Submitted by

M.Durga Prasad

The English and Foreign Languages university,

Hyderabad-500007

CONTENTS

TOPICS PAGE NO.

1 .ABSTRACT ……..... 6

2. INTRODUCTION…….....7

3. OVERVIEW OF TELUGU LANGUAGE…….8

3.1 NOUNS…….9

3.2VERBS……10

- FINITE VERBS…….10
- NON FINITE VERBS……13

4. INTRODUCTION TO MORPHOLOGICAL ANALYZER …….14

5. WHY PERL FOR LINGUISTICS.....27

6. METHODOLOGIES……28

7. CODING……..32

8.SAMPLE INPUT SPLIT OUTPUT SCREEN SHORTS

8. LIMITATIONS…….……

9. CONCLUSION…….……

10 REFERENCES…….…….....

1. INTRODUCTION

Natural language processing (NLP) is a subfield of artificial intelligence and linguistics. It studies the problems of automated generation and understanding of natural human languages. Natural language generation systems convert information from computer databases into normal-sounding human language, and natural language understanding systems convert samples of human language into more formal representations that are easier for computer programs to manipulate.

Telugu is a highly inflectional and agglutinative language providing one of the richest and challenging set of linguistic and statically features. For the purpose of analysis of such inflectionally rich languages, the root and the morphemes of each word have to be identified. There are few languages in the world that matches Telugu in this regard. Telugu is having large number of morphological variants for a given root. It is a free word order language (SOV word order). Each word in Telugu is inflected for hundreds of word forms. In Telugu each inflected word starts with root and is having so many suffixes. The word suffix is used here is to refers to inflections, post-positions and markers, indicating tense, number, person and gender, negatives, imperatives. These suffixes are affixed with each root word to generate word forms.

The term morphology comes from classical Greek (morpheme) and means the study of shape or form. In linguistics, it is concerned with the structure and arrangement of morphemes of a word, and how these morphemes are fit together to create a complete or meaningful word. The process of segmenting words into morphemes and analyzing word formation is called morphological analysis. It is a primary step for various types of text analysis of any language.

2. OVERVIEW OF TELUGU LANGUAGE

Historically Telugu Language is also known by the names, amdhram, tenu (m) gu and gen too .

2.1. Demographic Information :

Telugu is one of the major Scheduled Languages in India. It is the Second most popular language in India. Its speakers are mainly concentrated in South India. It is the official language of Andhra Pradesh and Secondly widely spoken language

in Tamilnadu, Karnataka. Considerable numbers of Telugu speaking minorities live in Maharashtra, Orissa, Madhya Pradesh and West Bengal. Considerable numbers of Telugu language speakers have migrated to Mauritius, South Africa and recently to U.S.A, UK, and Australia.

2.2.Generic affiliation and History:

Telugu belongs to South Central branch of Dravidian family of languages. It is the most widely spoken Dravidian language. It is the only literary language outside the South-Dravidian branch. Its literature goes back to 11th century A.D. Its ancient forms were attested through inscriptions dating back to 200A.D.

2.3.Telugu Alphabet

The primary units of Telugu [17] alphabet are syllables; therefore, it should be rightly called a syllabary and most appropriately a mixed alphabetic-syllabic script. Unlike in the Roman alphabet used for English, in the Telugu alphabet the correspondence between the symbols (graphemes) and sounds (phonemes) is more or less exact. However, there exist some differences between the alphabet and the phonemic inventory of Telugu. The overall pattern consists of 60 vowels, 3 vowel modifiers and 41 consonants.

Notable features

- Type of writing system**:** syllabic alphabet in which all consonants have an inherent vowel. Diacritics, which can appear above, below, before or after the consonant they belong to, are used to change the inherent vowel.
- When they appear at the beginning of a syllable, vowels are written as independent letters.
- When certain consonants occur together, special conjunct symbols are used which combine the essential parts of each letter.
- Direction of writing: left to right in horizontal lines.

3. Computational Grammar of Telugu

In Telugu, Morphology plays a crucial role in not only generating numerous word forms from nouns and verbs but determining their shapes as well. As head of Noun phrases, nouns carry distinct morphological inflections indicating various syntactic and semantic functions expressed in proposition. Word Order, unlike

English, does not determine the syntactic relations between a noun and its governing category verb.

3.1. Nouns

A noun in Telugu is inflected in a complex way. Nouns in Telugu characteristically carry the markings of gender, number, person and case.

A number of nouns in Telugu often change their form before the marking of gender, number, and person and case. Systematic changes occur in the base particularly when inflected for non-nominative cases such as accusative, dative, instrumental, ablative and locative. Conventionally noun-nominative base of a noun is also known as oblique base or oblique form. However, it should be noted that such a base is neither unique nor common.

- Gender marking on noun Though the inflections classes are insensitive to gender distinctions, there are distinctions of gender discernible from morphology of agreement on verbs, adjectives, possessives, predicate nominal, numerals and deictic categories. It is necessary to identify four distinctions in gender, viz. nouns indicating:

1. Human males
2. Other than human males, in singular and plural, nouns indicating
3. Humans, and
4. Non-humans.

This distinct is necessitated by the distribution of nouns indicating human females which are grouped with neuter nouns in singular, but human males in plural. However, a number of nouns denoting human males end in –du, and human females end in –di.

Telugu uses a wide variety of case markers and post-positions and their combinations to indicate various relations between nouns and verbs or nouns. Case suffixes and post-positions fall into two types viz. „Grammatical‟ and „Semantic or location and directional‟. Grammatical case suffixes are those which express grammatical case relations such as nominative, accusative, dative, instrumental, genitive, comitative, vocative and causal. The semantic cases include such as nouns inflected for location in time and space. Nouns when attached with various combinations of adverbial nouns and case markers or post-positions express many more such relations.

