CHAPTER 6

Computational Morphology and Manipuri

Computational morphology is one of the important parts of computational linguistics which deals with the processing of words in both their graphemic, i.e. written form and phonemic, i.e. spoken form [59]. It includes the analysis of word formation methods, morpheme segmentation, hyphenization and error correction etc. These tasks may appear very easy to a human but they create difficult problems to a computer program.

6.1 Roots and Affixes in Manipuri

As Manipuri is an agglutinative language, roots and affixes play a very important role in the processes of word formation. It is eagerly required to analyse the roots and affixes in the development of any computational linguistics applications like morphological analyser, POS tagger, etc. of the language.

6.1.1 Roots

A root is a form of word which cannot be further analysed either in terms of derivational and inflectional morphology. It is a base word, or a primary lexical unit of a word form that remains when all derivational or inflectional affixes are removed. Technically, it is the smallest unit of a word called morpheme and it cannot be reduced into smaller units. There are two types of root, viz; free and bound roots.

Free roots: Roots which can stand alone as a word without adding the affixes to it is called free roots. Most of the nouns in Manipuri are free roots.

e.g.; থা / thaa "moon" মী / mee "man"

চীঙ / cheeng "mountain" ঈশিং / eesing "water"

Bound roots: Roots which cannot stand alone as a word without adding the affixes to it is called bound roots. All verb roots in Manipuri are bound roots.

e.g.; পা /paa "read" চা /chaa "eat"

চৎ/chat "go" চেন/chen "run"

There are limited numbers of nominal bound roots in the language. Nominal bound roots are mainly found in the Manipuri kinship terms and animal body parts. Examples of kinship terms:

মা/ maa 'mother'

চে/ che 'sister'

Until and unless one of the personal pronominal ই-/i- (first person), ন-/na- (second person) and ম-/ma- (third person) is added as prefix, it does not give a complete meaning, for instance;

ইমা / i-maa 'my mother'

নমা / na-maa 'your mother'

মমা / ma-maa 'his/her mother'

Examples of animal body parts:

মচি / ma-chi 'horn'

মমৈ / ma-mei 'tail'

Here, -চি / -chi and -মৈ / -mei are the bound roots which cannot be stand independently as a word without prefixing the third person pronominal marker ম- / ma- .

6.1.2 Affixes

Affixes are bound morphemes which can only be attached to a root, stem and base word to form a new word. In an agglutinative language like Manipuri, affixes play very important role in the formation of various words and deriving several word classes. Prefixes and suffixes are two types of affixes in Manipuri. Both types are discussed below:

Prefix: A prefix is an affix which is placed before the root of a word, mostly used to form word class. Prefixes are very limited in Manipuri. There are limited numbers of prefixes available in the language. The following table shows the different types of prefixes used in Manipuri:

Table 6.1: Prefixes in Manipuri

Pronominal Prefixes	Non Pronominal Prefixes	
	Nominalizing Prefixes	Formative Prefixes
ই-/i- , ন-/na-, ম-/ma-	খু-/khu-, ম-/ma-	অ-/a-, ই-/i-, ম-/ma-,ত-/ta-, থ-/tha-, সে-/se-, পঙ-/pang, পুঙ-/pung, চ-/cha-, শুক-/suk, খঙ-/khang-

Examples:

য়ুম/yum	'house'	ইয়ুম/i-yum	'my house'
চা/chaa	'eat'	মচা/ ma -chaa	'the mode of his eating'
চা/chaa	'eat'	খুচা/khu-chaa	'the mode of eating'
তৌ/tou	'do'	খুতৌ/khu-tou	'the mode of doing'
শাংবা/saangbaa	'be long'	অশাংবা/ a- saang-baa	'long'

Suffix: A suffix is an affix which is placed after the root of a word. In Manipuri, suffix plays an important role to form new word class. So various word forms

can be constructed by suffixation of respective markers and various suffixes can be added one after another, by which meaning is also added. There are numerous number of suffixes used in Manipuri. Suffixes can be divided into two groups viz., nominal suffixes and verbal suffixes. The following table shows the nominal suffixes and verbal suffixes used in Manipuri.

