2014 International Conference on Advances in ICT for Emerging Regions (ICTer): 190 - 196

Morphological Analyzer and Generator for Tamil Language

S. Lushanthan[#1], A. R. Weerasinghe[#2], D.L. Herath[*3]

[#]*University of Colombo School of Computing,*
35, Reid Avenue, Colombo 07, Sri Lanka
[1]*lushanthanucsc@gmail.com*
[2]arw@ucsc.cmb.ac.lk

[*]*University of Queensland,*
Brisbane, Australia
[3]d.herath@uq.edu.au

***Abstract*— Morphological analysis is an essential component in Natural Language Processing (NLP) applications ranging from spell checker to machine translation. When performing a morphological analysis it leads to segmentation of a word into morphemes, combined with an analysis of the attachments of these morphemes. In English language the complexity of the formation of words is not much higher compared with Indic languages. Hence, Tamil language too does have its complexities when building up a NLP application. The morphemes in the language, the rules how these morphemes are connected and the changes occur when they attach together are the important factors that need to be considered when building up a Morphological Analyzer for any language. Our "Morphological Analyzer and Generator for Tamil Language" will be generating the word forms of a stem/ root, given a particular context and at the same time, a surface form in Tamil language should get analyzed into its proper context. This model tries to cover only the nouns and verbs in the Tamil language.**

This paper illustrates how the lexicon and the orthographic rules of Tamil language have been written as regular expressions using only finite state operations and how this approach has been implemented to develop a morphological analyzer/generator. This model is built using the Xerox toolkit, which uses "Two-level Morphology", and almost 2000 noun stems and 96 verb stems have been incorporated into the network. A noun stem now produces about 40 different forms and a verb stem produces up to 240 forms. We have also defined our own transliteration scheme for this purpose.

***Keywords*—Tamil Morphological Analyzer and Generator, Morphology, Finite State Transducer, Regular Expressions**

I. INTRODUCTION

Though Tamil Language has received focus in the field of Natural Language Processing, extensive analysis and experimentation have not been done so far. In order to build up any sort of NLP application for Tamil language, a morphological analyzer has almost become very essential. Like other Indic languages, Tamil language too faces its challenges in defining the morphotactics and the orthographic rules. Morphotactics is the order in which the morphemes combine. Orthographic rules models the changes when two morphemes combine. In other words, these are the discrepancies between the underlying abstract strings and the surface strings. Thus, Morphological analysis is simply a model that maps between the *surface form* and the *lexical form* of words.

Since Tamil is morphologically very rich, question may arise whether Tamil Morphology can be modelled using only finite state operations. So in order to clarify this, we built a model and tested whether it produces the appropriate results that are accepted by the language.

In a Morphological model we need three major components. The *lexicon* which has the stem and the affixes, the *Mophotactics* which defines the order of the morphemes for a particular language (In English language, plural morpheme follows noun stem) and the *Orthographic rules* or the spelling rules, which explains the variations found in the final surface form. We will discuss these in detail for Tamil language in section 5.

Our Model is based on finite state approaches to Morphology similar to [3] [8] [9], where the authors use regular expressions to model morphotactics and the orthographic rules. It is originally based on the implementation known as Two-level Morphology [2], which was significantly successful for many languages like Finnish [1] [2], French, English [7], Portuguese, Italian, Turkish and Arabic [4]. The two-level Morphology is implemented using Finite State Transducers (FST). A finite state network/machine consists of states, including a single start state and one or more final states. Transitions between states are possible only if the required input is recognized. A Path is a sequence of transitions over arcs to a particular state. A finite state transducer is a finite state machine that provides a set of outputs from an accepted input and expresses the relations between languages. In a FST we can analyze (look up) and generate (look down). Here, input is the lower side of symbol and output is the upper side of symbol. Thus, FST creates relations between strings. The finite state transducers built at Xerox are inherently bidirectional - there is no privileged input side [3]. We have implemented our model on the non-commercial Xerox tool and later we look forward to implementing in open platforms such as Open FST library[1] or foma[2]. Our model tries to cover the nouns and verbs in Tamil language.

The following sections of the paper are structured as follows. In Section 2, we will define our scope. Section 3 will speak of the related work. In section 4 we will focus on regular expression grammars. Section 5 and 6 will be about the implementation and the evaluation process

[1] http://www.openfst.org
[2] https://code.google.com/p/foma/

978-1-4799-7732-1/14/$31.00 ©2014 IEEE

respectively. Section 7 will illustrate the results of our model. Section 8 will be the conclusion and Section 9 speaks of the future work that we have intended to carry out. Finally, we will acknowledge the parties who have helped us in making this research worth.

