Konkanverter - A Finite State Transducer based Statistical Machine Transliteration Engine for Konkani Language

Vinodh Rajan
School of Computer Science
University of St Andrews
Scotland, UK
`vrs3@st-andrews.ac.uk`

Abstract

We have developed a finite state transducer based transliteration engine called *Konkanverter* that performs statistical machine transliteration between three different scripts used to write the Konkani language. The statistical machine transliteration system consists of cascading finite state transducers combining both rule-based and statistical approaches. Based on the limited availability of parallel corpora, this cascading approach is found to perform significantly better than a pure rule-based approach or pure statistical approach.

1 Introduction

Konkani is an Indian language spoken by approximately 2.5 million people (Gov. of India, 2001), mainly in the Indian state of Goa. It also has a substantial amount of linguistic minority population living in neighboring states of Karnataka and Kerala. In Goa, Konkani uses the Devanagari script and the Roman script (locally known as *Romi*). In Karnataka and Kerala, the dominant regional scripts, Kannada and Malayalam, are being used to write the language. Muslim sections of the Konkani population are also known to use a Perso-Arabic based alphabet. This polygraphic scenario is unique to Konkani in contemporary Indian linguistic milieu.

Among these different orthographies, Devanagari, Kannada and the Roman script are the mainstream orthographic systems. For all practical purposes, Konkani can therefore be considered to possess synchronic trigraphia. There are several important features that differentiate these orthographies. Consonants of Indic scripts carry an inherent schwa, which is unmarked, while other vowel combinations with a consonant are represented as combining signs. However, absence of schwa in a consonant is marked by explicit orthographic consonantal clusters or using a special sign called a *Virāma*. All orthographies other than Devanagari universally show explicit schwa deletion. Devanagari and Kannada distinguish vowel lengths, but their distribution and representation are very idiosyncratic to each orthography. In contrast to the Indic scripts, *Romi* does not differentiate vowel length at all. Most importantly, the *Romi* orthography does not distinguish schwa from vowel *o*. Both are represented using the same grapheme *o*. Several Indic graphemic combinations are also rendered as vowel digraphs or trigraphs in *Romi*. As a result, many Indic sequences are merged in *Romi* orthography. Table 1 lists some sample words in various orthographies.

Synchronic trigraphia is a major issue of political contention inside the community, each group favoring the usage of a particular script as the official orthography. Different orthographic communities exist in isolation with minimal interaction and with its own literary tradition, as very few people are fluent in multiple orthographies. A statistical machine transliteration engine with reasonable accuracy would greatly enable cohesion and interaction among the greater linguistic community. Facilitating the usage of multiple scripts would also encourage more linguistic diversity among the community.

Devanagari	*Kannada*	*Romi*
देवनागरी dēvanāgarī	ದೇವ್ನಾಗರಿ dēvnāgari	devnagri
झटको jhaṭakō	ಝಟ್ಕೊ jhaṭko	zhottko
दिवंचे divaṃcē	ದಿಂವ್ಚೆ diṃvce	dinvche
देवादयेन dēvādayēna	ದೆವಾದಯೆನ್ devādayen	devadoien
सत्तेवयल्या sattēvayalyā	ಸತ್ತೆವಯ್ಲ್ಯಾ sattevaylyā	sat'tevoilea

Table 1: Sample *Konkani* words in various orthographies

2 Related Work

Machine transliteration frequently occurs within machine translation when either named entities or out of vocabulary (OOV) words are encountered. Machine transliteration is also useful for cross-language information retrieval (CLIR). Consequently, a significant amount of work has been done in this field (Arbabi et al., 1994; Knight and Graehl, 1998). Machine transliteration can generally be classified as rule-based or statistical depending on the approach.

Rule-based transliteration is typically performed through hand-crafted rules and is usually graphemic in nature. Within Indic transliteration, there have been several attempts on rule-based approaches. Malik et al (2008) implemented a Hindi-Urdu transliteration system with finite-state transducers using a universal intermediate transcription (UIT). It was based on the graphemic equivalence between Perso-Arabic script and Devanagari script. Similarly, Kishorjit (2011) developed a rule-based transliteration system also based on direct graphemic correspondence between Meetei Mayek and Bengali script.