Table 6.2: Suffixes in Manipuri

Nominal suffixes	-ন/-na, -বু/-bu, -পু/pu, -গী/-gee, -কী/-kee, -দ/-da, -ত/-ta, -দ্গী/-dagee, -গ/-ga, -ক/-ka, -শিং/-sing, -খোয়/-khoy, -নে/-ne, -সু/-su, -র/-ra, -ল/-la ,-দি/-di, -তি/-ti, -তু/tu, -দু/-du, -মক/-mak, -মুক/-muk, -রক/-rak, -সে/-se, -নে/-ne, -লক/-lak, -রক/-rak, -নাউ/-naau, –নাও/-naao
Verbal suffixes	-য়/-y, –ই/-i, -রি/-ri, -লি/-li, -রে/-re, -লে/-le, -নিঙ/-ning, -নিং/-ning, -লোয়/-loy, -লোই/-loi, -রোয় /-roy, –রোই/-roi, -দে/-de, -তে/-te, -নু/-nu, -লো/-lo, -রো/-ro, –রু/-ru, -য়ু/-yu,-উ/-u, -মু/-mu, -পু/-pu, -ঙু/ngu-, -ঙো/-ngo, -পো/-po, -মো/-mo, -ও/-o, -র/-ra, -ল/-la, -রক/-rak, -খি/-khi, -রুই/-rui, -গে/-ge, -কে/-ke, -শি/-si, -সি/-si, -সনু/-sanu, -ন/-na, -না/-naa, -শিন/-sin, -জিন/-jin, -শোক/-sok, -শুক/-suk, -ব/-ba, -বা/-baa, -প/-pa, -পা/-paa, -রোক/-rok, -লোক/-lok, -শং/-sang, -শিৎ/-sit, -নি/-ni, -শিন/-sin, -থোক/-thok, -খ/-kha, -সন/-san, -শন/-san, -চন/-can, -জন/-jan, -দোক/-dok, -তোক/-tok, -খৎ/-khat, -খত/-khat, -গৎ/-gat, -গত/-gat, -কৎ/-kat, –কত /-kat, -থৌ/-thou, -খায়/-khaay, –খাই/-khaai, -গায়/-gaay, –গাই/-gaai, -থক/-thak, -থৎ/-that, -কায়/-kaay, কাই/-kaai, -থেক/-thek, -তৎ/-tat, -দৎ/-dat, -দেক/-dek, -মিন/-min, -মন/-man, -মল/-mal, -হন/-han, -হল/-hal, -বী/-bee, -পী/-pee, -বি/-bi, -পি/-pi, -কো/-ko, -কন/-kan, -গন/-gan, -হৎ/-hat, -চ/-ca, -জ/-ja, -কুম/-kum, -গুম/-gum, -লম/-lam, -রম/-ram, -নে/-ne, -ফেৎ/-phet, -প্রেক/-prek, -ঙৈ/-Ngei

Examples of nominal suffixes used in Manipuri:

উচেক/uceck ‘bird’ উচেকশিং/uceck- sing ‘birds’

নিংথৌ/ningthou	'king'	নিংথৌশিং/ningthou- sing	'kings'
ঐ/ei	'I'	ঐখোয়/ ei- khoy	'we'
নঙ/nang	'you'	নখোয়/na- khoy	'you (pl)'
মহাক/mahaak	'he'	মখোয়/ma-khoy	'they'
মনি/mani	'Mani'	মনিখোয়/mani-khoy	'Mani and others'
রাজু/raju	'Raju'	রাজুগী/raju-gi	'Raju's'

Examples of verbal suffixes used in Manipuri:

চা/chaa	'eat'	চারি/chaa-ri	'eating (progressive)'
চা/chaa	'eat'	চারে/chaa-re	'eaten'
চৎ/chat	'go'	চৎতে/chat-te	'does not go'
চৎ/chat	'walk'	চৎপা/chat-pa	'walking (vebal noun)'

6.2 Word Formation in Manipuri

In Manipuri, new words are formed by the following word formation processes. They are affixation, derivation and compounding. The majority of the roots found in the language are bound and the affixes are the determining factor to fix the classes of words in the language. The three processes of word formation in Manipuri are discussed below.

