*IOSR Journal of Computer Engineering (IOSR-JCE)*
*e-ISSN: 2278-0661,p-ISSN: 2278-8727,*
*PP 89-99*
*www.iosrjournals.org*

# Malayalam Morphological Analyzer Using String Matching Algorithm

Divya S Nair[1], Sumithra M D[2]
[1](Computer Science and Engineering, LBS Institute of Technology for Women, Poojappura, India)
[2](Computer Science and Engineering, LBS Institute of Technology for Women, Poojappura, India)

***ABSTRACT:*** *In Natural Language Processing (NLP), the morphological analyzer plays an important role in processing different forms of human language for formal writing and reading. Generally morphological analyzer's of natural languages provide information concerning grammatical properties of the word it analyses. Malayalam is morphologically rich and agglutinative language, natural language processing like, web searching, machine translation, speech recognition, speech synthesis etc. for Malayalam is complex in nature. The proposed system identifies grammatical information of Malayalam, depending upon its word category using string matching algorithm. Experimental results show that the proposed system provide an efficient tense categorization, while comparing with the existing algorithms available in NLP.*

***Keywords:*** *Morphological Analyzer, Natural Language Processing, Noun Analysis, Root Words, Sandhi, Verb Analysis*

## I. INTRODUCTION

Natural Language Processing (NLP) is a subfield of Artificial Intelligence and Linguistics. In NLP, the morphological analyzer plays an important role in processing different forms of human language for formal writing and reading [1]. The morphological analyzer mainly deals with the study of the internal structure of the words based on its grammatical features of any language. It will return its root/stem [1] (പ്രകൃതി [2]) word along with its grammatical information depending upon its word category. Morphological analysis is the process of splitting the surface form of a word into its lemma and grammatical information. For example, the word 'Cats' splits into its lemma 'cat' and grammatical information <noun><plural> [3].

Malayalam is a morphologically rich and agglutinative language. The design and implementation of the Malayalam morphological analyzer is a promising research area for various applications in NLP. The previous works carried out in the field of Morphological Analysis, is not complete. Eg: Brute Force Method [4]. As Malayalam requires many morphophonemic changes in the word formation, the above mentioned method is not sufficient. The proposed work use a method called Suffix Stripping [1] which uses String matching algorithm and Sandhi rules to find the root/stem (പ്രകൃതി[2]) form, also it can identify tense with all categories [5] like Simple Present (സാമാന?വർ? മാനകാലം), Present Definite (നി? യവർ? മാനകാലം), Present Continuous (തുടർവർ? മാനകാലം) , Present Immediate (ആസ? വർ? മാനകാലം), Simple Past(സാമാന?ഭൂതകാലം), Past Definite (നി? യഭൂതകാലം), Past Conditional (ഹേതുഹേതുമ? ഭൂതകാലം), Past Immediate (ആസ? ഭൂതകാലം), Past Indefinite (സ? ി? ധഭൂതകാലം), Past Perfect (പൂർ? ഭൂതകാലം), Past Continuous (തുടർഭൂതകാലം), Simple Future (സാമാന?ഭാവികാലം), Future Definite (നി? യഭാവികാലം), Future Conditional (വിക? പഭാവികാലം), Future Immediate (ആസ? ഭാവികാലം), Future Indefinite (സ? ി? ധഭാവികാലം), Future Continuous (തുടർഭാവികാലം) etc. This Malayalam Morphological Analyzer would help in automatic spelling and grammar checking, natural language understanding, web searching, machine translation, speech recognition, speech synthesis, part of speech tagging, and parsing applications.

## II. LITERATURE SURVEY

### 2.1 Brute Force Method

Brute force search is a very general problem solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement. Brute force stemmers [4] employ a lookup table which contains relations between root forms and

inflected forms. To stem a word, the table is queried to find a matching inflection. If a matching inflection is found, the associated root form is returned. Brute force approaches are criticized for their general lack of elegance in that no algorithm is applied that would more quickly converge on a solution. In other words, there are more operations performed during the search than should be necessary. The algorithm is only accurate to the extent that the inflected form already exists in the database. Given the number of words in a given language, like Malayalam, it is unrealistic to expect that all words and inflected forms to be recorded manually. Brute force algorithms are initially very difficult to design given the immense amount of relations that must be initially stored to produce an acceptable level of accuracy.

