Chapter 6 :FINITE STATE MORPHOLOGICAL ANALYSIS WITH XFST TOOL

Implementation of morphological analysis by compiling regular expressions into finite state networks is purposely defining a data structure rather than procedural programs. The skeleton for morphological analysis using finite-state technique has been the construction of lexical transducers. Karttunen, Kaplan and Zaenen 1992, described the standard way of constructing lexical transducers, and it consists of –

(1) a finite-state source lexicon that defines the set of valid lexical forms of the language (possibly infinite), and is a specification for the morphotactics of a category of words; and

(2) a set of finite state rules that assign the proper surface realization to all lexical forms and morphological categories of the language. The rules are compiled to transducers (called rule transducers) and merged with the source lexicon using intersection and composition operation. Every Rule transducer represents a single spelling change rule.

So a lexical transducer is a set of transducers for orthographical rules with a transducer encoding the source lexicon. The lexical transducers encodes the relation between the inflected surface forms of root/stems and their corresponding lexical forms or lemmas, each containing a citation form of the word followed by a part-of speech tag.

xfst uses a built-in stack to store and manipulate networks. The operations, features, and other characteristics of stack such as Last-In, First-Out (LIFO) data structure that stores networks, pushing and popping of networks on top of the stack are supported here. At the time xfst is launched, the stack is empty.

In defining natural-language morphotactics, union and concatenation are the basic operations required to perform on the source lexicon of different category of words.

Variation rules and long distance- dependency filters are applied using composition operation (Kenneth R. Beesley, 1998). The current chapter describes the creation and implementation of the source lexica of different Manipuri word categories and compilation of the regular expression for orthography or the spelling change rules for the various morphophonemic alternations.

6.1 MANIPURI SOURCE LEXICON

For implementation of the morphotactics, Manipuri source lexicon has been divided into two major sub-lexicons (or continuation classes in xfst terminology)- nominal and verbal lexicon. Again the nominal category sub-lexicon has been divided into eight categories – animate free nouns, inanimate free nouns, wh-words, personal pronouns, demonstrative pronouns, kinship terms, animal body parts, derived nouns (from verb roots), and numerals. Similarly the verb source lexicon has been divided into three sub categories- action, process and stative.

The source files have been written using Notepad text editor. Our source lexicon entry consists of root- both free and bound forms, the affixes in its lexical and surface form, muticharacter symbols and other morphemes like particles, etc. for each nominal category and their in the xfst regular expression notation. The following is the source lexicon for animate nouns defined in the xfst notation (see Table 5-13 for noun morphotactics) written in a text file, named **NF_Animate-lex.txt**.

```
Multichar_Symbols

+NOUN +SG +MASC +PL +FEM +NOM +GEN +ASS +ABL +COP +CON-W +CON-AL +CON-AN +DEF-P
+QM +EMP-S-P +EXC-S-P +EMP-ONLY-P

Lexicon Root
        Masculine;
        Feminine;

Lexicon  Feminine
        +NOUN+FEM:0 #;
        +NOUN+FEM:0 NUMBER;

Lexicon Masculine
        +NOUN+MASC:0 #;
        +NOUN+MASC:0 NUMBER;

Lexicon NUMBER
        +SG:0 #;
        +SG:0 INFLECTION;
        +PL:শিং INFLECTION;

Lexicon INFLECTION
```

```
            Nominative;
            Genitive;
            Associative;
            Ablative;
            Accusative;

Lexicon Nominative
            +NOM:ন #;
            +NOM:ন Connectives;
            +NOM:ন Accusative;
            +NOM:ন Copula;
            +NOM:ন Interrogative;
            +NOM:ন Definitive;
            +NOM:ন Emphatic-Self;
            +NOM:ন Exclusive-Self;
            +NOM:ন Emphatic-Only;

Lexicon Genitive
            +GEN:গী #;
Lexicon Associative
            +ASS:গ #;
Lexicon Ablative
            +ABL:দগী #;
Lexicon Accusative
            +ACC:বু #;
Lexicon Connectives
            +CON-W:গ #;
            +CON-AL:সু #;
            +CON-AN:নে #;
Lexicon Copula
            +COP:নি #;
Lexicon Interrogative
            +QM:র #;
Lexicon Definitive
            +DEF-P:দি #;
Lexicon Emphatic-Self
            +EMP-S-P:মক #;
Lexicon Exclusive-Self
            +EXC-S-P:দ #;
Lexicon Emphatic-Only
            +EMP-ONLY-P:দমক #;
END
```