3.2. Verbs

Verb denotes the state of or action by a substance. Telugu verb may be finite or non-finite. All finite verbs and some non-finite verbs can occur according to situation before the utterance final juncture /#/ characterized by of following terminal contours: rising pitch, meaning question; level pitch, falling pitch, meaning command. A finite verb does not occur before any of the non-final junctures. On the morphological level, no non- finite verb contains a morpheme indicating person; this statement should not, however, be taken to mean that all finite verbs necessarily contain a morpheme indicating person. Since any verb, finite or non-finite, occurs only after some marked juncture, by definition of these junctures, all verbs have phonetic stress or prominence on their first syllable, which invariably part of the root.

Almost every Telugu verb has a Finite and a non- finite form. A finite form is one that can stand as the main verb of a sentence and occur before a final pause (full stop). A non- finite form cannot stand as a main verb and rarely occurs before a final pause.

- **FINITE VERBS**

The eight finite forms of the modern Telugu verb may be arranged in three structural types, which are set up according to the differences in the grouping of the three substitution classes,

- Stem or inflection root
- Tense-mode suffix
- Personal suffix(es)

The paradigms of the finite forms of a simple verbal base are given below under the three structural types: ammu (to sell), with two allomorphs: amm- before a vowel.

Type 1: stem + personal suffix:

1. Imperative : singular –u amm-u (sell)

Plural - andi ammu - andi

Type 2: stem + tense-mode suffix:

2. Admonitive or abusive:

On account of semantic restrictions, many verbs cannot occur in this mood. A few bases like kAlu (to burn), kUlu (to fall), cAvu (to die), pagulu (to break), etc., occur

Eg: nIyilli kUlu - may your house fall

3. Obligative (in all persons): -Ali

amma – Ali I, we, you(sg, pl)

he, she, it

Type 3: stem + tense-mode suffix + personal suffix

4. Habitual- future or non-past: -t-

ammu – t - Anu I shall sell

ammu – t – Am we shall sell

ammu – t – Ava you shall sell

ammu – t – Aru he shall sell

ammu – t – Adu she shall sell

ammu – tun – di she sell

ammu – t – Ay they sell

-i-

ammu – i – Anu* I sold

ammu – i – Am we sold

ammu – i – Ava you sold (Singular)

ammu – i – Aru you sold (plural)

ammu – i – Adu he sold

ammu – in – di she/ it sold

ammu – i – Aru they sold

-d-

ammu – d – Am let us sell, or we shall sell

7. Negative tense:-a-

ammu – a – nu I (do, did, and shall) not sell

ammu – a –m we(do, did, and shall) not sell

ammu – a –va you (do, did, and shall) not sell

ammu – a – Du he(does, did, and shall) not sell

ammu – a – du she/ it(do, did, and shall) not sell

ammu – a – ru they (do, did, and shall) not sell

8. Negative imperative or prohibitive: -Ak-

ammu – Ak – a you(sg.) don‟t sell

ammu – Ak – andi you(pl.) don‟t sell

NON-FINITE VERBS

There are ten non- finite verbs which may be arranged into two structural types:

- Unbound
- Bound

Type 1:

1.	Present participle	-tu	ammu- tU	Selling
2.	Past participle	-i	ammu- i	having sold
3.	Concessive	-inA	ammu- inA	even though sold
4.	Conditional	-itE	ammu- itE	if sold
5.	Infinitive	-a	ammu- a	to sell
6.	Negative participle	-aka	amm-aka	not selling
7.	Habitual adjective	-E	amm-E	that sells
8.	Past adjective	-ina	amma-ina	that sold
9.	Negative adjective	-ani	ammu- ani	not selling

Type 2:

Bound present - t- : ammu- t – occurs with any finite form of the verb un- to be and also a few non- finite forms.

Eg: ammu- t- unnAnu	I am selling
ammu- t- un- nA	even selling(now)
ammu- t- un- tE	if selling
ammu- t- un- na	that selling

4. INTRODUCTION TO MORPHOLOGICAL ANALYZER

4.1 Introduction to Morphology:

Morphology is the study of the way words are built up from smaller meaning-bearing units, morphemes. A morpheme is often defined as the minimal meaning-bearing unit in a language. Words are the building blocks of the grammar of a natural language. The part of the computational grammar that deals with words is called morphology. Morphology is the branch of computational linguistics that studies the internal structure of words.

It is often useful to distinguish two broad classes of morphemes:

1. Stems

2. Affixes

The stem is the "main" morpheme of the word, supplying the main meaning, while the affixes add "additional" meaning of various kinds.

Affixes are further divided into

Affixes are further divided into

- Prefixes
- Suffixes
- Infixes
- Circumfixes.

Prefixes precede the stem, suffixes follow the stem, circumfixes do both, and infixes are inserted inside the stem. Prefixes and suffixes are often called concatenative morphology since a word is composed of a number of morphemes concatenated together. A number of languages have extensive non-concatenative morphology, in which morphemes are combined in more complex ways. Another kind of non-concatenative morphology is called templatic morphology or root- and pattern morphology.

A **stem** is the root or roots of a word, together with any derivational affixes, to which inflectional affixes are added. For example, tie and untie both are stem, to which inflectional -s may be added to the stem to form ties and unties.

Compounding is the formation of new words, which is made by combining two or more words. Each unit that combines in compounding is a lexeme in itself.

Examples: black-bird, fire-fighter, hard-hat, water-hose, gardenhose, rubber- hose, fire-hose.

Morphemes express concepts or relationships, which could be meaningful full words or could be sequence of character/s which is/are not meaningful until joined with another morpheme or word.

Examples: car, table, anti-, re-, -s, -ing are morphemes.

Morphemes convey information about syntactic features,
like:-Number (singular, plural),

Tense (present, past, future), Gender (masculine, feminine).