II. Scope

The noun/ verb categories, suffixes and roots were basically identified according to Tamil literature and later altered to suit our need. Our system is only limited to the nouns and verbs in written Tamil. The Morphological Analyzer and Generator (MAG) for nouns incorporates 40 forms, which is almost equal or close to the maximum possible number of basic single word forms that can be generated from a noun stem. However, given a verb stem, one cannot easily predict the number of verb forms that can be generated in Tamil language. We have mainly focused our system towards Inflectional Morphology, though we have tried to a certain extent to cover the Derivational Morphological forms of a verb stem. We have networked some important negative Adverbial forms as well. Further, our system only generates forms for single word stems.

Our system does not cover the spoken Tamil system. We have almost omitted the negative forms of a verb action as one form is almost the same to all the combination of tags. This will not cause any problems in the generation process, but it will be a major drawback in the analysis process. For example, a negative surface verb form may not get correctly analyzed. Moreover, as we are not modelling sandhi rules between two words we have not focused on that. However in general, the words which are joined using sandhi rules behave the same way the later word behaves. For example, the word "kaTirî" and the word "marakkaTirî" behave the same way. Thus, they can reside in the same category. Now our system covers up to a maximum 240 types of verb forms in a category.

III. Related Work

A. The Origins

Kimmo Koskenniemi for his dissertation [2] in 1983 proposed his constraint-based model and implemented it and showed that he did not need or depend on a rule compiler, composition or any other finite state algorithms. He named it "The Two- Level Morphology". The two-level system was based on a lexicon system and a set of two-level rules. Rather than using cascaded rules with intermediate stages, rules can be thought as statements that directly constrain the surface realization of lexical strings. The rules were applied in parallel. The two- level model is also fully bi-directional both conceptually and processually. The model is based on computationally simple machinery, mostly on finite state automata. His model was evaluated with Finnish language, which is morphologically very rich.

Koskenniemi's two-level morphology was based on three major ideas:

- Rules are symbol-to-symbol constraints that are applied in parallel, not sequentially.
- The constraints can refer to the lexical context, to the surface context, or to both contexts at the same time.
- Lexical lookup and morphological analysis are performed one after the other.

Here the relation between lexical and surface form is seen as correspondence, not as a transformation between different segments. (i.e.) They are symbol-to-symbol constraints, not string-to-string relations like general rewrite rules. The two-level model deals with two stages of the whole system like the Kaplan and Kay model: the lexicon and the surface representation. Thus, this model is highly influenced by the Kaplan and Kay model. The main difference is that there are no-intermediate stages between the lexicon and the surface forms in the two-level model, and that relation is generated with the rules applied in parallel which may refer to both of these two stages.

The levels are taken as two separate tiers written above each other. The lexical representation with its stems and suffixes will be stored in the upper row; where as in the lower row will be the morphemes of the surface form. The upper and the lower row characters are mapped pair-wise to each other.

B. Finite State Morphology

In Natural Language Processing finite state transducers have become an unavoidable aspect as many morphological analyzers have been developed using the finite state technology for many complex languages.

Arabic is such a challenging language that forced constraints on Natural language processing applications. These particular challenges of Arabic, and the limitations of some implementations of finite-state morphology, have made researchers to believe that this technology was not sufficient to handle Arabic. The work presented in [9] mainly addresses the problems faced when describing the morphotactics and the variation rules in Arabic. The paper argues that, though many researchers have concluded that finite state technology is not suitable for describing Semitic root- and-pattern morphology, Arabic root-and-pattern interdigitation can be performed through intersection and composition operations available in the Xerox technology. The most difficult challenge was to handle the weak roots. The system contains 4930 Arabic roots, 72,000,000 abstract, fully- voweled words, sixty-six finite-state variation rules and some other rules as well.

The issues that arise when building a finite state morphological analyzer for Urdu and Hindi are discussed in [6]. Potential ambiguity issues, handling reduplication which is very much common among South Asian languages, tokenization problems in Urdu grammar especially in future morphology of Urdu/ Hindi are three major issues that are discussed in that paper. The authors implement various techniques and finite state methods into the Urdu/ Hindi morphological analyzer in order to eliminate ambiguity and reduplication problems and propose solutions that would fit Urdu Grammar. They have implemented the methods in [8] (referred to as B&K) for Urdu and Hindi. Specially, the methods for introducing a hyphen and multi character brackets for reduplicated words and to specify their domains, bracket filter of B&K for eliminating unmatched brackets, flag diacritics notation (one example format is `@P.feature.value@` which sets and resets the feature to the given value) when introducing replace rules for Urdu echo reduplication are such examples. They introduce a set of replacement rules for Urdu morphology for echo reduplication. Other than the methodologies of B&K they also use some other standard techniques from previous