**2.2 Root Driven Approach**

In the root driven approach, the stem of the word should be firstly found in a lexicon before starting the morphological analysis. The major drawback of this approach is the cost of the searching process required to find the stem. In the work done by Solak and Oflazer [6], the word itself and its subparts, which have been obtained by removing the letters one by one from the end of this word, are looked up in a lexicon to find all the possible stems. The real stem is discovered after the morphological analysis made by using these possible stems. Even though there are different search methods improving the performance, the examining of each subpart is obviously a very time consuming process especially for agglutinative languages.

**2.3 Stemming Algorithm**

A stemming algorithm is a process of linguistic normalization, in which the variant forms of a word are reduced to a common form. Affix removal conflation techniques are referred to as stemming algorithms and can be implemented in a variety of different methods. All remove suffixes (ზuთ?ൌo [5]) and/or prefixes in an attempt to reduce a word to its stem. Suffixes that are concatenated to words are often done so in a certain order, such that a set of order classes will exist among suffices. An iterative stemming algorithm will remove suffixes one at a time, starting at the end of the word and working towards the beginning. A stemming algorithm, a procedure to reduce all words with the same stem to a common form, is useful in many areas of computational linguistics and information retrieval work.

**2.4 String matching algorithms**

String matching or searching algorithms try to find places where one or several strings (also called patterns) are found within a larger string (searched text). The single string matching problem is to search for all the occurrences of a string p, called the pattern, in the text T = t1 t2 t3…tn on the same alphabet S, where n is the length of the text. Multiple string matching extends the problem to search for the pattern set P = {p1, p2 …, pr} simultaneously in the text. During the search, a search window of the pattern length is moved along the text, and the pattern is searched for within the window. Pattern matching can be exact or approximate. An exact matching algorithm stipulates that the pattern and the matched text should be exactly the same, while an approximate matching algorithm allows a limited error between the pattern and the matched text.

Exact string matching algorithms can be categorized in various ways. One way of categorization is grouping them into three general approaches: prefix searching, suffix searching, and factor searching, depending on which part of the pattern is searched for within the search window. A string X is the prefix, suffix, and factor of XY, YX, and YXZ, respectively, where Y and Z are also strings.

Po-Ching el. [7] categorizes the algorithms into four categories to emphasize the data structure that drives the matching. These categories are automaton-based, heuristics-based, hashing-based, and bit-parallelism-based. An automaton-based algorithm builds a finite state automaton from the patterns in the pre-processing stage and tracks the partial match of the pattern prefixes in the text by state transition in the automaton. A heuristics-based algorithm allows skipping some characters to accelerate the search according to certain heuristics. Some algorithms require a verification algorithm following a possible match to verify if a true match occurs. A hashing based algorithm compares the hash values of characters in the text segment by segment with those of the characters in the patterns. If both hash values are equal, a possible match may occur. The characters in the text and those in the patterns are then compared to verify if a true match occurs. A bit-parallelism-based algorithm simulates the operation of a non-deterministic finite automaton that tracks the partial match of the prefix or the factor of the pattern by means of the parallel bit operations inside a computer register word in which the state transition status is encoded.

**2.4.1 Automaton-based**

The Aho-Corasick (AC) algorithm [8] was proposed for multi-pattern matching. A finite automaton that accepts all the strings in the pattern set is built in the pre-processing stage. Each character in the text is then fed sequentially to the automaton that tracks partially matched patterns through state transition. If one of the final states is reached, a match is claimed. Although the AC algorithm is theoretically independent of the pattern set size in efficiency, it will become slow for a large pattern set in practice because of the worse cache locality in accessing a large transition table. Effectively compressing the transition table to reduce the memory requirement and enhance the cache locality becomes active research in the implementation of the AC algorithm.