In word 'flower', the single morpheme 'flower' is recognized as the morph 'flower' to make the word 'flower'. However in word 'flowers', the morpheme 'flower' and the plural morpheme are recognized as 'flower' and '-s' respectively, to form the word 'flowers'.

Allomorphs are the different forms of the same morpheme.

Example, the plural morpheme in English has several allomorphs, i.e., -es, -s, e-deletion, germination, etc.

Free morphemes are those that can stand on their own as individual words, Example: book, knock, soft.

to same noun 'flower'. The words 'run', 'runs', 'running' refer to same verb 'run'. Thus 'flower' and 'run' are lexemes. Same free morphemes are categories as different lexeme, because lexeme refers to a single grammatical category, while morpheme is not associated with grammatical category. This means that the word 'print' is a free morpheme, which results in two lexemes, one as a noun and other as a verb.

4.2 Types of morphology

There are two broad classes of ways to form words from morphemes:

- Inflection
- derivation

Inflection is the combination of a word stem with a grammatical morpheme, usually resulting in a word of the same class as the original stem, and usually filling some syntactic function like agreement.

Derivation is the combination of a word stem with a grammatical morpheme, usually resulting in a word of a *different class*, often with a meaning hard to predict exactly.

Inflection morphology is the process of adding inflectional morphemes to a word. The inflectional morpheme adds some type of grammatical information, i.e., case, number, person, gender, mood, mode, tense, aspect, etc. Inflectional morphology doesn't change grammatical category of the word and thus the inflected words refers to same lexeme.

Derivational morphology, in contrast, adds derivational morphemes, which create a new word from an existing word, sometimes by simply changing grammatical category, i.e., changing a noun to a verb. Derivational morphology is thus the creation of new words out of other words and morphemes. The new word formed often belongs to a different part of speech, but not always.

Examples of Derivation morphology:

The words 'possible', 'possibly', 'impossible' are made by using derivational morphology. Similarly 'happy' and 'happiness', 'inform', 'informer' and 'information' are the words with derivational morphology.

Root, the lexical content morpheme having no affix, is the one that is not analyzed into smaller meaningful parts . Root is common to set of derived or inflected forms, when all of the affixes are removed. Root morpheme carries the main fraction of meaning, e.g., in words: disestablish, establishment, establishments. The word 'establish' is a root to which various derivational and inflection morphemes are attached.

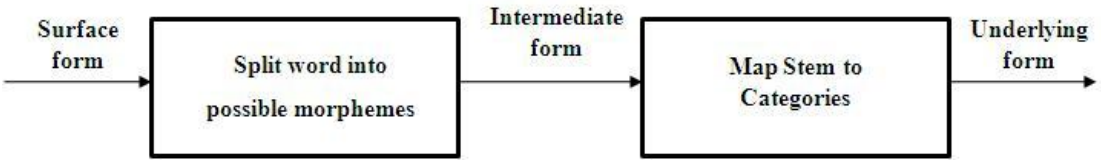
4.3 Morphological Analyzer

In the normal morphological analyzer [26] or generator there are actually 3 components:

1. **Lexicon**: The list of stems and affixes, together with basic information about then (whether a stem is a Noun stem or a Verb stem, etc.)
2. **Morph tactics**: The model of morpheme ordering that explains classes of morphemes can follow other classes of morphemes inside a word. For example, the rule that the English plural follows the noun rather than preceding it.

Orthographic Rules: these spellings rules are used to model the changes that occur in a word, usually when two morphemes combine

.

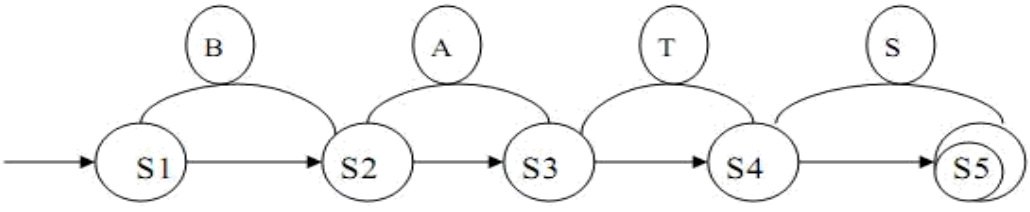


Two steps of morphological parser

4.3.1 Various methods of Morphological Analyzer

Various NLP research groups have developed different methods and algorithm for morphological analysis. Some of the algorithms are language dependent and some of them are language independent. A brief survey of various methods involved in Morphological Analysis includes the following:

- Finite State Automata (FSA)
- Two Level Morphology
- Finite State Trancendures (FST)
- Stemmer Algorithm
- Corpus Based Approach
- DAWG (Directed Acrylic Word Graph)



A finite state machine or finite automation is a model of behavior composes of state, transitions and actions. A finite state automaton is a device that can be in one of a

finite number of states. If the automation is in a final state, when it stops working, it is said to accept its input. The input is a sequence of symbols

FSA is used to accept or reject a string in a given language. It uses regular expressions. When the automaton is switched on it will be in the initial stage and start working. In the final state it will accept or reject the given string. In between the initial state and finite state there are transition a process of switching over to another state. Regular expressions are powerful tools text searching. FSA is used to represent morphological lexicon and recognition.

The KIMMO PARSER

The first implementation of a two-level system by Koskenniemi was in Pascal. Doing a Lisp implementation of the two-level model would be a good approach and Karttunen et.al completed the project and published a collection of papers on the topic, along with Lisp code (Karttunen 1983; Gajek et al. 1983). They called it the KIMMO system and it inspired many other KIMMO implementations. The most popular of these is PC-KIMMO, a free C implementation from the Summer Institute of Linguistics (Antworth 1990).

