Parsing Complementizer Phrases In Machine Translation System

T.Suryakanthi
Research Scholar of Computer Science and Engineering
Lingaya's University
Haryana, India
ch.kanthi@gmail.com

Dr. S.V.A.V Prasad
Dean (R&D and Industrial Consultancy)
Lingaya's University
Haryana, India
prasad.svav@gmail.com

***Abstract*— Every language has a finite number of words and finite number of rules but infinite number of sentences. Sentences are not formed by the words alone but by structural units known as constituents. Analysis of the sentence constituency begins at the larger units of grammar and then breaks the larger units down into smaller and smaller units. Syntax is the study of relationships between words and how they are put together to construct phrases and sentences**

Sentences can also take embedded form that is each sentence consist another sentence with in it. Embedded phrases are complements like NP or PP complements except they are united by the complementizer phrase and may be introduced by a complementizer like subordinate conjunction or a relative pronoun.

Paring is the process of identifying the structure of a sentence to know whether a sentence is well formed with respect to a language. It is a process to check whether a sentence can be derived from the given grammar of a language. The two basic approaches for parsing are top-down and bottom-up. This paper describes the process of parsing of complementizer phrases in a machine translation system.

Keywords-component Complementizer Phrase (CP), Machine Translation (MT), Government and Binding (GB), Parts of Speech (POS)

I. INTRODUCTION

The CP is a phrase having the non-lexical element Complementizer (C) as its head. A Complement is a conjunction which marks a complement clause. Complements include subordinate conjunctions, relative pronouns and relative adverbs [1].

Subordinate conjunctions are the conjunctions which act as connectors between a sentence and a clause. The complementizer will act as a head to the clause. Complementizers such as 'whether', 'for', 'if' and 'that' act as subordinate conjunctions [2].

a) I will ask if Ram eats Mango
b) I will say that Ram eats mango
c) I expect for Ram to eat Mango
d) I wonder whether Ram ate the mango

The subordinate clause if in (a) is interrogative that in (b) is declarative. The difference between the two is signaled by the choice of complementizer introducing the clause. In other words, the complementizer determines the type of clause. This suggests that we treat the complementizer C, as the head of a clause. Complementizers such as whether, if, that and for introduce a sentence (IP), C selects an IP-complement. The choice of the type of IP is determined by the choice of C. The complementizers that and if select a finite clause as their complement, for selects an infinitival clause and whether selects either type of clause.

A relative pronoun is a pronoun that marks a relative clause within a larger sentence. A relative pronoun links two clauses into a single complex clause. The choice of a relative pronoun depends on the way it is used. 'Who' is used when we talk about people, which is used when we talk about things and whose is used when we talk about third person or thing.

Ex: She is the girl who won the championship
It is the car which had an accident
He is the boy whose purse was lost

Adverbial clauses have a greater variety of lexical items as complementizers. Complements like while, since, because, although, if, when, so that, as, such, before, after, until, as long as, as soon as, by the time that, now that, once act as complementizers. "Like an adverbial clause, the wh-question always begins with a complementizer, in this case, who, whom, whose, what, which, why, when, where, and how. An empty complementizer is a

hypothetical phonologically null category with a function parallel to that of visible complementizers such as that and for. Its existence in English has been proposed based on the following type of alternation:

Ex: He hopes you go ahead with the speech
He hopes that you go ahead with the speech

II. LANGUAGE MODELING

Representation of a language is called language modeling. Modeling can be either Grammar-based or Statistical based. In grammar based modeling the language is represented in terms of a set of grammar rules. Of all the grammar formalisms Government Binding (GB) approach which lies emphasis on universal grammar has the most impact on NLP. Grammar based model involves writing a set of grammar rules for each language to describe which sentences are well formed within a language. These rules are used to expand and construct a tree structure for a sentence. These rules say that a certain symbol may be expanded in the tree by a sequence of other symbols. The set of re-write rules used to represent the language are phrase –structure grammar rules.

For phrase structures the symmetrical X-bar analysis is adopted in GB theory [3]. We need three levels of projection X0, X1, X11. X-bar phrase in a language is defined by the following rules. The X-bar analysis in GB theory for phrases is shown in figure 1.

$$XP \longrightarrow XPS\ X^{1}$$
$$X^{1} \longrightarrow (ZP);\ X^{11}$$
$$X^{11} \longrightarrow (YP);\ X^{0}$$

In the figure 1 X can be any lexical category like Verb, Noun, Adjective, Preposition. X0 is the head; XPS is XP specifier; X1 is the intermediate projection; ZP is Adjunct which is optional; YP (XCOMP) is Complement.

