Chapter 5

Marathi Link Grammar Parsing

5.1 Introduction

Parsing of natural language text has been explored extensively since the 1990’s. Most of the early parsers were tried for English or other fixed word order languages. Since the past decades parsing of languages other than English has been taken up by the wider natural language processing research communities. Parsing these morphologically rich, free word order languages is a challenging task. Challenges arises due to the non configurational nature of such languages. Non configurationality leads to the complex and distributed nature of syntactic cues necessary to identify various syntactic relations. The surge of addressing parsing in such languages is discussed in chapter 2. Inspite of this, parsing accuracies for these languages are still less when compared to accuracies of a fixed word order languages like English.

Link Grammar formalism and parsing algorithm based on it have been known to follow dependency structure. Dependency structures are better suited for representing the structure of languages that have a relatively free word order and morphologically rich. Another advantage of dependency structure is that they offer a straightforward interface between syntactic and sematic representation.

In this chapter we explore the parsing approach proposed for the formulated Marathi link grammar discussed in chapter 4. Section 5.1 gives the general idea. Section 5.2
discusses proposed parsing algorithm. Section 5.3 discusses how simple, complex and compound Marathi sentence types are modeled. Section 5.4 presents the experimental setup and results incurred for the samples selected from various domains followed by concluding remarks in Section 5.5.

5.2 The Proposed Parsing Algorithm

Parsing unfolds a linear string of words into a structure which shows explicitly the relation between words, formally known as syntactic analysis. Before syntactic analysis can be performed, input sentences must be preprocessed. Firstly, the units i.e. sentences and/or words need to be identified by segmentation. In order to assign structural descriptions to a sentence and the words in it, it is necessary first to identify these units. In the second place, it is necessary to perform Parts-of-speech (POS) tagging and morphological analysis. In POS tagging, the appropriate word class tag is automatically assigned to each word. Morphological processing deals with the analysis and generation of word forms, analyzing the surface form of a word and producing the output that represents the morphological features of any given word is the purpose of morphological analyzers. Next process is of disambiguation, which selects the correct tag from a set of possible tags or in a large context correct meaning for the particular word(s). Figure 5.1, illustrates the sub processes of syntactic parsing.

Each phase of the syntactic parsing has its own importance and effective processing at each phase complements in overall parsing effectiveness. Indian languages are morphologically rich and free word ordered hence the task of preprocessing becomes even more complicated.

The most complex task that a natural language parser has to perform is syntactic analysis. The two main parts of the syntactic analysis component of a parser are the grammar and the parsing algorithm. The grammar encodes the linguistic rules and specifies how each sentence is constructed from its parts. The parsing algorithm applies the rules defined by a grammar to a given input. The output scheme defines the
The attacker's responsibility is to score goals. The attacker generally restricts his play to the...

5.2.1 The Proposed Algorithm

The proposed algorithm takes a sentence as an input with its morphological information and builds a linkage having each word connected with another word in the Link Grammar Framework as an output.
Algorithm 1 MainAlgorithm(s)

1: Tokenize the Sentences
   //Split the sentence in the individual token
2: Remove the suffixes
   //Seperate any suffixes attached to the tokens
3: Get actual words, POS and Morphological Information
   //Removing suffixes gives Samanya Rupa or Kriyapadas so need to have root forms
4: Get Compound Chunk
   //In the token first priority is given to Compounding
   [List of Compound chunks = Conjunctive coordinator Aani, Va, An, Aanik Dis-
   juructive Coordinator= Kinva, Athva Adversative Coordinator= Pan, Parantu,
   tathapi]
5: if Compound Chunk found then
6:   Call ParseCompound(S)
   //Function for Parsing Compound Sentence
7: else
8:   Get Complex Chunks
   //If not compound, checking for Complex sentence
   [List of Complex Chunks = Hi, He, Ha, Asa, Asa Mhanun, Ase, Ji-Ti, Jya-Tya, Je-
   Te, Jine-Tine, Jyala-Tyala, Jila-Tila, Jo-To, Jevha-Tevha, JoParyant-ToParyant,
   Jethe-Tethe, Jitaki-Titaki, Jashi-Tashi, JyaRitine-TyaRitine, JyaJaagi-TyaJaagi,
   Jari-Turi, Jithe-Tithe, Jar-Tar]
9: if complex Chunks found in s then
10:   Call ParseComplex(S) // Function for parsing sentence
11: else
12:   ParseSimple(S) //if Neither Compound nor complex
13: end if
14: end if
15: Draw LinkageTree(S) // Drawing the Linkage tree received
The sentence which have sentential coordination have been treated as compound sentences and handled as step 1 of the algorithm e.g. consider sentence babu gela aani lili ghari aali. Here two simple sentences are joined with conjunctive coordinator aani to form a compound sentence. Our system finds these various coordinators like conjunctive, disjunctive and adversative coordinators identifying whether coordination is sentential coordination or a word level coordination.

Following is the algorithm proposed to parse the compound sentences,

<table>
<thead>
<tr>
<th>Algorithm 2 ParseCompound(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify the conjunctive style and obtain $s_1$, $s_2$, ..., $s_n$ or $w_1$, $w_2$, ..., $w_n$</td>
</tr>
<tr>
<td>// $s_1$, $s_2$, ..., in case of sentential compound and</td>
</tr>
<tr>
<td>// $w_1$, $w_2$, ..., in case of word level compounds</td>
</tr>
<tr>
<td>if S is sentential compound then</td>
</tr>
<tr>
<td>if $s_1$ compound $s_2$ compound ...$s_n$ then</td>
</tr>
<tr>
<td>//Identify the Type of compounding</td>
</tr>
<tr>
<td>ParseSimple($s_1$), ParseSimple($s_2$), ... ParseSimple($s_n$)</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>for $s_1$, $s_2$, ..., $s_n$ do</td>
</tr>
<tr>
<td>if $s_i$ have Complex Chunks then</td>
</tr>
<tr>
<td>ParseComplex($s_i$)</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>else if s have word level compound then</td>
</tr>
<tr>
<td>Group $w_1$, $w_2$, ..., $w_n$ into W by CompoundWord($w_1$, $w_2$, ..., $w_n$)</td>
</tr>
<tr>
<td>ParseSimple(S)</td>
</tr>
<tr>
<td>//including W as single entity</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>Assign Compound Links</td>
</tr>
</tbody>
</table>

In next phase the algorithm checks for the complex sentence markers. We have seen that complex sentences are of two major type one is complement structure and other is correlative structure. For example consider the following sentence which is of correlative structure which involves pair of relative marker and correlative marker. $Ji$ mulgi ghari geli $Ti$ mulgi dha aahe. Here Ji is the relative marker and Ti is correlative marker followed by simple sentences which then need to be parsed separately, following is the algorithm proposed to parse complex sentences.
Algorithm 3 ParseComplex(S)

Get all the Complex Markers in S
//A complex sentence contains either Header and Complementizer
//or Relative marker and Correlative markers
Identify the Clauses in S and assign as $C_1, C_2, ..., C_n$
// Separate clauses involved in a complex sentences
Identify sentence structure of S
Parse sentence structure using complex sentence rules.
//It is the set of rules from the Data Structure defined for Complex sentences
if Matched then
    ParseSimple($C_1$), ParseSimple($C_2$)... //Parse distinct cluases
end if
Assign Complex Links

Complex algorithm uses the data structure which we have designed specifically to deal with it, following is the structure.

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Rule No</th>
<th>Chunks Count</th>
<th>Condition List</th>
<th>Links List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition 1</td>
<td>Condition Name</td>
<td>Identifiers</td>
<td>Checker Condition</td>
<td>Checkers</td>
</tr>
<tr>
<td>Main clause</td>
<td>Ji/Jar/Jevha</td>
<td>verb form</td>
<td>Yachi/Yacha</td>
<td>Checkers</td>
</tr>
<tr>
<td>Condition 2</td>
<td>Condition Name</td>
<td>Identifiers</td>
<td>ending verb</td>
<td>Checkers</td>
</tr>
</tbody>
</table>

Table 5.1: Data Structure for Complex Sentence

Ultimately whichever may be the sentence type either complex or compound it calls the parsing algorithm used to parse simple sentences.

5.2.2 Explanation

Simple Sentences have been parsed with link satisfaction which either assigned to the words manually or assigned automatically using various approaches. The Karaka links gets established between verbs and other nominal of the sentence to form a meaningful sentence. A linking is specified as Karaka relations depending upon Vibhakti’s (i.e Postposition or suffix) associated with nominals. It depends on verbal semantics and the tense, aspect and modality labels.

Every verb or class of verb have analyzed corresponding to the TAM label and Karaka establishment ensures the vibhakti association of this process, the corresponding link
Algorithm 4 ParseSimple(s)

Get words $w_1, w_2, ..., w_n$ of S // $w_n = Kriyapada$
Identify Named Entity in list of words
Get links from $w_n$ to Named Entity
Get link From $w_n$ to $w_1, w_2, ..., w_{n-1}$
if link found then
    Assign the link //Karkartha links (Links From Kriyapada to nominal(s) in a sentence)
else
    Find links to each word to another word
    find the link from $w_i$ to $w_{i-1}$
    if link found then
        Assign the link // Up-pad sambandha links (Links which are not related with Kriyapada)
    else
        Assign NULL link
    end if
end if

is formed for example Raam aamba khato for verb khane following link Table 5.2 will be retrieved.