researches. The Hindu Urdu Machine Transliteration System (HUMTS) was used in the research. It is written as a cascade of finite-state transducers that transliterate from the Urdu and the Hindi scripts to SAMPA and vice-versa. Add to this, Malik's work on dealing with tokenization problems was used especially for the future morphology in Urdu and Hindi. Realizations from earlier researches of German Pargram grammar project were also referred. It was helpful in dealing with ambiguity in the syntax phase rather than in the morphology phase by adding some functional information to the `+Inf` tag and eliminating unwanted overflows based on the contextual information at the syntax. Glassman's transliteration system is another important technique used in that research. The research tries to focus mainly on the problems when it comes to building a morphological analyzer for Urdu and the issues in integrating this into the morphology-syntax interface.

The HAMSAH model [5] addresses the challenges faced when applying finite state morphology on Hebrew. It uses the XFST technology. The system is composed of two major components, a lexicon represented in Extensible Markup Language (XML) and a set of finite state rules implemented on XFST. HAMSAH is based on a lexicon of more than 20,000 entries, which is being constantly updated. The set of rules covers almost all the morphological, morpho-phonological and orthographic phenomena observed in modern Hebrew text. It produced a 91% success rate.

C. Tamil Morphological Analyzers – The Need

Although there are many versions of Morphological Analyzers and Generators reported in the web, only ATCHARAM and Amrita's MAG have demo versions. No tool other than Amrita's MAG is open source. It uses AT&T finite state machine techniques.

Machine learning approaches have also been used in building up Tamil Morphological Analyzers where SVM tool has been used for the classification task [11]. In this model, rules are handled using classification. Though better results have been obtained using this method, the training period for this model is long compared to other models.

IV. Regular Expressions

In order to build a Morphological Analyzer using finite state power, writing the morphotactics and the orthographic rules in terms of regular expressions must be a known factor. One of the results of formal language theory is the demonstration that finite state languages are precisely the set of languages that can be described by a regular expression.

A regular expression denotes a set of strings (i.e. a language) or a set of ordered string pairs (i.e. a relation). It can be compiled into a finite state network that compactly encodes the corresponding language or relation that may well be infinite. The language of regular expressions includes the common set of operators of Boolean logic and operators such as concatenation that are specific to strings. It follows from Kleene's theorem that for each of the regular expression operators for finite state languages, there is a corresponding operation that applies to finite state networks and produces a network for the resulting language. Any regular expression can in principle be compiled into a single finite state network.

The Xerox implementation of finite-state morphology includes a complete range of fundamental algorithms (concatenation, union, intersection etc.) plus higher level languages like lexc, twolc compiler, and Replace Rules [3] [8].

A. Finite State Operations

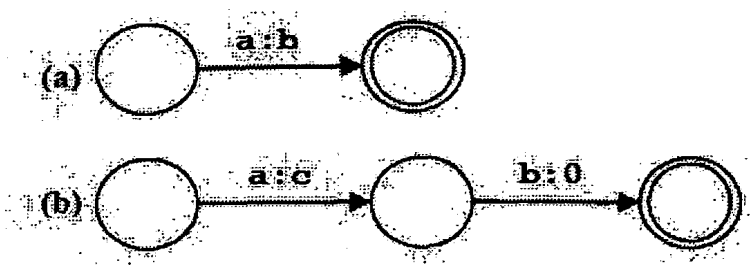

Fig. 1 Cross-product (a) a.x.b (b) [a b] .x. c

If `A, B` are regular languages then the cross-product of A and B which is denoted as `A.x.B`, relates to every string in the upper language `A` to every string in the lower language `B`, and vice versa. So, `A.x.B` denotes a relation between the two languages `A` and `B`. If `u` and `l` are symbols respectively from the upper and lower languages, then `u:l` is equivalent to `u.x.l`. The cross-product of two languages may also produce pairs of strings that are of different length, as in the second example, figure 1(b). The pair of strings <"ab", "c" > could be encoded in a network in other ways; for example, by a path in which successive labels are `a:0` and `b:c` instead of `a:c` and `b:0` as in figure 1(b). However, because it is convenient to have a unique encoding for cross product relations, the Xerox compiler pairs the strings from left to right, symbol to symbol, and introduces one-sided epsilon pairs only at the right end of the path if needed.