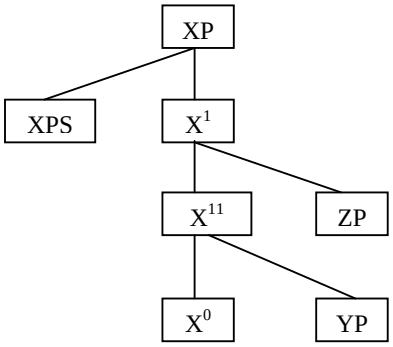

Figure 1. X-bar Analysis

III. SYSTEM DESIGN

The Parsing process is carried in a Systematic way. The process is carried out in four major modules: Tokenization, POS Tagging, Chunking and Parsing. Morphological analysis is done if the words are inflected in any form like plural or tense form. The Output of one module is fed as input to the next module. The systematic flow of the parsing process shown in fig (1) is explained using the following Complementizer phrase: “He hopes that you go ahead with the speech

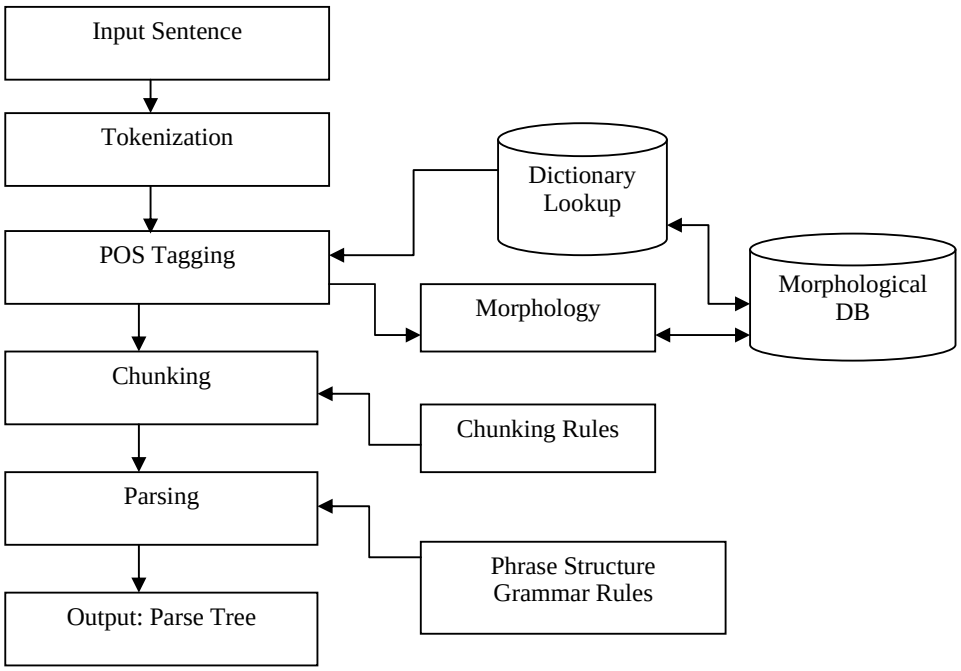

Figure 2. System Design

A. Tokenization

Sentences are formed of individual words called tokens. Tokenizing is the process of splitting sentences into tokens. Discourse input is divided into sentences using the delimiters ".", "!","?" Sentences are there by isolated into words whenever the delimiter white space is encountered

Algorithm for Tokenization:

Input: Discourse Text is taken as input

Step 1: Read the Discourse input.

Step 2: Split the discourse into array of sentences using the delimiters (".", "!","?")