<table>
<thead>
<tr>
<th>Link Type</th>
<th>Presence</th>
<th>Vibhakti</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ka_karta</td>
<td>Mandatory</td>
<td>0/ne,</td>
<td>Nominal</td>
</tr>
<tr>
<td>Ka_karma</td>
<td>Mandatory</td>
<td>0/La</td>
<td>Nominal</td>
</tr>
</tbody>
</table>

Table 5.2: Basic Karaka Table for Verb-Noun Agreement

The table above specifies the Vibhakti permitted for the applicable Karaka links for a verb when the verb has the basic TAM label. Correspondingly the POS tag of noun(s) present in the input sentence is drawn and agreement rules gets checked against each pair of appropriate Karaka link for particular pair of Verb-Noun or any other nominal.

Relation that exists between pairs of words of a sentence can signify various functions. Verb-Noun relation, Noun to its postposition are some examples of it. Based on various sentence types dependency labels between chunks has to be classified. We observed two major classification one is at clausal level i.e. main clause, subordinate clause, complement clause, relative clause or correlative clause and another is at chunk level i.e. there are various Up-padSambandha links except Karaka links, which can be established
justifying the relation between nouns to other category chunks except verbs.

The notion of *vibhakti* in Marathi can be captured using this local group chunking. The case markers on nouns and the TAM markers on the verb are lexicalized and appear as separate words. These case and TAM markers play an important role in identifying various links. Local modifications such as adjectives modifying a noun have no effect on the global dependency structure. Therefore, these elements along with function elements can be made part of a chunk. In general, all the nominal inflections, nominal modifications are treated as part of noun chunk, similarly verbal inflections, auxiliaries are treated as part of the verb chunk.

In chapter 4, Section 4.8.1 we have discussed about two level linking scheme. The section explains the purpose behind proposing such scheme, this is captured in Link Grammar Framework by treating clause as a minimal parsing unit. Once minimal parsing unit has been identified we can again divide the linking into chunk level. Treating a clause as a minimal parsing unit leads to two level linking scheme of an input sentence. In first level inter clausal boundaries are extracted and then in second level intra clausal i.e chunk level relations are handled.

It is clear that by treating chunk and clause as minimal parsing units the whole parsing task under Link Grammar Framework for Marathi language divides in layers, wherein specific tasks are subdivided into smaller linguistically motivated subtasks.

1. The first sub task is Part-of-speech tagging and chunking along with morphological analysis which treated as preprocessing step before the task of parsing.

2. Parse a POS tagged and chunked input in two levels. The parser first tries to extract inter clausal links and then intra clausal chunk level linkage is obtained in second level.

3. Finally, accumulated linkage gets identified and displayed as the post-processing to step (1) and (2).
5.3 Modeling Structures of Sentences

Before the meaning of a sentence can be determined, the meaning of its constituent parts must be established. This requires a knowledge of the structure of the sentence, the meaning of the individual words and how the words modifying each other. Structure of sentence plays very important role in parsing, as it involves finding a grammatical sentence structure from an input string. As per scope of our research work we focused on investigating all possible structures falling under simple sentences, complex sentences and compound sentences of Marathi language. We had set a basic model to investigate the structures manually. We allowed to have random sentences of any sentence type and identifying or observing any certain patterns in the structure of these sentences. Surprisingly, although Marathi is a free word order language we found that few constituents appears in some fixed order. We surveyed Marathi linguistic literature in this regard and found that there are various justifications available in this regard [Dhongade and Wali, 2009]. Here we discuss the observed patterns which we treated as modeling of sentences. This modeling of sentence structure will be useful in various research areas like Sentence Boundary Identification, Intelligent Tutoring System to advance learners, Machine Learning etc.

Given input sentence tries to build linkage i.e. each individual token is linked with some or the other way as per grammar with each other. As per Paninian framework link from Kriyapada to other nouns get identified which is known as Karaka relations and other words in chunk gets Uppadsambandh links with that particular nouns. Once the whole bunch of linkage is received the linkage can be observed in the set of structures identified through regression.

Our observation is more the set of structures any language have, more the language is free or flexible.
5.3.1 Simple Sentences

In this section we present the Marathi sentence structures handled in the proposed Marathi Link Grammar Framework.

To give an idea about such structures let us take example of English language simple sentences, which are classified in six basic sentence structures [Quirk 1973, Singh 2003] as follows,

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Sentence Structure</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC</td>
<td>Subject + Verb+ Complement</td>
<td>Monica is beautiful</td>
</tr>
<tr>
<td>SV</td>
<td>Subject + Verb</td>
<td>The egg is boiling</td>
</tr>
<tr>
<td>SVO</td>
<td>Subject + Verb+Object</td>
<td>I accepted the new offer</td>
</tr>
<tr>
<td>SVOO</td>
<td>Subject + Verb+Object+Object</td>
<td>He gave me a book</td>
</tr>
<tr>
<td>SVOC</td>
<td>Subject + Verb+ Object+Complement</td>
<td>The Mps elected him the president</td>
</tr>
<tr>
<td>SVOV</td>
<td>Subject + Verb+ Object+Verb</td>
<td>He helped me write the essay</td>
</tr>
</tbody>
</table>

Table 5.3: English Sentence Structures

Similarly, we have observed many variations in simple sentences, we have used Karaka relations to identify links between words to model Marathi sentences predominantly structures.

Simple Sentence structure 1: The minimum unit of a simple sentence consists a Karta and a Kriyapada, e.g. surya ugavato etc.

\[ \text{Karta} + \text{Kriyapada} \]

Figure 5.3: Simple Sentence Structure 1

Simple Sentence Structure 2: This structure is the most classic structure of Marathi simple sentences. In English it is SVO. Let us see, e.g. raam aamba khato etc.

\[ \text{Karta} + \text{Karma} + \text{Kriyapada} \]

Figure 5.4: Simple Sentence Structure 2
Simple Sentence Structure 3: This is the slight variation in basic unit of Karaka i.e. Karma is placed before Karta. Such structure many a times get jumbled with structure 2, let us see example sentence rajala mukut shobhato राजाला मुक्त शोभतो, In this example to identify Karta simply question is raised upon kriyapada i.e. Shobhanaare kaay? and answer is mukut and not raja. Hence mukut is Karta and raja is a Karma here.

\[
\text{Karma} + \text{Karta} + \text{Kriyapada}
\]

Figure 5.5: Simple Sentence Structure 3

Simple Sentence Structure 4: In this structure Karan Karaka is added e.g. in sentence structure 2 the variation liline surine kanda kapla लिलीने सूरिने कांदा काप्ला is valid so the Karan i.e. the instrument of Kriyapada can be identified

\[
\text{Karta} + \text{Karan} + \text{Karma} + \text{Kriyapada}
\]

Figure 5.6: Simple Sentence Structure 4

Simple Sentence Structure 5: Placing of Karan Karaka is valid in both ways i.e. in above structure 4 it is placed before Karma and in this structure it is placed after Karma e.g. Ramane ravanala banane marale रामाने रावणाला बाणाने मारले

\[
\text{Karta} + \text{Karma} + \text{Karan} + \text{Kriyapada}
\]

Figure 5.7: Simple Sentence Structure 5

Simple Sentence Structure 6: Here Sampradan Karaka is placed between Karta and Karma Karaka e.g. sentence is liline gurujina dakshina dili लिलीने गुरुजिना दक्षिणा दिली.

\[
\text{Karta} + \text{Sampradaan} + \text{Karma} + \text{Kriyapada}
\]

Figure 5.8: Simple Sentence Structure 6
Simple Sentence Structure 7: In this structure Sampradan and Karan Karaka are placed e.g. *liline minila sofarchand chakune kapn dile* लिली मिनीला सफ़रचंद चाकुने कापन दिले.

```
Karta + Sampradan + Karma + Karan + Kriyapada
```

Figure 5.9: Simple Sentence Structure 7

Simple Sentence Structure 8: *Aapadaan Karaka* participates in this structure where an action of separation takes place, e.g. *fule zadavarun padli* फुले झाडावरून पडली.

```
Karta + Aapadan + Kriyapada
```

Figure 5.10: Simple Sentence Structure 8

Simple Sentence Structure 9: This is the variation in above structure where *Karta* follows the *Aapaadaan*, e.g. *Zadavarun fule padli* झाडावरून फुले पडली.

```
Aapadaan + Karta + Kriyapada
```

Figure 5.11: Simple Sentence Structure 9

Simple Sentence Structure 10: *Adhikarna* represents the place or time of the *kri-ayapada* in a sentence e.g. *mini suratla geli* मिनी सूरतला गेली, *babu divasacha zopto* बाबु दिवसाचा झोपतो.

```
Karta + Adhikaran + Kriyapada
```

Figure 5.12: Simple Sentence Structure 10

Simple Sentence Structure 11: This structure gives separation activity and place or time of the activity indicated by the *Kriyapada*, e.g. *lili shaletun ghari aali* लिली शालेतून घरी आली.
Simple Sentence Structure 12: This is the variation of above sentence structure 11, an addition is, *karan Karaka* is also placed in it, e.g. *lili basne shaletun ghari aali* लिली बसने शालेतुन घरी आली.

