Multilingual *Akshar* Based Transducer for South and South East Asian Languages which Use Indic Scripts

Anil Kumar Singh
Language Tech. Research Centre
Int'l Inst. of Information Tech.
Hyderabad, India
anil@research.iiit.net

Harshit Surana
Language Tech. Research Centre
Int'l Inst. of Information Tech.
Hyderabad, India
surana.h@gmail.com

Abstract

Akshar is an orthographic unit in Indic scripts. In this paper, we show how this unit can be the basis of processing text in languages using Indic scripts. These languages are spoken by over a billion people. We describe a method for multilingual lexicon compilation in the form of an *akshar* based transducer. This method exploits the similarities among Indic scripts as well as a large overlap among the vocabularies of these languages. Apart from compression, this transducer also has the advantage that it can be used for many multilingual and crosslingual applications, some of which we briefly describe in this paper.

1 Introduction

Indic scripts are derived from the ancient Brahmi scripts (Sproat, 2002; Sproat, 2006). There is lot of similarity among them which can be exploited for text processing. A larger number of major languages of South and South East Asia use these scripts. One of the characteristics of these scripts is that they are non-linear and the text written in them can be broken up into *akshars*, which are the second level orthographic units in Indic scripts (the first being letters). The closest English term for such units is syllable.

An *akshar* is either an independent vowel or it begins with a consonant and ends with a vowel marker (*maatraa*), a schwa deletion marker (*halant*) or a nasalization marker (*anusvar* or *chandrabindu*) or (in scripts like Devanagari) a consonant modifier (*nukta*). These constraints are shown in the form of an FST in Figure-1.

In this paper, we first define an *aaksharik* (an adjective derived from *akshar*) model for Indic scripts. Then we describe a multilingual *akshar* based lexical transducer for languages using Indic scripts. Our work could be compared to that of Kwon and Karttunen (Kwon and Karttunen, 1994), who have described a lexical transducer for Korean, but it was not a multilingual transducer. Finally we discuss how this transducer can be used for applications like indexing of multilingual lexical resources and crosslingual study of lexicon and affixes, e.g. for similarities and variation. Our contributions here include using *akshar* as the unit for building a lexical transducer, giving an algorithm for fuzzy string alignment using the *akshar* transducer, presenting the results of some studies on this transducer and suggesting some applications.

2 *Aaksharik* Model

This model defines the grammar for *akshars* in Indic scripts. It also defines operations that can be performed on *akshars* such as **expand** and **collapse**. Here we will focus only on the grammar. The *aaksharik* model formalizes the available knowledge about *akshars* so that the model can be correctly implemented and used for computational purposes. *Akshar* has been described in the previous section, but we can

Akshar ::= Vowel Akshar | Consonant Akshar
Vowel Akshar ::= Vowel [Vowel Modifier]
Consonant Akshar ::= Pure Consonant
| Modified Consonant
Pure Consonant ::= Consonant [Nukta] Halant
Modified Consonant ::= Consonant [Nukta]
[Maatraa] [Vowel Modifier]

S: Start
C: Consonant
V: Vowel
M: Maatraa
N: Nukta
H: Halant
D: Vowel modifier

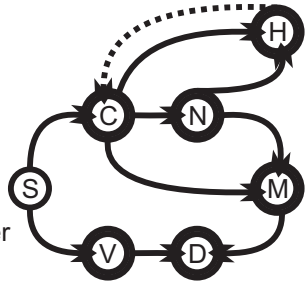


Figure 1: The *akshar* grammar represented as an FST and also in the Backus Naur Form (BNF), also called the Panini Backus Form. The dotted arc is for (optionally) allowing consonant clusters.

add here that two alternative variation of that definition are possible: one allows consonant cluster in an *akshar* and one does not.

Considering *akhsar* at the computational level ensures that we take into account one of the inherent characteristic of Indic scripts. These scripts are not free order scripts (unlike Latin scripts) in the sense that there are rules which govern the order of consonants, vowels, and other 'letters'.

There are several reasons why an *aaksharik* model is needed. One is that *akshar* is the most important unit in Indic orthography. Then, many combinations of letters are not possible in Indic scripts. Including the knowledge about how *akshars* can be formed and what operation can be performed on them can help us in many applications. Some such knowledge is already being used for display of text in Indic scripts. However, there is still a lot of potential for putting together this knowledge and using it for more Natural Language Processing (NLP) applications.

