The Festvox Indic Frontend for Grapheme-to-Phoneme Conversion

Alok Parlikar, Sunayana Sitaram, Andrew Wilkinson and Alan W Black
Carnegie Mellon University
Pittsburgh, USA
aup, ssitaram, aewilkin, awb@cs.cmu.edu

Abstract

Text-to-Speech (TTS) systems convert text into phonetic pronunciations which are then processed by Acoustic Models. TTS frontends typically include text processing, lexical lookup and Grapheme-to-Phoneme (g2p) conversion stages. This paper describes the design and implementation of the Indic frontend, which provides explicit support for many major Indian languages, along with a unified framework with easy extensibility for other Indian languages. The Indic frontend handles many phenomena common to Indian languages such as schwa deletion, contextual nasalization, and voicing. It also handles multi-script synthesis between various Indian-language scripts and English. We describe experiments comparing the quality of TTS systems built using the Indic frontend to grapheme-based systems. While this frontend was designed keeping TTS in mind, it can also be used as a general g2p system for Automatic Speech Recognition.

Keywords: speech synthesis, Indian language resources, pronunciation

1. Introduction

Intelligible and natural-sounding Text-to-Speech (TTS) systems exist for a number of languages of the world today. However, for low-resource, high-population languages, such as languages of the Indian subcontinent, there are very few high-quality TTS systems available. One of the bottlenecks in creating TTS systems in new languages is the development of a frontend that can take sentences or words in the language and assign a pronunciation in terms of phonemes from a phone set defined for the language.

In some languages, such a frontend may make use of a lexicon, which is a list of words in the language with their pronunciations, and of a Letter-to-Sound (LTS) model that predicts the pronunciation of Out-of-Vocabulary (OOV) words. Other frontends may not have a lexicon and may only use LTS or Grapheme-to-Phoneme (g2p) rules to predict the pronunciations of all words. Languages that have fairly close relationships between the orthography and pronunciation typically fall in the latter category.

In this work, we improve upon previous grapheme-based approaches to create a unified frontend that implements various g2p rules for Indian languages. The Indic frontend can be used for g2p conversion for building TTS systems in various Indian languages for use with the Festival Speech Synthesis engine (Taylor et al., 1998). While this frontend was designed keeping TTS in mind, it can also be used as a general g2p system for speech recognition.

2. Relation to Prior Work

The lack of lexical resources and clearly defined phone sets can be an impediment to building TTS and other speech-processing systems in new languages. Previously, two techniques have been proposed to build voices in new low-resource languages (Sitaram et al., 2015b). The first technique assumes no knowledge of the language and simply treats each grapheme or letter as a phoneme. This allows us to build a voice without having to define a phone set for the language. The disadvantage of this technique is that we lose out on phonetic features that typically give gains in models of the spectrum and the prosody. Another problem with this approach is that since each grapheme maps to a single "phoneme" in all contexts, this technique does not work well in the case of languages that have pronunciation ambiguities. We refer to this technique as "Raw Graphemes."

Another technique exploits a universal transliteration resource (UniTran) (Qian et al., 2010) that provides a mapping from graphemes to phonemes in the X-SAMPA phone set. The UniTran implementation in Festvox (Sitaram et al., 2015b) provides a single mapping from each grapheme in the Unicode specification to a phoneme, with the exception of a few scripts such as Chinese and Japanese. While this technique allows us to use phonetic features in models downstream, it has the same limitation as the previous technique because there is a single mapping between a grapheme and a phoneme.

(Choudhury, 2003) describes a rule-based g2p mapping scheme for Hindi, which can be used to build a TTS system for Hindi. Schwa deletion, syllabification, and contextual nasalization are handled by the rules stated in this work, with exceptions to these rules being handled by listing in a lexicon.

(Bali et al., 2004) also describes a rule-based Hindi frontend to be used with the Festival Speech Synthesis system. Schwa deletion is handled with rules. However, these rules fail when dealing with compound words. In light of this, an algorithm to detect compound words is described which results in an improvement in the g2p conversion when schwa deletion is applied to the individual constituents.