The KIMMO parser had two analytical components: the rules component and the lexical component, or lexicon. First, the rules component consisted of two-level rules that accounted for regular phonological or orthographic alternations, such as *chase* versus *chas*. Second, the lexicon listed all morphemes (stems and affixes) in their lexical form and specified morph tactic constraints. The Generator would accept as input a lexical form such as *spy+s* and return the surface form *spies*. Recognizer would accept as input a surface form such as *spies* and return an underlying form divided into morphemes, namely *spy+s*, plus a gloss string such as

N+PLURAL.

In Europe, two-level morphological analyzers became a standard component in several large systems for natural language processing such as the British Alvey project, SRI's CLE Core Language Engine, the ALEP Natural Language Engineering Platform and the MULTEXT project.

In his dissertation Koskenniemi [20] introduced a formalism for two-level rules. The semantics of two-level rules was well-defined but there was no rule compiler available at the time. Koskenniemi and other early practitioners of two-level morphology constructed their rule automata by hand. This is tedious in the extreme and very difficult for all but very simple rules.The first two-level rule compiler was written in InterLisp by Koskenniemi and Lauri Karttunen in 1985-87 using Kaplan's implementation of the finite-state calculus (Koskenniemi 1986; Karttunen, Koskenniemi, and Kaplan 1987). The current C version two-level compiler, called TWOLC, was created at PARC (Karttunen and Beesley 1992). It has extensive systems for helping the linguist to avoid and resolve rule conflicts, the bane of all large-scale two-level descriptions.

The PC-KIMMO parser

In 1990, the Summer Institute of Linguistics produced PC-KIMMO [23] version 1, an implementation of the two-level model that closely followed Karttunen's KIMMO. Written in C, it ran on personal computers such as IBM PC compatibles and the Macintosh as well as UNIX. PC-KIMMO was quite good at what it was designed to do-tokenize a word into a sequence of tagged morphemes. But it had a serious deficiency: it could not directly determine the part of speech of a word or its inflectional categories.

For example, given the word *enlargements*, PC-KIMMO could tokenize it into the sequence of morphemes *en+large+ment+s* and gloss each morpheme, but it could not determine that the entire word was a plural noun. This meant that PC-KIMMO was not adequate to act as a morphological front end to a syntactic parser- its most desirable application.

It is of interest to computational linguists, descriptive linguists, and those developing natural language processing systems. The program is designed to generate (produce) and/or recognize (parse) words using a two-level model of word structure in which a word is represented as a correspondence between its lexical level form and its surface level form.

The PC-KIMMO [23] program is actually a shell program that serves as an interactive user interface to the primitive PC-KIMMO functions. These functions are available as a C-language source code library that can be included in a program written by the user.

A PC-KIMMO description of a language consists of two files provided by the user:

(1) a rules file, which specifies the alphabet and the phonological (or spelling) rules, and

(2) a lexicon file, which lists lexical items (words and morphemes) and their glosses, and encodes morphotactic constraints.

The theoretical model of phonology embodied in PC-KIMMO is called two-level phonology. In the two-level approach, phonology is treated as the correspondence between the lexical level of underlying representation of words and their realization on the surface level. For example, to account for the rules of English spelling, the surface form *spies* must be related to its lexical form *`spy+s* as follows (where ` indicates stress, + indicates a morpheme boundary, and 0 indicates a null element):

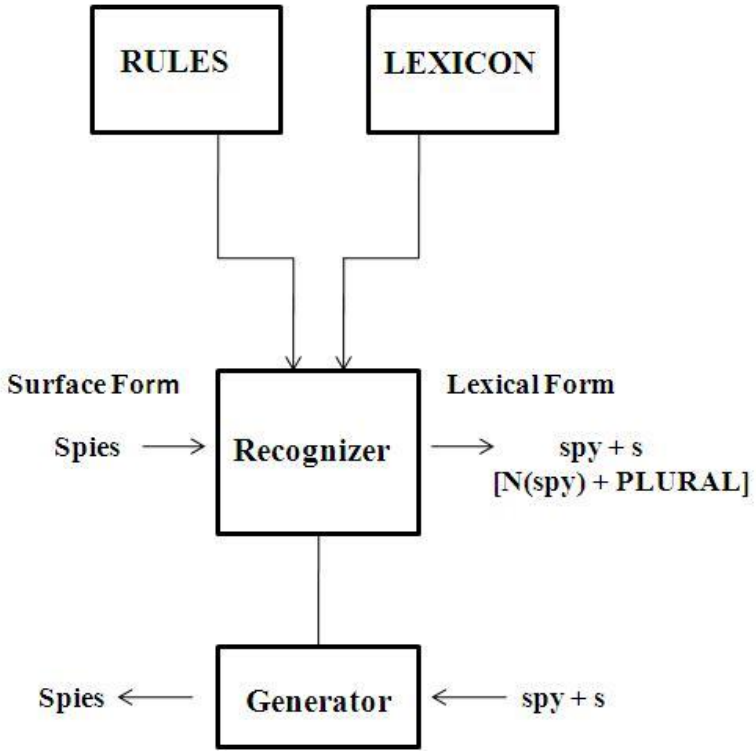
Lexical Representation: ` s p y + 0 s

Surface Representation: 0 s p i 0 e s

Rules must be written to account for the special correspondences `:0, y:i, +:0, and 0:e.

The two functional components of PC-KIMMO are the generator and the recognizer. The generator accepts as input a lexical form, applies the phonological rules, and returns the corresponding surface form. It does not use the lexicon. The recognizer accepts as input a surface form, applies the phonological rules, consults

the lexicon, and returns the corresponding lexicon form with its gloss. Figure shows the main components of PC-KIMMO system.