**2.4.2 Heuristics-based**

The Boyer-Moore (BM) algorithm is a single string matching algorithm. It allows skipping over characters that cannot be a match in the text according to two heuristics: the bad-character heuristic and the good-suffix heuristic [9]. Two shift functions are pre-computed according to the two heuristics before the search. The characters within the search window are searched backward from the last character for the longest suffix that is also a suffix of the pattern. The two shift functions are referred to only when a character mismatch is found. The maximum of these two function values is the shift distance of the search window. If no mismatch is found, a pattern match is claimed. The BM algorithm has the best performance when the shift distance is close to the pattern length.

**2.4.3 Hashing-based**

The Rabin-Karp (RK) algorithm [10] is designed to handle single string matching. During the search, the hash value of each segment of m characters in the text, $h(t_{s\ldots s+m-1})$, is compared with the hash value of the pattern, h(p), for $s = 1\ldots n-m+1$, where m is the pattern length and h is the hash function. If $h(t_{s\ldots s+m-1}) = h(p)$, a possible match may occur at the position of s. The pattern is then compared with this segment character by character to verify whether a true match occurs or not.

The RK algorithm can be extended for multiple string matching by storing the hash values of the patterns in the pattern set in an ordered table. The comparison of the hash values becomes binary search in the ordered table, to find whether the hash value of a text segment is equal to that of some pattern in the ordered table.

**2.4.3.1 SHA 256**

An n bit hash is a map from arbitrary length messages to n bit hash values. An n bit cryptographic hash is an n bit hash which is one way and collision- resistant. Such functions are important cryptographic primitives used for digital signature and password protection. The SHA (Secure Hash Algorithm) is a cryptographic hash function. A cryptographic hash is like a signature for a text or a data file. SHA-256 algorithm generates an almost unique, fixed size 256-bit (32-byte) hash value. The result is represented as 64 hex digits. SHA-256 is the strongest hash function available. SHA-256 function operates on a 512 bits message blocks and generates a 256 bits message digest.

**2.4.4 Knuth–Morris–Pratt (KMP) algorithm**

The KMP algorithm searches the occurrences of a word within a text by sliding the window of length m (word) over the text (of length n) from left to right. Compare successive characters within the window from left to right until a mismatch occurs. If a mismatch occurs then the word itself realizes sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters.

The idea behind KMP [11] approach to pattern matching is perhaps easiest to grasp if we imagine placing the pattern over the text and sliding it to the right in a certain way. Consider for example a search for the pattern “abcabcacab” in the text “babcbabcabcaabcabcabcacabc”; initially place the pattern at the extreme left and prepare to scan the leftmost character of the input text:

abcabcacab
babcbabcabcaabcabcabcacabc
↑
The arrow here indicates the current text character; since it points to “b”, which doesn’t match that “a”, shift the pattern one space right and move to the next input character:

abcabcacab
babcbabcabcaabcabcabcacabc
↑
Now have a match, so the pattern stays put while the next several characters are scanned. Soon come to another mismatch

abcabcacab
babcbabcabcaabcabcabcacabc
↑
At this point the first three pattern characters have matched but not the fourth, so the last four characters of the input have been a b c x where x ≠ a; there is no need to remember the previously scanned characters, since the position in the pattern yields enough information to recreate them. In this case, no matter what x is (as long as it's not a), the pattern can immediately be shifted four more places to the right; one, two, or three shifts couldn't possibly lead to a match.
Soon get to another partial match, this time with a failure on the eighth pattern character:

abcabcacab
babcbabcabcaabcabcabcacabc
↑
The last eight characters were a b c a b c a x, where x ≠ c. The pattern should therefore be shifted three places to the right:

abcabcacab
babcbabcabcaabcabcabcacabc
↑
Then try to match the new pattern character, but this fails too, so we shift the pattern four (not three or five) more places. That produces a match, and continues scanning until reaching another mismatch on the eighth pattern character:

abcabcacab
babcbabcabcaabcabcabcacabc
↑
Again shift the pattern three places to the right; this time a match is produced and eventually discovers the full pattern:

abcabcacab
babcbabcabcaabcabcabcacabc
↑

## III. OBJECTIVE

The proposed system, "Malayalam Morphological Analyzer", specifically aims to return 'root/stem' (പകൃതി [2]) of a word along with its grammatical information [5] depending upon its word category without the need of a database to store root/stem words. This work extracts information like gender (ലിംഗം), number (ഏകവചനം, ബഹുവചനം), case (വിഭ? ി) information for nouns (നാമ? ൾ) and all categorization of tense (കാലം) is identified in case of verbs (ക്രിയ) for a given sentence.

