# Recognizing English Grammar Using Predictive Parser

**Magdum P. G.[1], Kodavade D. V.[2]**

[1] (Department of Computer Science and Engineering, RMCET (Ratnagiri), Mumbai University,
[2] (Department of Computer Science and Engineering, DKTE (Ichalkanji), Shivaji University,

## ABSTRACT

**We describe a Context Free Grammar (CFG) for English language and hence we propose a English parser based on the context free grammar. Our approach is very much general to apply in English Sentences and the method is well accepted for parsing a language of a grammar. The proposed parser is a predictive parser and we construct the parse table for recognizing English grammar. Using the parse table we recognize syntactical mistakes of English sentences when there is no entry for a terminal in the parse table. If a natural language can be successfully parsed then grammar checking from this language becomes possible. The proposed scheme is based on Top down parsing method and we have avoided the left recursion of the CFG using the idea of left factoring.**

***Keywords-*** Context Free Grammar (CFG), Predictive Parser, Parse Table, Top down and Bottom up Parser, Left Recursion.

## I. INTRODUCTION

Parsing is the process of using grammar rules to determine whether a sentence is legal, and to obtain its syntactical structure. Tree structure provides two information viz. it divides the sentence into constituents (in English, these are called phrases) and it puts them into categories (Noun Phrase, Verb Phrase, etc).To process any natural language, parsing is the fundamental problem for both machines and humans. In general, the parsing problem includes the definition of an algorithm to map any input sentence to its associated syntactic tree structure[1]. A parser analyzes the sequence of symbols presented to it based on the grammar [2]. Natural language applications namely Information Extraction, Machine Translation, and Speech Recognition, need to have an accurate parser[3]. Parsing natural language text is more difficult than the computer languages such as compiler and word processor because the grammars for natural languages are complex, ambiguous and infinitynumber of vocabulary. For a syntax based grammar Checking the sentence is completely parsed to check the correctness of it. If the syntactic parsing fails, the text is considered incorrect. On the other hand, for statistics based approach, Parts Of Speech (POS) tag sequences are prepared from an annotated corpus, and hence the frequency and the probability[4]. The text is considered correct if the POS-tagged text contains POS sequences with frequencies higher than some threshold[5]. Natural languages like English and even Hindi are rapidly progressing as far as work done in processing by computers is concerned.

In this paper, we proposed a context free grammar for the English language and hence we proposed a predictive English parser constructing a parse table. We have adopted the top down parsing scheme and avoided the problem of left recursion using left factoring for the proposed grammar. We implemented the English dictionary ms access format using the corresponding word as tag name and it's POS as value. It helps to search the dictionary very fast. English grammar has huge amount of forms and rules .We believe the proposed grammar and parser can be applicable to any forms of English sentences and can be used as grammar checker.

A rule based English parser has been proposed in [1] that handles semantics as well as POS identification from English sentences and ease the task of handling semantic issues in machine translation. The system is based on analyzing an input sentence and converting into a structural representation. A parsing methodology for English natural language sentences is proposed and shows how phrase structure rules can be implemented by top-down and bottom-up parsing approach to parse simple sentences of English. A comprehensive approach for English syntax analysis was developed [4] where a formal language is defined as a set of strings. Each string is a concatenation of terminal symbols. Some other approaches such as Lexical Functional Grammar (LFG) [4] and Context Sensitive Grammar (CSG) [5] have also been developed for parsing English sentences. Some developers developed English parser using SQL to check the correctness of sentence; but its space complexity is inefficient. Besides it takes more time for executing SQL command. As a result that Parser becomes slower.