Components of PC-KIMMO

Around the components of PC-KIMMO shown in figure 1 is an interactive shell program that serves as a user interface. When the PC-KIMMO shell is run, a command-line prompt appears on the screen. The user types in commands which PC-KIMMO executes. The shell is designed to provide an environment for developing, testing, and debugging two-level descriptions. The PC-KIMMO functions are available as a source code library that can be included in another program. This means that the user can develop and debug a two-level description using the PC-KIMMO shell and then link PC-KIMMO's functions into his own program.

When you parse a word, you must first tokenize it into morphemes. This tokenizing is done by the rules and lexicon. When a surface word is submitted to PC-KIMMO's

Recognizer, the rules and lexicon analyze the word into a sequence of morpheme structures. A morpheme structure consists of a lexical form, its gloss, its category, and its features. For example, the word *enlargement* is tokenized into this sequence of morpheme structures:

PC-KIMMO is a significant development for the field of applied natural language processing. Up until now, implementations of the two-level model have been available only on large computers housed at academic or industrial research centers. As an implementation of the two-level model, PC-KIMMO is important because it makes the two-level processor available to individuals using personal computers. Computational linguists can use PC-KIMMO to investigate for themselves the properties of0 the two-level processor. Theoretical linguists can explore the implications of two-level phonology, while descriptive linguists can use PC-KIMMO as a field tool for developing and testing their phonological and morphological descriptions. Finally, because the source code for the PC-KIMMO's generator and recognizer functions is being made available, those developing natural language processing applications (such as a syntactic parser) can use PC-KIMMO as a morphological front end to their own programs.

4.3.3 Finite State Trancedures (FST)

FST [22] is an advanced version of FSA. FST is used to represent the lexicon computationally. It can be done by accepting the principle of two level morphology.

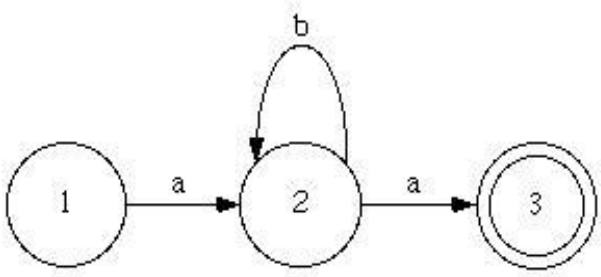
The two level morphology represents a word as a correspondence between lexical level and surface level. An FST is represented as a two tape automaton. We can combine lexicon, orthographic rules and spelling variations in the FST to build a morphological analyzer .Tamil morphological analyser utilizes this principle side by side with paradigm.

How finite state machines work

The basically mechanical procedure for applying two-level rules makes it possible to implement the two-level model on a computer by using a formal language device called a finite state machine. The simplest finite state machine is a finite state automaton (FSA), which recognizes (or generates) the well-formed strings of a regular language. While finite state machines are commonplace in computer science

and formal language theory, they may not be familiar to all linguists. They have, however, had their place in the linguistic literature.

For example, consider the regular language L1 consisting of the symbols a and b and the "grammar" *abNa* where *N >= 0*. Well-formed strings or "sentences" in this language include aa, aba, abba, abbba, and so on. The language L1 is defined by the figure.



FSA for L1

valid correspondences (or translations) of each other. For example, assume the first input string to an FST is from language L1 above, and the second input string is from language L2, which corresponds to L1 except that every second b is c.

L1:

abbbba

L2:

abcbca

4 Stemmer

Stemmer is used to stripping of affixes. It uses a set of rules containing list of stems and replacement rules.

E.g: writing → write + ing

For a stemmer program me we have to specify all possible affixes with replacement rules.

E.g. ational →ate relational→ relate

tional → tion conditional→ condition

The most widely used stemmer algorithm is Potter algorithm. The algorithm is available free of cost http://www.tartarus.org/martin/PotterStemmer/. There are some attempts to develop stemmer for Indian Languages also. IIT Bombay & NCST Bombay have developed stemmer for Hindi [Manish, Anantha].

3.3.5 Corpus Based Approach

Corpus is a large collection of written text belongs to a particular language. Raw corpus can be used for morphological analysis .It takes raw corpus as input and produces a segmentation of the word forms observed in the text. Such segmentation resembles morphological segmentation. Morfessor1.0 developed in Helsinki University is a corpus based language independent morphological segmentation program. The LTRC Hyderabad successfully developed a corpus based morphological analyzer. The program combines paradigm based approach as well corpus based approach.

Paradigm Approach

A paradigm defines all the word form of a given stem and also provides a feature structure with every word form .The paradigm based approach is efficient for inflectionally rich languages. The ANUSAARAKA research group has developed a language independent paradigm based morphological compiler program for Indian Languages.

This or a variant of this scheme has been used widely in NLP. The linguist or the language expert is asked to provide different tables of word forms covering the words in a language. Each word-forms table covers a set of roots which means that the roots follow the pattern (or paradigm) implicit in the table for generating their word forms. Almost all Indian language morphological analyzers are developed using this method. Based on paradigms the program generates add delete string for analyzing. Paradigm approach rely on findings that the different types of word paradigms are based on their morphological behavior. Words are categorized as nouns, verbs, adjectives, adverbs and postpositions. Each category will be classified

into certain types of paradigms based on their morphophonemic behavior. For example noun *Uru* (village) belongs to a paradigm class is different form *Abbayi* (boy) which belongs to a different paradigm class as they behave differently morphophonemic ally.

We have used Machine learning using SVMTool for implementing Morphological Analyzer.

5. Why Perl for Linguistics

Perl, Practical Extraction Report Language, is a programming language which can be used for a large variety of tasks. It was designed by Larry Wallin the mid 1980s as a tool for writing programs in the UNIX environment. A typical use of Perl would be for extracting information from a text file and printing out a report or for converting a text file into another form. But Perl provides a large number of tools for quite complicated problems, including systems programming. Programs written in Perl are called Perl scripts. Perl is implemented as an interpreted (not compiled) language.

