A Compiler for Morphological Analyzer Based on Finite-State Transducers

Bhuvaneshwari C. Melinamath, A.G. Math, and Sunanda D. Biradar

B.L.D.E.A.'s Engg College and Tech. Bijapur 586103,
Karnataka
bmelinamath@yahoo.co.in,
agmath@yahoo.com,
sunanda_biradar@rediffmail.com

Abstract. Morphological analyzers are an essential parts of many natural language processing (NLP) systems such as machine translation systems. They may be efficiently implemented as finite state transducers. This paper describes a morphological system that can be used as stemmer, lemmatizer, spell checker, POS tagger, and as E-learning tool for Kannada learning people giving detailed explanation of various morphophonemics changes that occur in saMdhi. The language specific components, the lexicon and the rules, can be combined with a runtime engine applicable to all languages. Building Morphological analyzer/generator for morphologically complex and agglutinative language like Kannada is highly challenging. The major types of morphological process like inflection, derivation, and compounding are handled in this system.

Keywords: FST, POS, NLP, DFA, NFA.

1 Introduction

Morphological analyzers are essential parts of many natural language processing systems such as machine translation systems. Morphological analysis reads the inflected surface form of each word in a text and writes its lexical form consisting of of a canonical form of the word and a set of tags showing its syntactic category and morphological characteristics. The analyzer relies on two sources of information: a dictionary of valid lemmas of the language and a set of rules for inflection handling.

Finite state transducers (FST) is a most efficient approach to morphological analysis (M. Mohri 1997: Oncina et al. 1993) a class of finite state automata, is a complete example using an intuitive pattern matching approach which tries first to decompose the word in number of stem inflection pairs which are subsequently validated. There are a number of tools for the construction of FST based morphological analyzers the best known being developed at Xerox (Karttmen 1994: Karttmen 1993: Chanod 1994) for a review in Spanish on finite state morphology.

In this work a FST based morphological analyzer is developed. It is a compiler that reads a morphological dictionary containing static description of lemmas and inflections and writes a PERL program that implements a compact FST based analyzer performing the task. This allows the linguist to focus on describing the lexicon and morphology of the language in question in a simple format.

C. Singh et al. (Eds.): ICISIL 2011, CCIS 139, pp. 81–85, 2011.

2 Finite State Transducers

The morphological analyzers are based on finite state transducers; in particular, we use string or pattern transducers instead of letter transducers (Roche & Schabes 1997). Any finite-state transducer may always be turned into an equivalent letter transducer. Instead of transition on letters we have transitions on sequences of letters i.e, strings, and generally valid suffixes in the language. The machine starts in the specified initial state and reads in a string of symbols from its alphabet. The automaton uses the state transition function to determine the next state using the current state, and the symbol just read or the empty string. However, "the next state of an NFA depends not only on the current input event, but also on an arbitrary number of subsequent input events. Until these subsequent events occur it is not possible to determine which state the machine is in. If, when the automaton has finished reading, it is in an accepting state, the NFA is said to accept the string, otherwise it is said to reject the string. When the last input symbol is consumed, the NFA accepts if and only if there is *some* set of transitions that will take it to an accepting state. Equivalently, it rejects, if, no matter what transitions are applied, it would not end in an accepting state. Unlike a DFA, it is non-deterministic in that, for any input symbol, its next state may be any one of several possible states. Thus, in the formal definition, the next state is an element of the power set of states.

The transducer is defined as $T = (Q, L, \delta, qI, F,)$ where Q is a finite set of states, L a set of transition labels, $qI \in Q$ the initial state, $F \subseteq Q$ the set of final states, and $\delta : Q \times L \rightarrow 2^Q$ the transition function (where 2^Q represents the set of all finite sets of states). The set of transition labels is $L = (\Sigma \cup \{\varepsilon\} \times (\Gamma \cup \{\varepsilon\})$ where Σ is the alphabet of input symbols, Γ the alphabet of output symbols, and ε represents the empty symbol. According to this definition, state transition labels may therefore be of four kinds: $(\sigma : \gamma)$, meaning that symbol $\sigma \in \Sigma$ is read and symbol $\gamma \in \Gamma$ is written $(\sigma : \varepsilon)$, meaning that a symbol is read but nothing is written; $(\varepsilon : \gamma)$, meaning that nothing is read but a symbol is written; and $(\varepsilon : \varepsilon)$ means that a state transition occurs without reading or writing. The last kind of transitions are not necessary neither convenient in final FSTs, but may be useful during construction. It is customary to represent the empty symbol ε with a zero ("0"). A letter transducer is said to be deterministic when $\delta : Q \times L \rightarrow Q$. Note that a letter transducer which is deterministic with respect to the alphabet $L = (\Sigma \cup \{\varepsilon\} \times (\Gamma \cup \{\varepsilon\})$ may still be non-deterministic with respect to the input Σ.