## II. A PARSING SCHEME FOR ENGLISH GRAMMAR RECOGNITION

A predictive parser is an efficient way of implementing recursive decent parsing by handling the stack of activation record. The predictive parser has an input, a stack, a parse table and output. The input contains the string to be parsed or checked, followed by a $, the right end marker. The stack contains a sequence of grammar symbols, the parse table is a two dimensional array M[A,n] where A is nonterminal and n is a terminal or $ sign.

| **Tag name(Symbol)** | **Examples** |
|---|---|
| Determiner | A, an, the |
| Noun | bus, home |
| Pronoun | he, she |
| Adjective | beautiful, |
| Adverb | slowly, fast |
| Verb | run, keep |
| Auxiliary verb | is ,was |
| Conjunction | and, but |

Table 1: Tag set Description for English grammar

# III. ENGLISH GRAMMAR DESIGN

Once constituents have been identified, the productions for Context Free Grammar (CFG) are developed for English sentence structures. As English grammar has different forms, the same production term can be used only by reorganizing the in the grammar.

## 1.1. Context Free Grammar

A context-free grammar (CFG) is a set of recursive rewriting rules (or productions) used to generate patterns of strings.

A CFG consists of the following components:

- A set of terminal symbols, which are the characters of the alphabet that appear in the strings generated by the grammar.
- A set of nonterminal symbols, which are placeholders for patterns of terminal symbols that can be generated by the nonterminal symbols.
- A set of productions, which are rules for replacing (or rewriting) nonterminal symbols (on the left side of the production) in a string with other nonterminal or terminal symbols (on the right side of the production).
- A start symbol, which is a special nonterminal symbol that appears in the initial string generated by the grammar.

To generate a string of terminal symbols from a CFG, we:

- Begin with a string consisting of the start symbol;
- Apply one of the productions with the start symbol on the left hand size, replacing the start symbol with the right hand side of the production;

Repeat the process of selecting nonterminal symbols in the string, and replacing them with the right hand side of some corresponding production, until all nonterminals have been replaced by terminal symbols.

## 1.2. Left Factoring

The parser generated from this kind of grammar is not efficient as it requires backtracking. To remove the ambiguity from the grammar we have used the idea of left factoring and reconstruct the grammar productions. Left factoring is a grammar transformation useful for producing a grammar suitable for predictive parsing. The basic idea is that when it is not clear which of the productions are to use to expand a non terminal then it can defer to take decision until we get an input to expand it. In general, if we have productions of form

$A \rightarrow \alpha\beta1|\ \alpha\beta2$

We left factored productions by getting the input α and break it as follows

$A\rightarrow \alpha A',\ \ A' \rightarrow \beta1|\ \beta\ 2$

| |
|---|
| S→ NP.VP |
| NP→ a NP1.VP4\|pronoun.NP4\| the.NP6\| an.NP7\| propernoun.NP3\| I\|noun |
| NP1→ noun\| adjective.NP2 |
| NP2→ noun |
| NP3→ conjuction.NP5\| ε |
| NP4→ conjuction.NP5\| noun\|ε |
| NP5→ noun\|pronoun\|propernoun |
| NP6→ propernoun.NP4\| adjective NP2 |
| NP7→ adjective1.NP2 |
| VP→ verb1. vp'\| verb2.vp'\| aux31.VP3\| aux32.VP6\| aux21.VP4\| aux22.VP9\| aux11.VP5 |
| VP→ aux11.VP7\|adverb.VP6\| adverb.VP6 |
| VP'→NP1.VP2\| adverb.VP2\| PP.NP\|ε\|pronoun |
| VP1→ adjective.NP2 |
| VP2→ PP.NP\| ε |
| VP3→ verb4.VP'\| adverb.VP6\| pronoun1.VP1 |
| VP4→ verb1.VP'\| be.VP6\| aux11.VP7\| have.VP8 |
| VP5→verb3.VP'\| been.VP6 |
| VP6→ verb4.VP' |
| VP7→ verb3.vp' |
| VP8→been.VP6 |
| VP9→be.VP6 |
| PP→ preposition |

Table 2: Left factored grammar

# IV. PARSER DESIGN

A parser for a grammar G is a program that takes a string as input and produces a parse tree as output if the string is a sentence of G or produces an error message indicating that the sentence is not according to the grammar G. To construct a predictive parser for grammar G two functions namely FIRST() and FOLLOW() are important. These functions allow the entries of a predictive parse table for G. Once the parse table has been constructed we can verify any string whether it satisfy the grammar G or not. The FIRST() and FOLLOW() determines the entries in the parse table.