As per our knowledge, there has been no prior work in creating a freely available, common frontend for all the Indian languages that can be used for g2p conversion. Our motivation for creating such a frontend is to be able to address many common pronunciation issues in Indian languages using a single frontend to facilitate rapid development of TTS systems in Indian languages.

To build the Indic frontend, we build upon the Festvox UniTran implementation. The UniTran mappings

have some limitations when it comes to handling the g2p rules in Indian languages. From a pronunciation point of view, most Indian languages have a fairly consistent mapping from graphemes to phonemes. However, some contextual rules need to be applied in some languages. For example, the UniTran mapping assigns an inherent schwa to all consonants by default. However, in some languages such as Hindi and Bengali, schwas are deleted at the ends of words and sometimes in the middle of words. This is something we need to handle explicitly. Still, the UniTran mappings provide a good starting point for building a frontend for Indian languages.

3. Indic Frontend Description

The Indic frontend has been developed on top of the Festvox voice-building tools (Black and Lenzo, 2002), to be used in the Festival Speech Synthesis engine (Taylor et al., 1998). We will also describe implementations of the Indic frontend for Flite (Black and Lenzo, 2001), a low-footprint speech synthesizer; and Flite for Android (Parlikar, 2014), an Android application that can be used on a smartphone. For the next few sections, we focus on the design and implementation of the Indic frontend for Festvox.

All Unicode characters defined for Indian languages are first mapped to a phoneme from the X-SAMPA set, similar to the mappings provided by UniTran. In addition, each Unicode character is mapped to its corresponding ordinal for ease of processing. Lastly, a set of rules are defined which are associated with language lists. If a language is in the list for a particular rule, the rule is fired for that language.

Next, we describe the rules implemented in the frontend to handle various phenomena.

3.1. Schwa Deletion

Indo-Aryan languages such as Hindi, Bengali, Gujarati, et cetera, exhibit a phenomenon known as *schwa deletion*, in which a final or medial schwa is deleted from a word in certain cases. For example, in Hindi, the final schwa (realized as the sound [ə]) in the word कमल (pronounced 'kamal') is deleted. None of the consonants क, म, or ल have an attached vowel; hence, they have inherent schwas, and the inherent schwa on the last consonant ल gets deleted. The word लगभग (pronounced 'lagbhag') has consonants ल ग भ ग, from which both the medial schwa on the first consonant ग and the final schwa on the second consonant ग get deleted. If schwa deletion did not take place, these words would erroneously be pronounced as 'kamala' and 'lagabhaga' respectively. In both these cases, the orthography does not indicate which inherent schwas should be deleted.

Schwa deletion has been well studied in the context of TTS systems. There are well defined linguistic rules to predict when a schwa gets deleted and when it does not. However, there are exceptions to these rules that reportedly affect around 11% of the vocabulary (Narasimhan et al., 2004), including cases such as consonant clusters.

Previous work on schwa deletion includes approaches that take into account morphology to preserve schwas that may otherwise be deleted (Narasimhan et al., 2004). Other approaches have used syllable structure and stress assignment to assign schwa deletion rules (Naim R and Nagar, 2009). (Choudhury et al., 2004) uses ease of articulation, acoustic distinctiveness and ease of learning to build a constrained optimization framework for schwa deletion.

Recently, (Sitaram et al., 2015a) proposed a technique to automatically discover LTS rules such as schwa deletion using acoustics for low-resource languages. They use the UniTran mappings and a cross-lingual technique to automatically discover schwa deletion rules for Hindi, which is considered to be a low-resource language in this case, by using acoustic models trained on other Indian languages and schwa deletion rules for Assamese.

Schwa deletion for Hindi occurs in both word-final and word-medial positions, while for languages like Bengali it occurs only in word-final positions. The schwa deletion rules we implemented in the Indic frontend are based on a simpler version of (Narasimhan et al., 2004) and are as follows:

- Process input from right to left
- If a schwa is found in a VC_CV context, delete it

Taking the examples of कमल 'kamal' and लगभग 'lagbhag' mentioned earlier, we now see how these rules apply. In the case of कमल, the consonants क, म, and ल all have inherent schwas, and the last schwa is deleted according to the first rule. Since none of the other schwas are in a VC_CV context, they remain. In the case of लगभग, the consonants ल ग भ ग also have inherent schwas, and once again the final schwa gets deleted. The schwa attached to the second ग is in a VC_CV context, and hence also gets deleted. This rule gives us the correct pronunciation in both these cases.