On the other hand, statistical machine transliteration systems typically use procedures familiar from statistical machine translation, including character alignments and subsequent training on the aligned data. Jia et al (2009) also developed a noisy channel model for the English-Chinese language pair using Moses, an SMT tool. Malik et al (2013) evaluated 28 different kinds of statistical models for Hindi-Urdu machine transliteration using GIZA++ and Moses. Similarly, Chinnakotla et al (2009) used the same tools for three language pairs - English-Hindi, English-Tamil and English-Kannada, focusing on fine-tuning the character sequence model (CSM). Singh (2012) evaluated both rule-based and statistical methods for bidirectional Bengali script and Meetei Mayek transliteration. A hybrid approach combining FSM based techniques with a statistical word language model with better performance was proposed by Malik et al (2009).

3 Initial Attempts

Konkani, being a minor language did not have any parallel corpora that could have been harnessed for statistical machine transliteration. The World Konkani Center, Mangalore had attempted to manually mine transliteration rules by studying the three orthographies and analyzing the differences. They also developed schwa deletion and schwa insertion rules for the orthographies, modeled on Hindi schwa deletion rules. For the initial machine transliteration system, we refined and improved upon these rules and implemented a rule-based transliteration system. In the absence of corpora, we iterated the rule-based system in an ardent attempt to improve accuracy. However, the performance of the transliteration engine was not very satisfactory and could not be improved beyond a certain limit. The performance of this rule-based engine is discussed in section 5.

Even though significant effort had to be spent towards the creation of a parallel corpus, it was finally decided to incorporate statistical models to improve accuracy. A monolingual untagged text corpus was obtained for Konkani in Devanagari script. In liaison with linguistic experts, we manually constructed a substantially sized corpus, despite several practical difficulties. Table 2 lists the word count of parallel word lists in each orthographic pair. This corpus will be released into the public domain, after some revisions and proof-checking. Currently it is available on

Orthography	*Word Count*
Devanagari - Kannada	23 187
Devanagari - Romi	38 550
Kannada - Romi	14 396

Table 2: Word count of parallel corpora

request.

We initially contemplated using an interlingua-like approach by choosing Devanagari as the intermediate script, thus reducing the need for a dedicated Kannada-Romi transliteration module and parallel corpus. However, we were skeptical of the error propagation that might occur if the conversion to the intermediate script itself is not very accurate.

4 Architecture

The implementation of the framework has been done entirely using OpenFst (Allauzen et al., 2007). Thrax (Tai et al., 2011) was used to define context-dependent rewrite rules and compile those rules into finite-state transducers compatible with Openfst. Thrax was also used to define finite state acceptors. We found Thrax to be particularly robust and flexible in generating various FSTs. Character alignments were performed using Phonetisaurus (Novak et al., 2011). N-gram models were created with the OpenGrm Ngram library (Roark et al., 2012), which also generates the models as FSTs.

Below, we describe the detailed architecture for each transliteration pair.

4.1 Devanagari to Kannada

Input text was first converted into an intermediate romanized encoding, where schwa is explicitly denoted. This also results in converting the script from syllabic to alphabetic form. This eases defining rules to a considerable extent, since independent vowels and dependent vowel signs need not be dealt with separately. We have used a customized encoding which only uses monographs and hence maintains a direct grapheme-to-grapheme correspondence with Indic scripts. However, to increase readability standardized Indic romanization scheme ISO 15919 has been used instead for presentation in this paper.

Before proceeding with schwa deletion rules, let us define the necessary sets. Let $\mathcal{V}$ stand for the class of vowels, $\mathcal{C}$ for the class of consonants, $\mathcal{C}_{nj}$ be a list of consonants that cannot occur as a second element of a triconsonantal cluster and $\mathcal{D}$ for the dependent signs *Chandrabindu*, *Anusvara* and *Visarga*. The morphemic boundary is denoted by $\mathcal{M}$. The beginning of string and end of string is denoted by s_i and s_f respectively. $\mathcal{E}$ denotes a null character with Σ denoting the entire alphabet.

Let $\mathcal{S}$ be an orthographic syllable, while $\mathcal{S}_{-a}$ is the set of syllables that do not contain schwa. $\mathcal{S}_{nj}$ is the set of syllables that do not start with $\mathcal{C}_{nj}$, while $\mathcal{S}_{nj-a}$ is the set that excludes the syllables with schwa. Using regular expressions, these are defined as:

$$\mathcal{S} \leftarrow \mathcal{C}*\mathcal{V}\mathcal{D}? \quad , \quad \mathcal{S}_{-a} \leftarrow \mathcal{C}*(\mathcal{V}-a)\mathcal{D}? \quad , \quad \mathcal{S}_{nj} \leftarrow \mathcal{C}_{nj}?\mathcal{C}\mathcal{V}\mathcal{D}? \quad , \quad \mathcal{S}_{nj-a} \leftarrow \mathcal{C}_{nj}?\mathcal{C}(\mathcal{V}-a)\mathcal{D}?$$

The initial word boundary $\mathcal{B}_i$ and final word boundary $\mathcal{B}_f$ are: $\mathcal{B}_i \leftarrow \mathcal{M}|s_i$ and $\mathcal{B}_f \leftarrow \mathcal{M}|s_f$.