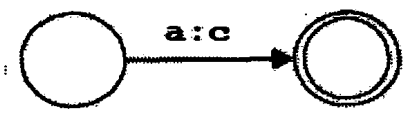

Fig. 2 Composition- a:b .o. b:c

Composition operator is different from cross-product. It is an operator that deals with relations and not with languages. Let `A`, `B` and `C` be regular languages. If `X` denotes the relation between the upper side language and the lower side language, namely `A` and `B`; and if `Y` denotes the relation between the upper side language `B` and the lower side language `C`; then the Composition of `X` and `Y`, written as `X.o.Y` will denote some other regular relation say `Z`, that maps between the languages `A` and `C`. Here the intermediate language gets disappeared.

B. Regular expression grammars

1) *Concatenative Morphotactics :* Morphemes normally concatenate together with the stems. For example, the present continuous morpheme `-ing`, third person singular morpheme `-s` and the past morpheme `-ed` can be written as concatenation of `[ {ing} | s | {ed} | 0 ]` for English verbs. Here the final symbol, denotes the epsilon (in Xerox, 0 symbol is used) which has no morpheme. `[{abc}]` is realized as `[ a b c]` by the Xerox tool as different symbols. Assume we are trying to model a network for the verb stems `kick`, `try` and `bore` with the above given morphemes concatenating with them. When concatenating, the network looks like, `[{kick} | {try} | {bore} ] [ {ing} | s | {ed} | 0 ].`

However, we define tags like +Verb, +Prog (present continuous), +Pres3PSg (third person singular), +Past, +Bare (stem itself) to increase the readability. So, for any surface realizations like "kicking" or "kicks", the upper side of the network will return with "kick+Verb+Prog" or "kick+Verb+Pres3PSg" respectively, and vice versa. So, using regular expression grammars our lexicon can be compiled as, [{kick} | {try} | {bore}] [%+Verb:0] [%+Prog:{ing} | %+ Pres3PSg:s |%+ Past:{ed} | %+ Bare:0];.

Note that, since this is a two-level network, the stem "kick" should be written as [k:k i:i c:c k:k]. In Xerox's regular notations, the relation [a:a b:b] can be simply written as [a b] or [{ab}]. However, this is not the final form as the final outputs of this network contains some inappropriate words like "tryed", "trys", "boreing" and "boreed". So, these corrections are made using a rule transducer on top of the lexicon transducer.

2) *Orthographic Rules:* A cascade of rule transducers can be mapped into a single transducer. This is the concept used in compiling a rule transducer. This is applied on top of the lexicon transducer, so that the results from the lexicon transducer get changed according to the rules defined for the language. The rule, $\alpha \rightarrow \beta \;||\; \gamma \_ \delta$ is read as "α is rewritten as β between γ and δ" . Here, $\alpha \rightarrow \beta$ is the rule and $\gamma \_ \delta$ is the context in which the rule has to be applied. Here, most of the rules are applied in parallel. In Xerox's notation, parallel rules are indicated by ",," - double commas.

For example, for the above lexicon transducer, "y" has to be rewritten as "i" for the underlying abstract word "tryed", "y" has to be rewritten as "ie" for the word "trys" and "e" has to be deleted from the abstract words using regular expressions would be,

```
[ y -> i || Cons _ {ed}.#. ,, y -> {ie}
|| Cons _ s .#. ] .o.  e ->  0  ||
Cons _  [{ing}.#.|{ed}.#.];.
```

This rule set will result the intermediate word forms, "tryed", "trys", "boreing" and "boreed" getting changed to "tried", "tries", "boring" and "bored".

V. Implementation

Data was collected from various sources, especially majority of them from the parliamentary notes as the pre- process before implementing the system on Xerox tool. Implementation was done in three steps. First was defining a transliteration scheme as the non-commercial Xerox tool version did not support Unicode at the time of this research was conducted. The transliteration tool generates an encoded ASCII string given a Unicode string and vice versa using the custom defined scheme. Then, networking the Morphological Analyzer and Generator in Xerox for Nouns and finally building up the MAG for verbs.

In the pre-process, Noun and Verb stems with the highest frequencies were identified from the available data and they were categorized into their respective groups as per linguistic sources. These categorizations of nouns and verbs will be explained under their respective MAGs.

A. Transliterator

The transliterator serves two purposes - as an encoder and decoder. We defined our own transliteration scheme which is a symbol to symbol mapping. This is to facilitate the Xerox tool, as it treats each letter as a symbol and it does not support a one-to-many relationship between symbols. The transliteration scheme was defined in a manner that supports and increases readability.