Simple Sentence Structure 13: This structure is variation above structure 12 the difference is in the placing of the *Karakas*, e.g. *lili shaletun ghari basne aali* लिली शालेतुन घरी बसने आली.

Simple Sentence Structure 14: This structure is variation of structure 8 and 9, *Adhikarna Karaka* is added into the said structure e.g. *zadavarun baget fule padli* शाड़वररु बागेत फुले पडली.

Simple Sentence Structure 15: This structure is variation of structure 10 *Karun Karaka* can be placed after *Adhikarna Karaka*, e.g. *mini suratla basne geli* मिनी सुरतला बसने गेली, *babu shales payi nighala* बाबू शालेस पाई निघाला.
Simple Sentence Structure 16: In this structure Adhikarna is placed before basic unit of sentence structure i.e. Karta and Kriyapada, e.g. eka gavat poor yeto एका गावात पूरे येतो.

Simple Sentence Structure 17: Similar to structure 16, in place of Adhikarna, Sampradan is placed and makes an another variation, e.g. Arogyasathi panyache mahatva ananyasadharan aahe आरोग्यासाठी पण्याचे महत्त्व अनन्यासाधारण आहे.

Simple Sentence Structure 18: This is the variation 12 and 13, the difference in this structure, Aapaadaan Karaka is absent, e.g. lili basne ghari aali लिली बसने घरी आली.

5.3.2 Complex Sentences

The complex sentence structure dealt through Marathi Link Grammar Formalism are discussed below on observing and studying various complex sentences. To understand complex sentences in detail one may refer Section 4.8.2.
Complex Sentence Structure 1: Header and Complementizer words are treated as a starting point for Main clause and Complement clause respectively. This structure is used commonly in Marathi for eg *Hi goshta vichitra aahe ki liline lagna kela* ही गोष्ट विचित्र आहे की लिलीनेन लगन केले. *hi* and *ki* are header and complementizer followed by main clause and complement clause.

![Figure 5.21: Complex Sentence Structure 1](image)

Complex Sentence Structure 2: In this structure complementizer is absent and sentence starts with complement clause followed by header and main clause e.g. *liline lagna kela hi goshta vichitra aahe* लिलीनेन लगन केले ही गोष्ट विचित्र आहे

![Figure 5.22: Complex Sentence Structure 2](image)

Complex Sentence Structure 3: This structure is variation of complex structure 1, here header is absent e.g. *he vichitra aahe ki liline lagna kela* हे विचित्र आहे की लिलीनेन लगन केले.

![Figure 5.23: Complex Sentence Structure 3](image)

Complex Sentence Structure 4: This structure is also known as center embedded complement clause in Marathi linguistics because complement clause separates subject of main clause from the main clause. Often subject is separated by comma punctuation to separate it from following complement clause, e.g. *lilila mini ithe nahi asa vatat* लिलिला मिनी इथे नाही असे वाटते.

105
**Complex Sentence Structure 5**: In Marathi, certain verbs of communication employ pronouns in complement clause to refer to the subject and object of the main clause. Such communicative verbs tend to use quotative word ‘mhanun’. The order in which the clauses occur is shown in the structure. Sentence *lili baralt hoti ki mala maru naka asa mhanun* लिली बरळ्यांना होती कि मला मारू नका असेल is the example of this structure.

**Complex Sentence Structure 6**: Object complement clauses are placed in subject and main clause, complementizer or header word is not used specifically. Sometimes it may lead to treat this sentence as simple sentence. But object complement clauses of verbs like *sang, laav, suru kar, shik* mark the infinitive with dative marker -ay, -la. This markers helps in identifying object complement clause and can be separated from main clause. *liline minila nachayla shikavala lili* लिली मिळेला नाचायला शिकवला.

**Complex Sentence Structure 7**: This structure is quite similar to complement structure in sense that this structure also do not includes complementizer or header word. Here the difference is, nonfinite forms of complex NP’s employ the perfective participle in -l followed by the possessive markers like -ch, -chi, chya.
Complex Sentence Structure 8: This structure is another variation of complex sentence structure 6 and 7. Subject of main clause is placed before complement clause. Header and complementizer words are not used e.g. *lilila babune minishi lagna kelyach aavadala nahi* here *lilila* is subject of main clause *aavadala nahi* and *babune minishi lagna kelyacha* is a verb ending possessive marked complement clause.

Complex Sentence Structure 9: This is another commonly used sentence structure which uses relative and correlative markers followed by relative and correlative clauses. In a basic correlative structure the relative and the main clause are juxtaposed forming a relative-correlative pair e.g. *ji mulgi ghari geli ti dha aahe*.

Complex Sentence Structure 10: This complex sentence structure exhibits one of the pattern of restrictive correlative clauses. Gap relatives are generated by deleting the relative marker along with the nouns, e.g. *ghari geli ti mulgi dha ahe*.
Complex Sentence Structure 11: This structure is postposed relative clause. The only difference is correlative marker and correlative clause respectively placed before relative marker and relative clause. There are certain variation do also exists such as deletion of nouns or placing additional correlative marker at the end etc. e.g. *ti mulgi dha aahe ji mulgi ghari geli* ती मुलगी द आहे जी मुलगी घरी गेली ती

Complex Sentence Structure 12: This kind of structure is known as appositive correlative. Appositive correlative differs from the restrictive correlative in several respect such as in basic appositive structure the relative clause follows the head while in basic restrictive correlative the relative clause in a preceding position etc. Detail study the subrules to get various structure under this sentence structure, e.g. *to maza mitra jo hindi bolto, amerikan aahe* तो माझा मित्र जो हिंदी बोलतो, अमेरिकन आहे

Complex Sentence Structure 13: This is the various of appositive correlative structure where correlative marker is replaced with header *Ha, Hi, He, Asa jo.......*, e.g. *ha maza mitra jo hindi shikvato ek prasidha lekhak aahe* हा माझा मित्र जो हिंदी शिकवतो, एक प्रसिद्ध लेखक आहे
Complex Sentence Structure 14: This structure is variation of structure 12 and 13 of appositive correlation where restrictive correlative may be antecedeced by quantifiers, proper nouns and first or second person pronouns, e.g. *joshi buwa je aamchya kade raahaat far majedaar maanus aahe*.

Complex Sentence Structure 15: Adverbial clauses may precedes or follow the main clause. They maintain the verb final order. In this complex sentence structure adverbial clause is preceding the main clause. The end of adverbial clause present past and conjunctive or completive participle are used e.g. verb+ta *chalata*, verb+ch *kshanich*, etc. e.g. *tu gelyapasun mala chain padat nahi*.

Complex Sentence Structure 16: This structure is variation of sentence structure 15, where conjunctive particles *karan, mhanun, sabab, hetu*, are placed as the middle term in main clause and adverbial clause, e.g. *lili ushira aali karan (ka / ka ki) ti kamat hoti*.
**Complex Sentence Structure 17**: This is the another of complex structure 16 only difference is adverbial clause precedes conjunctive particle and main clause respectively, e.g. *lili kamat hoti mhanun ushira aali* लिली कामात होती म्हणून उशिरा आली

![Complex Sentence Structure 17](image)

**5.3.3 Compound Sentences**

The details about compound sentences are discussed in Section 4.4.3. In following section we have discussed about the structures that are ensured through the Marathi Link Grammar Parser.

**Compound Sentence Structure 1**: When two separate sentences are compounded through conjunctive coordinator *aani, va, an, anik* it achieves the sentential coordination, e.g. *babu gela aani lili ghari aali* बबू गेला आणि लिली घरी आली The structure is as follows,

![Compound Sentence Structure 1](image)

**Compound Sentence Structure 2**: The structure is about use of disjunctive coordinator. There are two disjunctive coordinator *kinva* and *athava* in sentential coordination e.g. *lili ghari geli asel kiwa baget basali asel* लिली घरी गेली असेल किवा बागेत बसली असेल. Disjunctive coordination structure is shown as below

![Compound Sentence Structure 2](image)
Compound Sentence Structure 3: There is another type of coordinator in use called as adversative coordinator. *pan, parantu, tadhapi* are the three adversative coordinator used alternatively, e.g. *lili hushaar aahe pan abhyas karat nahi* लिली हुशार आहे पण अभ्यास करत नाही. Following is the structure.

![Figure 5.40: Compound Sentence Structure 3](image)

Compound Sentence Structure 4: This structure is word level structure. It is a constitution coordination, nouns of all categories may be coordinated. It follows agreement rules for coordination. We observed that subject are coordinated with conjunctive coordinator, e.g. *sudha aani mini gharat hotye* सुधा आणि मिनी घरात होत्या.

![Figure 5.41: Compound Sentence Structure 4](image)

Compound Sentence Structure 5: This structure is slight variation of above structure 4 i.e. in this structure conjunctive coordinator is replaced with disjunctive coordinator kinva, e.g. *lili kiwa babu ghari jail* लिली किवा बाबू घरी जैल.