The general schwa deletion rules have been given below as context dependent rewrite rules. These rules are expressed in the form $\phi \rightarrow \psi/\lambda\_\_\rho$. Here, ϕ is replaced by ψ whenever it is preceded by λ and succeeded by ρ, λ and ρ being the left and right contexts respectively. The rules below are listed with a corresponding sample case. These rules were compiled as finite-state transducers (Narasimhan et al., 2004)

We have produced a list of possible suffixes and prefixes from the collected corpus. Let $\mathcal{P}_{re}$ denote the set of prefixes and $\mathcal{S}_{uf}$ the set of prefixes. We then mark the morphemic boundary

$\mathcal{M}$ as:

$$\mathcal{B}_m = \mathcal{E} \rightarrow \mathcal{M}/\mathcal{P}_{re}\underline{\quad}\mathcal{S}_{uf}$$

The schwa deletion rules can be effectively summarized as follows.

$$\begin{gathered}
\mathcal{W}_f = a \rightarrow \mathcal{E}/(\mathcal{B}_i\mathcal{S} + \mathcal{C}+)\underline{\quad}\mathcal{B}_f \quad | \quad \text{āsāta} \rightarrow \text{āsāt} \\
\mathcal{W}_3 = a \rightarrow \mathcal{E}/(\mathcal{B}_i\mathcal{S}\mathcal{C})\underline{\quad}(\mathcal{S}_{-a}\mathcal{B}_f) \quad | \quad \text{āṃvaḍō} \rightarrow \text{āṃvḍō} \\
\mathcal{W}_{3vy} = a \rightarrow \mathcal{E}/(\mathcal{B}_i\mathcal{S}(y|v))\underline{\quad}(\mathcal{S}\mathcal{B}_f) \quad | \quad \text{payasa} \rightarrow \text{pays} \\
\mathcal{W}_4 = a \rightarrow \mathcal{E}/(\mathcal{B}_i\mathcal{S}\mathcal{C})\underline{\quad}(\mathcal{S}_{nj}\mathcal{S}\mathcal{B}_f) \quad | \quad \text{āḍakātī} \rightarrow \text{āḍkātī} \\
\mathcal{W}_{4y} = a \rightarrow \mathcal{E}/(\mathcal{B}_i\mathcal{S}\mathcal{S}y)\underline{\quad}(\mathcal{S}_{nj}\mathcal{B}_f) \quad | \quad \text{ulayatāṃ} \rightarrow \text{ulāytāṃ} \\
\mathcal{W}_{4_3} = a \rightarrow \mathcal{E}/(\mathcal{B}_i\mathcal{S}\mathcal{S}_{-a}\mathcal{C})\underline{\quad}(\mathcal{S}_{nj}\mathcal{B}_f) \quad | \quad \text{vicāratā} \rightarrow \text{vicārtā} \\
\mathcal{W}_{P2} = a \rightarrow \mathcal{E}/(\mathcal{B}_i\mathcal{S}\mathcal{C})\underline{\quad}(\mathcal{S}_{nj}\mathcal{S}\mathcal{S} + \mathcal{B}_f) \quad | \quad \text{vācatakūca} \rightarrow \text{vāctakūc} \\
\mathcal{W}_{Pl} = a \rightarrow \mathcal{E}/(\mathcal{B}_i\mathcal{S}\mathcal{S}(y|l|v))\underline{\quad}(\mathcal{S}_{nj}\mathcal{S}\mathcal{S} + \mathcal{B}_f) \quad | \quad \text{vājayatālō} \rightarrow \text{vājaytālō} \\
\mathcal{W}_{P3} = a \rightarrow \mathcal{E}/(\mathcal{B}_i\mathcal{S}_{-a}\mathcal{S})\underline{\quad}(\mathcal{S}_{nj}\mathcal{S}\mathcal{S} + \mathcal{B}_f) \quad | \quad \text{juvānapana} \rightarrow \text{juvānpan} \\
\mathcal{W}_{tri} = a \rightarrow \mathcal{E}/(\mathcal{S}\mathcal{C})\underline{\quad}\mathcal{S}_{nj} \quad | \quad \text{gōyāṃtalyā} \rightarrow \text{gōyāṃtlyā} \\
\mathcal{W}_{grm} = a \rightarrow \mathcal{E}/(\mathcal{B}_i\mathcal{S}\mathcal{S}\mathcal{S} + (w|y))\underline{\quad}l\mathcal{V}\mathcal{D}?) \quad | \quad \text{gōyāṃtalē} \rightarrow \text{gōyāṃtlē}
\end{gathered}$$