6.2.1 Affixation

Affixation is the morphological process of forming a new word by addition of prefix or suffix to an already existing morpheme or root or word. In Manipuri, more than one suffix can be added to the root or morpheme or word. Some rules of the formation of new words by affixation are discussed below:

Formation of adjective (MJ):

1. In Manipuri, most of the adjectives are formed by affixation of formative prefix "অ / a" and nominalizer "বা / baa" or "পা / paa" as suffix to a verb root (VR).

   a + verb root + baa/paa → adjective

   অ/ a + ফ/pha + বা/baa → অফবা / a-pha-baa 'good'

   অ/ a + শাং/saang + বা/baa → অশাংবা / a-saang-baa 'long'

   অ/ a + পাক/paak + পা / paa → অপাকপা / a-paak-paa 'wide'

The above rule can be coded as:

```
if (word.startsWith ("a") && word.endsWith ("baa"))
    {
    word.tag.set ("MJ")
    }
```

2. In the case of polysyllabic verb root, an adjective is formed without the formative prefix "অ / a".

   verb root + baa → Adjective

   ফজ / phaja + বা / baa → ফজবা/ pha-ja-baa 'beautiful'

   হরাও / haraw + বা / baa → হরাওবা / haraw-baa 'joyful'

3. Adjective can also be formed by attaching a suffix between verb root and nominalizer.

   verb root + suffix + baa → adjective

   শাৎ/saat + লি/li + বা/baa → শাৎলিবা/saat-li-baa 'blooming'

চেন/chen +লি/ li+বা/baa → চেনলিবা/chen-li-baa ‘running’

পৎ/pat + ল/la + বা/baa → পৎলবা/pat-la-baa ‘rotten’

Computational Model of the above rule can be written as:

```
if (word.tag.startsWith (“VR”)
                    && word.contains (suffix)
                    && word.endsWith (“baa”))
        {
        word.tag.set (“MJ”)
        }
```

Formation of adverb (ADV):

In Manipuri adverb is formed by addition of case marker “না/na” as suffix to verb root.

verb root + naa → adverb

তপ/tap + না /naa → তপনা/ tap-naa ‘slowly’

য়াং/yaang + না/naa → য়াংনা/yang-naa ‘quickly’

কপ/kap +না/naa → কপনা/ kap-naa “cryingly”

Computational Model of the above rule can be written as:

```
if (word.tag.startsWith (“VR”) && word.endsWith (“naa”))
        {
        word.tag.set (“ADV”)
        }
```

6.2.2 Compounding

It is also a word formation process in which a word is formed by joining more than one free roots or words. In Manipuri, it may also be the combination of free and bound roots. Most of the compound words in Manipuri are nouns. Some examples of compound words are given below:

1. ***noun + noun → noun***

   চাক/chaak + শঙ/sang → চাকশঙ/ chaak-sang

   'rice' 'shade' 'kitchen'

   য়োৎ/yot + চৈ/chei → য়োৎচৈ/yotchei

   'iron' 'stick' 'iron rod'

2. ***noun + verb root → noun***

   ৱা/waa + হং/hang → ৱাহং/waa-hang

   'word' 'ask' 'question'

   খোং/khong + চৎ/chat → খোংচৎ/khong-chat

   'leg' 'go' 'trip'

3. ***noun + augmentative → noun***

   য়ুম/yum + চাও/chaw → য়ুমজাও/yum-jaw

   'house' 'big' 'big house'

   উ/u + চাও/chaw → উজাও/u-jaw

   'tree' 'big' 'big tree'

4. ***noun + nutive → noun***

ষন/san + নাও/naw → ষননাও/san-naw

'cow' 'small' 'calf'

থোং/thong + নাও/naw → থোংনাও/ thong-naw

'door' 'small' 'window'

5. ***noun + adjective → noun***

উ/u + অশাংবা/asaangba → উশাং/u-saang

'tree' ' long' 'long tree'

ফি/fi + অঙৌবা/angouba → ফিঙৌ/fi-ngou

'cloth' 'white' 'white cloth'

6.2.3 Derivation

Derivation is a morphological process of forming a new word from an existing word by addition of derivational affixes and the word category of the existing word changes to another category. In Manipuri, there are cases of derivation of nouns from verbs by the addition of the derivative suffix "বা / baa" or "পা / paa" to the verb root directly.

verb root + baa → verbal noun

চেন/chen + বা/baa → চেনবা/chen-ba 'running'

চৎ/chat + পা/paa → চৎপা/chat-paa 'walking'

চা/chaa + বা /baa → চাবা/chaa-baa 'eating'

Computational Model of the above rule can be written as:

***if (word.tag.startsWith ("VR") && word.endsWith ("*baa")*)**

{

word.tag.set ("MJ")