## IV. PROPOSED METHODOLOGY

The existing systems available for Malayalam morphological analyzer deal with whether the word is noun or a verb. But the tense categorization is not completely achieved.

Apart from identifying noun and verb in the given sentence, the proposed work categorizes tense. If the word is noun, then it will identify case, number and gender information [12].

If the word is verb, then it will identify tense with all categories [5] like Simple Present (സാമാന?വർ? മാനകാലം), Present Definite (നി? യവർ? മാനകാലം), Present Continuous (തുടർവർ? മാനകാലം) , Present Immediate (ആസ? വർ? മാനകാലം), Simple Past (സാമാന?ഭൂതകാലം), Past Definite (നി? യഭൂതകാലം), Past Conditional (ഹേതുഹേതുമ? ഭൂതകാലം), Past Immediate (ആസ? ഭൂതകാലം), Past Indefinite (സ? ി? ധഭൂതകാലം), Past

Perfect (പൂർ? ഭൂതകാലം), Past Continuous (തുടർഭൂതകാലം), Simple Future (സാമാന?ഭാവികാലം), Future Definite (നി? യഭാവികാലം), Future Conditional (വിക? പഭാവികാലം), Future Immediate (ആസ? ഭാവികാലം), Future Indefinite (സ? ി? ധഭാവികാലം), Future Continuous(തുടർഭാവികാലം) etc.

**4.1 Noun Analysis:** The general format of input and output of the morphological analyzer of Malayalam is as follows: Word —> stem + suffixes

Nouns are linguistic categories [5], which takes Cases [12] (വിഭ? ികൾ), Number (ബഹുവചനം, ഏകവചനം) and Gender (സലിംഗം, അലിംഗം, നപുംസകലിംഗം) information. The categorical information in noun is listed in Table 4.1 [5].

**Table 4.1:** Categorical Information in Noun

| *വിഭ? ികൾ* | |
|---|---|
| നിർ?ശിക | ø |
| പതിഗാഹിക | എ |
| സംയോജിക | ഓ? |
| ഉ?ശിക | ? ്, ? |
| പയോജിക | ആൽ |
| സംബ? ിക | ഒ? , ഉടെ |
| ആധാരിക | ഇൽ, കൽ |

| ബഹുവചനം |
|---|
| കൾ |
| മാർ |
| അർ |

**4.2 Verb Analysis:** Verb is a grammatical category, which takes tense with it. Many markers [5] are present in the Tense is listed in Table 4.2.

**Table 4.2:** Categorical Information in Past Tense

| ഉപവിഭാഗങ്ങൾ | പ്രത്യയം | ഉദാഹരണം |
|---|---|---|
| സാമാന്യഭൂതകാലം | ഇ, തു | പോയി, കണ്ടു |
| നിശ്ചയഭൂതകാലം | ആയിരുന്നു | കാണുകയായിരുന്നു,<br>പറയുകയായിരുന്നു |
| തുടർഭൂതകാലം | കൊണ്ടിരുന്നു | കണ്ടുകൊണ്ടിരുന്നു<br>പറഞ്ഞുകൊണ്ടിരുന്നു |
| ആസന്നഭൂതകാലം | ഇട്ടുണ്ട് | കണ്ടിട്ടുണ്ട് |
| പൂർണ്ണഭൂതകാലം | ഇരുന്നു,<br>ഇട്ടുണ്ടായിരുന്നു | വന്നിരുന്നു,<br>പോയിരുന്നു,<br>കണ്ടിട്ടുണ്ടായിരുന്നു |
| സന്നിഗ്ധഭൂതകാലം | ഇട്ടുണ്ടാണ്ടിരിക്കും | കണ്ടിട്ടുണ്ടായിരിക്കും<br>പറഞ്ഞിട്ടുണ്ടായിരിക്കും |
| ഹേതുഹേതുമത് ഭൂതകാലം | ഉം+ആയിരുന്നു | കാണുമായിരുന്നു<br>വരുമായിരുന്നു |