The grammar is represented as an FST and in the Backus Naur Form (BNF) Figure-1. Some terms need to be explained here. Vowel modifier is a letter which usually represents nasalization. In Devanagari script, there are two such letters: *anusvar* and *chandrabindu*. *Nukta* is a kind of consonant modifier which is added to some consonants to represent sounds borrowed from Persian or English. *Halant* represents explicit absence of a vowel sound after a consonant. *Maatraa* is a letter representing a bound vowel, i.e., a vowel that comes after a consonant. In Indic scripts, bound and free vowels are written differently.

3 Multilingual *Akshar* Based Transducer

In multilingual *akshar* based transducer, we take an *akshar* as a unit, i.e., the nodes of the transducer are *akshar*. One of the important characteristics of Indic scripts is that they have very similar alphabets, even though the shapes of the letters differ. To allow words from different languages to be included in the same transducer, we can map the letters of different Indic scripts to letters of one selected script (say, Devanagari), or we can use a standard 'super encoding' like ISCII which can represent text in all Indic languages. Figure-2 shows how a multilingual *akshar* based transducer can accommodate words from different languages. Though this transducer can be applied for all languages which use Indic scripts, it might need some minor modifications for some scripts. The main reason it can be easily used for all Indic scripts is that the alphabet which we are considering is the superset of the alphabets used for Indic scripts.

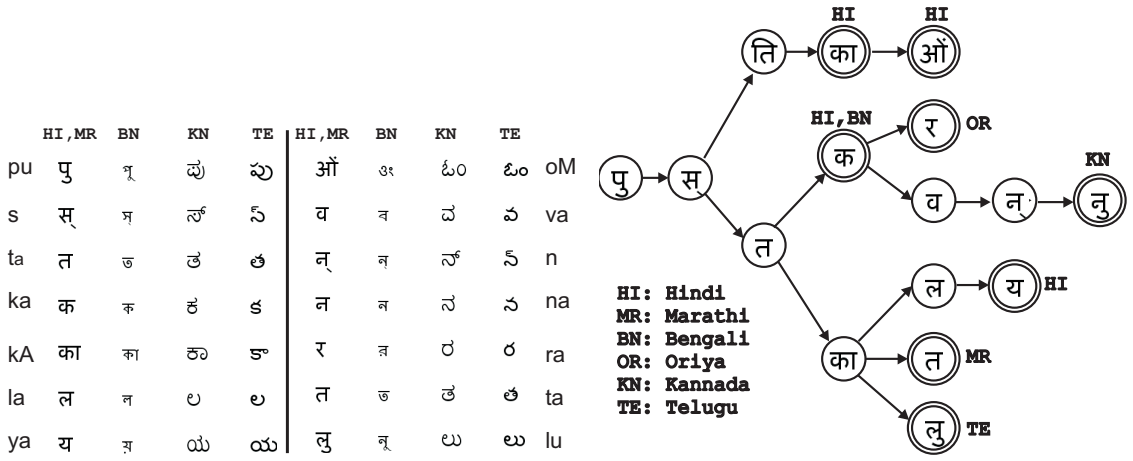


Figure 2: Multilingual *akshar* based transducer. The nodes are the *akshars*. Most of the *akshars* are common among all languages. The nodes with double circles are the accepting states. The input symbols are the *akshars*. The output symbols are the words and the codes of the languages in which they occur. Some words may be common among different languages. This is shown by (HI, BN), i.e., the node concerned is ending a word which occurs in both Hindi and Bengali.

4 Alignment Algorithm for *Akshar* Transducer

An alignment algorithm called the DTW algorithm (Myers and Rabiner, 1981) is heavily used in speech recognition and for problems like gene sequencing. One version of this algorithm can be roughly described as follows:

```
Let the query string be Sq
Let the retrieval string be Sr

m = stringLength(Sq)
n = stringLength(Sr)

initMatrix DTW[m,n]

for i = 1 to n
 for j = 1 to m
   cost = SDF[Sq[i], Sr[j]]
      * K(i,j)

   // insertion
   DTW[i,j] = min(DTW[i-1, j]
      + cost,

   // deletion
   DTW[i, j-1] + cost,

   // substitution
   DTW[i-1, j-1] + cost)
```

Here, $K(i, j)$ is a heuristic function which can take into account language specific issues like the inflectional nature of a language, e.g. giving the last two characters (which are most likely to represent an inflection) a lesser weight for Hindi. $SDF[Sq[i], Sr[j]]$ is the cost between two letters $Sq[i]$ and $Sr[j]$ of the two strings which are being compared at a particular node in the trellis.