3.2. Contextual Nasalization

The *anuswaar* character is used to indicate nasalization, which can be realized as different phonemes depending on the context. For example, in the words एवं, पंप, तंतु, and अंक ('evam,' 'pump,' 'tantu,' 'ank') the dot above the consonants indicates nasalization, which is realized as different nasal phonemes in each of these words depending on the consonant. We implemented rules for contextual nasalization as follows:

- If it's a schwa, it's not nasalized. nX becomes m
- nX followed by velar becomes nG
- nX followed by palatal becomes n
- nX followed by alveolar becomes nr
- nX followed by dental becomes nB
- nX followed by labial becomes m
- all other nX become nB

3.3. Tamil Voicing

Most Indian languages have distinct representations in their orthography for voiced and unvoiced sounds. However, this is not the case with Tamil, which does not have distinct letters for voiced and unvoiced stops. There are well defined rules for predicting voicing in Tamil. For example, the voiceless stop [p] occurs at the beginning of words, while the voiced stop [b] does not.

(Ramakrishnan and Laxmi Narayana, 2007) describes a frontend for Tamil with rules for predicting voicing, similar to those described below. They also use a lexicon for foreign words of Sanskrit and Urdu origin, which do not follow these rules.

The rules that we implemented for Tamil voicing are taken from (Albert and others, 1985) and are as follows:

- Initial and geminated stops are voiceless
- Intervocalic and postnasal stops are voiced
- Stops after voiced stops are voiced

3.4. Lexical Stress

Prosody and lexical stress have not been well studied in Indian languages. A technique for automatically identifying stress based on power, energy, and duration by clustering units is described in (Laxmi Narayana and Ramakrishnan, 2007). Experiments were carried out on Tamil for syllable-level lexical stress, based on which a rule was created for assigning stress in Tamil as follows: The first syllable is stressed if it does not contain a short vowel; otherwise, the second syllable is stressed.

Our implementation of stress rules for Indian languages is based on (Hussain, 1997). This is based on the concept of syllable weights, which are decided by vowel context. A light syllable ends in a short vowel, while a heavy syllable ends in either a long vowel, or a short vowel and a consonant. An extra-heavy syllable ends in either a long vowel and a consonant, or a short vowel and two consonants. This is similar to the ideas presented in (Pandey, 2014), who also describe pitch and amplitude based cues for schwa deletion, which we did not implement.

Stress is based on the syllable with the highest weight. In the case of a tie, the last syllable with the highest weight is stressed. The last syllable of the word does not participate in tie-breaking; it is stressed only when there are no ties. For example, in the word कारीगरी ('kaarigarii'), the syllables with highest weights are 'kaa,' 'rii' and 'rii.' Since the last syllable is not considered, the first 'rii' is stressed in this case.

3.5. Other Post-Lexical Rules

The *halant* character under a consonant indicates that a schwa is deleted, so we remove schwas after consonants that have this character under them.

We handle consonants with nukta characters under them by mapping them to the consonant without the nukta, as these characters are usually very rare in our training corpora.

We also added rules for schwa realization, in which the schwa is replaced with another vowel (*rahna* vs *rehna*), for which we replace the schwa with the phoneme /e/ in the post-lexical rules.

For Malayalam, we added rules to process Chillu letters. Consonants represented by Chillu letters are never followed by an inherent vowel, and we added appropriate mappings in the frontend.

3.6. English Language Support

We extended the Indic frontend to be capable of synthesizing not just words written in Indian languages in Unicode, but also English words. It is often seen on Indian-language websites such as newspapers and Wikipedia that a few English words are written in the Latin script.

The task was to synthesize test sentences containing mixed-script sentences, but the training synthesis databases contained no English in the recordings and their corresponding prompts. However, there may have been some words written in the native script that were not truly native, such as proper names, technical terms, et cetera.

Since we only had data in the Indian languages to train from, we employ a straightforward approach to handle English words in Indian-language sentences. When we detect a word in the Latin script, we use the US English text-processing frontend to process it. This means that all Non-Standard Words that are covered by the (much higher-resource) US English frontend are also available to us, including special symbols except numbers, which we describe next.