**Table 4.3:** Categorical Information in Present Tense

| ഉപവിഭാഗ? ൾ | ?പത?യം | ഉദാഹരണം |
|---|---|---|
| സാമാന?വർ? മാനകാലം | ഉ? ു | കാണു? ു, പറയു? ു |
| നി? യവർ? മാനകാലം | ആകു? ു,ആ? | കാണുകയാ? , പറയുകയാകു? ു |
| തുടർവർ? മാനകാലം | കൊ? ിരി? ു? ു | ക? ുകൊ? ിരി? ു? ു,<br>പറ? ുകൊ? ിരി? ു? ു |
| ആസ? വർ? മാനകാലം | ഉ? ു? ് | കാണു? ു? ്, പറയു? ു? ് |

**Table 4.2:** Categorical Information in Future Tense

| ഉപവിഭാഗ? ൾ | ?പത?യം | ഉദാഹരണം |
|---|---|---|
| സാമാന?ഭാവികാലം | ഉം | കാണും, പറയും |
| നി? യഭാവികാലം | ഇരി? ും | ക? ിരി? ും,<br>പറ? ിരി? ും |
| തുടർഭാവികാലം | കൊ? ിരി? ും | ക? ുകൊ? ിരി? ും,<br>പറ? ുകൊ? ിരി? ും |
| ആസ? ഭാവികാലം | ആൻ + പോവുകയാ? | കാണാൻപോവുകയാ? ,<br>പറയാൻപോവുകയാ? |
| സ? ി? ധഭാവികാലം | ആയിരി? ും | കാണുമായിരി? ും,<br>പറയുമായിരി? ും |
| വിക? പഭാവികാലം | ഏ? ും, ഏ? ാം | ക? ? ും, പറ? ? ാം |

## V. SYSTEM DESIGN

### 5.1 SYSTEM ARCHITECTURE

Morphological Analyzer identifies the root word from the user input. The processing is continued under the assumption that user input is a valid Malayalam word. Suffix of input is extracted using Malayalam grammatical rules. The suffix dictionary is searched for the identified suffix, based on hash values. Finally validity of user input may be verified using stem dictionary.

In this method each word is scanned character by character from right to left. The Unicode [13] decimal value for the substring is formed during each iteration and verified whether it represents a valid suffix in Malayalam. If it is a valid suffix, the corresponding hash value is generated using SHA-256 hash function and compared with the hash values present in the suffix dictionary. Inclusion of hashing techniques improves the system efficiency by reducing the search time. If a matching hash value is identified, the related grammatical information is conveyed to the user.

The method used for analysis makes use of a stem dictionary (for identifying a valid stem), a suffix dictionary containing all possible suffixes that nouns/verbs in the language can have (to identify a valid suffix), morphotactic [1] rules (case, gender, number) and morphophonemic [1] rules or sandhi rules [5]. Even if the item does not exist in the dictionary, the analyzer can identify the suffixes and stem (?പകൃതി [2]). Once the suffixes are identified, removing the suffixes and applying proper sandhi rules can obtain the stem. Suffix stripping [1] algorithms do not rely on a lookup tables; instead, rules are stored to find the root/stem.

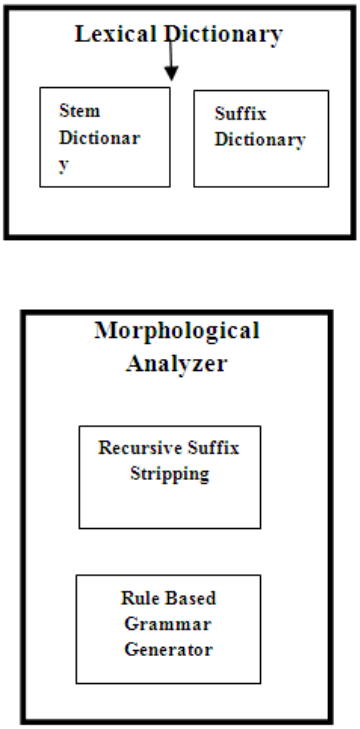


**Figure 5.1:** System Architecture

**5.1.1 Sandhi Rules**
Sandhi [5] is classified as Elision, Augmentation, Substitution and Reduplication.

a) Elision: When two sounds join together one sound will be lost is called Elision.