Let $\mathcal{B}_r$ denote the removal of morphemic boundaries. The overall schwa deletion can be constructed by composing all of the above transducers:

$$\mathcal{W}_d = \mathcal{B}_m \circ \mathcal{W}_3 \circ \mathcal{W}_{3vy} \circ \mathcal{W}_{4y} \circ \mathcal{W}_{4_3} \circ \mathcal{W}_{P2} \circ \mathcal{W}_{Pl} \circ \mathcal{W}_{P3} \circ \mathcal{W}_{tri} \circ \mathcal{W}_{grm} \circ \mathcal{W}_f \circ \mathcal{B}_r$$

Devanagari has two additional vowels ऍ *ĕ* and ऑ *ŏ*, and two special conjunct characters ऱ्य *ṛy* and ऱ्ह *ṛh*. They were directly mapped to the Kannada characters ಎ *e*, ಒ *o*, ಯಽ *ry* and ಹಽ *rh* respectively. Let this mapping be $\mathcal{R}_{vc}$.

In Kannada, *i* and *u* are used at word endings, where *ī* and *ū* are found in Devanagari.

$$\begin{gathered}
\mathcal{R}_i = \bar{i} \rightarrow i/(\mathcal{C} + \underline{\quad}\mathcal{D}?\mathcal{B}_f \quad | \quad \text{satī} \rightarrow \text{sati} \\
\mathcal{R}_u = \bar{u} \rightarrow u/(\mathcal{C} + \underline{\quad}\mathcal{D}?\mathcal{B}_f \quad | \quad \text{vastū} \rightarrow \text{vastu}
\end{gathered}$$

The Devanagari sequence *vaṃk* corresponds to *ṃvk* in Kannada, let this be $\mathcal{R}_{vmk}$ The overall rule-based transduction is:

$$\mathcal{R}_{kn} = \mathcal{W}_d \circ \mathcal{R}_{vc} \circ \mathcal{R}_i \circ \mathcal{R}_u \circ \mathcal{R}_{vmk}$$

The precedence for the rule-based compositions were decided based on emperical observations.

The input word $\mathcal{I}$ is composed with the overall rule-based transducer. The second (output) projection of this transducer is used for later operations.

$$\mathcal{R}_{knp} = \pi_2(\mathcal{R}_{kn} \circ \mathcal{I})$$

When either *ī* or *ū* appears before a final schwa-consonant, with or without a preceding *r* or *ṃ*, it can retain its length or become short. Also, *ē* and *ō*, which usually transform into short vowels *e* and *o*, are retained in case of loan words. Additional arcs were added to the transducer $\mathcal{R}_{knp}$ to reflect this. The new transducer $\mathcal{R}_{knm}$ contains multiple paths for a given input.

We then created a lexical acceptor $\mathcal{A}_{Lkn}$ from the Kannada words in the corpus. For a given lattice, this removes all non-lexical entries.

$$\mathcal{T}_{knl} = \mathcal{R}_{knm} \circ \mathcal{A}_{Lkn}$$

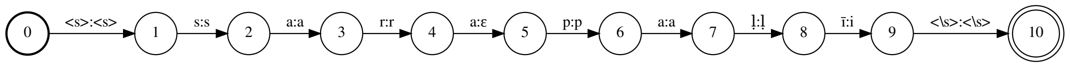

Figure 1: Transducer $\mathcal{R}_{kn}$ for the Devanagari input सरपळी *sarapaḷī*