The 2-step process of interpreted languages makes them slightly easier to work with. You can constantly check your code by running it after every change. Thus, the execution of a Perl script tends to use more CPU time than a corresponding C program. Perl has the power and flexibility of a high-level programming language such as C. Perl (like COBOL and some other languages) was designed to sound as much as possible like English, so it's relatively user-friendly. Perl is a very convenient language that allows for multiple forms of expression. That means that you can say 104 something in various ways, as long as the syntax is correct. Perl's original main use was text processing. It is exceedingly powerful in this regard, and can be used to manipulate textual data, reports, email, news articles, log files, or just about any kind of text, with great ease. In morphological analyzer, it is obvious that we are handling texts, or words of a Natural Language. Dealing with the patterns or we can say, morphemes, in the word would be easy if Perl is used.

The program for "Morphological analyzer for Telugu Verbs" is written in Perl. Because the program has to compare the suffix with the list of possible inflections and to go for checking Sandi changes many times in the course of analyzing a word. Pearl can be called from a Java program. The Graphical User Interface (GUI) of the program is designed in Java.

6. Methodologies

The purpose of this project is to find the automatic extraction of telugu verbs through their suffix stripping by using Perl program.

Input data: sakshi.txt(moral stories taken from sakshi news paper)

Total no. Of words: 4487

Total no. of verbs: 1321

Accuracy:95%

***PROCEDURE*:**

In Telugu,verbs can be identified in two types:

1. Based on the termination of their suffixes from root

2. Based on termination of Tense+PNG(person-number-gender) .

It can be explained in below

6.1 Based on the termination of their suffixes from root:

The simple verbs in telugu are in 3 forms

1. tam ending words	ex: cheaya+tam (do)
2. dam ending words	ex: raya+dam (write)
3. ta ending words	kottu+ta (strike or beat)
4. yu ending words	ex: che+yu (do)

6.2 Based on termination of Tense+PNG(person-number-gender):

In this we obsrved in 3 types:

1. Progressive present tense:

   In this the word ending with the roots :

   nanu,nnaavu,nnaaru,nnaadi,nnaadu,nnaamu.

Ex: kottutu+nnaanu

Kottu=verb nnanu=Tense+PNG

2. Habitual present and future tense:

In this the word ending with the roots :

taanu,taavu,taaru,taadi,taadu,taamu.

ex:kottu+taanu.

Taanu	Vachaaanu	Presente sl+1 st m.f
Taavu	Vastaavu	Future sl+2nd m.f
Taadi	Vastaadi	Future sl+2nd female
Taadu	Vastaadu	Future sl+ 2nd male

3. Past tense:

In this the word ending with the roots :

tindi,aadu,,aaru,aanu,…..etc.

ex:kottu+tini.

Tindi	Thittindhi
Taadu	Chesaadu
Taaru	Vastaadi
Aanu	Techaanu

Nnadhi	Vasthunnadhi
Nnavi	Vastunnavi
Nnamu	Vastunnamu
Naavu	Vastunnavu

6.4Person number gender categorization

	Past	*Nonpast*	*Negative*
1s	tinnānu	tiṇṭānu	tinanu
2s	tinnāvu	tiṇṭāvu	tinavu
3s.mh	tinnāḍu	tiṇṭāḍu	tinaḍu
3s.nmh	tinnādi	tiṇṭundi	tinadu
1pl	tinnāmu	tiṇṭāmu	tinamu
2pl	tinnāru	tiṇṭāru	tinaru
3pl.h	tinnāru	tiṇṭāru	tinaru
3pl.nh	tinnāyi	tiṇṭāyi	tinavu

6.5 Programming design:

In this section I can do only two steps:

They are

1. Splitting, tokenization and sorting.

2. Automatic extraction of verbs based on roots of words.

Explanation:

Splitting can be done on following logic:

@element = split/\s+/,$line;

Tokenization can be done on following logic:

```
next if ($line =~ /^\s*$/);
        for($line){
        s/<\/?.+?>//g;
        s/\'(.+?)\'/$1/g;
        s/[!.,::?()"\/]//g;
        s/[a-z\/]//g;
      s/[A-Z\/]//g;
       s/[0-9\/]//g;
        s/\s~\s//g;
        s/^\s*//;
```

```
s/\s+$//;

s/\'//;

s/\-//;

s/\=//;
```

}

Extracting verbs can done on regular expressions

In this I can use only while and if loop, logics in regular expressions

While used for assigning the input line to each operation in loop.

If loop is assigned in while loop for checking the given condition is true or false.

Use utf8: this function can be encode and decode the given data.

Regular expressions logics are:

i$line =~ /(\w+)చాడు$|(\w+)చారు$|(\w+)చావు$|(\w+)చాను$/

#matching and extract the root words chaadu,chaaru etc.

$line =~ /(\w+)**టాడు**$|(\w+)**టాను**$|(\w+)**టాడు**$|(\w+)**టాము**$|(\w+)**టారు**$|(\w+)**టవి**$/

#matching and extract the root words taamu,taadu etc.