ക? ു+ഇല?ക? ില?

b) Augmentation: When two words join together one sound comes between these two words.

താളി+ഓല=താളിയോല

c) Substitution: When two words join together, one phoneme is substituted by another

ഹിമ+അങ്ങി=ഹിമാങ്ങി

d) Reduplication: When two words combined together, the letter gets duplicated. This process is called reduplication.

നീല+താമര=നീല? ാമര

**5.2 Algorithm**
1. Accept Input Sentence.
2. Separate each word in the input.
3. Scan each word in the input from right to left character by character.
4. Check the Unicode Decimal value of each scanned character to form a Suffix
5. Check the validity of the formed suffix

If suffix not valid, scan the next character,
GOTO step 4

Else:
a. Find the hash value of Suffix.
b. Separate Suffix from remaining portion.
6. The remaining input characters are considered as the root word by applying Sandhi rules
7. Display the grammar
8. End

**6. Experiment and Results**

NetBeans IDE 7.2.1 and Java was used to implement the system. PostgreSQL is used to store root words and suffixes required for processing. Some of the verified inputs and results are included in Appendix I and II. For the input "അ? ?ാപിക കു?ികളോ? പറ? ു" has three words which are processed separately. Each word is scanned character by character from right to left. The Unicode [13] for the substring formed during each iteration is generated and verified whether it represents a valid suffix in Malayalam. If it is a valid suffix, the corresponding hash value is generated and compared with the hash values present in the suffix dictionary. If a matching hash value is identified, the related grammatical information is conveyed to the user. So the output will be

Input :
*വാചകം* : *അ? ?ാപിക കു?ികളോ? പറ? ു*

**1 . വാചക? ിലെ വാ? ുകൾ**

| വാ? ് |
|---|
| അ? ?ാപിക |
| കു?ികളോ? |
| പറ? ു |

**2. വിഭ? ി പരിശോധന**

| വാ? ് | വിഭ? ി വിഭാഗം | ?പത?യം |
|---|---|---|
| അ? ?ാപിക | നിർ?ശിക | ?പത?യമില |
| കു?ികളോ? | സംയോജിക | ഓ? |

**3. ബഹുവചന? ളുടെ പരിശോധന**

| വാചക? ിലെ വാ? ് | പരിശോധി?? ബഹുവചനം പദം | ബഹുവചന വിഭാഗം | ?പത?യം |
|---|---|---|---|
| അ? ?ാപിക | അ? ?ാപിക | ഏകവചനം | ******* |
| കു?ികളോ? | കു?ികൾ | അലിംഗ ബഹുവച നം | കൾ |

**4 . വാചക? ിലെ നാമ? ൾ**

| വാചക? ിലെ നാമ? ൾ |
|---|
| അ? ?ാപിക |
| കു?ി |

**5 . വാചക? ിലെ ുകിയ പരിശോധന**

| വാചക? ിലെ ുകിയ | ുപത?യം | വിഭാഗം |
|---|---|---|
| പറ? ു | തു | സാമാന?ഭൂതകാലം |

**6 . ുപകൃതി വാ? ുകൾ**

| ുപകൃതി |
|---|
| അ? ?ാപിക |
| കു?ി |
| പറ? |

## VI. CONCLUSION

The suffix stripping method proposed here for identifying the grammar from any of the agglutinative word from Malayalam vocabulary gives a fine tuned result. The method also support fast convergence compared with the existing brute force method and root driven method. Grammatically followed translation techniques from English to Malayalam or from Malayalam to English is not available now. The proposed system can be utilized for the implementation of the semantically followed translation, and an efficient web search using root words.

## ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Mrs Sumithra M D for the continuous support, patience and motivation. Her guidance helped me in all the time of research and writing of this paper. My sincere thanks also go to Mr. Ramachandran, Retd. Associate Prof., Dept. of Malayalam, University of Kerala, for his valuable lectures in Malayalam grammar. Last but not the least; I would like to thank my friend Mr Rajesh for his support and suggestions throughout this work.