### 4.1. Rules for computing FIRST

I. If X is a terminal symbol then FIRST(X)={X}
II. If X is a non-terminal symbol and X → ε is a production rule then ε is in first(X).
III. If X is a non-terminal symbol and X → Y1Y2...Yn is a production rule then first(X)=first(Y1).

FIRST(S) ={noun, pronoun,a, an, the,propernoun}
FIRST(NP) ={noun, pronoun,a,an,the}
FIRST(NP1) ={ noun, pronoun,adjective}
FIRST(NP2) = { noun }
FIRST(NP3) = {conjunction,ε}
FIRST(NP4) ={conjunction,noun, ε }
FIRST(NP5) = { noun, pronoun,propernoun }
FIRST(NP6) = { propernoun,adjective }
FIRST(NP7) = {adjective1}
FIRST(VP) ={verb1,verb2,verb3,verb4, aux11, aux12, aux21, aux22,aux31, aux32}
FIRST(VP') ={noun,adjective,pronoun,adverb, ε}
FIRST(VP1) ={preposition, adverb, ε }
FIRST(VP2) = { preposition,ε }
FIRST(VP3) = {verb4,adverb}
FIRST(VP4) ={verb1,be,aux11,have }
FIRST(VP5) = {verb3,been}
FIRST(VP6) = {verb4}
FIRST(VP7) = {verb3}
FIRST(VP8) = {been}
FIRST(VP9) = {be}
FIRST(PP) = {preposition}

Table 3: FIRST function Computation

### 4.2. Rules for computing FOLLOW

I. If S is the start symbol then $ is in FOLLOW(S)
II. If A→ aBb is a production rule then everything in FIRST(b) is FOLLOW(B) except ε.
III. If ( A → aB is a production rule ) or ( A →aBb is a production rule and ε is in first(b) ) then everything in FOLLOW(A) is in FOLLOW(B).

FOLLOW(S) ={$}
FOLLOW(NP) ={ verb1,verb2,verb3,verb4, aux11, aux12, aux21, aux22,aux31,aux32,adjective,adverb}
FOLLOW(NP1) ={ noun, conjunction, verb1,verb2,verb3,verb4, aux11, aux12, aux21, aux22,aux31, aux32,adjective,adverb }
FOLLOW(NP2) = { noun, conjunction, verb1,verb2,verb3,verb4, aux11, aux12, aux21, aux22,aux31, aux32,adjective,adverb }
FOLLOW(NP3) = { verb1,verb2,verb3,verb4, aux11,aux12,aux21, aux22,aux31,aux32,adjective,adverb }
FOLLOW(NP4) ={ verb1,verb2,verb3,verb4, aux11,aux12, aux21,aux22,aux31, aux32,adjective,adverb }
FOLLOW(NP5) = { verb1,verb2,verb3,verb4, aux11,aux12,aux21,aux22,aux31, aux32,adjective,adverb }
FOLLOW(NP6) = { verb1,verb2,verb3,verb4, +aux11,aux12,aux21,aux22,aux31, aux32,adjective,adverb }
FOLLOW(NP7) = { verb1,verb2,verb3,verb4, aux11, aux12,aux21,aux22,aux31,aux32,adjective,adverb }
FOLLOW(VP) ={verb1,verb2,verb3,verb4, aux11,aux12,aux21,aux22,aux31, aux32,adjective,adverb,$}
FOLLOW(VP') ={ verb1,verb2,verb3,verb4, aux11,aux12,aux21,aux22,aux31,aux32,adjective,adverb }
FOLLOW(VP1) ={ verb1,verb2,verb3,verb4, aux11, aux12,aux21,aux22,aux31,aux32,adjective,adverb }
FOLLOW(VP2) = { verb1,verb2,verb3,verb4, aux11, aux12,aux21,aux22,aux31,aux32,adjective,adverb }
FOLLOW(VP3) = { verb1,verb2,verb3,verb4, aux11, aux12,aux21,aux22,aux31,aux32,adjective,adverb }
FOLLOW(VP4) ={ verb1,verb2,verb3,verb4, aux11, aux12, aux21,aux22,aux31,aux32,adjective,adverb }
FOLLOW(VP5) = { verb1,verb2,verb3,verb4, aux11,aux12,aux21, aux22,aux31,aux32,adjective,adverb }
FOLLOW(VP6) = { verb1,verb2,verb3,verb4, aux11,aux12,aux21,aux22,aux31, aux32,adjective,adverb }
FOLLOW(VP7) = { verb1,verb2,verb3,verb4, aux11,aux12,aux21,aux22,aux31, aux32,adjective,adverb }
FOLLOW(VP8) = { verb1,verb2,verb3,verb4, aux11,aux12,aux21,aux22,aux31, aux32,adjective,adverb }
FOLLOW(VP9) = { verb1,verb2,verb3,verb4, aux11, aux12,aux21,aux22,aux31,aux32,adjective,adverb }
FOLLOW(PP) = { verb1,verb2,verb3,verb4, aux11, aux12,aux21,aux22,aux31,aux32,adjective,adverb }