First, the gathered Noun and Verb stems in Unicode were encoded into ASCII based characters which will be the input to the Xerox tool. Further, the suffixes were also transliterated for the tool to produce the correct output. Then the final surface forms which the tool produces were transliterated (decoded) into Tamil words (Tamil surface forms).

B. MAG for Nouns

According to Tamil grammar [10], it has eight types of noun suffix classes. However, each of them holds more than one suffix. Type 1, has no suffixes which is the Nominative form in Tamil. Type 2 speaks of the Accusative form. Type 3 is about Instrumental and Social cases. Type 4 is the Dative form in Tamil. Type 5 contains the Ablative case, whereas Type 7 is about the Locative case. Type 8 is Vocative which is used for addressing or calling a person.

These cases should be applied for both singular and plural forms. All these eight types have about 20 suffixes [12], which means all together for both singular and plural we would be able to generate about 40 forms.

Seven noun stem categories were identified according to their morphological behaviour. Below given table shows the classification method for each category.

TABLE I
NOUN STEM CATEGORY CLASSIFICATION

Noun Stem Category	Classification Type
Category 1	Mute consonant endings
Category 2	"m" ending
Category 3	" i " , " i: " , " ai " endings
Category 4	"l", "L", "n" ,"N" endings – Consonant doubling
Category 5	"u" , "vu" endings
Category 6	"du" , "Ru" endings - Consonant doubling
Category 7	"a:" , "u" , " u:" , "o" endings

Shown below is the regular expression of the lexicon transducer for category 2 noun stems "maram", "caraNam" and "pAram". Note that the examples are chosen at random.

```
[ {maram} | {caraNam} | {pAram} ]
[[%+Noun:0] [%+Sin:0 | %+Plu:{kaL}]
[ %+Nom:0 | %+Acc:{Î} | %+Acc:{inÎ}
| %+Dat:{ukku}| %+Dat:{iRku}
| %+Genetive:{in} | %+Genetive:{inaTu}
| %+Genetive:{inutÎya} | %+Abl:{itam}
| %+Abl:{itaTTil} | %+Abl:{iliruñTu}
| %+Abl:{itámiruñTu}
| %+Abl:{itaTTiliruñTu} | %+Abl:{utan}
| %+Abl:{Otu} |%+Abl:{inAl} | %+Abl:{Al}
| %+Abl:{il}]];
```

The Noun transducer is modelled as the root/stem coming first , then the Noun arc which will have no suffixes (epsilon), followed by the number (Singular/ Plural) morpheme finally concatenated with the case morpheme. Here to make the regular expression more readable, we have shown the stems "maram", "caraNam", "pAram" also in the same regex file. However, it can be maintained in a separate lexicon database, in which any number of new noun stems of that category can be added later and then the network is applied. Notice that the same +Abl tag has been used ten times in the network as there are ten suffixes available in Tamil for the Ablative case. We have not distinguished them as +Abl1,+Abl2 and so on. We have written the regular expressions for the lexicon of all the seven categories in the above manner.

The regular expression of the orthographic rule for category 2 is written as,

```
read regex [ m -> {TT} || ?* _ [ {î}.#.
| {ukku}.#.     |  {in}.#.     |
{itam}.#.|{inî}.#.   |  {iRku}.#.   |
{ukkAka}.#.| {iRkAka}.#.| {inaTu}.#. |
{inutîya}.#.   |{itaTTil}.#.      |
{iliruñTu}.#.|{itamiruñTu}.#.|{itaTTilir
uñTu}.#.|{utan}.#.|{Otu}.#.|{inAl}.#.|{A
l}.#.|{il}.#.] ,, m -> G  || ?*
_{kaL}[.#.| {î}.#.| {ukku}.#.    | {in}.#.
| {itam}.#.|{inî}.#.   |  {iRku}.#.   |
{ukkAka}.#.|{iRkAka}.#.| {inaTu}.#. |
{inutîya}.#.   |{itaTTil}.#.     |
{iliruñTu}.#. |{itamiruñTu}.#. |
{itaTTiliruñTu}.#.|{utan}.#.|{Otu}.#.|{i
nAl}.#.|{Al}.#.| {il} .#.] ];
```

Here two rules have been applied for different contexts in singular and plural forms. First rule set is for the singular cases and the second is for the plural cases.