Step 3: Split the sentence into array of words using the delimiter (Space)

Output: An array of words

- Input of Tokenization: He hopes that you go ahead with the speech
- Output of Tokenization: He | hopes | that | you | go | ahead | with | the | speech

B. POS Tagging

POS tagging is assigning parts of speech to each individual token of a sentence. The output of Tokenization module, array of words is fed to POS tagging module. Each word is searched in the Dictionary and parts of speech are assigned. If a word is not found in the dictionary it is sent to Morphological analysis. After morphology is applied a base word is obtained which is again searched in the dictionary. If the word is found then POS is assigned and if not found it requires post processing. The Penn Tree bank tag set (Annexure 1) is used as a naming convention for parts of speech in the lookup dictionary. The POS is assigned as a key/value pair for each word and fed into an array. The key is the word itself and the value is the POS

- Input of POS Tagging: He | hopes | that | you | go | ahead | with | the | speech
- Output of POS Tagging: He/PRP hopes/VBZ that/IN you/PRP go/VBP ahead/RB with/IN the/DT speech/NN

C. Chunking

Chunking is a process where the input sentence is partitioned into a sequence of non-overlapping chunks [4] [5]. The output of POS Tagging, array of words along with the POS is taken as input to Chunking. Chunking is a process where the array of words is grouped as non-overlapping chunks. Chunks are meaningful sequences of words typically consisting of a content word surrounded by zero or more function words. Chunking reduces the complexity of full parsers and enhances the performance of full parser. Chunks are sequences of adjacent words grouped on the basis of linguistic properties. The rules of chunking should made in line with grammar of the language [6] [7]. The Rules of Chunking are shown in Table I.

TABLE I. RULES FOR CHUNKING

Phrase (LHS)	Constituents (RHS)	Details
NP	Pronoun/ Noun/ Numeral Zero Article+ Noun Pre-Determiner + Determiner + Post-determiner + Noun Determiner + Noun	Numeral- 13,12, 1
Complex NP	Pre-Modifier + Noun + Post- Modifier	Pre-Modifier –Adjective Post-Modifier--PP, Finite clause, Non-Finite clause, AP, NP, AdvP
VP	[Auxiliary] + Verb + [Object] + [Complement] + [Modifier]	Object—Direct/ Indirect/ PP Complement—Subject/ Object/ Sentence Modifier—NP,AP,PP, AdvP
AdvP	Adverb Degree Adverb + Adverb Intensifier + Adverb	Intensifier—very, quite, rather, too, more/most, only, somewhat
PP	Preposition + NP	
AP	Adj Intensifier + Adj Adj + PP Adj + Clause Adj + Infinitival Adv + Adj + PP	Intensifier—very, quite, rather, too, more/most, only, somewhat
CP	Subordinate conjunction Relative Pronoun	that, if, for, whether

Algorithm for Chunking: This algorithm uses bottom up approach to find non overlapping chunks also called as Phrases.

Input: An array of words W with corresponding POS as a key/ value pair, which is the output of POS module.

Step 1: From the array W, take a sequence of words (key) and extract the corresponding POS (value) and match them to the right hand side of the rules of chunking given in table 1

Step 2: If a match is found group them as a chunk of left hand side phrase. Add this phrase to an array CHUNKS. Add the group of words into an array and make a link to the phrase that is added into the array, CHUNKS

Step 3: If a match is not found continue the matching process as a search process.

Step 4: Repeat step 2 and 3 until all the symbols of the array P are exhausted.

Output: An array of chunks of the Input sentence, CHUNKS

- Input of Chunking: He | hopes | that | you | go | ahead | with | the | speech
- Output of Chunking: [NP He/PRP] [VP hopes/VBZ] [CP that/IN] [NP you/PRP] [VP go/VBP] [ADVP ahead/RB] [PP with/IN] [NP the/DT speech/NN]

IV. PHRASE STRUCTURE GRAMMAR RULES FOR PARSING COMPLEMENTIZER PHRASE

Phrase structure rules also called as re-write rules are used to represent the grammar of a language. These are associated with the syntax of a language. With these rules the sentences can break down into constituents called the phrasal categories or lexical categories. Phrase structure grammar can be thought of as an alternative way of expressing the information found in the tree diagram with rewrite rules [8]. Unlike the dependency grammars the phrase structure grammars are based on constituency relation. Using this model the linguist formalizes the grammar by means of generative rules which explicitly assign the correct constituent structure to sentences. This model of grammar shows not only the terminal elements or constituents of a linear structure but also specifies the subunits and the level at which these units form natural groups. Using these grammar rules the pattern underlying the sentence and the syntactic devices used to link the constituents together, and the ways in which various parts relate to one another is clearly understood. An important aspect of phrase structure rules is that they view sentence structure from top down. Phrase structure rules result in constituency grammars. The constituency tree or parse tree is generated using this grammar.