A string $w' \in \Gamma^*$ is considered to be a transduction of an input string $w \in \Gamma^*$ if there is at least one path from the initial state qI to a final state in F whose transition labels form the pair $w : w'$ when concatenated. There may in principle be more than one of such paths for a given transduction; this should be avoided, and is partially eliminated by determinization. On the other hand, there may be more than one valid transduction for a string w (in analysis, this would correspond to lexical ambiguity; in generation, this should be avoided). In analysis, the symbols in Σ are those found in texts, and the symbols in Γ are those necessary to form the lemmas and special symbols representing morphological information, such as <noun>, <feminine>, <first person p1. second person p2. Third person p3> for pronouns, etc. In generation, Σ and Γ are exchanged. The general definition of letter transducers is completely

parallel to that of non-deterministic finite automata (NFA) and that of deterministic letter transducers, parallel to that of DFA; accordingly, letter transducers may be determinized and minimized (with respect to the alphabet L) using the existing algorithms for NFA and DFA (Hopcroft & Ullman 1979; Salomaa 1973; van de Snepscheut1993). Transitions labeled (ε : ε) may be eliminated during determinization using a technique parallel to ε closure. For all one writes if and only if q can be reached from p by going along zero or more ε arrows. For any , the set of states that can be reached from p is called the **epsilon-closure** or ε **-closure** of p, and written as

$$E(\{p\}) = \{q \in Q : p \xrightarrow{\epsilon} q\}.$$

For any subset, define the ε -closure of P as for any P⊂Q subset, define the ε-closure of P as

$$E(P) = \bigcup_{p \in P} E(\{p\}).$$

Which allows a transformation to a new state without consuming any input symbols. For example, if it is in state 1, with the next input symbol an a, it can move to state 2 without consuming any input symbols, and thus there is an ambiguity: is the system in state 1, or state 2, before consuming the letter a. Because of this ambiguity, it is more convenient to talk of the set of possible states the system may be in. Thus, before consuming letter a, the NFA-epsilon may be in any one of the states out of the set {1,2}. Equivalently, one may imagine that the NFA is in state 1 and 2 'at the same time': and this gives an informal hint of the power set construction 2^Q.

Unlike other compilers like Karttunen's (1993), the compiler described in this paper builds transducers having no cycles (transitions form a directed acyclic graph) which, in addition, have a unique final state. The absence of cycles is due to the fact that only concatenations and alternations are allowed in the morphological dictionary (see section 3)1. To minimize the resulting transducer, we use an algorithm described by van de Snepscheut (1993), which has two identical steps which may be summarized as follows: in each step, the transition arrows in the letter transducer are reversed, so that the final state is initial and the initial state is final, and the resulting transducer is determinized with respect to L (that is, new states are formed with sets of old states so that the new δ is δ : Q×L→ Q). The transducer resulting from the double reversal determinization process is minimal. This algorithm is particularly efficient in the case of acyclic letter transducers. Moreover, the two steps have a simple interpretation: the first step joins common endings (finds regularities in suffixes) and the second one joins common beginnings of transduction (finds regularities in prefixes). FST-based analyzers output all possible analyses.

3 Morphological Dictionary

The morphological dictionary is a text file. Any text starting with “#" is ignored and may be used as a comment. The dictionary has the following three sections: 1. The symbol declaration section representing the actual root words in the language. 2. Second section represent categories of the word followed by morphological features