![Figure 5.42: Compound Sentence Structure 5](image)

Compound Sentence Structure 6: This structure is another variation of constituent variation. Here the constituent is object. An objects are coordinated with conjunctive coordinator, e.g. *lili ne aambe aani keli khaale* लिली ने आंबे आणि केली खाले.
**Figure 5.43: Compound Sentence Structure 6**

| Object 1 | + | Conjunctive coordinator | + | Object 2 |

**Compound Sentence Structure 7:** In this structure two objects are coordinated with disjunctive coordinator, forming the structure as follows, e.g. *lili pen kiwa pensil aanel* लिली पेन किवा पेनसील आणेल.

| Object 1 | + | Disjunctive coordinator | + | Object 2 |

**Figure 5.44: Compound Sentence Structure 7**

**Compound Sentence Structure 8:** Similarly as subject and objects are coordinated, adjectives are also coordinated with conjunctive coordinator, e.g. *lili jara baawali aani vedi aahe* लिली जरा बावळी आणि वेडी आहे.

| Adjective 1 | + | Conjunctive coordinator | + | Adjective 2 |

**Figure 5.45: Compound Sentence Structure 8**

**Compound Sentence Structure 9:** Adjectives when coordinated with disjunctive coordinator, e.g. *te biskit god kiwa kharat asu shakate* ते बिस्किट गोड किवा खारट अशा शकते we get following structure,

| Adjective 1 | + | Disjunctive coordinator | + | Adjective 2 |

**Figure 5.46: Compound Sentence Structure 9**

**Compound Sentence Structure 10:** This structure is another Adverb constituent is also coordinated with conjunctive coordinator e.g. *lili halu halu aani mand swarat bolte* लिली हाळ हाळ आणि मंद स्वरात बोलते as follows,

| Adverb 1 | + | Conjunctive coordinator | + | Adverb 2 |

**Figure 5.47: Compound Sentence Structure 10**
Compound Sentence Structure 11: Adverb constituent with disjunctive coordinator is also possible structure e.g. *lilin hihi hale kiwa jorane bolayala have* लिलिहेन हैली हैली किवा जोरानें बोलायला हवे which is as follows,

\[
\text{Adverb 1} \quad + \quad \text{Disjunctive coordinator} \quad + \quad \text{Adverb 2}
\]

Figure 5.48: Compound Sentence Structure 11

We have observed that adversative coordinator *pan, parantu, tathapi* are not in the use for constituent level coordination.

5.4 Experimental Setup and Results

In this section we present illustration of sentences parsed through the proposed system and the empirical results. The proposed algorithm works for many variations of simple, complex and compound sentences. The structure modeled through our system is discussed in previous Section 5.2. Few complex sentence structures make parsing quite difficult because of their complex nature. One may study the linking patterns of possible sentence structure of Marathi sentences or English sentences having similar structure to develop rules so that parsing of such sentences can be possible.

For parsing purpose by random selection, first, sentences from each type of structures were selected to parse. Around 150 such sentences were parsed to check its possibility, correctness and for identifying loopholes in the proposed parsing system. We have handled agreement rules necessary to parse but still some structures need more language specific information.

For further evaluation we have taken paragraphs from three different plain corpus i.e. from Newspaper article, a story and a paragraph from text book chapters.
### 5.4.1 Illustration

In this section we illustrate the working of proposed parsing algorithm with some examples. We present verity of sentences signifying the different aspects that can be handled by the parsing algorithm, we have selected sentences from experimental samples only to illustrate the working nature of it. Stepwise parsing output is discussed as follows.

**Illustration 1 : ekada eka mandirachya pujarachya gavaat poor yeto**

(Simple Sentence)

**Step 1 -** Tokenize the sentence.

Each word gets separated and gets tokens as $t_0$-ekada, $t_1$-eka, $t_2$-mandirachya, $t_3$-pujarachya, $t_4$-gavaat, $t_5$-poor, $t_6$-yeto

**Step 2 -** Remove the suffixes.

Each token is scanned and checked for suffixes attached with it. If found then separate it. Here $t_2$-mandirachya, $t_3$-pujarachya do have suffix Chya which will be separated.

**Step 3 -** Get Saamanya Rupa, POS and Morphological information of tokens under process. We get Table 5.4,

<table>
<thead>
<tr>
<th>Token</th>
<th>Word</th>
<th>Saamanya rupa</th>
<th>Prtyaya</th>
<th>POS TAG</th>
<th>Morphological Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>ekada</td>
<td>ekada</td>
<td>QT</td>
<td>QTO</td>
<td><code>&lt;fs af='1.ekada,unk,,,,,'/&gt;</code></td>
</tr>
<tr>
<td>$t_1$</td>
<td>eka</td>
<td>eka</td>
<td>QT</td>
<td>QTQTC</td>
<td><code>&lt;fs af='ek,n,m,sg,o,-A,-A'&gt;'</code></td>
</tr>
<tr>
<td>$t_2$</td>
<td>mandirachya</td>
<td>mandir</td>
<td>chya</td>
<td>N,NN</td>
<td><code>&lt;fs af='mandir,n,n,sg,o,Chya,A,Chya,'&gt;'</code></td>
</tr>
<tr>
<td>$t_3$</td>
<td>pujyryachya</td>
<td>pujari</td>
<td>chya</td>
<td>N,NN</td>
<td><code>&lt;fs af='pujari,n,m,sg,o,Chya,A,Chya'&gt;</code></td>
</tr>
<tr>
<td>$t_4$</td>
<td>gavat</td>
<td>gaav</td>
<td>At</td>
<td>N,NN</td>
<td><code>&lt;fs af='gaav,n,n,sg,o,A,A,T'&gt;</code></td>
</tr>
<tr>
<td>$t_5$</td>
<td>poor</td>
<td>poor</td>
<td>N</td>
<td>NN</td>
<td><code>&lt;fs af='poor,n,m,sg,d,,'/&gt;</code></td>
</tr>
<tr>
<td>$t_6$</td>
<td>yeto</td>
<td>yene</td>
<td>V</td>
<td>VM</td>
<td><code>&lt;fs af='ye,v,m,sg,3,,To,To' t='pre' a='h' type='ak'&gt;</code></td>
</tr>
</tbody>
</table>

Table 5.4: Illustration of Sentence 1

**Step 4 -** Checking tokens for the Compound chunk. Here compound chunk/word not found so go to next step.
Step 5 - Now Checking tokens for Complex chunk. Here Complex chunk not found so go to next step.

Step 6 - Neither Compound chunk nor Complex chunk is present hence the tokens are given as input to ParseSimple().

Step 7 - The ParseSimple Procedure identifies verb or verb form at token $t_6$. From $t_6$ program proceeds from right to left. Takes a pair of tokens and checks for link. If link found, builds the link else checks the next token. Building link has to be proceed by underlying rules written for each type of link.

Linkage - The parsed output generated through the parsing system is shown in Figure 5.49,

![Figure 5.49: Parsed Output 1](image.png)

Illustration 2: jevha te tyala aaplyabarobar gayla sangatat tevha to nakarato

(Complex Sentence)

Step 1 - Tokenize the sentence.

Each word gets separated and gets tokens as $t_0$-jevha, $t_1$-te, $t_2$-tyala, $t_3$-aaplyabarobar, $t_4$-gayla, $t_5$-sangatat, $t_6$-tevha, $t_7$-to, $t_8$-nakarato.

Step 2 - Remove the suffixes.

Each token is scanned and checked for suffixes attached with it. If found then separate it. Here $t_3$-aaplyya, $t_{31}$-barobar is received.
**Step 3** - Get Saamanya Rupa, POS and Morphological information of tokens under process. We get Table 5.5,

<table>
<thead>
<tr>
<th>Token</th>
<th>Word</th>
<th>Saamanya rupa</th>
<th>Prtyaya</th>
<th>POS TAG</th>
<th>Morphological Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀</td>
<td>jevha</td>
<td>jevha</td>
<td>PR_PRP</td>
<td></td>
<td>&lt;fs af='jevha,unk,...'/&gt;</td>
</tr>
<tr>
<td>t₁</td>
<td>te</td>
<td>te</td>
<td>PR_PRP</td>
<td></td>
<td>&lt;fs af='to,pn,m,pl,d,,Te'/&gt;</td>
</tr>
<tr>
<td>t₂</td>
<td>tyala</td>
<td>to</td>
<td>Lya</td>
<td>PR_PRP</td>
<td>&lt;fs af='mandir,n,n,sg,_,o,Chya,A,Chya'/&gt;</td>
</tr>
<tr>
<td>t₃</td>
<td>apalyabarobar</td>
<td>apala</td>
<td>Barobar</td>
<td>PR_PRP</td>
<td>&lt;fs af='apala,pn,m,sg,_,o,Barobar,Ya,Barobar'/&gt;</td>
</tr>
<tr>
<td>t₄</td>
<td>yayla</td>
<td>yene</td>
<td>PR_PRP</td>
<td></td>
<td>&lt;fs af='ye,v,m,sg,d,la,la,type=kr'/&gt;</td>
</tr>
<tr>
<td>t₅</td>
<td>sangatat</td>
<td>sangane</td>
<td>V_VM</td>
<td></td>
<td>&lt;fs af='sang,v,m,pl,3,,tat,tat' t='pre' a='h' type=ak'/&gt;</td>
</tr>
<tr>
<td>t₆</td>
<td>tevha</td>
<td>tevha</td>
<td>PR_PRP</td>
<td></td>
<td>&lt;fs af='tevha,avy,...'/&gt;</td>
</tr>
<tr>
<td>t₇</td>
<td>to</td>
<td>to</td>
<td>PR_PRP</td>
<td></td>
<td>&lt;fs af='to,pn,m,sg,d,'/&gt;</td>
</tr>
<tr>
<td>t₈</td>
<td>nakarato</td>
<td>nakarane</td>
<td>V_VM</td>
<td></td>
<td>&lt;fs af='nakar,v,m,sg,3,,to,to' t='pre' a='h' type=ak'/&gt;</td>
</tr>
</tbody>
</table>