Compilation of the text file into a finite-state network after invoking lexc is done at the **lexc** prompt, by entering

lexc> compile-source NF_Animate-lex.txt

The following is a glimpse of lexc interface after the above command is executed.

```
      ******************************************************
      *  Finite-State Lexicon Compiler 3.7.14 (2.25.11)     *
      *                  created by                         *
      *          Lauri Karttunen and Todd Yampol            *
      *    Copyright ┬⌐ 1993-2014 by the Xerox Corporation.  *
      *                All Rights Reserved.                 *
      ******************************************************

Input/Output ---------------------------------------------------------------

   Source:      compile-source, merge-source, read-source, result-to-source,
              save-source.
   Rules:       read-rules.
   Result:      merge-result, read-result, save-result, source-to-result.
   Properties:  add-props, reset-props, save-props.

Operations -----------------------------------------------------------------
   Composition: compose-result, extract-surface-forms, invert-source, invert-result.
   Checking:    check-all, lookdown, lookup, random, random-lex, random-surf.
   Switches:    ambiguities, duplicates, failures, obey-flags, print-space, recode-cp1252, quit-on-fail, show-
flags, singles, time, utf8-mode.
   Scripts:     begin-script, end-script, run-script.
Display --------------------------------------------------------------------
   Misc:        banner, labels, props, status, storage.
   Help:        completion, help, history, ?.

Type 'quit' to exit.

Starting in utf8-mode.
lexc> compile-source NF_Animate-lex.txt
opening "NF_Animate-lex.txt"
Opening 'NF_Animate-lex.txt'...
Root...9, Feminine...2, Masculine...2, NUMBER...3, INFLECTION...5, Nominative..9, Genitive...1,
Associative...1, Ablative...1, Accusative...1, Connectives...3, Copula...1, Interrogative...1, Definitive...1,
Emphatic-Self...1, Exclusive-Self...1, Emphatic-Only...1Building lexicon...Minimizing...Done!
SOURCE: 3.3 Kb. 25 states, 50 arcs, 288 paths.

lexc>
```

The compiled lexc file is saved as an fst file by the following command from the lexc prompt *save -source NF_Animate-lexc.fst.*

```
lexc> save-source NF_Animate-lex.fst
opening "NF_Animate-lex.fst"
Opening 'NF_Animate-lex.fst'...
Done.
lexc>
```

All the source lexica are compiled as FST network from the lexc interface with a .fst extension file. These FST files are loaded into the stack from the xfst interface. The command and the xfst interface for loading the NF_Animate-lex.fst file is shown below:

```
Copyright ┬⌐ Palo Alto Research Center 2001-2014
PARC Finite-State Tool, version 2.15.7 (libcfsm-2.25.11) (svn 34269)
Type "help" to list all commands available or "help help" for further help.
xfst[0]: load stack NF_Animate-lex.fst
Opening input file 'NF_Animate-lex.fst'
April 22, 2014 17:38:29 GMT
Closing input file 'NF_Animate-lex.fst'
xfst[1]:
```