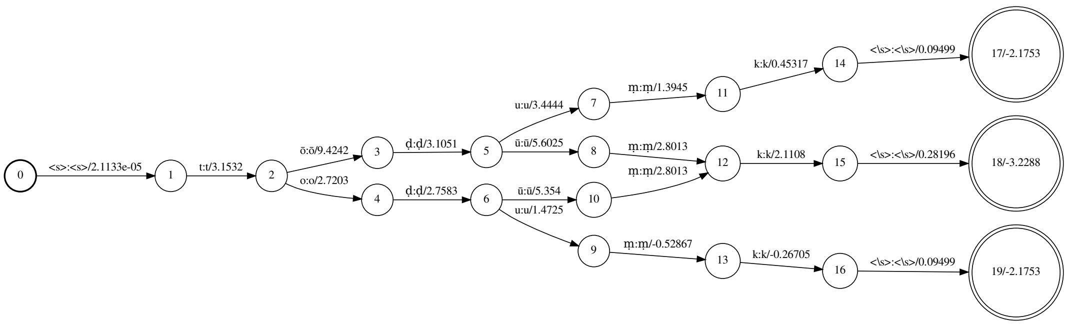

Figure 2: Weighted Transducer $\mathcal{T}_{dv2kn}$ for the Devanagari input तोड़ूंक *tōḍūṃka*. Weights indicate negative log n-gram probabilities

In case none of the outputs are present in the lexicon, or if multiple paths are lexically valid (in case of several standard variants), we proceed to choose the best path by utilizing n-gram probabilities. An n-gram word model $\mathcal{N}_{kn}$ was generated based on the list of Kannada words in the corpus.

If $\mathcal{T}_{knl}$ is null then,

$$\mathcal{T}_{dv2kn} = \mathcal{R}_{knm} \circ \mathcal{N}_{kn}$$

Else,

$$\mathcal{T}_{dv2kn} = \mathcal{T}_{knl} \circ \mathcal{N}_{kn}$$

$\mathcal{T}_{dv2kn}$ is the final transducer that transliterates Devanagari to Kannada. The best path was chosen from the lattice as the most probable transliteration.

4.2 Kannada to Devanagari

Kannada to Devanagari transliteration is more complex than Devanagari to Kannada. Since Kannada orthography shows explicit schwa deletion, schwa needs to be inserted in this case.

First, we inverted the schwa deletion transducer $\mathcal{W}_d$, effectively making it perform schwa insertion. However, reversing the schwa deletion results in multiple alternate paths, all of which are theoretically viable. In order to prune the lattice, a cluster acceptor $\mathcal{A}_{Cdv}$ was created. $\mathcal{A}_{Cdv}$ rejects all paths that contain non-standard consonantal clusters for the Devanagari orthography. This list of non-standard clusters was manually created by analyzing the Devanagari corpus.

For example, the word ಚಿಕ್ಟುನ್ ciktun could hypothetically have resulted from schwa deletion of चिकटून cikaṭūna or चिक्टून cikṭūna (ignoring additional hypotheses for word-internal *u*) . However, the cluster क्ट kṭ is a non-standard ligature and is highly unlikely to appear in Devanagari orthography. The acceptor would prune any path that would result in the cluster kṭ. Similarly, the word ವಸ್ತು vastu could have resulted from वसतू vasatū or वस्तू vastū. But स्त st being a standard consonantal cluster, both are equally plausible. In this case, a lexicon lookup or n-gram model is necessary to choose the most probable output.

Similarly, inverting $\mathcal{R}_{kn}$ also results in multiple alternate paths for *i* and *u* among others.

Since Devanagari only uses long ē and ō, let $\mathcal{R}_{eo}$ be the transducer that replaces the short vowels *e* and *o* with the corresponding long vowels.

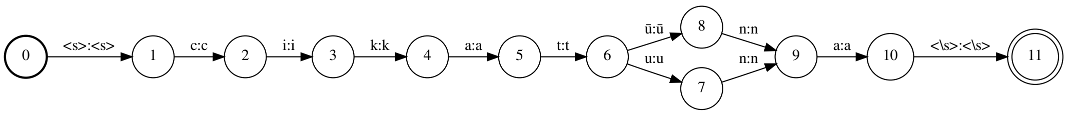

Figure 3: Transducer $\mathcal{R}_{dv}$ for the Kannada input ಚಿಕ್ಟುನ್ *ciktun*

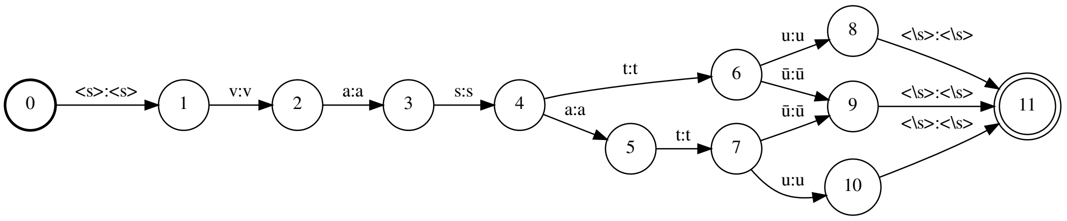

Figure 4: Transducer $\mathcal{R}_{dv}$ for the Kannada input ವಸ್ತು *vastu*

The final rule-based transducer $\mathcal{R}_{dv}$ is,

$$\mathcal{R}_{dv} = \mathcal{R}_{kn}^{-1} \circ (\mathcal{W}_d^{-1} \circ \mathcal{A}_{Cdv}) \circ \mathcal{R}_{eo}$$