We use an algorithm adapted for the *akshar* transducer which has the same conceptual basis, i.e., it is a dynamic programming algorithm and provides for operations like insertion, deletion and substitution. It can be visualized as in Figure-3. The figure shows a small fragment of the transducer in which fuzzy search is being performed for a word *piCalI*. Since this

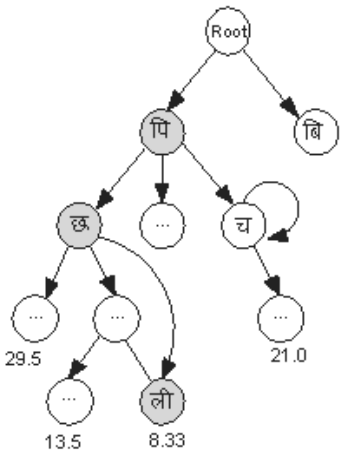


Figure 3: Alignment algorithm for the akshar transducer. There are three kinds of arcs. The arc which skips a node represents deletion, the one which points back to the same node represents insertion, and the straight normal arc with non-zero cost represents substitution.

word is not in the dictionary compiled as the transducer, another word *piCa*lI* has matched, where * denotes some *akshar* which is not *lI*. The figure also shows the three possible arcs, representing insertion, deletion and substitution. The darkened nodes are those which have matched with one of the *akshars* in the word being aligned or searched.

The figure indicates the basic formulation of the alignment algorithm. However, several optimization techniques were used to increase the speed of search or alignment.

5 Applications

A *aaksharik* transducer can be used as a computational tool for many applications. A multilingual transducer for related languages using similar scripts can be even more useful. Some of the applications are described here briefly.

5.1 Compression of Dictionaries in Many Languages

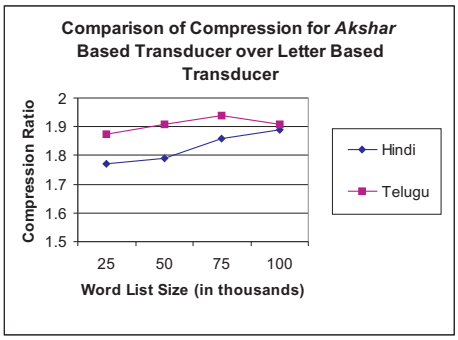
Lexical resources are now needed even on devices like mobile phones (Jagannathan and Jawahar, 2005), which may have limited memory or storage space. The transducer described earlier can be used to compress lexical resources for many languages. Though there are other compression techniques, with a multilingual transducer we do not need to decompress. The time complexity of searching a word is of the order of $max(l_w)$, where l_w is the length of a word (or lexical item) w in terms of *akshars*. The results of our experiment on comparing compression are shown in Figure-4. 'All' includes 10 languages from the CIIL corpus. 'Indo-Aryan' include 6 languages. The second case shows an improvement, which means that compression is more for more related languages. 'Related pairs' include two very close languages like as Assamese and Bengali. This case shows even more improvement. We have not yet actually tried using the transducer for small mobile devices.

5.2 Indexing of Multilingual Lexical Resources

Lexical resources may have a lot of information associated with a lexical item. This information may differ depending on the kind of resource we are interested in. For example, a monolingual dictionary may have part of speech, possible meanings, origin, example sentences etc. A WordNet like resource might have hypernyms, synsets, etc. For languages using similar scripts, we can use a single multilingual transducer as an indexing mechanism in which each lexical item will have pointers to different kinds of information.

5.3 Error Tolerant Finite State Recognition

Error tolerant finite state recognition as described by (Oflazer, 1996) can be applied more efficiently for Indic scripts using our method



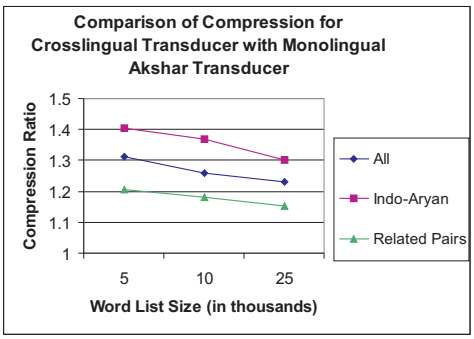


Figure 4: Comparison of compression in terms of compression ratio. Note that compression for *akshars* based transducer is more than 1.7 times better than for letter based compression. Also, compression for multilingual transducer is also significantly more than monolingual transducer. Therefore, compression for *akshars* based multilingual transducer is more than twice that for letter based monolingual transducer.

as *akshar* is the most relevant unit for these scripts. The insights from other works on string search (Wu and Manber, 1992; Baeza-Yates, 1999) may be useful in improving our work.

5.4 Crosslingual Study of Lexicon and Affixes

Since words from different languages are being stored in the same transducer in the same encoding, the similarities and differences among languages can be studied and analyzed easily for linguistics or computational purposes. By applying some heuristics or by using statistical techniques, we can study the roots and affixes of words in different languages. In other words, this transducer can be very useful for crosslingual lexical and morphological study.