## REFERENCES

[1]. Rajeev R R, Dr Elizabeth Sherly, "Morphological Analyzer for Malayalam Language: A Suffix Stripping approach", Proceedings of the 20th Kerala Science Congress, Thiruvananthapuram, 2008.
[2]. Dr. K. Sukumara Pillai, "Kairali Sabdhanusanam", Published by Kerala Bhasha Institute, Thiruvananthapuram, July 1980.
[3]. Vinod P M, Jayan V, Bhadran V K, CDAC Thiruvananathapuram, "Implementation of Malayalam Morphological Analyzer Based on Hybrid Approach", Proceedings of the 24thConference on Computational Linguistics & Speech Processing , 2012.
[4]. Dinesh Kumar, Prince Rana, "Stemming of Punjabi Words By Using Brute Force Technique", International Journal of Engineering Science and Technology, Vol. 3 No. 2 Feb 2011
[5]. A.R Raja Raja Varma, "Keralapanineeyam", Published by National Book Stall, Kottayam, July 1969.
[6]. Solak & K. Oflazer, Design and Implementation of a Spelling Checker for Turkish, Literary and Linguistic Computing, 8(3), 1993, 113-130.
[7]. Po-Ching Lin, Zhi-Xing Li, Ying-Dar Lin, "Profiling And Accelerating String Matching Algorithms in Three Network Content Security Applications", IEEE Communications Surveys & Tutorials, 2nd Quarter 2006.
[8]. Aho and M. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," Commun. ACM, vol. 18, no. 6, 1975, pp. 333–40.
[9]. R. Boyer and S. Moore, "A Fast String Searching Algorithm," Commun. ACM, vol. 20, no. 10, 1977, pp. 762–72.
[10]. R. Karp and M. Rabin, "Efficient Randomized Pattern-Matching Algorithms," IBM J. Research and Development, vol. 31, no. 2, Mar. 1987, pp. 249–60.
[11]. D. Knuth, J. Morris, and V. Pratt, "Fast pattern matching in strings," SIAMJ. Computing, vol. 6, pp. 323–350, 1977.
[12]. Ravi Sankar S Nair, Ph.D, "A Grammar of Malayalam",Language in India, www.languageinindia.com, ISSN 1930-2940, 12 : 11 November 2012.
[13]. symbolcodes.tlt.psu.edu/bylanguage/malayalamchart

APPENDIX I
SAMPLE VERB INPUTS
**വാചക? ിലെ ങ്കിയ പരിശോധന**

| വാചക? ിലെ വാ? ് | ൂപത?യം | വിഭാഗം |
|---|---|---|
| പോയി | ഇ | സാമാന?ഭൂതകാലം |
| ക? ു | ഉ | സാമാന?ഭൂതകാലം |
| വ? ു | ഉ | സാമാന?ഭൂതകാലം |
| കാണുകയായിരു? ു | ആയിരു? ു | നി? യഭൂതകാലം |
| പറയുകയായിരു? ു | ആയിരു? ു | നി? യഭൂതകാലം |
| ക? ുകൊ? ിരു? ു | കൊ? ിരു? ു | തുടർഭൂതകാലം |
| പറ? ുകൊ? ിരു? ു | കൊ? ിരു? ു | തുടർഭൂതകാലം |
| ക? ി?ു? ് | ഇ?ു? ് | ആസ? ഭൂതകാലം |
| പറ? ി?ു? ് | ഇ?ു? ് | ആസ? ഭൂതകാലം |
| കേ?ി?ു? ് | ഇ?ു? ് | ആസ? ഭൂതകാലം |
| വ? ിരു? ു | ഇരു? ു | പൂർ? ഭൂതകാലം |
| പോയിരു? ു | ഇരു? ു | പൂർ? ഭൂതകാലം |
| ക? ി?ു? ായിരു? ു | ഇ?ു? ായിരു? ു | പൂർ? ഭൂതകാലം |
| പറ? ി?ു? ായിരു? ു | ഇ?ു? ായിരു? ു | പൂർ? ഭൂതകാലം |
| പറ? ി?ു? ായിരി? ും | ഇ?ു? ാ? ിരി? ും | സ? ി? ധഭൂതകാലം |
| ക? ി?ു? ായിരി? ും | ഇ?ു? ാ? ിരി? ും | സ? ി? ധഭൂതകാലം |
| കാണുമായിരു? ു | ഉം+ ആയിരു? ു | ഹേതുഹേതുമ? ൂതകാലം |
| വരുമായിരു? ു | ഉം+ ആയിരു? ു | ഹേതുഹേതുമ? ൂതകാലം |
| കാണു? ു | ഉ? ു | സാമാന?വർ? മാനകാലം |
| പറയു? ു | ഉ? ു | സാമാന?വർ? മാനകാലം |
| കേൾ? ു? ു | ഉ? ു | സാമാന?വർ? മാനകാലം |
| ? നേഹി? ു? ു | ഉ? ു | സാമാന?വർ? മാനകാലം |
| കാണുകയാ? | ആ? | നി? യവർ? മാനകാലം |