We faced problems in consonant doubling in category 4 in which we wanted the "L" to get doubled, followed by the case marker. This caused all the plural forms in this category with errors. The rule was not powerful enough to distinguish the root-suffix boundary and the suffix-suffix boundary, where the plural suffix "kaL" too is followed by the case marker. For this, the plural suffix was inserted with number symbols and then the unwanted symbols in the intermediate surface forms were eliminated to produce correctly spelled words. The suffix pairs "î", "inî" and "Al", "inAl" also faced the same issue in consonant doubling in the same category and was eliminated in the same fashion. Thus, the rule FST of this category is very complex compared to other noun categories.

The final network is composed by applying the rule transducer on top of the lexicon transducer. So far, 2000 noun stems have been incorporated into the network which can cover up to 80,000 words.

C. MAG for Verbs

The transducer for handling verbs in the Tamil language is very complex, especially the rule transducer. A combination of tags may sometimes have only one suffix which was not the case for nouns. We had, each suffix having only one tag for nouns. In verbs for example, the suffix "An" would have the tag combination, +ThirdPr+Mas+Plu. Likewise, plenty of such combination tags were identified, and modelled into the network. Add to this, the number of forms a verb stem can produce is not limited. The Tamil verb system cannot be directly modelled as one network due to its complexity. So, we tried to build smaller networks using incremental approach and later joined them using the "|" operator. It is actually a network of networks. It tries to grow bigger, due to derivational morphological patterns and negative forms of a verb. So, we restricted our scope to avoid derivational patterns and negative forms. Here 24 tags were identified with 32 suffixes. Tags were identified for the Verb, Non Causative, Active Causative, Passive Causative, Causative, Present, Past, Future, First Person, Second Person, Third Person, Masculine, Feminine, Singular, Plural, Common, Other Class, Infinitive, Verbal Adjective, While, Adverbial Participle, Negative Adverbial Participle, Perfect and Bare forms.

Eleven verb stem categories were identified according to their morphological behaviour. Below given table shows the classification method for each verb stem category.

TABLE III
VERB STEM CATEGORY CLASSIFICATION

Verb Stem Category	Classification Type
Category 1	"y" ending
Category 2	"zu", "u" endings
Category 3	"ti", "tu", "ai" endings (2,3 letter words)
Category 4	"tu", "Ru" and some "u" endings
Category 5	"l" ending
Category 6	"l" ending (a few words)
Category 7	"n" ending (a few words)
Category 8	"L" ending ("L" becoming "N" rule)
Category 9	"la", "ta", "ru", "Ra" endings
Category 10	"tu", "du" endings
Category 11	"L" ending ("L" becoming "t" rule)

108 surface forms of the verb stem "cey" can be produced by the below given sub-network.

```
[cey][[%+Verb:0]
[%+NonCause:0| %+ActCause:{vi}| %+PasCau
se:{vikkappatu}][%+Pre:{kinR}
| %+Past:{T}
| %+Future:{v}][[ %+FirstPr%+Sin:{En}
| %+FirstPr%+Plu:{Om}| %+SecondPr%+Sin:{
Ay}| %+SecondPr%+Plu:{IrkaL}]|[ %+ThirdP
r%+Mas%+Sin:{An}
| %+ThirdPr%+Mas%+Plu:{ArkaL}|%+ThirdPr%
+Fem%+Sin:{AL}
| %+ThirdPr%+Fem%+Plu:{ArkaL}
| %+ThirdPr%+Common%+Sin:{Ar}
| %+ThirdPr%+Common%+Plu:{ArkaL}| %+Thir
dPr%+OtherClass%+Sin:{aTu}|%+ThirdPr%+Ot
herClass %+Plu:{ana} ]]];
```

The regular expression of the orthographic rule for this sub- network is written as,

```
read regex [{uT} -> {t}
|| ?*{vikkappat}_[ {En}.#. | {Om}.#.|
{Ay}.#.| {IrkaL}.#.|{An}.#. | {ArkaL}.#.
|{AL}.#. | {ArkaL}.#. |{Ar}.#. |
{ArkaL}.#. | {aTu}.#. |{ana}.#. |{a}.#.
```

```
{avARu}.#.] ,, v -> {pp} || ?*{vi} _
[ {En}.#. | {Om}.#.| {Ay}.#.|
{IrkaL}.#.|{An}.#. | {ArkaL}.#.|{AL}.#.
| {ArkaL}.#. | {Ar}.#. | {ArkaL}.#. |
{aTu}.#. |{ana}.#. ] ,, [..] -> k
|| ?*{vi} _ {kinR}[ {En}.#. |{Om}.#.|
{Ay}.#.| {IrkaL}.#.|{An}.#. | {ArkaL}.#.
|{AL}.#. | {ArkaL}.#. | {Ar}.#. |
{ArkaL}.#. | {aTu}.#.|{ana}.#. |{a}.#.|
{avARu}.#.] ,, [..] -> T || ?*{vi} _
{T}[ {En}.#. | {Om}.#.| {Ay}.#.|
{IrkaL}.#.|{An}.#. | {ArkaL}.#. |{AL}.#.
| {ArkaL}.#. | {Ar}.#. | {ArkaL}.#. |
{aTu}.#. |{ana}.#.|{a}.#.|
{avARu}.#. ] ] ;
```