In case of non-standard input such as words which already have a schwa in a position where none is possible, the composition fails. In this case we have a rule-based Schwa insertion transducer $\mathcal{W}_{ir}$ that inserts an additional arc that inserts schwa to non-standard consonantal clusters. In this case,

$$\mathcal{R}_{dv} = \mathcal{R}_{kn}^{-1} \circ \mathcal{W}_{ir} \circ \mathcal{R}_{eo}$$

Similar to $\mathcal{T}_{dv2kn}$, the Kannada to Devanagari transducer $\mathcal{T}_{kn2dv}$ is formed with a Devanagari lexical acceptor and if needed, a corresponding Devanagari n-gram word model.

$$\mathcal{T}_{kn2dv} = (\pi_2(\mathcal{R}_{dv} \circ \mathcal{I}) \circ \mathcal{A}_{Ldv}) \circ \mathcal{N}_{dv} \quad or$$
$$\mathcal{T}_{kn2dv} = \pi_2(\mathcal{R}_{dv} \circ \mathcal{I}) \circ \mathcal{N}_{dv}$$

4.3 Kannada to Romi

As in previous transliterations, we romanized the input. Since Romi and Kannada share different graphemic sets, we proceed with performing a character alignment with a Kannada-Romi parallel wordlist. Romanizing the input very marginally improves the character alignment process, since both input and output are then alphabetic scripts (as compared to syllabic to alphabetic alignment). We initially experimented with tools such as GIZA++, but found Phonetisaurus produced better alignments compared to other tools as it uses many-to-many alignments developed specifically for grapheme to phoneme systems (Jiampojamarn et al., 2007).

A sample alignment sequence from Phonetisaurus is given below:

meḷillem̩ | mellil'lem → m}m e}e ḷ}l|l i}i l}l|' l}l e}e m̩}m
gaḍyem̩tlyān | gaddientlean → g}g ā}a ḍ}d|d y}i e}e m̩}n t}t l}l y}e ā}a n}n
bhāratāsārkyā | bharotasarkea → bh}b|h ā}a r}r a}o t}t ā}a s}s ā}a r}r k}k y}e ā}a
bhiyeli | bhieli → bh}b|h i|y}i e}e l}l i}i

Where } denotes individual character alignment, | between characters indicates grapheme chunks, and ḍ}d|d implies that the source grapheme «ḍ» is mapped to the target graphemic chunk «dd».

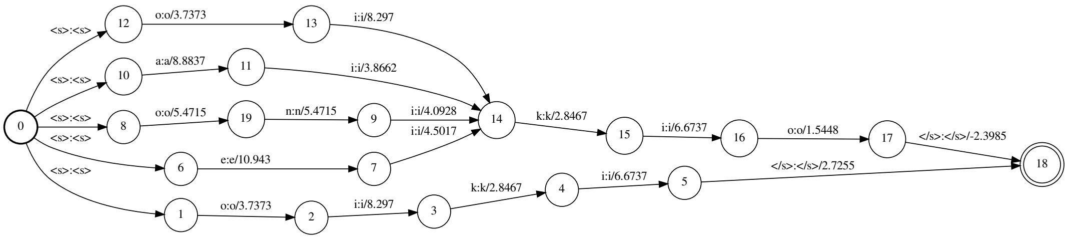

Figure 5: 5 best paths of $\mathcal{L}_p$ for Kannada input ಐಕ್ಯ *aiky*

This alignment lattice was then used to create a joint sequence n-gram model (Galescu and Allen, 2002) $\mathcal{N}_{knrm}$. This is then composed with the input word $\mathcal{I}$, whose output projection we use. We also modify the resulting transducer by removing edges with grapheme chunks and replacing it with succeeding edges with the individual graphemes of the chunk. This is necessary for later operations.