Table 4: FOLLOW function Computation

### 4.3. Algorithm to construct Predictive table

a. set Input Pointer(IP) to point to the first word of w;
b. set X to the top stack word;
while ( X!= $ ) begin /* stack is not empty */
if X is a terminal, pop the stack and advance IP;
if X is a Nonterminal and M[X,IP] has the production X → Y1Y2…Yk
output the production X -> YlY2 -Yk;
pop the stack;
c. push Yk, Yk-1,. . . , Yl onto the stack, with Yl on top;
d. if X=$ ,Sentence is Accepted. end

| | A | an | The | noun | i | pronoun | pnoun | adj | adject1 | Adv | prep | conj |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **S** | NP. VP | NP. VP | NP.VP | | NP.VP | NP. VP | NP.VP | | | | | |
| **NP** | A.NP1. VP4 | an.NP7 | The. NP6 | | I | Pronoun. NP3 | | | | | | |
| **NP1** | | | The. noun | noun | | | | adj.NP2 | | | | |
| **NP2** | | | | noun | | | | | | | | |
| **NP3** | | | | | | | | | | | | Con. NP5 |
| **NP4** | | | | noun | | | | | | | | Con. NP5 |
| **NP5** | | | | noun | | Pronoun | pnoun | | | | | |
| **NP6** | | | | | | Pronoun. NP5 | | adj.NP2 | | | | |
| **NP7** | | | | | | | | | Adj1.NP2 | | | |
| **VP** | | | | | | | | adj.VP6 | | Adv.VP4 | prep. NP1 | |
| **VP'** | | | NP1. VP1 | NP1. VP1 | | Pronoun | | NP1. VP1 | | Adv.VP2 | prep. NP | |
| **VP1** | | | | | | | | adj.NP2 | | | | |
| **VP2** | | | | | | | | | | | | |
| **VP3** | | | | | | Pronoun. VP1 | | | | Adv.VP6 | | |
| **VP4** | | | | | | | | | | | | |
| **VP5** | | | | | | | | | | | | |
| **VP6** | | | | | | | | | | | | |
| **VP7** | | | | | | | | | | | | |
| **VP8** | | | | | | | | | | | | |
| **VP9** | | | | | | | | | | | | |
| **PP** | | | | | | | | | | | prep | |