}

6.3 Morpheme Segmentation in Manipuri

It is generally agreed among the researchers that morpheme segmentation is a necessary first step in agglutinative languages like Malayalam, Tamil, Basque, Finnish and Manipuri etc. Most of the words in Manipuri are formed by the morphological processes called affixation, compounding and derivation. Therefore, it will almost certainly have to segment the sequence of characters in the word into morphemes. This is not a trivial task. Although certain simple cases look uncomplicated:

চৎলি/ chatli → চৎ/ chat [VR] + লি/li [PRG] 'going'

উচেকশিং/uchek → উচেক/uchek [NC] + শিং/sing [PL] 'birds'

The segmentation cannot be done by spotting a familiar affix and detaching it – consider the following examples:

ঈশিং/ eesing → ঈ/ee [NC] + শিং/sing [PL]

Here, the word ঈশিং/eesing is water in English, it is the valid Manipuri word having its own POS as Common Noun (NC). But it is wrongly segmented in the above example.

চৎপদ্গী/chatpdagee → চৎ/cat [VR] + প/pa [NMZ] + দ/da [LOC] + গী/gee [GEN]

Similarly, the above example is also wrong segmentation process though 'chat', 'pa', 'da' and 'gi' all are the suffixes. It should be segmented as shown below:

চৎপদ্গী/chatpadagee → চৎপা/chatpaa [NV] + দগী/dagee [ABL]

Therefore, many linguistics rules are applied for handling such kind of ambiguity issues facing in the language.

A further complication for the segmentation process is the fact that minor alterations in spelling often occur at the boundaries between morphemes. For example:

চৎপগী/chatpagee → চৎপা/chatpaa + গী/gee 'to go'

ফম্বগী/fambagee → ফমবা/fambaa + গী/gee 'to seat'

There are lots of long words formed by affixation which is eagerly needed to segment into morphemes. For example:

চৎলুরবনি → চৎ লু র ব নি

catlurabani → cat lu ra ba ni 'has been gone'

তৌহনজরমগদবনিকো → তৌ হন জ রম গ দ ব নি কো

touhanjaramgadabaniko → tou han ja ram ga da ba ni ko

'I could cause him to do'

In Manipuri, it is especially necessary to apply different linguistics rules to overcome the wrong segmentation of morphemes. There are many stemming algorithms viz; affix removal stemming algorithms, statistical stemming algorithms and mixed stemming algorithms which are applied in morpheme segmentation process. Among them affix removal stemming algorithms are more suitable for an agglutinating language like Manipuri.

6.3.1 Affix Removal Stemming Algorithms

The theory of affix removal stemming algorithm is to remove the endings of the word keeping first n letters i.e. to shorten a word up to nth character and remove the rest. In this method words shorter than n are kept as it is. The chances of over stemming increase when the word length is small. Different affix removal stemming algorithms are discussed below.

6.3.1.1 Lovins Stemming Algorithm

This was the first popular and effective stemmer proposed by Julie Beth Lovins of Massachusetts Institute of Technology in 1968. It performs a look up on a table of 294 endings, 29 conditions and 35 transformation rules, which have been arranged on a longest match principle [54]. The Lovins stemmer removes the longest suffix from a word. Once the ending is removed, the word is recoded using a different table that makes various adjustments to convert these stems into valid words. It always removes a maximum of one suffix from a word, due to its nature as single pass algorithm. The advantages and disadvantages of this algorithm are given in the table below [46]:

Table 6.3: Advantages and disadvantages of Lovins Stemmer

Advantages	Disadvantages
1. Fast – single pass algorithm. 2. Handles removal of double letters in words like 'getting' being transformed to 'get'. 3. Handles many irregular plurals like – mouse and mice etc.	1. Time and data consuming. 2. Not all suffixes available 3. Not very reliable and frequently fails to form words from the stems. 4. Dependent on the technical vocabulary being used by the author.

6.3.1.2 Porter Stemming Algorithm

Porter's algorithm is most popular and widely used for stemming English that has repeatedly shown to be empirically very effective [61]. The original algorithm consists of 5 phases of word reduction. Each phase has a set of rules written beneath each other, among which only one is obeyed. The rules for removing a suffix will be given in the form

(condition) S1 → S2

This means that if a word ends with the suffix S1 and the stem before S1 satisfies the given condition, S1 is replaced by S2. The condition is usually given in terms of m, e.g;

(m>1) EMENT →

Here S1 is 'EMENT' and S2 is null. This would map REPLACEMENT to REPLAC, since REPLAC is a word part for which m=2 [50].