| പറയാൻപോവുകയാ? | ആൻ + പോവുകയാ? | ആസ? ഭാവികാലം |
|---|---|---|
| കേൾ? ാൻപോവുകയാ? | ആൻ + പോവുകയാ? | ആസ? ഭാവികാലം |
| കാണുമായിരി? ു | ആയിരി? ു | സ? ി? ധഭാവികാലം |
| പറയുമായിരി? ു | ആയിരി? ു | സ? ി? ധഭാവികാലം |
| ക? ? ു | ഏ? ു | വിക? പഭാവികാലം |
| പറ? ? ാ | ഏ? ാ | വിക? പഭാവികാലം |
| ഓടുകയാ? | ആ? | നി? യവർ? മാനകാലം |
| വ? ു | തു | സാമാന?ഭൂതകാലം |
| കാണാൻപോവുകയാ? | ആൻ + പോവുകയാ? | ആസ? ഭാവികാലം |
| പറ? ു | തു | സാമാന?ഭൂതകാലം |
| അടി? ു | തു | സാമാന?ഭൂതകാലം |
| കേ?ുകൊ? ിരി? ു | കൊ? ിരി? ു | തുടർഭാവികാലം |

APPENDIX II
SAMPLE NOUN INPUTS

**വാചക? ിലെ നാമ പരിശോധന**

വിഭ? ി പരിശോധന

| **വാ? ്** | **വിഭ? ി വിഭാഗം** | **പ്രത?യം** |
|---|---|---|
| രാധയുടെ | സംബ? ിക | ുടെ |
| കു?ികൾ | നിർ?ശിക | പ്രത?യമില |
| അ? യുടെ | സംബ? ിക | ുടെ |
| വീ?ിൽ | ആധാരിക | ിൽ |
| വീ?ിൻറെ | സംബ? ിക | ൻറെ |
| അ? ൻറെ | സംബ? ിക | ൻറെ |
| കട | നിർ?ശിക | പ്രത?യമില |
| അവളുടെ | സംബ? ിക | ുടെ |
| അ? ?പികമാർ | നിർ?ശിക | പ്രത?യമില |
| ?ശീല? മി? ് | ഉ?ശിക | ? ് |
| സുമി?തയെ | പ്രതി?ഗാഹിക | എ |
| ദിവ?യുടെ | സംബ? ിക | ുടെ |
| അവരുടെ | സംബ? ിക | ുടെ |
| അവ? | ഉ?ശിക | ? |
| കൂ?ുകാരുടെ | സംബ? ിക | ുടെ |
| അവൾ? ് | ഉ?ശിക | ? ് |
| ഞാൻ | നിർ?ശിക | പ്രത?യമില |
| സിനിമ | നിർ?ശിക | പ്രത?യമില? |
| സിനിമ? ് | ഉ?ശിക | ? ് |
| ഷിജിന? ് | ഉ?ശിക | ? ് |
| ഷിജിനയെ | പ്രതി?ഗാഹിക | എ |
| അശ?തി | നിർ?ശിക | പ്രത?യമില |