Four such sub-networks were integrated using the union operator to generate the 240 surface forms of the stem "cey". So far, 96 verb stems have been incorporated into the network which can cover up to 17,166 words. Each verb stem category will have a separate FST network. Filtering out unwanted forms was given major importance due to its redundancy in the verb MAG. However, all the categories did not generate 240 forms as we omitted the forms that do not occur frequently in common text.

We faced several problems in almost all the verb stem categories in combining the rules when networks were merged. Duplicate rules were eliminated. However, Over-Generation happened because of the conflicts in the rules in failing to recognize patterns. To avoid this ambiguity we changed our lexicons and altered our suffixes. For example, the Passive Causative suffix "vikkappatu" was changed into "vikkappatlu", for category 2 to avoid conflicts. Then, a rule was applied sequentially to delete the unwanted numbers in the network. A "u" deletion rule was also applied for the same category when it concatenates with the Negative Adverbial Participle suffixes "Amal", "ATu" and the Infinitive suffix. In category 3, "kk" insertion rule was applied before the verb stem concatenates with the Negative Adverbial Participle suffixes.

The rule of "tu" becoming "ttu" at the verb root boundary in category 4, also mistakenly got applied for the Passive Causative suffixes "patu", "vikkapatu or "pikkapatu". So rules were carefully written considering the context. While "L" deletion rule should be applied for the Causative forms in Category 8, consonant doubling was applied for the Infinitive forms in Category 7 and the non-Causative forms in Category 9. "u" deletion rule was applied for Past non-Causative forms in Category 10. Add to the above mentioned special rules, several other rules were also applied in the root-suffix boundary and in the suffix-suffix boundary to produce the correct words. The Verb Transducer had many sequential rules other than the normal parallel rules.

VI. Evaluation

An iterative evaluation was carried out throughout the development of the model. The Morphological Generator evaluation was done by comparing the output that the machine produces with the human output. Unwanted forms and wrongly spelled words were eliminated and re-networked according to the stem category behaviour.

Due to Xerox tool's limitation we were able to test our generation correctness on 3500 noun surface forms which resulted in a 78.3% success rate. Below given shows, the number of correct forms for each noun stem category.

TABLE IIIII
Evaluation Results

Noun Stem Category	Correct forms (out of 500)
Category 1	220
Category 2	500
Category 3	480
Category 4	500
Category 5	340
Category 6	380
Category 7	320

For verbs, due to the Xerox tool's limitation of generating only 500 words, the results will be very limited as the generation evaluation only covers very few different forms of verb roots (some categories can have only 2 roots). Therefore, it is not significant enough to report. We are planning to eliminate this limitation in our future work.

VII. Results

In the Xerox toolkit one can check, analyze and generate for each word that has been networked using the apply up or apply down commands or else can print the whole words that are in the lower side of the network – words in the lower language using print lower-words command. Since, there is a FST for each noun and verb stem categories, commands must be applied separately to produce results. For example, for nouns when we look down (generate) "maram+Noun+Sin+Acc", it will result in "maraTTî" and when we look up (analyze) for the word "maraGkaL" it will result us with "maram+Noun+Plu+Nom" in the XFST command line interface. For verb generation process, it is rather a long tag set. The input "cey+Verb +Avp+Perfect+Past+ThirdPr+OtherClass+Plu" will produce "ceyTiruñTana" as the surface form. In this model, we can look up or look down for 80,000 noun surface forms and 17,166 verb surface forms, which will cover up to 97,166 Tamil words in total.

The results of this network for each category can also be printed into an external word list file with all the possible forms in it. Since, we have several forms for the same tag, the model may produce several results or forms in the look down (apply down) process for nouns. For verbs, the same surface form will produce many number of possible combination of tags in the look up (apply up) process. This is the nature of Tamil language. For example, in the noun lexicon FST the tag "+Acc" occurs several times, and in the verb lexicon FST the suffix "ArkaL" occurs several times in the regular expression.