Table 5.5: Illustration of Sentence 2

**Step 4** - Checking tokens for the Compound chunk. Here compound chunk/word not found so go to next step.

**Step 5** - Now Checking tokens for Complex chunk. Here Complex chunk found at t₀-jevha and t₆-tevha.

Checks token t₆-₁ = t₅ for verb, gets t₅=sangane.

separates clauses gets C₁ = t₁...t₅ and C₂ = t₇...t₈

Then C₁ and C₂ is given as a input to t₀ ParseSimple().

**Step 6** - The ParseSimple builds links as explained in illustration one and upper level links will be assigned on t₀ - C₁ - t₆ - C₂

**Linkage** - The parsed output generated through the parsing system is shown in Figure 5.50,
Figure 5.50: Parsed Output 2

Illustration 3: to tyana sangato ki tyacha tyachya devavar vishwas aahe aani dev tyach nakki rakshan karel

(Complex + Compound Sentence)

Step 1 - Tokenize the sentence.

Each word gets separated and gets tokens as \(t_0\)-to, \(t_1\)-tyana, \(t_2\)-sangato, \(t_3\)-ki, \(t_4\)-tyacha, \(t_5\)-tyachya, \(t_6\)-devavar, \(t_7\)-vishwas, \(t_8\)-aahe, \(t_9\)-aani, \(t_{10}\)-dev, \(t_{11}\)-tyach, \(t_{12}\)-nakki, \(t_{13}\)-rakshan, \(t_{14}\)-karel.

Step 2 - Remove the suffixes.

Each token is scanned and checked for suffixes attached with it. If found then separate it. Here \(t_6\)-deva, \(t_{61}\)-var is received.

Step 3 - Get Saamanya Rupa, POS and Morphological information of tokens under process. We get Table 5.6.

Step 4 - Checking tokens for the Compound chunk. Here compound chunk/word found at \(t_9\)-aani.

Step 5 - Now Checking tokens for Complex chunk. Here Complex chunk found at \(t_3\)-ki (Complementizer)
Table 5.6: Illustration of Sentence 3

Step 6 - separates clauses as follows

\[
C_1 = t_0 \ldots t_2 \\
C_2 = t_4 \ldots t_8 \\
C_3 = t_{10} \ldots t_{14}
\]

Then \(C_1\), \(C_2\) and \(C_3\) is given as a input to ParseSimple().

Step 7 - The ParseSimple builds inner links : \(C_2 - t_9\) - \(C_3\) and will get upper level links as contains compound links.

The ParseSimple now builds inner links of \(C_1\) and will get upper level links as contains complex chunk.

Linkage - The parsed output generated through the parsing system is shown in Figure 5.51,
Illustration 4: thodya velane ek hodi yete pan to tyathi basat nahi

(Compound Sentence)

Step 1 - Tokenize the sentence.
Each word gets separated and gets tokens as $t_0$-thodya, $t_1$-velane, $t_2$-ek, $t_3$-hodi, $t_4$-yete, $t_5$-pan, $t_6$-to, $t_7$-tyathi, $t_8$-basat, $t_9$-nahi.

Step 2 - Remove the suffixes.
Each token is scanned and checked for suffixes attached with it. If found then separate it. Here $t_7$-tyat, $t_{71}$-hi is received.

Step 3 - Get Saamanya Rupa, POS and Morphological information of tokens under process. We get Table 5.7,

Step 4 - Checking tokens for the Compound chunk. Here compound chunk/word found at $t_5$-pan.

Step 5 - Now Checking tokens for Complex chunk. Here Complex chunk not found.
Table 5.7: Illustration of Sentence 4

Step 6 - Separates clauses as follows

\[ C_1 = t_0 \ldots t_4 \]
\[ C_2 = t_6 \ldots t_9 \]

Step 7 - The ParseSimple builds inner links: \( C_1 \) and \( C_2 \) and will get upper level links: \( C_1 - t_5 - C_2 \) as contains compound links.

Linkage - The parsed output generated through the parsing system is shown in Figure 5.52,

![Figure 5.52: Parsed Output 4](image-url)
Illustration 5: *shevati poorach pani vadhata aani tyach ghar budata va to marto*  
(Compound Sentence)

**Step 1** - Tokenize the sentence.
Each word gets separated and gets tokens as $t_0$-shevati, $t_1$-poorach, $t_2$-pani, $t_3$-vadhata, $t_4$-aani, $t_5$-tyach, $t_6$-ghar, $t_7$-budata, $t_8$-va, $t_9$-to, $t_{10}$-marto.

**Step 2** - Remove the suffixes.
Each token is scanned and checked for suffixes attached with it. If found then separate it. No separate pratyayas found

**Step 3** - Get Saamanya Rupa, POS and Morphological information of tokens under process. We get Table 5.8,

<table>
<thead>
<tr>
<th>Token</th>
<th>Word</th>
<th>Saamanya rupa</th>
<th>Prtyaya</th>
<th>POS TAG</th>
<th>Morphological Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>shevati</td>
<td>shevat</td>
<td>N,NN</td>
<td>&lt;fs af='thodya,unk,...,'/&gt;</td>
<td></td>
</tr>
<tr>
<td>$t_1$</td>
<td>purach</td>
<td>poor</td>
<td>Ch,NN</td>
<td>&lt;fs af='shevati,unk,...,'/&gt;</td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>paani</td>
<td>paani</td>
<td>N,NN</td>
<td>&lt;fs af='poor,n,n,sg,o,Ch,A,Ch'&gt;</td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td>vadhat</td>
<td>vadhat</td>
<td>V,VM</td>
<td>&lt;fs af='paani,n,n,sg,d,'/&gt;</td>
<td></td>
</tr>
<tr>
<td>$t_4$</td>
<td>aani</td>
<td>aani</td>
<td>CC,CCD</td>
<td>&lt;fs af='aani,avy,...,'/&gt;</td>
<td></td>
</tr>
<tr>
<td>$t_5$</td>
<td>tyach</td>
<td>to</td>
<td>Ch,PRP</td>
<td>&lt;fs af='to,pn,m,sg,o,Ch,Ya,Ch'&gt;</td>
<td></td>
</tr>
<tr>
<td>$t_6$</td>
<td>ghar</td>
<td>ghar</td>
<td>N,NN</td>
<td>&lt;fs af='ghar,n,n,sg,d,'/&gt;</td>
<td></td>
</tr>
<tr>
<td>$t_7$</td>
<td>budat</td>
<td>budane</td>
<td>V,VM</td>
<td>&lt;fs af='vud,v,n,sg,3,T,T' t='pre' a='h' type='ak'/&gt;</td>
<td></td>
</tr>
<tr>
<td>$t_8$</td>
<td>va</td>
<td>va</td>
<td>CC,CCD</td>
<td>&lt;fs af='va,avy,...,'/&gt;</td>
<td></td>
</tr>
<tr>
<td>$t_9$</td>
<td>to</td>
<td>to</td>
<td>PR,PRP</td>
<td>&lt;fs af='to,pn,m,sg,d,'/&gt;</td>
<td></td>
</tr>
<tr>
<td>$t_{10}$</td>
<td>marato</td>
<td>marane</td>
<td>V,VM</td>
<td>&lt;fs af='mar,v,m,sg,1,To,To' t='pre' a='h' type='ak'/&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.8: Illustration of Sentence 5

**Step 4** - Checking tokens for the Compound chunk. Here compound chunk/word found at $t_4$-aani and $t_8$-va.

**Step 5** - Now Checking tokens for Complex chunk. Here Complex chunk not found.

**Step 6** - Separates clauses as follows

\[
C_1 = t_0\ldots t_3 \\
C_2 = t_5\ldots t_7 \\
C_3 = t_9\ldots t_{10}
\]
Step 7 - The ParseSimple builds inner links of : C1, C2 and C3 and will get upper level links : C1 - t4 - C2 - t8 - C3 as contains compound links.

Linkage - The parsed output generated through the parsing system is shown in Figure 5.53,

Figure 5.53: Parsed Output 5

Illustration 6: *Panyane mendutalyla peshina chalana milate tyamule sharirache karya vyavasthitpane chalate*

(Complex Sentence)

Step 1 - Tokenize the sentence.