Phrase Structure grammar rules for parsing complementizer phrases in English are as shown in Table II

TABLE II: PHRASE STRUCTURE GRAMMAR RULES FOR ENGLISH

Complementizer phrase	**Noun phrase**
CP -> (CPS) + C1 C^1 -> C^0 + CCOMP CCOMP -> IP CPS -> ADV	NP -> (NPS) + N^1 N^1 -> N^0 + NCOMP N^1 -> N^0 NCOMP -> IP / PP/ CP NPS -> DET
Verb phrase	**Prepositional phrase**
VP-> V^1 V^1-> V^0 + VCOMP VCOMP-> NP/PP/CP/VP/IP	PP -> PPS + P^1/P^1 PPS -> ADV P^1 -> P^0 + PCOMP /P^0 PCOMP -> NP P^1 -> P PCOMP
Inflection phrase	**Adjective phrase**
IP -> (IPS) + I^1 I^1 -> I^0 + ICOMP ICOMP -> VP IPS -> NP	AP -> APS + A^1 / A^1 APS -> ADV A^1 -> A^0 / A^0 + ACOMP ACOMP -> PP / AP

V. PARSING

Parsing is a process of automatically building syntactic analysis of a sentence in terms of a given grammar and a lexicon. The output of parsing is something logically equivalent to a tree, displaying dominance and precedence relations between constituents of a sentence, perhaps with further annotations in the form of attribute value equations (features) capturing other aspects of linguistic description [9] [10].

To parse a string according to a grammar means to reconstruct the production tree that indicate how the given string can be produced from the given grammar, a sentence can produce more than one such tree for a given grammar. The parse tree describes how the grammar was used to produce the sentence. The technique used for construction of a parse tree for a sentence, given a grammar is called a parsing technique. The two basic methods of parsing are top-down and bottom-up [11].

The phrase structure of a sentence is generally represented by a tree diagram. This representation of the phrase structure of a sentence is known as its `phrase marker' or `P marker' for short. The points that are joined by the lines or branches are called `Nodes'. Each of the nodes, except those on the bottom line (which are the terminal nodes) is given a label that represents a grammatically definable constituent - N, V, NP, VP, etc. where one mode is higher than another and joined to it by branches, it is said to `Dominate' it, if it is placed immediately above it and joined by a single line, it `Immediately' dominates it. `Dominance' then shows how a larger constituent may consist of one or more constituents of a smaller kind. It is also important to note that the tree structure preserves the linear order of the constituents, just as plain IC analysis does. The first noun phrase precedes the verb phrase; the verb precedes the second noun phrase. The determiner precedes the noun. 'Precedence' thus like 'Dominance' is clearly shown in the tree diagram.

A. Algorithm for Parsing Complementizer phrases

The algorithm takes the array of chunks as input. Searches the complementizer phrase (CP) and locates its position. Divides the chunk array into two sub arrays, one consisting of chunks before CP and other consisting of chunks after CP. Parse procedure is called with the two sub arrays as input. The outputs of the procedure parse are pushed into array parse along with the CP respectively.

A simple parsing algorithm for parsing complementizer phrases is as follows:

Input: Output of Chunking module, an array of CHUNKS is taken as input
Step 1: Search CP and its index position, i in the array CHUNKS
Step 2: Split the array CHUNKS into two sub arrays CHUNKS [0...i-1] and CHUNKS [i+1...n]
Step 3: Define a procedure Parse(c, p, q): Ptemp with the following input and output
Input: c- Array of chunks, p starting index, q ending index
Output: Array Ptemp with parsed chunks.
Step 4: call Parse(c, p, q) on the sub array CHUNKS [i+1...n]. Set the output Ptemp into an array Temp1
Step 5: call Parse(c, p, q) on the sub array CHUNKS [0...i-1]. Set the output Ptemp into an array Temp 2
Step 6: Declare an array PARSE as a Stack
PUSH '(' into array PARSE
PUSH Temp2 into array PARSE
PUSH '(' into array PARSE
PUSH CP into array PARSE
PUSH '(' into array PARSE
PUSH Temp1 into array PARSE
PUSH ')', ')', ')' into array PARSE
Procedure Parse (c, p, q): Ptemp

```
Declare a Dynamic array Ptemp
FOR k:=p TO q
        DO flag:=i
        FOR r:=k+1 TO q
        DO IF c[r] is COMPLEMENT of c[k]
        //refer table 2 to get the phrase structure rules for complements
                THEN Merge c[r] into c[k]
                        Ptemp = c[r]
                        flag=flag+1
                        r=r+1
                ELSE
                        BREAK;
        k=flag+1
RETURN Ptemp
```