The 'condition' part may also contain the following:

*S – the stem ends with S (and similarly for the other letters).

v- the stem contains a vowel.

*d – the stem ends with a double consonant (e.g. –TT, -SS).

*o – the stem ends with cvc, where the second c is not W, X or Y (e.g. –WILL, -HOP)

(*d and not (*L or *S or *Z))

tests for a stem ending with a double consonant other than L, S or Z [36]. Dr. Porter himself has suggested several improvements to the original algorithm. The suggested changes are:

i. Terminating 'y' changed to 'i' seldom occurrence.

ii. Suffix 'us' does not lose its 's'.

iii. Removal of additional suffixes, including suffix 'ly'.

iv. Add step 0 to handle apostrophe.

v. A small list of exceptional forms is included.

Although, these changes do not make the algorithm very extensive, however, failed to improve the performance and minimize the errors to great extend. As shown in table 2, the accuracy of correctly stemmed words increased only from 31.9% to 34.76% [61]. The advantages and limitations of this algorithm are given in the table below [46]:

Table 6.4: Advantages and limitations of Porter Stemmer

Advantages	**Limitations**
1. Produces the best output as compared to other stemmers. 2. Less error rate. 3. Compared to Lovins it's a light stemmer. 4. The Snowball stemmer framework designed by Porter is language independent approach to stemming.	1. The stems produced are not always real words. 2. It has at least five steps and sixty rules and hence is time consuming.

6.3.1.3 Dawson Stemming Algorithm

The Dawson Stemmer was developed by J.L. Dawson of the Literary and Linguistics Computing Centre at Cambridge University in the year 1974. It is an extension of Lovins approach except that it has much more comprehensive list of about 1200 suffixes. Like Lovins, it is also a single-pass context-sensitive suffix removal stemmer. The main objective of the stemmer was to refine the rule sets and techniques originally proposed in Lovins stemmer and to correct any basic errors that exist. It has two phases.

Phase I: All plurals and combinations of the simple suffixes are included. This increases the size of the ending list to approximately five hundred.

Phase II: The Dawson stemmer has employ the completion principle in which any suffix contained within the ending list is completed by including all variants, flexions and combinations in the ending list. This increased the ending list once more to approximately one thousand two hundred terms.

The Dawson stemmer applies the technique of partial matching which attempts to match stems that are equal within certain limits. The advantages and disadvantages of this algorithm are given in the table below [46]:

Table 6.5: Advantages and disadvantages of Dawson Stemmer

Advantages	Disadvantages
1. Covers more suffixes than Lovins. 2. Fast in execution.	1. Very complex. 2. Lacks a standard implementation.

6.3.1.4 Paice/Husk Stemming Algorithm

The Paice/Husk Stemmer was developed by Chris Paice at Lancaster University in the late 1980s and was originally implemented with assistance from Gareth Husk. It is a simple iterative Stemmer – that is to say, it removes the endings from a word in an indefinite number of steps. The Stemmer uses a separate rule file, which is first read into an array or list. This file is divided into a series of sections, each section corresponding to a letter of the alphabet. The section for a given letter, say "e", contains the rules for all endings ending with "e", the sections being ordered alphabetically. An index can thus be built, leading from the last letter of the word to be stemmed to the first rule for that letter.

When a word is to be processed, the stemmer takes its last letter and uses the index to find the first rule for that letter. The rule is examined, and is accepted if:

- It specifies an ending which matches the last letters of the word.
- Any special conditions for that rule are satisfied (e.g, the so-called 'intact' condition, which ensures that the rule is only fired if no other rules have yet been applied to the word).
- Application of the rule would not result in a stem shorter than a specified length or without a vowel.

If a rule is accepted then it is applied to the word. If it is not accepted, the rule index is incremented by one and the next rule is tried. However, if the first letter of the next rule does not match with the last letter of the word, this implies that no ending can be removed, and so the process terminates.

When a rule is applied to a word, this usually means that the ending of the word is removed or replaced. For example, the rule e1> { -e - }.