Some graphemic sequences such as geminate vowel graphemes *aa*, *ii* etc. do not occur in the orthography. We construct a transducer $\mathcal{A}_{rm}$, which accepts only paths with standard graphemic sequences. Thus effectively creating a pruned lattice $\mathcal{L}_p$.

$$\mathcal{L}_p = \pi_2(\mathcal{I} \circ N_{knrm}) \circ \mathcal{A}_{rm}$$

We created a Romi lexical acceptor $\mathcal{A}_{Lrm}$ which is composed with $\mathcal{L}_p$. If all the paths are non-lexical, we select the cheapest path from $\mathcal{L}_p$.

$$\mathcal{T}_{kn2rm} = \mathcal{L}_p \circ \mathcal{A}_{Lrm} \quad or \quad \mathcal{T}_{knr2rm} = shortest(\mathcal{L}_p)$$

4.4 Romi to Kannada

We produced a similar set of transducers as in Kannada → Romi. The lattice pruner here rejects non-standard Indic forms like digraphic vowel sequences. We generated a new joint sequence n-gram model by swapping the original training data and retraining it. We initially attempted to invert $\mathcal{N}_{knrm}$, to avoid re-training, but the accuracy was found to be 18% lower than a retrained model.

$$\mathcal{L}_p = \pi_2(\mathcal{I} \circ N_{rmkn}) \circ \mathcal{A}_{ind}$$
$$\mathcal{T}_{kn2rm} = \mathcal{L}_p \circ \mathcal{A}_{Lkn} \quad or \quad \mathcal{T}_{kn2rm} = shortest(\mathcal{L}_p)$$

4.5 Devanagari to Romi

We performed an initial schwa deletion on the input using $\mathcal{W}_d$. We found that schwa-deleted input improves the joint sequence n-gram model. Schwa deletion being a grammatical process, removing one of the underlying uncertainties effectively improves the performance. As a result of this, the joint sequence n-gram model performs better with preprocessed input.

Following Schwa deletion, a similar process to that described in section 4.3 is performed, to generate the transducer $\mathcal{T}_{dv2rm}$.

$$\mathcal{L}_p = \pi_2((\mathcal{W}_d \circ \mathcal{I}) \circ N_{dvrm}) \circ \mathcal{A}_{rm}$$
$$\mathcal{T}_{dv2rm} = \mathcal{L}_p \circ \mathcal{A}_{Lrm} \quad or \quad \mathcal{T}_{dv2rm} = shortest(\mathcal{L}_p)$$

4.6 Romi to Devanagari

We performed a rule-based schwa insertion on the input here. However, we did not see any substantial improvement in the performance in terms of accuracy, as compared to the raw input. However, we retained the preprocessing, to take advantage of even the marginal improvement. Also, the preprocessing model could be improved in the future. We performed a similar set of

Script Pair	*Rules-bases System*	*Statistical System*	*Cascading System*
Devanagari - Kannada	83.9%	84.59%	90.383%
Kannada - Devanagari	79.49%	90.16%	96.66%
Devanagari - Romi	74.88%	78.02%	95.39%
Romi - Devanagari	54.02%	74.04%	83.41%
Kannada - Romi	81.29%	87.63%	96.12%
Romi - Kannada	68.01%	82.21%	97.87%

Table 3: Accuracy of three different systems

transductions as in section 4.4.

$$\mathcal{L}_p = \pi_2((\mathcal{W}_{ir} \circ \mathcal{I}) \circ N_{rmdv}) \circ \mathcal{A}_{ind}$$
$$\mathcal{T}_{rm2dv} = \mathcal{L}_p \circ \mathcal{A}_{Ldv} \quad or \quad \mathcal{T}_{rm2dv} = shortest(\mathcal{L}_p)$$

5 Evaluation

For the procedures involving statistical methods, we split the corpus with 90% being used for training and the remaining 10% for testing. The accuracy results for the orthographies are reported in table 3. For the rule-based system, the initial system developed was used for the evaluation. For the pure statistical approach, we used Phonetisaurus's in-built g2p system. The cascading system was that discussed in this paper.

As expected, the initial rule-based system has the least accurate performance. Although it is theoretically possible to mine all rules that can apply to a system, in practice the rule-mining process is highly inefficient and user-dependent. The statistical system performs better than the rule-based system. The mediocre performance of the statistical system can be mainly attributed to the limited corpora that we possess. Konkani being an inflectional language, the effective number of unique words in the corpus is considerably lower than the absolute word count. The hybrid system performs best when compared to others.

