-down parse

Terminology is

1. push arcs is
Arcs that are labelled with networks
2. cat arcs is
Arcs labeled with word categories
3. jump arc is
An arc can always be followed.

Top-Down Parsing With Recursive Transition Networks :-

An algorithm for parsing with RTNs can be developed along the same lines as the algorithms for parsing Context free Grammars (CFGs). The state of the parse 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 network.

return points - a stack of nodes in other networks where you will continue if you pop from the current network.

→ First, consider an algorithm for searching an RTN that assumes that if you can follow an arc, it will be the correct one in the final parse. Say you are in the middle of a parse and know the three pieces of information just cited. You can leave the current node and traverse an arc in the following cases:

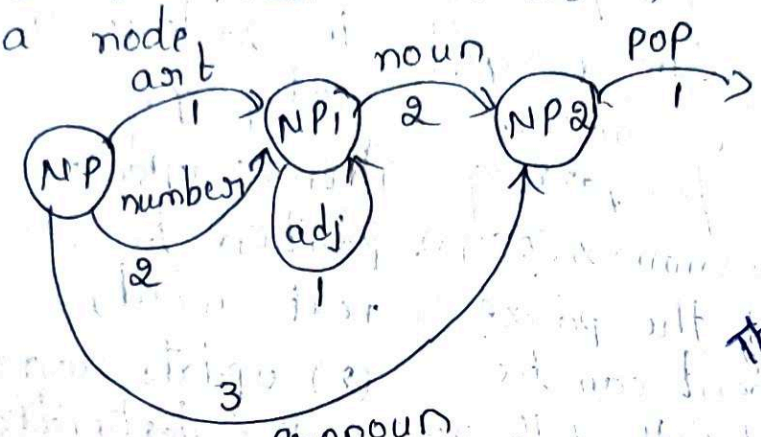
case 1 : If arc names word category and next word in sentence is in that category;
Then (1) update current position to start at the next word;
(2) update current node to the destination of the arc.

case 2 : If arc is a push arc to a network N,
Then (1) add the destination of the arc onto return points;
(2) update current node to the starting node in network N.

case 3 : If arc is a pop arc and return points list is not empty,
Then (1) remove first return point and make it current node.

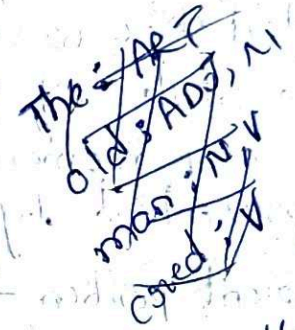
case 4 : If arc is a pop arc, return points list is empty and there are no words left,
Then (1) parse completes successfully.

1. Consider the network grammar. The numbers on the arcs simply indicate the order in which arcs will be tried when more than one arc leaves a node.



NP: RTN

Fig 3.1



S: CFG

1 one 2 saw 3 the 4 man 5

Consider the sentence "one saw the man". obtain a top-down RTN parse with backtracking.

Sol:- one: NP, P, ~~ADJ~~, NUMBER
saw: V, N
the: ART
man: N, V

pronoun

1 one 2 saw 3 the 4 man 5

1 ~~The~~ 2 ~~old~~ 3 man 4 ~~saw~~ 5

	current state	Arc to be followed	Backup States
1.	(S, 1, NIL)	S/1	NIL
2.	(NP, 1, (S1))	NP/2 (& NP/3 for backup)	NIL
3.	(NP1, 2, (S1))	NP1/2	(NP2, 2, (S1))
4.	(NP2, 3, (S1))	NP2/1	(NP2, 2, (S1))
5.	(S1, 3, NIL)	no arc can be followed	(NP2, 2, (S1))
6.	(NP2, 2, (S1))	NP2/1	NIL
7.	(S1, 2, NIL)	S1/1	NIL
8.	(S2, 3, NIL)	S2/2	NIL
9.	(NP, 3, (S2))	NP/1	NIL

Step	Current State	Arc to be followed	Backup states
10.	(NP1, 4, (S2))	NP1/2	NIL
11.	(NP2, 5, (S2))	NP2/1	NIL
12.	(S2, 5, NIL)	S2/1	NIL
13.	Parse succeeds		NIL

consider following sentence

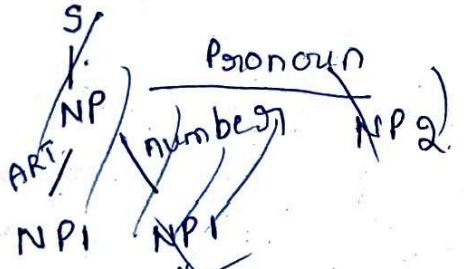
1 one 2 saw 3 the 4 man 5

one - Number, Pronoun, Noun, ADJ

saw - V, N

the - ART

man - N, V



→ The parser initially attempts to parse the sentence as beginning with the NP "one saw", but after failing to find a verb, it backtracks and finds a successful parse starting with the NP one. The trace of the parse is shown above table, where each stage the current parse state is shown in the form of a triple (current node, current position, return points), together with possible states for backtracking. The figure also shows the arcs used to generate the new state and backup states.