Each word gets separated and gets tokens as t0- Panyane, t1- mendutalyla, t2- peshina, t3- chalana, t4- milate, t5- tyamule, t6- sharirache, t7- karya, t8- vyavasthitpane, t9- chalate.

Step 2 - Remove the suffixes.

Each token is scanned and checked for suffixes attached with it. If found then separate it. Here t0- paanya, t071- ne, t1- menduta, t11- lya, t2- peshi, t21- na, t3- tyamule, t51- mule, t6- sharira, t61- che, t8- vyavasthit, t81- pane is received.
Step 3 - Get Saamanya Rupa, POS and Morphological information of tokens under process. We get Table 5.9,

<table>
<thead>
<tr>
<th>Token</th>
<th>Word</th>
<th>Saamanya rupa</th>
<th>Prtyaya</th>
<th>POS TAG</th>
<th>Morphological Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀</td>
<td>panyane</td>
<td>paani</td>
<td>Ne</td>
<td>N_NN</td>
<td>&lt;fs af='paani,n,n,sg,o,Ne,,,'/&gt;</td>
</tr>
<tr>
<td>t₁</td>
<td>mendutalya</td>
<td>mendu</td>
<td>Talya</td>
<td>N_NN</td>
<td>&lt;fs af='mendu,n,m,sg,o,Lya,T_Lya'/&gt;</td>
</tr>
<tr>
<td>t₂</td>
<td>peshina</td>
<td>peshi</td>
<td>Na</td>
<td>N_NN</td>
<td>&lt;fs af='peshi,n,f,pl,o,Na,A_Na'/&gt;</td>
</tr>
<tr>
<td>t₃</td>
<td>chalana</td>
<td>chalana</td>
<td>N_NN</td>
<td>&lt;fs af='chalana,n,f,sg,d,/'&gt;</td>
<td></td>
</tr>
<tr>
<td>t₄</td>
<td>milate</td>
<td>milane</td>
<td>V_VM</td>
<td>&lt;fs af='mil,v,f,sg,1,,te,te' t='pre' a='h' type='ak'/&gt;</td>
<td></td>
</tr>
<tr>
<td>t₅</td>
<td>tyamule</td>
<td>tyamule</td>
<td>CC_CCD</td>
<td>&lt;fs af='tyamule,avy,,,,,'/&gt;</td>
<td></td>
</tr>
<tr>
<td>t₆</td>
<td>sharirache</td>
<td>sharir</td>
<td>Che</td>
<td>N_NN</td>
<td>&lt;fs af='sharir,n,n,sg,o,che,A_che'/&gt;</td>
</tr>
<tr>
<td>t₇</td>
<td>karya</td>
<td>karya</td>
<td>N_NN</td>
<td>&lt;fs af='karya,n,n,sg,d,,'/&gt;</td>
<td></td>
</tr>
<tr>
<td>t₈</td>
<td>vyavasthitpane</td>
<td>vyavasthit</td>
<td>Pane</td>
<td>&lt;fs af='vyavasthitpane,adv,,,,,'/&gt;</td>
<td></td>
</tr>
<tr>
<td>t₉</td>
<td>chalate</td>
<td>chalane</td>
<td>V_VM</td>
<td>&lt;fs af='chal,v,n,sg,3,te,te' t='pre' a='h' type='ak'/&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.9: Illustration of Sentence 6

Step 4 - Checking tokens for the Compound chunk. Compound chunk/word not found.

Step 5 - Now Checking tokens for Complex chunk. Here Complex chunk found at t₅-tyamule which is a marker for adverbial clause along with t₅₋₁ = t₄ = Verb.

Step 6 - Separates clauses as follows

\[ C₁ = t₀,\ldots,t₄ \]
\[ C₂ = t₆,\ldots,t₉ \]

Step 7 - The ParseSimple builds inner links of : \( C₁ \) and \( C₂ \) and will get upper level links : \( C₁ - t₅ - C₂ \)

Linkage - The parsed output generated through the parsing system is shown in Figure 5.54,

Illustration 7: divasbhar kabadkashta kelyane aapan thakun jato

(Complex Sentence)
Figure 5.54: Parsed Output 6

**Step 1** - Tokenize the sentence.

Each word gets separated and gets tokens as $t_0$-divasbhar, $t_1$-kabadkashta, $t_2$-kelyane, $t_3$-aapan, $t_4$-thakun, $t_5$-jato.

**Step 2** - Remove the suffixes.

Each token is scanned and checked for suffixes attached with it. If found then separate it. Here $t_0$-divasbhar, $t_0$-bhar, $t_2$-kelyane, $t_2$-ne is received.

**Step 3** - Get Saamanya Rupa, POS and Morphological information of tokens under process. We get Table 5.10,

<table>
<thead>
<tr>
<th>Token</th>
<th>Word</th>
<th>Saamanya rupa</th>
<th>Prtyaya</th>
<th>POS TAG</th>
<th>Morphological Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>diwasbhar</td>
<td>diwas</td>
<td>bhar</td>
<td>N,NN</td>
<td>$&lt;fs\ af='diwasbhar,unk,.....,'&gt;$</td>
</tr>
<tr>
<td>$t_1$</td>
<td>kabadkashta</td>
<td>kabadkashta</td>
<td></td>
<td>N,NN</td>
<td>$&lt;fs\ af='kabadkashta,n,n,sg,d,,'&gt;$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>kelyane</td>
<td>karnae</td>
<td>Lyane</td>
<td>V,VM</td>
<td>$&lt;fs\ af='kar,v,m,sg,1,<em>,Lyae,Ne,</em>,Lyae,Ne'\ t='pas'\ a='h'\ type='ak'&gt;$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>apan</td>
<td>apan</td>
<td></td>
<td>PR,PRP</td>
<td>$&lt;fs\ af='apan,pm,sg,d,,'&gt;$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>thakun</td>
<td>thakun</td>
<td></td>
<td>V,VM</td>
<td>$&lt;fs\ af='thak,k,_,Un,Un'\ type='kr'&gt;$</td>
</tr>
<tr>
<td>$t_5$</td>
<td>jato</td>
<td>jato</td>
<td></td>
<td>VAUX</td>
<td>$&lt;fs\ af='ja,v,m,sg,1,_,to,to'\ t='pre'\ a='h'\ type='ak'&gt;$</td>
</tr>
</tbody>
</table>

Table 5.10: Illustration of Sentence 7

**Step 4** - Checking tokens for the Compound chunk. Compound chunk/word not found.
Step 5 - Now Checking tokens for Complex chunk. Here Complex chunk found at $t_2$-kelyane which is a marker for adverbial clause along with $t_2 = \text{Verb}$.

Step 6 - separates clauses as follows

$C_1 = t_0....t_2$

$C_2 = t_3....t_5$

Step 7 - The ParseSimple builds inner links of : $C_1$ and $C_2$ and will get upper level links : $C_1 \rightarrow C_2$

Linkage - The parsed output generated through the parsing system is shown in Figure 5.55,

Illustration 8: *divasbhar aakashat bharkatnaare pakshi ratri aaplya ghartyat zop ghetat* (Simple Sentence)

Step 1 - Tokenize the sentence.

Each word gets separated and gets tokens as $t_0$-divasbhar, $t_1$-aakashat, $t_2$-bharkatnaare, $t_3$-pakshi, $t_4$-ratri, $t_5$-aaplya, $t_6$-ghartyat, $t_7$-zop, $t_8$-ghetat.

Step 2 - Remove the suffixes.

Each token is scanned and checked for suffixes attached with it. If found
then separate it. Here $t_0$-divasbhar, $t_0$-bhar, $t_2$-bharkat, $t_{21}$-naare, $t_6$-gharty, $t_{61}$-at is received.

**Step 3** - Get Saamanya Rupa, POS and Morphological information of tokens under process. We get Table 5.11,

<table>
<thead>
<tr>
<th>Token</th>
<th>Word</th>
<th>Saamanya rupa</th>
<th>Prtyaya</th>
<th>POS TAG</th>
<th>Morphological Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>diwashcar</td>
<td>diwas</td>
<td>bhar</td>
<td>N,NN</td>
<td>&lt;fs af='diwashcar,unk,...'&gt;</td>
</tr>
<tr>
<td>$t_1$</td>
<td>aakashat</td>
<td>aakash</td>
<td>At</td>
<td>N,NN</td>
<td>&lt;fs af='aakash,n,n,sg,o,t,A,T'&gt;</td>
</tr>
<tr>
<td>$t_2$</td>
<td>bharkatnare</td>
<td>bharkatne</td>
<td>Nare</td>
<td>JJ</td>
<td>&lt;fs af='bharkat,v,m,pl,d,naare,naare' type='tn'&gt;</td>
</tr>
<tr>
<td>$t_3$</td>
<td>pakshi</td>
<td>pakshi</td>
<td></td>
<td>N,NN</td>
<td>&lt;fs af='pakshi,n,m,pl,d,,' &gt;</td>
</tr>
<tr>
<td>$t_4$</td>
<td>ratri</td>
<td>ratra</td>
<td></td>
<td>N,NN</td>
<td>&lt;fs af='mil,v,f,sg,1,te,te t='pre' ratra,n,f,sg,o, I,I'&gt;</td>
</tr>
<tr>
<td>$t_5$</td>
<td>aplya</td>
<td>apla</td>
<td></td>
<td>PR,PRP</td>
<td>&lt;fs af='apla,adj,m,,o,Ya'&gt;</td>
</tr>
<tr>
<td>$t_6$</td>
<td>ghartyat</td>
<td>gharte</td>
<td>At</td>
<td>N,NN</td>
<td>&lt;fs af='gharte,n,n,sg,o,t,Ya,T'&gt;</td>
</tr>
<tr>
<td>$t_7$</td>
<td>zop</td>
<td>zop</td>
<td></td>
<td>N,NN</td>
<td>&lt;fs af='zop,n,f,sg,d,' &gt;</td>
</tr>
<tr>
<td>$t_8$</td>
<td>ghetat</td>
<td>ghetat</td>
<td>Pane</td>
<td>N,NN</td>
<td>&lt;fs af='ghe,v,m,pl,3,tat,tat' t='pre' a='h' type='ak'&gt;</td>
</tr>
</tbody>
</table>