| | verb1 | Verb2 | Verb3 | verb4 | aux11 | aux12 | aux21 | aux22 | aux31 | aux32 | have | been | be | $ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **S** | | | | | | | | | | | | | | |
| **NP** | | | | | | | | | | | | | | |
| **NP1** | | | | | | | | | | | | | | |
| **NP2** | | | | | | | | | | | | | | |
| **NP3** | ε | E | E | ε | E | ε | ε | ε | E | E | E | ε | ε | ε |
| **NP4** | ε | E | E | ε | E | ε | ε | ε | E | E | E | ε | ε | ε |
| **NP5** | | | | | | | | | | | | | | |
| **NP6** | | | | | | | | | | | | | | |
| **NP7** | | | | | | | | | | | | | | |
| **VP** | verb1. VP’ | verb2. VP’ | | | aux11. VP5 | aux12. VP7 | aux21. VP4 | aux22. VP9 | aux31. VP3 | Aux32. VP6 | | | | |
| **VP’** | ε | ε | E | ε | E | ε | ε | ε | E | E | E | E | ε | ε |
| **VP1** | | | | | | | | | | | | | | |
| **VP2** | ε | ε | E | ε | E | ε | ε | ε | E | ε | E | E | ε | ε |
| **VP3** | | | | verb4. VP’ | | | | | | | | | | |
| **VP4** | | | | | aux11. VP7 | | | | | | have. VP8 | | be. VP6 | |
| **VP5** | | | verb3. VP’ | | | | | | | | | been. VP6 | | |
| **VP6** | | | | verb4. VP’ | | | | | | | | | | ε |
| **VP7** | | | verb3.VP’ | | | | | | | | | | | |
| **VP8** | | | | | | | | | | | | been.VP6 | | |
| **VP9** | | | | | | | | | | | | | be.VP6 | |
| **PP** | | | | | | | | | | | | | | |

Table 5: Predictive Parsing table

### 4.4 Parse Tree Generation

A parse tree for a grammar G is a tree where the root is the start symbol for G, the interior nodes are the non terminals of G and the leaf nodes are the terminal symbols of G. The children of a node T (from left to right) correspond to the symbols on the right hand side of some production for T in G. Every terminal string generated by a grammar has a corresponding parse tree and every valid parse tree represents a string generated by the grammar. We store the parse table M using a two-dimensional array. To read an element from a two-dimensional array, we must identify the subscript of the corresponding row and then identify the subscript of the corresponding column.

Example 1: consider the on e English sentence
Ram was going to home.

| Stack | Input | Action |
|---|---|---|
| $ S | Propernoun aux32 verb4 preposition noun$ | |
| $ VP.NP | Propernoun aux32 verb4 preposition noun$ | S→ NP.VP |
| $ VP. NP3.propernoun | Propernoun aux32 verb4 preposition noun$ | NP→propernoun. NP3 |
| $ VP.NP3 | aux32 verb4 preposition noun$ | Popped |
| $ VP.NP3 | aux32 verb4 preposition noun$ | NP3→ε |
| $ VP6.aux32 | aux32 verb4 preposition noun$ | VP→ aux32.VP6 |
| $ VP6 | verb4 preposition noun$ | Popped |
| $ VP'. Verb4 | verb4 preposition noun$ | VP6→ verb4.VP' |
| $ VP' | preposition noun$ | Popped |
| $ NP.PP | preposition noun$ | VP'→PP.NP |
| $ NP. Preposition | preposition noun$ | PP→preposition |
| $ NP. Preposition | preposition noun$ | Popped |
| $ noun | Noun$ | NP→noun |
| $ | $ | Popped |
| $ | $ | Accepted |