$line =~ /(\w+)**చ్చారు**$|(\w+)**చ్చాడు**$|(\w+)**చ్చను**$/) #pattern matching

my $root2 = substr$line,0,-7; #extracting

$base3 = $root2."**వ్యటం**"; #adding root

Remaining part of the program can explained on same logic(shown on above)

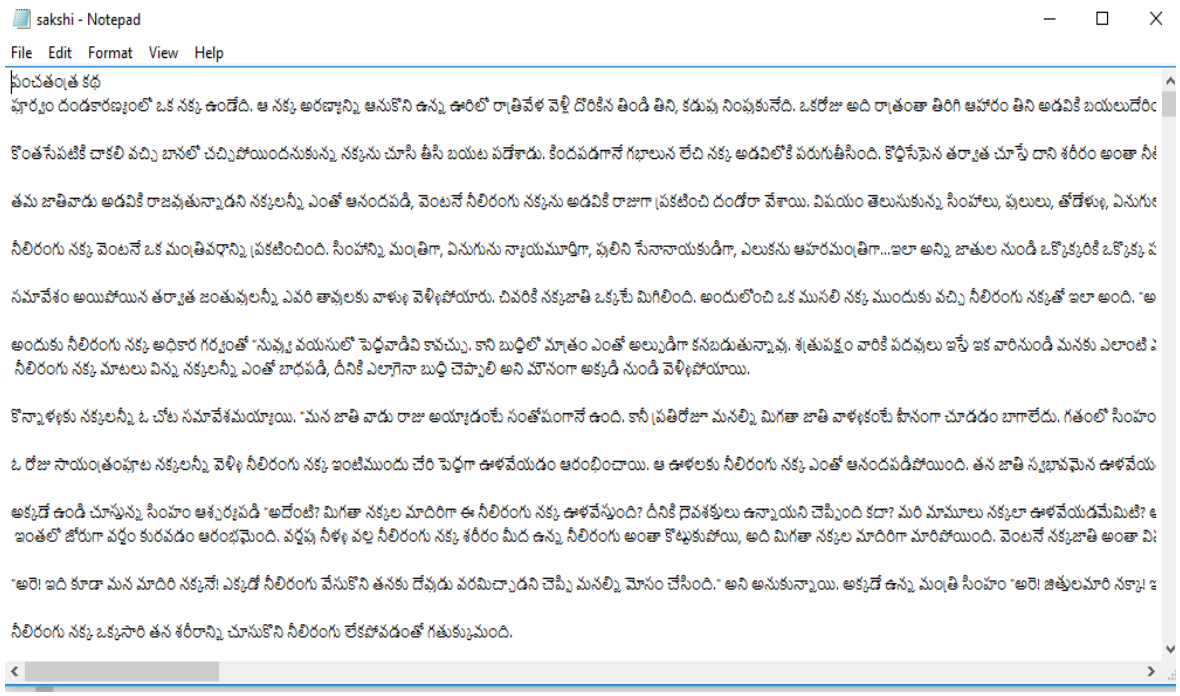
7. Code

```
#!/usr/bin/env perl

use strict;

use warnings;

use utf8;           #used for encoding and decoding purpose


my @words = ();      #assing the array of words and vocab_list

my @vocab_list = ();

my $infile = shift or die "please provide an input file name!\n";

open my $IN,"<encoding(utf8)",$infile or die "unable to open the input file $infile!\n";

my $outfile = shift or die "please provide an output file name!\n";

open my $OUT,">encoding(utf8)",$outfile or die "unable to open the output file $outfile!\n";


while (my $line = <$IN>) {                    #assign input data to each operation in loop and checks
true or false


if ($line  =~
/(\w+)చాడు$|(\w+)టాడు$|(\w+)చ్చాడు$|(\w+)నాడు$|(\w+)న్నాడు$|(\w+)తాడు$|(\w+)స్తాడు$/){

#pattern matching

my $root1 = substr$line,0,-1;

print $OUT "$root1\t\t  VB+T+3s.M\n\n";

}

if ($line  =~
/(\w+)నది$|(\w+)న్నది$|(\w+)స్తుంది$|(\w+)తుంది$|(\w+)టుంది$|(\w+)యింది$|(\w+)న్నది$|(\w+)స్తుంది$

/) { #pattern matching

my $root7 = substr$line,0,-1;

print $OUT "$root7\t\t  VB+T+3s.F\n\n";

}if ($line  =~
/(\w+)చారు$|(\w+)టారు$|(\w+)చ్చారు$|(\w+)నారు$|(\w+)న్నారు$|(\w+)తారు$|(\w+)దురు$|(\w+)తిరు$|(\

w+)స్తారు$/){   #pattern matching

my $root2 = substr$line,0,-1;

print $OUT "$root2\t\t  VB+T+3pl.M&F\n\n";

}
```