6 Speed and Accuracy

To compare the performance of an *akshar based transducer* (ABT) with a letter based transducer (LBT), we conducted some experiments. These experiments were meant to evaluate speed and accuracy.

The parameter used to estimate speed was trie depth, as speed should increase exponentially with increase in trie depth. The results for Hindi as well Telugu are shown in Figure-5. Word lists of 50,000 from the CIIL (Central Institute of Indian Languages) corpus were compiled for both the languages. It is clear from this figure that the average depth of a word is significantly lower for ABT (Hindi: 3.62, Telugu: 4.31), even in linear terms. This implies a much higher increase in speed over LBT (Hindi: 6.22, Telugu: 7.69). Note that the average depth for Telugu is more mainly because it is morphologically richer and more agglutinative than Hindi. There is scope for further work on improving the transducer to take care of agglutination, which is common in South Indian languages.

The accuracy is likely to depend on the application. For example, the accuracy will be different for spell checking and cognate identification. Our experiments was conducted for cognate identification. We took 100 word pairs for Hindi to Bengali. The precision for LBT was 46%, whereas that for ABT was 52%. Thus, there is an improvement even in accuracy, at least for cognate identification.

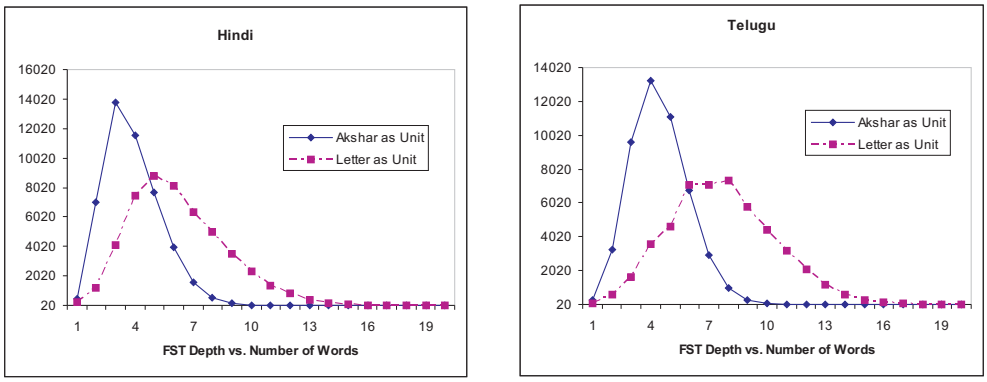


Figure 5: Comparison of tree depth vs. number of words. The mean depth for *akshar* and letter based transducers for Hindi is 7252.24 and 12454.84, respectively. The corresponding values for Telugu are 10833.3 and 9722074.25. Thus, the depth of *akshar* based transducer is 0.52 of letter based transducer for Hindi and 0.56 for Telugu.

7 Conclusion

We presented an *akshar* based model of Indic scripts and described an *aaksharik* transducer which can be used to compile multilingual lexicon for languages using Indic scripts. In addition to being a way to compress multilingual lexical resources, it can be used for various applications, e.g for indexing of multilingual lexical resources or as a versatile tool for study of lexicon and affixes across languages. It can help in significantly reducing the effort required in building resources for several languages which are related and have a large overlap in vocabulary. We also presented the results of some experiments which show that using an *akshar* transducer can lead improvements in compression, speed as well as accuracy.

References

R. Baeza-Yates. 1999. Faster Approximate String Matching. *Algorithmica*, 23(2):127–158.

L. Jagannathan and C.V. Jawahar. 2005. Crosslingual access of textual information using camera phones. In *Proceedings of the International Conference on Cognition and Recognition*.

Hyuk-Chul Kwon and Lauri Karttunen. 1994. Incremental construction of a lexical transducer for korean. In *Proceedings of the 15th conference on Computational linguistics*, pages 1262–1266, Morristown, NJ, USA. Association for Computational Linguistics.

C. S. Myers and L. R. Rabiner. 1981. A comparative study of several dynamic time-warping algorithms for connected word recognition. In *The Bell System Technical Journal, 60(7)*, pages 1389–1409.

Kemal Oflazer. 1996. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1):73–89.

Richard Sproat. 2002. Brahmi scripts. In *Constraints on Spelling Changes: Fifth International Workshop on Writing Systems*, Nijmegen, The Netherlands.

Richard Sproat. 2006. Brahmi-derived scripts, script layout, and phonological awareness. *Written Language and Literacy*.

S. Wu and U. Manber. 1992. Fast text searching: allowing errors. *Communications of the ACM*, 35(10):83–91.