such as (such as <feminine> or <singular>) are explicitly declared. First second fields are separated by two vertical bars "||". 3. The information section, where any morph relevant information like real u, past participle form of irregular verbs information etc. are declared: when lemmas in the dictionary share a common infection pattern, this pattern may be given a name in another file handling the inflection of the words in the language. While generating the inflections for a word the information field is looked upon. Rules may be indefinitely nested, that is, the names of rules previously defined may be used to define derived forms of the words. (Category wise set of rules are compiled into sub transducers that are then integrated to build the complete transducer). The transition on suffix 'a' in Kannada stand for a genitive case of nouns also for negation for verbs and another sense of imprecate meaning of verb in some case. Hence we get more than one analyzes. All are valid transitions. Since context is not considered here.

```
taavu || PRO-REF-P23.MFN.PL-NOM
tamma || PRO-REF-P23.MFN.PL-GEN||N-COM-COU-M.SL-NOM::TYPE-kinship
biMdu || N-COM-COU-N.SL-NOM::LV-real-u
cakshu || N-COM-COU-N.SL-NOM::LV-real-u
caru || N-COM-COU-N.SL-NOM::LV-real-u
daaru || N-COM-UNC-N.SL-NOM::LV-real-u
dattu || N-COM-COU-N.SL-NOM::LV-real-u
```

Fig. 1. Sample of the morphological dictionary

The null transitions are allowed since it is a NFA automata. We are not preserving any fixed length pattern strings since we perform transitions on suffixes not on single letter; Separate rule file is used to holds set of orthographic rules, it is not a part of dictionary. The idea behind holding separate file is to make the system language independent, the code is not hard wired i.e, SaMdhi rules governing insertion or deletion of vowels are put in separate file and not written as part of code. And another advantage is the same code can be used for other languages too just by replacing orthographic rule file with their language morphophonemic. The valid transitions may be an entry in the another file called FST transitions file.

4 The Compiler

The compiler has been developed under Linux using Perl. Which reads in the morphological dictionary file and combines the partial transducers corresponding to the declared paradigms and the dictionary entries into a single transducer containing one initial and one final state using (ε: ε) transition.

Error messages are designed to help the linguist correct possible errors in the format of the morphological dictionary. The back end minimizes the resulting transducer and combines the resulting code with a standard skeleton to produce a Perl program which is ready to be used on its own or included in a larger application such as a machine translation system.

5 Experiments and Comparisons

We perform the experiments to evaluate the heuristic used by the compiler and used the morphological dictionary of 10000 words compiled by us following hierarchical tag set for Kannada which covers more detailed analysis of the word, design of Hierarchical tag set for Kannada was an another stage towards developing the morphological analyzer for Kannada, and for testing, the first 5000 most frequent Kannada words of Department of Electronics (DoE), Central Institute of Indian Languages (CIIL) corpus words are selected which included all the nominal, pronominal, adjectival and verbal inflections of the Kannada language. Around 80% words were analyzed correctly, manually checked and verified, remaining 20% words were a mixture of spelling variations, dialect variations, compound words, hence such words in the raw corpus selected for testing are not analyzed, regarding ambiguity error is very less and missing entries in dictionary. Currently we are not handling compound words and dialectic variations. This system is first of its kind for Kannada using FST.

6 Concluding Remarks

A compiler to automatically build finite state transducer based morphological analyzers using set of rules consisting of features for categories and another file with FST transition rules and morphological dictionaries has been described. This tool may be of great interest when building natural language processing systems such as machine translation programs. When the linguist does not supply an explicit alignment between surface forms and lexical forms, the compiler uses a simple heuristic to produce an alignment that has been experimentally shown to be equally efficient. We are currently testing an extended version the program which to handle spelling variations, few dialectic variations and to improve dictionary as per the need of hierarchical design of tag set.

References

1. Karttunen, L.: Finite-state lexicon compiler. Technical Report ISTL-NLTT, Xerox Palo Alto Research Center, Palo Alto, California (1993-04-02)
2. Oncina, J., García, P., Vidal, E.: Learning subsequential transducers for pattern recognition interpretation tasks. IEEE Transaction on Pattern Analysis and machine Intelligence 15, 448–458 (1993)
3. Hopcroft, J.E., Ullman, J.D.: Introduction to automata theory, languages and computation. Addition -Wesley, Reading (1979)
4. Mohri, M.: Finite-state transducers in language and speech processing. Computational Linguistics 23(2), 269–311 (1977)
5. Roche, E., Schabes, Y.: On the use of sequential transducers in natural language processing. In: Finite State Language Processing, pp. 353–382. MIT Press, Cambridge (1997b)
6. Salomaa, A.: Formal Languages. Academic Press, New York (1973)
7. Van de Snepscheut, J.L.A.: What computing is all about. Springer, New York (1993)
8. Chanod, J.-P.: Finite state composition of French verb morphology. Technical Report Technical Report MLTT-005, Xerox Research Centre Europe, Meylan, France (1994)