Implementation for the compilation of the Adjective lexicon as per the Adjective morphotactics is shown below:

```
Multichar_Symbols

+FORM +NZR +INC +PRGASP +HAB +PROS +PERF +DEST +EVI +DCT +POT +PERF +DIR-UP

+DIR-OUT +DIR-IN +NG1 +NG2 +NG3 +NG4 +DUBT

Lexicon Root
        VR;
        Prefix;
Lexicon VR
        Verbs;
Lexicon Verbs
        +VR:0 Suffixes;
Lexicon Suffixes
        +INC:রক NG2;
        +INC:রক NG3;
        +INC:রক NG4;
        +INC:রক NZR;
        +NOM:ন NG2;
        +NOM:ন NG3;
        +NOM:ন NG4;
        +NOM:ন NZR;
        +HAB:গন NG2;
        +HAB:গন NG3;
        +HAB:গন NG4;
        +HAB:গন NZR;
        +EVI:রম NG2;
        +EVI:রম NG3;
        +EVI:রম NG4;
        +EVI:রম NZR;
        +DCT:রু NG2;
        +DCT:রু NG3;
        +DCT:রু NG4;
        +DCT:রু NZR;
        +PERF:খি NG2;
        +PERF:খি NG3;
```

```
        +PERF:খি NG4;
        +PERF:খি NZR;
        +DIR-UP:খৎ NG2;
        +DIR-UP:খৎ NG3;
        +DIR-UP:খৎ NG4;
        +DIR-UP:খৎ NZR;
        +DIR-OUT:থোক NG2;
        +DIR-OUT:থোক NG3;
        +DIR-OUT:থোক NG4;
        +DIR-OUT:থোক NZR;
        +DIR-IN:সিন NG2;
        +DIR-UP:সিন NG3;
        +DIR-UP:সিন NG4;
        +DIR-UP:সিন NZR;
        +PROS:র NZR;
        +DEST:গাই PROS;
        +DEST:গাই NG2;
        +DEST:গাই NG3;
        +DEST:গাই NG4;
        +POT:গ DUBT;
        +NG1:রোই DUBT;
Lexicon Prefix
        +FORM:অ Verb;
Lexicon Verb
        চা NZR;
Lexicon NG2
        +NG2:ত NZR;
Lexicon NG3
        +NG3:ত্র NZR;
Lexicon NG4
        +NG4:ত্রি NZR;
Lexicon PROS
        +PROS:র NZR;
Lexicon DUBT
        +DUBT:দ NZR;
Lexicon NZR
        +NZR:ব #;
END
```

The wh-words in Manipuri resembles the features of English wh-words such as **wh**at, **wh**en, **wh**ere, etc. These Manipuri words take certain nominal suffixes and are very particular in their behavior at the sentence level. The source lexicon to model this nominal sub category is presented below for five Manipuri wh-words. (The text file is saved as Wh-lex.txt).

```
Multichar_Symbols
+GEN +ACC +LOC +TOO +ABL +ASS +NOM +INQ
LEXICON Root
        Noun;
LEXICON Noun
        WhWords;
LEXICON WhWords
        কনা     Wh0;
        করি     Wh1;
        কদায়    Wh2;
        কদোম    Wh3;
        করমব    Wh4;
LEXICON Wh0
        +INQ:^নো #;
        CaseInfl;
LEXICON Wh1
        +INQ:^নো #;
        CaseInfl;
LEXICON Wh2
        +INQ:^দ #;
        CaseInfl;
LEXICON Wh3
        CaseInfl;
LEXICON Wh4
        +INQ:নো #;
        CaseInfl;
LEXICON CaseInfl
        +GEN:^গী #;
        +NOM:^ন #;
        +ASS:^গ #;
        +ACC:^বু #;
        +ABL:^দগী #;
        +LOC:^দ #;
        +GEN+INQ:^গী^নো # ;
        +NOM+INQ:^ন^নো #;
        +ASS+INQ:^গ^নো #;
        +ACC+INQ:বু^নো #;
        +ABL+INQ:^দগী^নো #;
        +LOC+INQ:^দ^নো #;
END
```

The above text file is compiled using xfst from the xfst interface. It may be noted here that compilation of lexicon text files into FSTN can also be performed from xfst interface (Beesley & Karttunen, 2003, Chapter 3). The command for the same is

xfst[0]: read lexc <Wh-lex.txt

And the interface is shown below:

PARC Finite-State Tool, version 2.15.7 (libcfsm-2.25.11) (svn 34269)

```
Type "help" to list all commands available or "help help" for further help.

xfst[0]: read lexc <Wh-lex.txt
Opening input file 'Wh-lex.txt'
May 05, 2014 17:17:51 GMT
Reading UTF-8 text from 'Wh-lex.txt'
Root...1, Noun...1, WhWords...5, Wh0...2, Wh1...2, Wh2...2, Wh3...2, Wh4...2, Ca
seInfl...12
Building lexicon...Minimizing...Done!
3.4 Kb. 27 states, 55 arcs, 64 paths.
Closing 'Wh-lex.txt'
xfst[1]: write spaced-text >result.txt
Opening 'result.txt'
Closing 'result.txt'
xfst[1]:
```

The result of the compilation is written out to a text file called result.txt, which contains the model for the five wh-words. (See Annexure IV). Here special mention can be made that no spelling rules has been applied to this model and yet the resulting model has all the word forms correct.