It means, if the current word/stem ends with "e" then delete 1 letter and continue' (the curly brackets just contain a comment showing the rule in another form). So this is a simple 'e-removal' rule, which for example would convert "estate" to "estat". After applying this rule, the new final letter (now "t") would be taken and used to access a different section of the rule table. If, however, the final symbol had been "." instead of "> ", the process would have terminated, and "estat" would have been returned at once. Suppose now that the rule had said:

e1i> { -e -i }

In this case, the "e" would have been removed and then replaced by the letter "i" – giving, in the present case, "estati".

Once a rule has been found to match, it is not applied at once, but must first be checked to confirm that it would leave an acceptable stem. For example, it would not be sensible to apply the 'e-removal' rule to the word "me", since the remaining stem would be too short - and would not even contain a vowel [41]. The advantages and disadvantages of this algorithm are given in the table below [46]:

Table 6.6: Advantages and limitations of Paice/Husk Stemmer

Advantages	**Disadvantages**
1. Simple form. 2. Each iteration takes care of deletion and replacement.	1. Heavy algorithm. 2. Over stemming may occur.

6.3.2 The Proposed Affix Stripping Algorithm for Manipuri

Before discussing the proposed model, it can be noted that most of the Manipuri words are formed by attaching a prefix which always stand for an independent meaningful word with a different POS category, so it is not required to separate the prefix. So the model gives emphasis on suffix stripping technique.

In this model, a list of all the 132 suffixes of Manipuri as mentioned in the Table 6.2 and a lexicon containing morpheme and its POS category are prepared. Many linguistics rules i.e. orthographic rules, disambiguation rules and other morphological rules are applied to segment the word into morphemes with an accuracy rate.

The architecture of the proposed model is as follows:

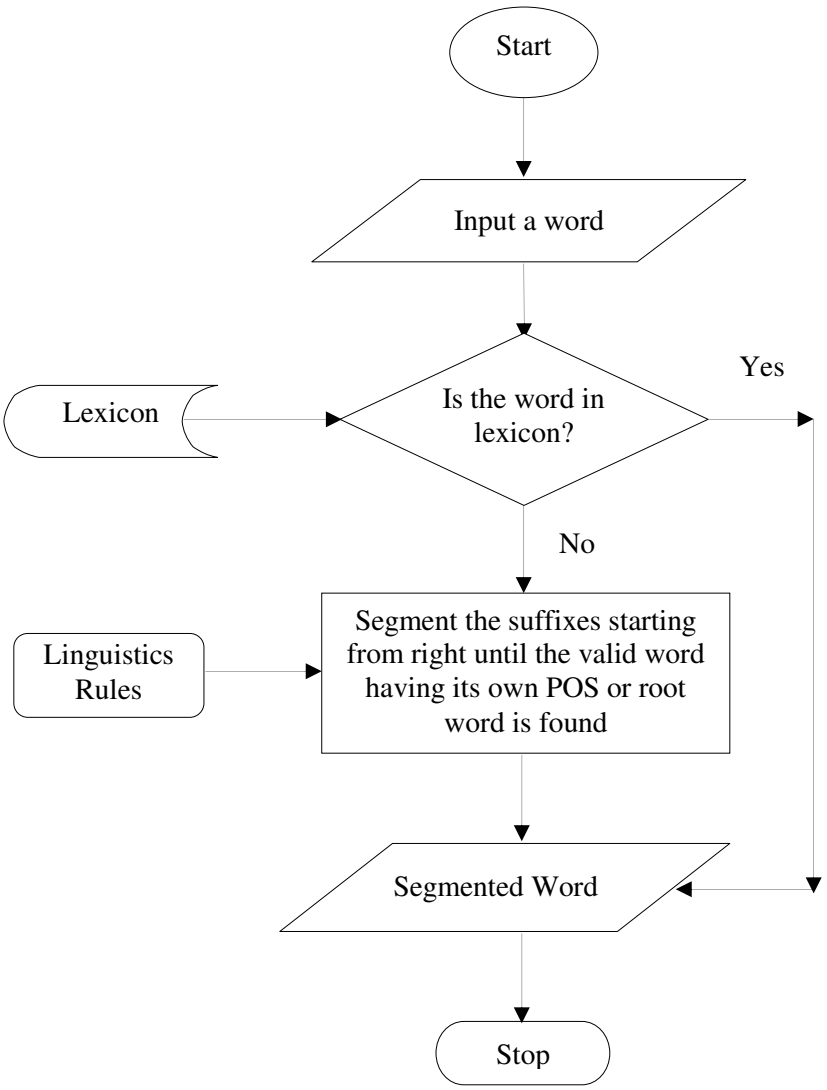

Figure 6.1 Architecture of proposed morpheme segmenter

The step by step procedure of the proposed model is given below:

Step I: Enter the word.