Output: The PARSE will contain the list representation of PARSE TREE

- Input of Parsing: [NP He/PRP] [VP hopes/VBZ] [CP that/IN] [NP you/PRP] [VP go/VBP] [ADVP ahead/RB] [PP with/IN] [NP the/DT speech/NN]
- Output of Parsing: (TOP (S (NP (PRP He)) (VP (VBZ hopes) (CP (IN that) (S (NP (PRP you)) (VP (VB go) (ADVP (RB ahead)) (PP (IN with) (NP (DT the) (NN speech))))))))

B. Parse Tree

Parse tree for the complementizer phrase "He hopes that you go ahead with the speech" is shown in figure 3

CONCLUSIONS

The process has been successfully implemented for Complementizer phrases with one subordinate conjunction. This work can be extended to complex Complementizer phrases consisting of two or three subordinate conjunctions. The accuracy achieved in parsing the sentences is above 80%. The parsing system is extended as translation system in the next phase of the project.

ACKNOWLEDGMENT

I would like to thank my guide Dr. S.V. A.V Prasad for timely help and support in doing the project. I also want to make a special mention of Dr. T.V Prasad for his valuable reviews and comments.

APPENDIX-1: Penn Tree-bank Tag set

CC-Coordinating conjunction	NNS-Noun, plural	TO-to
CD-Cardinal number	NP-Proper noun, singular	UH-Interjection
DT-Determiner	NPS-Proper noun, plural	VB-Verb, base form
EX-Existential there	PDT-Pre-determiner	VBD-Verb, past tense
FW-Foreign word	POS-Possessive ending	VBG-Verb, gerund or present participle
IN-Preposition or subordinating conjunction	PP-Personal pronoun	VBN-Verb, past participle
JJ-Adjective	PP$-Possessive pronoun	VBP-Verb, non-3rd person singular present (like)
JJR-Adjective, comparative	RB-Adverb	VBZ-Verb, 3rd person singular present (likes)
JJS-Adjective, superlative	RBR-Adverb, comparative	WDT-Wh-determiner
LS-List item marker	RBS-Adverb, superlative	WP-Wh-pronoun
MD-Modal	RP-Particle	WP$-Possessive wh-pronoun
NN-Noun, singular or mass	SYM-Symbol	WRB-Wh-adverb