Table 6: Moves made by a English parser on input

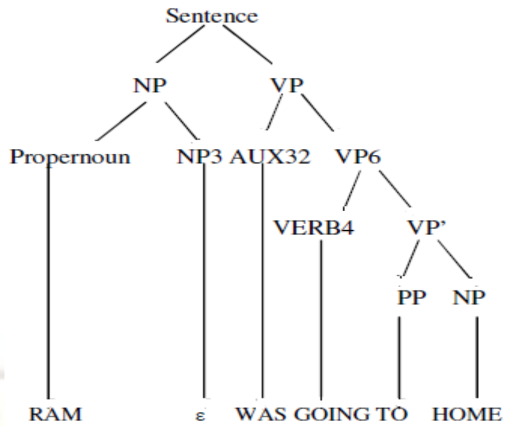


Fig 1. Parse tree for English parser on input *"ram was going to home"*

Example 2:
The old man walking on the road.

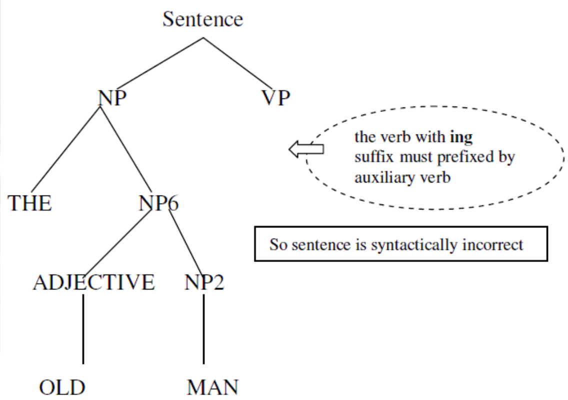


Fig. 2. Parse tree for English parser on input *"the old man walking on the road"*.

## I. EXPERIMENTAL RESULTS

In this Section we show some input sentences that is used for performance analysis. We have used three types of sentences namely simple and regular form, some nontraditional form and Paragraphs. By nontraditional form we mean the same meaning of another sentence having structural similarity.

In this way many example have been checked using this predictive parser. Also some paragraphs have been checked. Both give the results as follows:

| **Sentence type** | **Total no of sentences/ paragraphs. (N)** | **Valid/ invalid (V)** | **Accuracy rate A=(V/N)*100** |
|---|---|---|---|
| Regular | 251 | 197 | 78.48 |
| paragraph | 9 | 6 | 66.67 |

Table 7: Experimental result

## II. 6. CONCLUSION

In this paper we describe a context free grammar for English language and hence we develop a English parser based on that grammar. Our approach is very much general to apply in English Sentences and the method is well accepted for parsing a language of a grammar. The structural representation that has been built can cover the maximum simple, complex and compound sentences. But there are some sentences composed of idioms and phrases are beyond the scope of this paper. Also mixed sentences are of out of the discussion. But further increasing and modifying the production rule it can be possible to remove the above limitations .We believe the proposed method can be applied to check most of the English grammar to parse English language.

## References

**Journal Papers:**

[1] K. M. Azharul Hasan, "Recognizing bangla grammar using predictive parser", IJCSIT, Vol 3, No 6, Dec 2011, pp-61-73.

[2] Amba P. Kulkarni, "Design and Architecture of 'Anusaaraka'- An Approach to Machine Translation", Volume 1, Q4, April, 2003, pp-57-64.

[3] Vishal Goyal, Gurpreet Singh Lehal, Hindi Morphological Analyzer and Generator, 978-0-7695-32677/08©2008*IEEE,DOI10.1109/ICETET.2008*.11 pp-1154-1157.

[4] K. Saravnan, Rajani Parthisarathi, "Syntax parser for Tamil", *tamil internet 2003*, pp-28-37.

**Books:**

[5] By T. Dean, J. Allen, and Y. Aloimonos, *artificial intelligence: theory and practice* [The Benjamin/Cummings Publishing Company, 1995].

[6] A. V. Aho, R. Sethi and J. D. Ullman , *Principles of Compiler Design* [Pearson Education, 2002].

[7] A. Cawsey, *The essence of Artificial Intelligence* [Prentice Hall Europe 1998].