All the the spelling rules specific to a lexicon of a particular word class can be applied through composition operation.

6.2 THE RULE TRANSDUCER

The morphotactical irregularities are captured by xfst 'replace rules which are part of the extended regular expression notation xfst offers. They are shorthand notations for complicated and rather opaque regular expressions built with more basic operators. These replace rules do not increase the descriptive power of regular expressions, but they represent a simple and straightforward method to define complicated finite-state relations in rule-like notation (Beesley and Karttunen, 2003).

In the xfst interface replace rules are compiled using the **read regex** command or **define** utilities just like other regular expression compilation. However in this study we have written the grammars of replace rules in regular-expression files, to be compiled with **read regex *<filename>*** commands on an individual basis from xfst prompt. The complex sequences of **xfst** commands for compiling and manipulating rule grammars are written in **xfst** script files, which are run using the **source** command (Beesley and Karttunen, 2003, Chapter 3).

The simplest example of such a replace rule is [a -> b]. This rule denotes a relation wherein every symbol a in the strings of the upper-side language is related to a symbol b of the lower-side language. When this rule is applied downward to the string aardvark of a finite-state network, every a will be replaced by b and so the output is bbrdvbrk. The upper-side language of such a relation is the universal language; and any upper-side string that does not contain **a** simply maps to itself on the lower side. For every upper-side string that does contain **a**, it is mapped to a lower-side string that contains **b** in the place of **a** (Beesley and Karttunen, 2003).

It is possible to define contexts for linguistic rules with more fine-grained mechanisms. In Manipuri, it is more often that the occurrence of a morpheme is restricted by its neighboring morphemes or we can say that occurrence of morphemes are context sensitive. e.g. xfst has the provision for defining context sensitive replace rules to formalize the alternation rules traditionally used in phonology. The rules specified for the context sensitive replace rules are of the form:

[a -> b || L _ R]

The replace rule can handle context sensitive orthographic changes; where L denotes the language that specifies the left context whereas R denotes the language that restrains the right context for the replacement. The double-bar operator between the replacement specification and the context expressions indicates that both context expressions must match on the upper side of the relation. Contexts L and R are optional, if L or R is empty, the resulting context is treated as the universal language (any possible context is allowed) (Beesley and Karttunen, 2003).

In the xfst prompt, compilation of the replace rule for regular expression (PR1) defined in section 5.4, chapter 5,Chapter 0 which is written in a text file, **PR1.txt**, as ব->প || [প|ক|ত|ৎ] _; is shown in the xfst interface as the following:

```
Copyright ┬⌐ Palo Alto Research Center 2001-2014
PARC Finite-State Tool, version 2.15.7 (libcfsm-2.25.11) (svn 34269)
Type "help" to list all commands available or "help help" for further help.
xfst[0]: source PR1.txt
Opening input file 'PR1.txt'
April 19, 2014 07:30:59 GMT
2.3 Kb. 2 states, 12 arcs, Circular.
Closing file PR1.txt...
```

6.3 APPLYING COMPOSITION OPERATION

Composition operation can be performed on two or more networks. In our context we use composition operator in order to apply the spelling rules to the source lexicons. It can either be performed directly from the xfst prompt on the available networks on the stack or by writing the script in a script file. We prefer the later option as different lexicons have different rule sets and stored them as a .fst file to be composed together with networks of other word classes. A source file (a text file named ASRC.txt) to compile and apply the rules applicable to adjective lexicon is as follows:

Table 6-1: A script file to compose the adjective lexicon with rules

```
clear
read regex <PR1.txt
read regex <PR11.txt
read regex <PR9.txt
read regex <PR12.txt
read regex <PR13.txt
read regex <PR40.txt
read regex <PR41.txt
read regex <PR42.txt
compose net
read lexc <Adj-lex.txt
compose net
write spaced-text >result.txt
echo <<Done>>
```

The above script file not only compile and compose the rule and source lexicon together to a finite state network, it also writes out the mapped form of the lexical to the surface form in a plain text file called result.txt. So when ASRC.txt is invoked from xfst prompt:

Copyright ┬⌐ Palo Alto Research Center 2001-2014

PARC Finite-State Tool, version 2.15.7 (libcfsm-2.25.11) (svn 34269)

Type "help" to list all commands available or "help help" for further help.

xfst[0]: source ASRC.txt
Opening input file 'ASRC.txt'
April 26, 2014 22:51:25 GMT
Opening file PR1.txt...
2.4 Kb. 2 states, 22 arcs, Circular.
Closing file PR1.txt...
Opening file PR11.txt...
2.6 Kb. 4 states, 34 arcs, Circular.
Closing file PR11.txt...
Opening file PR9.txt...
2.6 Kb. 4 states, 34 arcs, Circular.
Closing file PR9.txt...
Opening file PR12.txt...
2.6 Kb. 4 states, 31 arcs, Circular.
Closing file PR12.txt...
Opening file PR13.txt...
2.6 Kb. 4 states, 31 arcs, Circular.
Closing file PR13.txt...
Opening file PR40.txt...
2.3 Kb. 2 states, 10 arcs, Circular.
Closing file PR40.txt...
Opening file PR41.txt...
2.4 Kb. 2 states, 16 arcs, Circular.
Closing file PR41.txt...
Opening file PR42.txt...
2.4 Kb. 2 states, 16 arcs, Circular.
Closing file PR42.txt...
5.4 Kb. 13 states, 238 arcs, Circular.
Opening input file 'Adj-lex.txt'
April 27, 2014 04:13:59 GMT
Reading UTF-8 text from 'Adj-lex.txt'
Root...2, VR...1, Verbs...1, Suffixes...43, Prefix...1, Verb...1, NG2...1, NG3...1, NG4...1, PROS...1, DUBT...1, NZR...1
Building lexicon...Minimizing...Done!
3.6 Kb. 36 states, 58 arcs, 44 paths.
Closing 'Adj-lex.txt'
3.8 Kb. 39 states, 69 arcs, 44 paths.
Opening 'result.txt'
Closing 'result.txt'
<<Done>>
Closing file ASRC.txt...
xfst[1]:

Annexure- III shows adjectives word forms derived from the verb root ଚା/ca (eat) due to the above compilation.

Our strategy has been that we compose the source lexicon of each word category separately with their applicable rule sets. Separate source files are used for compiling each word class and their respective rules. The resulting lexical transducers altogether are composed to form a single lexical transducer.

Afterwards all the resulting lexical transducers of the word categories are composed together to form the final lexical transducer for a language model of Manipuri morphological analysis.

6.4 THE LEXICAL TRANSDUCER

The pictorial representation of how lexical transducers are formed is depicted as shown in the following figure (courtesy: Kenneth R. Beesley and Lauri Karttunen, 2000)

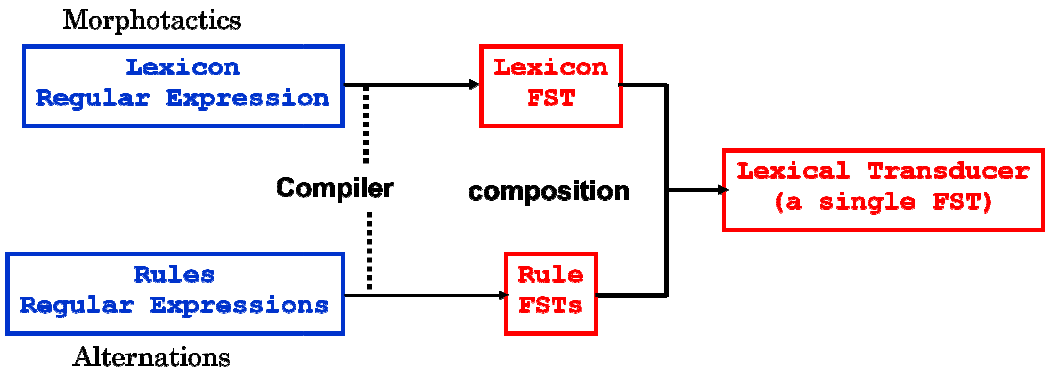

Figure 6-1: The Lexical Transducer