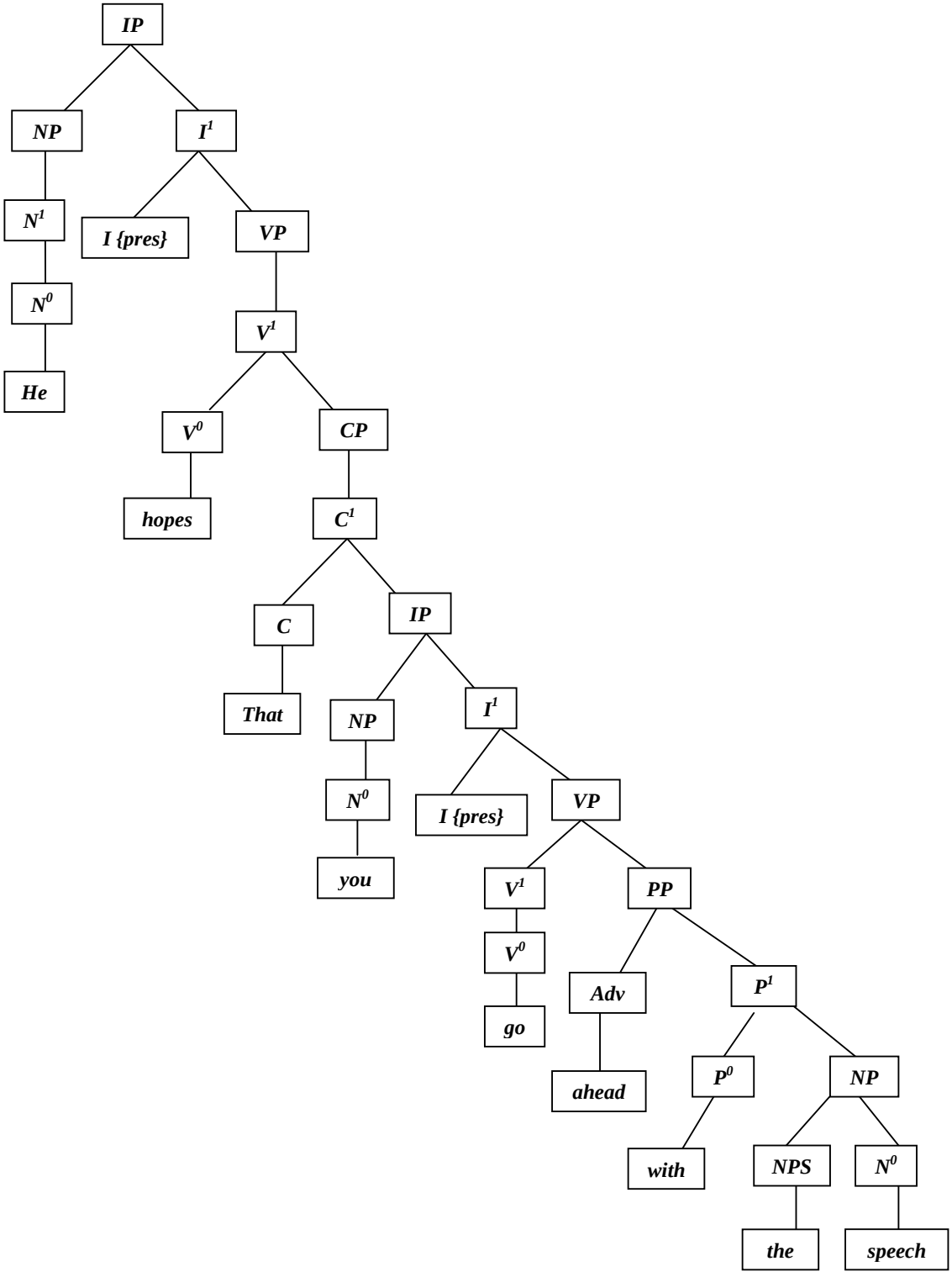

Figure 3. Parse Tree

REFERENCES

[1] Gertrude Stein, "Linguistics 110 Linguistic Analysis: Sentences and Dialects, Representing Phrase Structure," http://www.departments.bucknell.edu/linguistics/lectures/10lect04.html

[2] Virginia Hill, "Complementizer Phrases (CP) in Romanian," Italian Journal of Linguistics, vol. 14, no. 2, 2002.

[3] Cheryl A. Black, A step-by-step introduction to the Government and Binding theory of syntax, SIL - Mexico Branch and University of North Dakota, 1998
[4] Steven Abney. Tagging and Partial Parsing. In: Ken Church, Steve Young, and Gerrit Bloothooft (eds.), Corpus-Based Methods in Language and Speech. Kluwer Academic Publishers, Dordrecht. 1996.
[5] Giuseppe Attardi, Felice Dell'Orletta, "Chunking and Dependency Parsing", Workshop at LTRC- Partial Parsing, Marrakech, Morocco, June 2008
[6] English grammar- From Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/english_grammar
[7] Nguyen Thi Van Lam, " Structure of English Noun phrases," TIL (Tun Institute of Learning, Yangon, Myanmar), May 2004 http://www.tuninst.net/English/MaLam04.htm
[8] P. Schlenker, Introduction to Language - Lecture Notes 4B Sentence Structure II:Phrase Structure Grammars ," http://www.linguistics.ucla.edu/people/schlenker/LING1-06.html
[9] Eric Bill, "Automatic Grammar Induction and Parsing Free Text: A Transformation Based Approach," In Proceedings of the 31st annual meeting of the association for computational linguistics, 1993
[10] Antonio Zamora, Chevy Chase, Michael D. Gunther, Gaithersburg, Elena M. Zamora, U.S. Patent 4,887,212, 1989. [PARSER FOR NATURAL LANGUAGE TEXT]
[11] Dick Grune and Ceriel J.H Jacobs , Parsing Techniques- A Practical Guide , 2nd edition, Ellis Horwood, Chichester, England, 1990

AUTHORS PROFILE

Author 1: T. Suryakanthi is a Research scholar in Lingaya's University She completed her Masters degree in Computer Applications. She has work experience of 2years in Industry and 1 year in Academics.

Author 2: Dr. S.V.A.V Prasad is a professor in Electronics and Communication Engineering in Lingaya's University. He also designates as Dean R&D and Industrail Consultancy projects in Lingaya's University. He has got good experience in both Industry and Academics.