Table 5.11: Illustration of Sentence 8

**Step 4** - Checking tokens for the Compound chunk. Compound chunk/word not found.

**Step 5** - Now Checking tokens for Complex chunk. No Compound chunk found.

**Step 6** - Neither Compound chunk nor Complex chunk is present hence the tokens are given as input to ParseSimple().

**Step 7** - The ParseSimple Procedure identifies verb or verb form at token $t_8$.

From $t_8$ program proceeds from right to left.

**Linkage** - The parsed output generated through the parsing system is shown in Figure 5.56,
Illustration 9: *sarvach sajiv prani saynakalnantar vishranti ghetat va ratri zoptat*  
(Compound Sentence)

**Step 1** - Tokenize the sentence.

Each word gets separated and gets tokens as $t_0$-sarvach, $t_1$-sajiv, $t_2$-prani, $t_3$-saynakalnantar, $t_4$-vishranti, $t_5$-ghetat, $t_6$-va, $t_7$-ratri, $t_8$-zoptat.

**Step 2** - Remove the suffixes.

Each token is scanned and checked for suffixes attached with it. If found then separate it. Here $t_0$-sarva, $t_0$-ch, $t_3$-saynakal, $t_{31}$-nantar, is received.

**Step 3** - Get Saamanya Rupa, POS and Morphological information of tokens under process. We get Table 5.12,

**Step 4** - Checking tokens for the Compound chunk. Compound chunk/word found at $t_6$-va.

**Step 5** - Now Checking tokens for Complex chunk. No Complex chunk found.

**Step 6** - Separates clauses as follows

$C_1 = t_0,...,t_5$

$C_2 = t_7,...,t_8$

**Step 7** - The ParseSimple builds inner links of: $C_1$ and $C_2$ and will get upper level links: $C_1 - t_6 - C_2$
<table>
<thead>
<tr>
<th>Token</th>
<th>Word</th>
<th>Saamanya rupa</th>
<th>Prtyaya</th>
<th>POS TAG</th>
<th>Morphological Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀</td>
<td>sarvach</td>
<td>sarva</td>
<td>Ch</td>
<td>QT_QTF</td>
<td><code>&lt;fs af='diwashhar,unk,,,,'&gt;</code></td>
</tr>
<tr>
<td>t₁</td>
<td>sajiv</td>
<td>sajiv</td>
<td>JJ</td>
<td></td>
<td><code>&lt;fs af='aakash,n,n.sg,o,t,A_T'&gt;</code></td>
</tr>
<tr>
<td>t₂</td>
<td>prani</td>
<td>prani</td>
<td>N_NN</td>
<td></td>
<td><code>&lt;fs af='bharkat,v,m.pl,d,nare,nare' type='tn'&gt;</code></td>
</tr>
<tr>
<td>t₃</td>
<td>sayankalantar</td>
<td>saynakal</td>
<td>Nantar</td>
<td>V_VM</td>
<td><code>&lt;fs af='fs af='pakshi,n,m.pl,d,'</code></td>
</tr>
<tr>
<td>t₄</td>
<td>vishranti</td>
<td>vishranti</td>
<td>N_NN</td>
<td></td>
<td><code>&lt;fs af='mil,v,sg,1,te,te' t='pre' ratra,n,f.sg., o,I,I'</code></td>
</tr>
<tr>
<td>t₅</td>
<td>ghetat</td>
<td>ghene</td>
<td>V_VM</td>
<td></td>
<td><code>&lt;fs af='apla,adj,m,,,o,,Ya'</code></td>
</tr>
<tr>
<td>t₆</td>
<td>va</td>
<td>va</td>
<td>C_CCD</td>
<td></td>
<td><code>&lt;fs af='gharte,n,n.sg,o,t,Ya_T'</code></td>
</tr>
<tr>
<td>t₇</td>
<td>ratri</td>
<td>ratra</td>
<td>N_NN</td>
<td></td>
<td><code>&lt;fs af='ghe,v,m,pl,3,,tat,tat' </code>pre' a='h' type='ak'&gt;`</td>
</tr>
<tr>
<td>t₈</td>
<td>zoptat</td>
<td>zopane</td>
<td>N_NN</td>
<td></td>
<td><code>&lt;fs af='ghe,v,m,pl,3,,tat,tat' </code>pre' a='h' type='ak'&gt;`</td>
</tr>
</tbody>
</table>

Table 5.12: Illustration of Sentence 9

**Linkage** - The parsed output generated through the parsing system is shown in Figure 5.57,

**Illustration 10**: *mala vatate hi goshta tumhala changali thaaak asali pahije*  
(Complex Sentence)

**Step 1** - Tokenize the sentence.

Each word gets separated and gets tokens as t₀-*mala*, t₁-*vatate*, t₂-*hi*, t₃-*
goshta, t₄-tumhala, t₅-changali, t₆-thauk, t₇-asali, t₈-pahije.

Step 2 - Remove the suffixes.

Each token is scanned and checked for suffixes attached with it. If found then separate it. Here t₄-tumha, t₄₁-la, is received.

Step 3 - Get Saamanya Rupa, POS and Morphological information of tokens under process. We get Table 5.13,

<table>
<thead>
<tr>
<th>Token</th>
<th>Word</th>
<th>Saamanya rupa</th>
<th>Prtyaya</th>
<th>POS TAG</th>
<th>Morphological Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₀</td>
<td>mala</td>
<td>mi</td>
<td>PR</td>
<td>PRP</td>
<td>&lt;fs af=’mala,unk,,,,,’&gt;</td>
</tr>
<tr>
<td>t₁</td>
<td>vatate</td>
<td>vatane</td>
<td>V</td>
<td>VM</td>
<td>&lt;fs af=’vaat,v,f,sg,1,,te,te’ t=’pre’ a=’hi’ type=’ak’&gt;</td>
</tr>
<tr>
<td>t₂</td>
<td>hi</td>
<td>hi</td>
<td>DM</td>
<td>DMD</td>
<td>&lt;fs af=’hi,unk,,,,,’&gt;</td>
</tr>
<tr>
<td>t₃</td>
<td>goshta</td>
<td>goshta</td>
<td>Nantar</td>
<td>N,NN</td>
<td>&lt;fs af=’gosht,n,f,sg,d,’ &gt;</td>
</tr>
<tr>
<td>t₄</td>
<td>tumhala</td>
<td>tumhala</td>
<td>La</td>
<td>PR_PRP</td>
<td>&lt;fs af=’tu,pp,,pl,,o,La,tumha_La’&gt;</td>
</tr>
<tr>
<td>t₅</td>
<td>changali</td>
<td>changli</td>
<td>JJ</td>
<td></td>
<td>&lt;fs af=’changala,adj,f,sg,d,,l’&gt;</td>
</tr>
<tr>
<td>t₆</td>
<td>thank</td>
<td>thank</td>
<td>JJ</td>
<td></td>
<td>&lt;fs af=’thauk,n,,,,,’&gt;</td>
</tr>
<tr>
<td>t₇</td>
<td>asli</td>
<td>asane</td>
<td>V</td>
<td>VM</td>
<td>&lt;fs af=’as,v,f,sg,3,,li,li’ t=’pas’ a=’p’ type=’ak’&gt;</td>
</tr>
<tr>
<td>t₈</td>
<td>pahije</td>
<td>pahije</td>
<td>VAUX</td>
<td></td>
<td>&lt;fs af=’pahije,v,s,sg,1,,E,E’ type=’ak’&gt;</td>
</tr>
</tbody>
</table>

Table 5.13: Illustration of Sentence 10

Step 4 - Checking tokens for the Compound chunk. Compound chunk/word not found.

Step 5 - Now Checking tokens for Complex chunk. Complex chunk found at t₂-hi (header).

Step 6 - separates clauses as follows

C₁ = t₀....t₁
C₂ = t₃....t₈s

Step 7 - The ParseSimple builds inner links of: C₁ and C₂ and will get upper level links: C₁ - - t₂ - C₂

Linkage - The parsed output generated through the parsing system shown in Figure 5.58,
5.4.2 Experimental Results

In order to investigate our proposed link grammar based parser for Marathi language first, we formulated Marathi Link Grammar then we modeled structures possible in Marathi Simple, Complex and Compound types of sentences. Implementation of proposed algorithm presented herein is done on PHP and is tested on system running Windows 7 equipped with 4GB RAM and Third Generation Intel core i3-311CM processor. In this section we present experimental results and discuss various issues observed while working on the parsing scheme.