→ The trace behaves identically to the previous example except in two places.

→ In step 2, two arcs leaving node NP could accept the word "one". Arc NP/2 classifies "one" as a number and produces the next current state.

→ Arc NP/3 classifies it as a pronoun and produces a backup state

→ This backup state is actually used later in step 6 when it is found that none of the arcs leaving node S1 can accept the input word "the".

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

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

→ Each time a pop is followed, the constituent is placed on the WFST, and every time a push is found, the WFST is checked before the subnetwork is invoked.

→ If the chart contains constituent(s) of the type being pushed for, these are used and the subnetwork is not reinvoked.

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

Problems :-

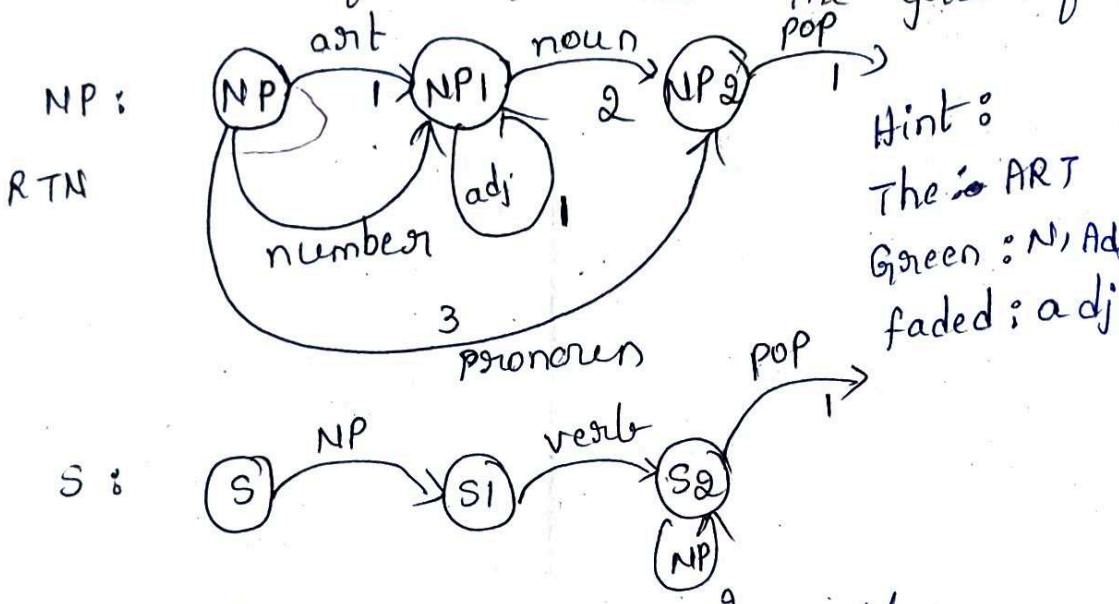
Given the CFG in grammar

1. S → NP VP
2. NP → ART N
3. NP → ART ADJ N
4. VP → V
5. VP → V NP

define an appropriate lexicon and show a trace in the top-down CFG parse of the sentence "The man walked the old dog".

Soln

2. Given the RTN in Grammar below and a lexicon in which "green" can be an adjective or a noun, show a trace in the form of a top-down RTN parse of the sentence "The green faded



Sol:- Top down parsing is not possible.

3. Map the following context-free grammar into an equivalent recursive transition network that uses only three networks - an S, NP and PP network. Make your networks as small as possible.

S → NP VP
VP → V
VP → V NP
VP → V PP
NP → ART NP2
NP → NP2
NP2 → N
NP2 → ADJ NP2
NP2 → NP3 PREPS
NP3 → N

PREPS → PP
PREPS → PP PREPS
PP → NP

Given above CFG the following lexicon, construct a trace of a pure top-down parse and pure bottom up parse of the sentence "The herons fly in groups". Make your traces as clear as possible, select the rules in the order above and indicate all parts of the search. The lexicon entries for each word are

the : ART
herons : N
fly : N V ADJ
in : P
groups : N V

Sol:

4. Consider the following CFG:

S → NP V

S → NP AUX V

NP → ART N

with sentence "1 The 2 man 3 is 4 laughing 5 with the lexicon entries:

the : ART

man : N

is : AUX

laughing : V

show every step of top-down parsing with backtracking; a simple top down parsing and bottom up parsing