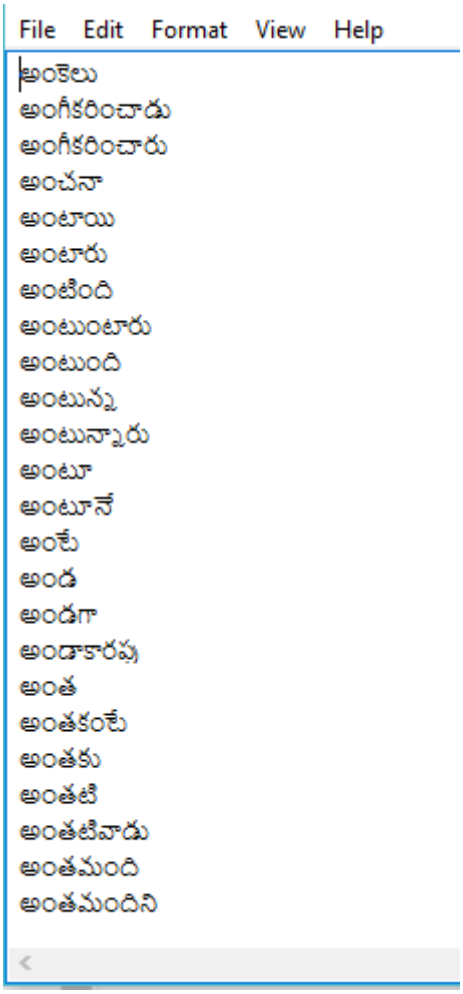
```
if ($line  =~ /(\w+)చావు$|(\w+)నావు$|(\w+)న్నావు$|(\w+)తావు$|(\w+)దువు$|(\w+)స్తావు$/){
#pattern matching
my $root3 = substr$line,0,-1;
print $OUT "$root3\t\t  VB+T+2s.M&F\n\n";
}
if ($line  =~
/(\w+)చాను$|(\w+)చ్చను$|(\w+)నాను$|(\w+)న్నాను$|(\w+)తాను$|(\w+)దును$|(\w+)స్తాను$/){
#pattern matching
my $root4 = substr$line,0,-1;
print $OUT "$root4\t\t  VB+T+1s.M&F\n\n";
}
if ($line  =~ /(\w+)టాము$|(\w+)న్నాము$|(\w+)తాము$|(\w+)దుము$|/) { #pattern matching
my $root5 = substr$line,0,-1;
print $OUT "$root5\t\t  VB+T+1pl.M&F\n\n";
}
if ($line  =~ /(\w+)టవి$|(\w+)నవి$|(\w+)న్నవి$|(\w+)న్నవి$|(\w+)తావి$|(\w+)తివి$|(\w+)స్తావి$/) {
#pattern matching
my $root6 = substr$line,0,-1;
print $OUT "$root6\t\t  VB+T+3pl.M&F\n\n";
}
if ($line  =~ /(\W+)ట$|(\W+)చు$|(\W+)యు$|(\w+)డం$|(\w+)టం$/&$line
!~/(\w+)డ్డం$|(\w+)ట్టం$|ట్టం$/&$line !~/\bఖం(\w+)\b/){ #pattern matching
my $root8 = substr$line,0,-1;
print $OUT "$root8\t\t  Verb\n\n";
}
if ($line  =~ /(\w+)తిమి$/) { #pattern matching
my $root9 = substr$line,0,-1;
print $OUT "$root9\t\t  VB+T+2pl.M&F\n\n";
}
if ($line  =~ /(\w+)తిని$/) { #pattern matching
my $root10 = substr$line,0,-1;
print $OUT "$root10\t\t  VB+T+1pl.M&F\n\n";
}
```

```
if ($line  =~ /(\w+)స్తాము$/) { #pattern matching

my $root11 = substr$line,0,-1;

print $OUT "$root11\t\t  VB+T+2pl.M&F\n\n";

}


}

close $IN;
```

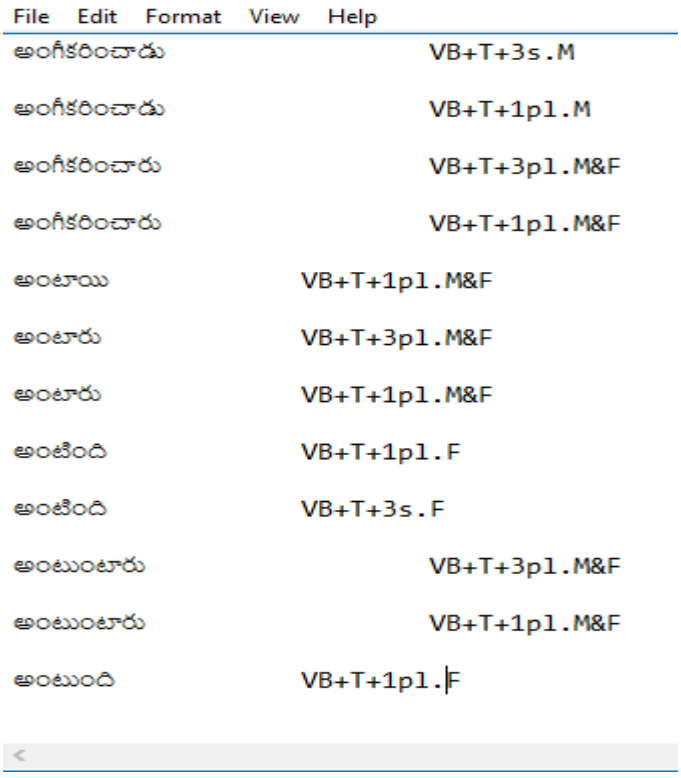
8. SAMPLE RAW TEXT, INPUT And OUTPUT



SPLIT OUTPUT



Sample Output



9. LIMITATIONS:

- some accuracy missing on root word (exact verb not Comes, comes only root word)
- While executing programming I am getting wrong analysis in very few words because the copied raw text. If it is properly copied, it gets correct output.

10. CONCLUSION AND FUTURE WORK

Telugu Morphology generates morphological forms of the two main lexical classes: noun and verbs of Telugu language. Except for a few irregular classes of nouns, some of which have only a few words per class, the process is automatic. For these exceptions, the user needs to identify the class to which the noun belongs, based on what form it takes for genitive case or based on how its plural is formed. This program Resulted accuracy level 80 percentage.

Plural generation is done in two ways: based on informal rules after identifying the class of the noun, or through formal descriptive rules without the requirement for identification of class in which case no user interaction is required. For the latter however, there are 2 syllable endings, for which the generated plural is incorrect. Verb generation is fully automatic and is correct for all words. There are some irregular verbs which are erroneously generated, but this is what makes them irregular words. Current version of the toolkit is available in Open Source for review and enhancement by the World Wide Web community.

11. References

1. Rajendran S, Arulmozi S, Ramesh Kumar, Vishwanathan S. "Computational Morphology of Verbal Complex Language". Language in India Volume 3:4 April 2003.
2. Maya Report on Telugu Morphology.
3. Brown, C.P., The Grammar of the Telugu Language. New Delhi: Laurier Books Ltd, 2001.
4. Uma Maheshwar Rao G, Rajeev Sangal, P V H M L Narasimham, S C Babu, J Satyanarayana. Subcommitee report on standards for the Implementation of Telugu in Information Technology, 2001.
5. B.N.Patnaik," Computational Linguistics for Indian Languages", Department of Humanities and Social Sciences & Centre for Creative Writing and Publication Indian Institute of Technology Kanpur, 2004.