Step II: Check whether the entered word is availability in the lexicon.

Step III: If the word is found in the lexicon then the entered word itself will be the output. Because the lexicon contains only the morpheme, root word and the valid word which has its own POS category.

Step IV: If the entered word is not found in the lexicon then a greedy search routine walks through the word trying to find the morpheme starting from the right side of the entered word that matches a suffix in the suffix list. When it finds the suffix, it will stop searching and inserted a space between the suffix and the remaining word. Step I, II, III and IV will be repeated for the remaining word until and unless the valid word or morpheme or root word is found.

Step V: The segmented word is displayed as output.

6.3.3 Experimental Results and Discussions

By applying the above algorithm, a morpheme segmenter tool is developed. One thousand five hundred individual words are tested and attain a good performance of accuracy. In order to measure the performance of the system, an evaluation set of manually segmented words is used. Accuracy percentage of the segmenter is calculated using the simple formula given below:

$$\text{Accuracy Percentage} = \frac{\text{Correctly segmented words}}{\text{No. of words in evaluation set}} \text{ X } 100$$

Results generated by applying different nos. of rules to a standard test of 1500 words are given in the table below:

Table 6.7: Segmentation experimental Results

No. of rules applied	Size of Lexicon (in words)	No. of Words in evaluation set	No. of correctly segmented words	Accuracy Percentage %
0	200	1500	1000	66%
10	200	1500	1100	73%
20	200	1500	1250	83%
35	200	1500	1450	96%

It gives the accuracy of 96% and it is clear that accuracy percentage is increased with the increment of the number of linguistics rules applied. The screenshot view of the Manipuri Morpheme Segmenter tool is shown below:

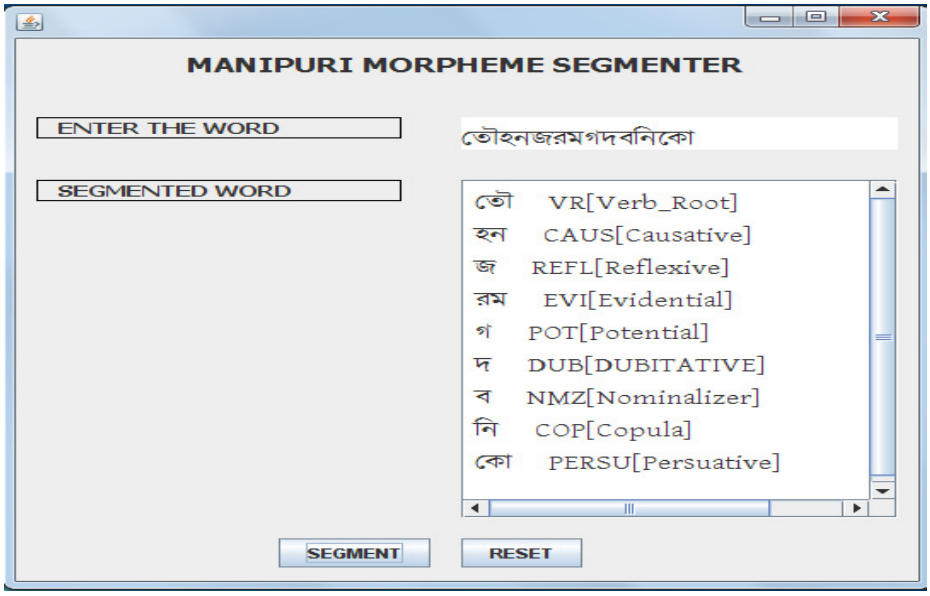

Figure 6.2 Manipuri Morpheme Segmenter

6.4 Chapter Summary

In this chapter, general definition of computational morphology is discussed. It then gives a brief description on roots and affixes of Manipuri clearly. Three major word formation processes viz., Affixation, Compounding and Derivation are discussed with examples. It also explains different linguistics rules which help to handle the word segmentation issues commonly faced in Manipuri. Again this chapter discusses different algorithms of affix stripping and proposed a new affix stripping algorithm. A morpheme segmenter tool is developed by applying the proposed algorithm. Further, the chapter presents some experimental results.