Test Run 1: At first test run, we tested the algorithm on approximately 150 Marathi sentences of type Simple, Complex and Compound taken randomly from the book, Marathi [Dhongade and Wali, 2009]. As we do not have any link grammar parsed output for Marathi, these sentences were parsed manually. Then they are parsed through our system.
Table 5.14: Summary of Test Run 1

The summary of test results for 50 such sentences is as shown in Table 5.14. Sentences which contains repetition of structures have not considered for evaluation purpose. Value 0 indicates that Partially correct parsed and 1 indicates completely correct parsed. In column Sentence structure mapped S - Simple, CX - Complex and CO- Compound.

We have observed following

- At test run 1, correctly parsed sentences were 39, most of the incorrect parses accounted in complex sentences. For example in some complex sentences chunks like complementizer, header etc. were absent thus assigned wrong links to the word pair. Latter it was corrected by introducing a special character with a heuristic in such cases.

- Parsing of sentences involving new words took comparatively more time. Dictionary look up obviously took more time for words which were either not present or entered on when such new word encountered.

- For some simple sentences identification of compound verbs for eg “radat aahe”, “piiun takale” etc. was unconvincing. More verb morphological information can be used to improve the performance in such cases.

**Test Run 2 :** For test run 2 we have chosen paragraphs as a sample from three distinct types of documents listed as below,

1. Story
2. News Article
3. Text book Chapter
The observations are as follows:

Table 5.15 shows the observation as sample 1 i.e. story, Table 5.16 shows the observation of sample2 i.e. news article and Table 5.17 of sample 3 i.e text book chapter.

<table>
<thead>
<tr>
<th>Particular</th>
<th>Total no of words</th>
<th>Type of sentence</th>
<th>Structure/ Rule</th>
<th>Parse</th>
<th>Not Parsed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Complete</td>
<td>Partial</td>
</tr>
<tr>
<td>S1</td>
<td>7</td>
<td>Simple</td>
<td>SI15</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S2</td>
<td>6</td>
<td>Simple</td>
<td>SI2</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>S3</td>
<td>10</td>
<td>Complex</td>
<td>CX9</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S4</td>
<td>15</td>
<td>Complex + Compound</td>
<td>CX3, CO1</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>S5</td>
<td>8</td>
<td>Compound</td>
<td>CO1</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S6</td>
<td>10</td>
<td>Simple</td>
<td>SI9</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S7</td>
<td>12</td>
<td>Compound</td>
<td>CO3</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S8</td>
<td>11</td>
<td>Compound</td>
<td>CO3</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S9</td>
<td>13</td>
<td>Compound</td>
<td>CO1, CO3</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>S10</td>
<td>11</td>
<td>Compound</td>
<td>CO1</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S11</td>
<td>15</td>
<td>Compound</td>
<td>CO3</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S12</td>
<td>10</td>
<td>Simple</td>
<td>SI6</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S13</td>
<td>8</td>
<td>Simple</td>
<td>SI9</td>
<td>Y</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.15: Sample 1 Story

<table>
<thead>
<tr>
<th>Particular</th>
<th>Total no of words</th>
<th>Type of sentence</th>
<th>Structure/ Rule</th>
<th>Parse</th>
<th>Not Parsed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Complete</td>
<td>Partial</td>
</tr>
<tr>
<td>S1</td>
<td>6</td>
<td>Complex</td>
<td>CX2</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S2</td>
<td>5</td>
<td>Simple</td>
<td>SI6</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S3</td>
<td>5</td>
<td>Simple</td>
<td>SI3</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S4</td>
<td>8</td>
<td>Simple</td>
<td>SI7</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>S5</td>
<td>6</td>
<td>Simple</td>
<td>SI15</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S6</td>
<td>17</td>
<td>Compound</td>
<td>-</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>S7</td>
<td>23</td>
<td>Complex + Compound</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S8</td>
<td>10</td>
<td>Complex</td>
<td>CX15</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S9</td>
<td>7</td>
<td>Simple</td>
<td>SI6</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S10</td>
<td>7</td>
<td>Compound</td>
<td>CO1</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S11</td>
<td>7</td>
<td>Simple</td>
<td>SI7</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>S12</td>
<td>7</td>
<td>Simple</td>
<td>SI8</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>S13</td>
<td>8</td>
<td>Simple</td>
<td>SI15</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S14</td>
<td>4</td>
<td>Simple</td>
<td>SI6</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S15</td>
<td>7</td>
<td>Simple</td>
<td>SI5</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S16</td>
<td>13</td>
<td>Complex</td>
<td>CX15</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S17</td>
<td>8</td>
<td>Simple</td>
<td>SI9</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S18</td>
<td>13</td>
<td>Simple</td>
<td>-</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>S19</td>
<td>7</td>
<td>Simple</td>
<td>SI20</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S20</td>
<td>14</td>
<td>Complex</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>S21</td>
<td>5</td>
<td>Simple</td>
<td>SI1</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>S22</td>
<td>17</td>
<td>Complex</td>
<td>CX15</td>
<td>-</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 5.16: Sample 2 News
Table 5.17: Sample 3 Text Book Chapter

Table 5.18 shows the summary of results incurred on parsing the samples.

Table 5.18: Empirical Result Summary

The graphical representation gives more clear idea as shown in following Figure 5.59.

Figure 5.59: Result Summary

From these statistics, the overall performance of the samples in specific is about 78.96%, which we feel is quite encouraging. As we look upon the percentage of partially parsed and not parsed sentences which is 25.36% and 11.36% respec-
tively, we have done following error analysis,

**Partially parsed sentences -**

- Simple sentences consisting compound verbs. It needs more morphological information to improve the performance of such sentences. At present, they are linked under *Uppadsambandha* links.

- Some structures like involving complex+compound type of sentences need careful alignment. At present parser checks compound chunks first and then goes for complex chunks at such case parse may need one more level of linking i.e. third level which may result in further complication.

- Cardinal numbers mostly appear as an adjective in a sentence, which are not handled presently but can be adapted in system with less efforts, e.g. *ssat te aath glass paani* etc.

**Not parsed sentences-**

- Interrogative sentences, negative sentences as well as Imperative sentences are not in the scope and hence not handled at present which accounts to non parsing of sentences.

Further we use Precision, Recall, F-Measure to evaluate performance of our system. F-Measure is harmonic mean of Precision and Recall. These values are calculated as follows based on [Jurafsky and Martin, 2000].

\[
\text{Precision}(p) = \frac{\text{# of Correct Answer given by system}}{\text{# of Answers given by System}} \quad (5.1)
\]

\[
\text{Recall}(R) = \frac{\text{# of Correct Answers given by system}}{\text{Total # of Possible Correct Answers in Text}} \quad (5.2)
\]

\[
F - \text{Measure} = \frac{(\beta^2 + 1)P \times R}{\beta^2 \times P + R} \quad (5.3)
\]
Using this three formulas we calculated the Precision, Recall and F-Measure values for samples under consideration which is given in Table 5.19 below and followed by graphical representation in Figure 5.60.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Precision (P)</th>
<th>Recall (R)</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample 1</td>
<td>76.92%</td>
<td>76.92%</td>
<td>76.00%</td>
</tr>
<tr>
<td>Sample 2</td>
<td>70.00%</td>
<td>63.63%</td>
<td>66.10%</td>
</tr>
<tr>
<td>Sample 3</td>
<td>77.77%</td>
<td>58.33%</td>
<td>63.54%</td>
</tr>
<tr>
<td>Average</td>
<td>74.89%</td>
<td>66.29%</td>
<td>68.54%</td>
</tr>
</tbody>
</table>

Table 5.19: P, R and F-Measure Summary

The average Precision is 74.89%, average Recall is 66.29% and average F-Measure is 68.54%. Hence one can conclude that the sentence structure mapped through the system contributes positively to improve parsing results of free word order languages. In our system it contributes to have good Precision score, certainly recall can be improved in due course of time as it will learn more words and phenomenon of language under consideration.

5.5 Concluding Remarks

In this chapter we presented algorithms for parsing Marathi Link Grammar. Typically the link grammar is introduced for fixed word language like English. The primary
objective of our work was to formulate Link Grammar for Marathi language which is free word order in nature and to parse it through the parsing algorithm proposed specifically for it. Algorithm have been developed to parse an input sentence using the knowledge learned and presented through our system. In order to make it more efficient we have studied as many possible sentences and modeled it through our system. Our system can map total 46 structures comprising simple, complex and compound types of sentences. We observe that more the set of structures any language have, more the language is free in nature.

we have presented step by step illustration of some of the sample sentences parsed by our proposed parsing system to give more idea of it. We have presented experimental results performed in two test runs with the error analysis. Performance of the samples parsed is evaluated in the form of Precision, Recall and F-Measure score. Precision score results in encouraging score i.e. 74.89%. We have observed that more the sentence structure mapped through the parsing system, results in good precision score.