Of all the transliteration pairs, Romi to Devanagari appears to have the lowest accuracy of all the three systems. Compared to other transliteration pairs, Romi to Devanagari is the most complex system as it involves both schwa insertion and disambiguation of confounded graphemic sequences. The poor performance of the hybrid system can be attributed to the fact that we had used a rule-based schwa insertion as a part of preprocessing. While this system worked well for an Indic system such as Kannada, it turned out to be not very efficient for Romi, where consonantal sequences are rendered as vowel digraphs or trigraphs.

6 Conclusion

We have developed a finite state transducer based transliteration engine called *Konkanverter* which performs statistical machine transliteration between three different scripts used to write the Konkani language. We have explained the detailed architecture of this statistical machine transliteration system, which consists of cascading finite state transducers combining both rule-based and statistical approaches. The transliteration engine was evaluated and its performance was reported. This cascading approach is found to perform significantly better than a pure rule-based approach or pure statistical approach.

Acknowledgements

This work was supported by the World Institute of Konkani Language (WIKL) part of the World Konkani Centre, Mangalore, India. The author would also personally like to thank Mr Gurudath Bantwalkar, Asst Director of WIKL for all his linguistic expertise regarding the Konkani language.

References

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer. `http://www.openfst.org`.

Mansur Arbabi, Scott M Fischthal, Vincent C Cheng, and Elizabeth Bart. 1994. Algorithms for Arabic name transliteration. *IBM Journal of research and Development*, 38(2):183–194.

Manoj Kumar Chinnakotla and Om P Damani. 2009. Experiences with English-Hindi, English-Tamil and English-Kannada transliteration tasks at news 2009. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, pages 44–47. Association for Computational Linguistics.

Lucian Galescu and James F Allen. 2002. Pronunciation of proper names with a joint n-gram model for bi-directional grapheme-to-phoneme conversion. In *INTERSPEECH*.

Ministry of Home affairs Gov. of India. 2001. Abstract of speakers' strength of languages and mother tongues – 2001. `http://www.censusindia.gov.in/Census_Data_2001/Census_Data_Online/Language/Statement1.aspx`. Accessed: 2014-05-30.

Yuxiang Jia, Danqing Zhu, and Shiwen Yu. 2009. A noisy channel model for grapheme-based machine transliteration. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, pages 88–91. Association for Computational Linguistics.

Sittichai Jiampojamarn, Grzegorz Kondrak, and Tarek Sherif. 2007. Applying many-to-many alignments and hidden markov models to letter-to-phoneme conversion. In *HLT-NAACL*, volume 7, pages 372–379.

N Kishorjit. 2011. Manipuri transliteration from Bengali script to Meitei Mayek: A rule based approach, c. singh et al.(eds.): Icisil 2011, ccis vol. 139, part 2.

Kevin Knight and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4):599–612.

M. G. Abbas Malik, Christian Boitet, and Pushpak Bhattacharyya. 2008. Hindi Urdu machine transliteration using finite-state transducers. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 537–544. Association for Computational Linguistics.

M. G. Abbas Malik, Laurent Besacier, Christian Boitet, and Pushpak Bhattacharyya. 2009. A hybrid model for Urdu Hindi transliteration. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, pages 177–185. Association for Computational Linguistics.

M. G. Abbas Malik, Christian Boitet, Laurent Besacier, and Pushpak Bhattcharyya. 2013. Urdu Hindi machine transliteration using SMT. In *the Proceedings of the 4th Workshop on South and Southeast Asian Natural Language Processing, International Joint Conference on Natural Language Processing*, pages 43—57.

Bhuvana Narasimhan, Richard Sproat, and George Kiraz. 2004. Schwa-deletion in Hindi text-to-speech synthesis. *International Journal of Speech Technology*, 7(4):319–333.

Josef Novak, Dong Yang, Nobuaki Minematsu, and Keikichi Hirose. 2011. Initial evaluations of an open source WFST-based phoneticizer. *The University of Tokyo, Tokyo Institute of Technology.*

Brian Roark, Richard Sproat, Cyril Allauzen, Michael Riley, Jeffrey Sorensen, and Terry Tai. 2012. The OpenGrm open-source finite-state grammar software libraries. In *Proceedings of the ACL 2012 System Demonstrations*, pages 61–66, Jeju Island, Korea, July. Association for Computational Linguistics.

Thoudam Doren Singh. 2012. Bidirectional bengali script and meetei mayek transliteration of web based manipuri news corpus. In *the Proceedings of the 3rd Workshop on South and Southeast Asian Natural Language Processing (SANLP) of COLING*, pages 181–189.

Terry Tai, Wojciech Skut, and Richard Sproat. 2011. Thrax: An open source grammar compiler built on openfst. ASRU.