VIII. Conclusion

Our current system has been implemented on the Xerox tool. It has been implemented with 2000 noun stems and 96 verb stems, considering 7 different noun classes and 11 verb classes. Each noun stem produces 40 different forms of noun words including both singular and plural cases. Each verb

stem produces around 240 or less different forms. So, this model now covers up to 80,000 noun surface forms and 17,166 of verb surface forms in the Tamil language. Here, each noun and verb class have their own orthographic rules, most of them applied to the network in parallel. However, sequential rules were also applied to avoid over- generation and error forms. Various forms of suffixes or combination of suffixes have been networked and have been given their tags.

In our system we have almost covered all the variations of noun and verb cases within our scope like different consonant endings, consonant doubling and deletion rules. However, we had problems in dealing with the consonant doubling rules in the noun FST and the over generated illegal forms in the verb FST.

Our model uses a common fully automated transliteration scheme and is implemented on the Xerox's non-commercial XFST interface. A user needs to know the commands for analyzing or generating a string. In our system rule transducer is applied on top of the lexicon transducer, which contains the stems, suffixes and their tags. Since our system contains all the stems of a particular class in a single file, it is easy to add new stem entries into the network.

IX. Future Work

Our major future work is to evaluate the analyzer part of our model with the CIIL corpus using the Open FST network. We also look forward to add more noun and verb stems as possible that would widen the coverage of our Morphological Analyzer. The more the coverage of words, the better the model becomes. Add to this, we look forward of the ways in which this Morphological Analyzer can fit into other NLP applications which uses Tamil morphology.

Another major goal of our future work would be to implement our morphological analyzer in the open FST library or foma and make it open source. In this way, we can eliminate the limitations we are facing now with the Xerox tool. We look forward to incorporating more than 5000 stems in our Morphological analyzer.

Acknowledgment

We would like to sincerely thank Prof. S. Suseendirarajah for helping us with the linguistic knowledge in Tamil language and the Language Technology Research Laboratory (LTRL) of UCSC for providing with linguistic resources and other facilities.

References

[1] H. Jäppinen, A. Lehtola, E. Nelimarkka, and M. Ylilammi, "Knowledge engineering approach to morphological analysis," in *Proc. first conference on European chapter of the Association for Computational Linguistics*, Pisa, Italy, Sept. 01-02, 1983.

[2] K. Koskenniemi, "A General Computational Model for Word-Form Recognition and Production," in *Proc. 22nd annual meeting on Association for Computational Linguistics*, Stanford, California, July 02-06, 1984.

[3] L. Karttunen, "Applications of Finite-State Transducers in Natural Language Processing," presented at the *5th Int. Conf. on Implementation and Application of Automata*, July 24-25, 2000, pp.34-46.

[4] M. Attia, "An Ambiguity-Controlled Morphological Analyzer for Modern Standard Arabic Modelling Finite State Networks," presented at the *Conf. on Challenge of Arabic for NLP/MT Conference*, Oct. 2006.

[5] S. Yona and S.Wintner, "A finite-state morphological grammar of Hebrew" in *Proc. ACL Workshop on Computational Approaches to Semitic Languages*, Ann Arbor, Michigan , 2005, pp 9-16.

[6] T. Bögel, M. Butt, A. Hautli and S. Sulger, "Developing a Finite-State Morphological Analyzer for Urdu and Hindi," in *Proc. Sixth Int. Workshop on Finite-State Methods and Natural Language Processing*, Potsdam, Sept. 2007.

[7] Yamashita, Tatsùo, Matsumoto and Yuji, "Language Independent Morphological Analysis." in *Proc. sixth conference on Applied natural language processing*, Seattle, Washington , Apr. 29-May 04, 2000, pp. 232-238.

[8] K. R. Beesley & L. Karttunen, "The xfst Interface," in *Finite State Morphology*, CSLI Publications (distributed by the University of Chicago Press), 2003, pp 81-202.

[9] K. R. Beesley, "Arabic Morphology Using Only Finite-State Operations," in *Proc. Workshop on Computational Approaches to Semitic Languages*, Montreal, Quebec, Canada, Aug. 16-16, 1998.

[10] S. Aarumuganavalar, "Elakkanach Churukkam," Sabanayagar press, 1873, pp 84-215.

[11] M.AnandKumar V. Dhanalakshmi, S. Rajendran , K.P. Soman "A Novel Approach to Morphological Analysis for Tamil Language," in the *8th Tamil Internet Conference*, Germany, Oct. 2009.

[12] A. W. L. Silva, "Teach yourself Tamil - A complete Course for Beginners," Pubudhu Printers, 2003.