Then, we use a mapping between the US English phone set and the Indic phone set (which is common for all the lower-resource Indian languages) to convert the US English phonemes to Indic phonemes. This is a simple one-to-one mapping, which has its limitations, since some phonemes in English do not exist in the Indian languages and vice-versa. Mapping phonemes between English and Hindi is a one-time cost, but ideally such mappings should be done automatically. We are exploring techniques to automatically map phonemes cross-lingually using knowledge about phonetic features, context, and acoustics.

3.7. Numbers

There has been very little work in creating text-processing frontends for Indian languages that can handle numbers, abbreviations, and other non-standard words. (Ramakrishnan and Laxmi Narayana, 2007) describes a text-processing frontend for Tamil that categorizes and expands numbers into ordinary numbers, phone numbers, dates, times, and currency, based on delimiters and length.

We provide support for synthesizing numbers written as numerals in different scripts. Indian language texts may employ the numerals native to the script of the language, or may employ the standard numerals common to most of the world today (known as "Arabic" or "Indo-Arabic" numerals; we refer to them as "English" numerals for simplicity). In modern Indian-language texts, English numerals are more commonly used than native numerals—sometimes much more commonly, depending on the language. In most cases, there is a one-to-one correspondence between native and English numerals, so it is simple to map from one numeric representation to the other. One

exception is the Tamil system, which has distinct numerals for 10, 100, and 1000, and hence is not a true base-10 system. Another is certain traditional systems of writing fractions, such as those of Telugu and Bengali. Writing rules to handle these exceptions is future work.

In the case of integers, we currently synthesize numbers written with English numerals in English, and numbers written in a native script in the corresponding language. This reflects a compromise between respecting the desires both of authors who use English numerals and wish the text to be accessible to a wide audience (including people who may not have full familiarity with the native number system), and of authors who use native numerals and wish to continue the traditions of the language. We plan to make these representation options (English, native, or mixed numbers) a choice in the future for the user.

Speaking numbers in Indian languages requires use of a pronunciation lookup table for all numbers between zero and one hundred, because these numbers take idiosyncratic forms that cannot be deterministically generated. For large numbers, one issue is when to speak numbers in multiples of "lakh" and "crore" (corresponding to one hundred thousand and ten million, respectively), and when to use "thousand," "million," et cetera. When a number in English holds to the lakh/crore pattern in comma-separated groupings of digits, it is spoken according to that paradigm. Thus, "12,34,56,789" is "twelve crore, thirty-four lakh, fifty-six thousand, seven hundred eighty-nine"; whereas "123,456,789" is "one hundred twenty-three million, four hundred fifty-six thousand, seven hundred eighty-nine." Numbers over twelve digits are spoken one digit at a time. For native-script numbers, numbers up to nine digits are mapped to lakh and crore, and for longer strings are spoken one digit at a time.

3.8. Creating Support for New Languages

The Indic frontend has explicit support for a number of Indian languages and has been designed to make it simple to add support for new languages. To add support, all the Unicode characters in the language need to mapped to an ordinal as described earlier, and each ordinal needs to be mapped to a phoneme from X-SAMPA. The UniTran mappings can be used as a starting point for doing this. The rules described above that have been implemented for other languages can be toggled for any new language, and any language-specific rules can be created in a similar manner. For example, schwa deletion does not occur in all Indian languages, and can be toggled for Hindi, with both word-final and medial schwa deletion, and Bengali, with only word-final schwa deletion.

3.9. Creating an Offset-Based Frontend

So far, we described the design of the frontend available in Festival. Our Flite implementation of the frontend follows an offset-based approach instead of having to create explicit support for each Indian language. Chapter 9 of the Unicode specification (Unicode Staff, 1991) has offset-based character tables for each script, with each script containing up to 128 characters. Within each script, there is a fixed sequence of characters which makes it easy to build general rules for the phenomena described above. This makes it possible to have a single mapping with offsets for all scripts for Indian languages.

4. Data and Experiments

The Blizzard Challenge (Black and Tokuda, 2005) is an annual community-wide evaluation of TTS systems. Participants are provided with common databases to build synthetic voices, and a common test set to synthesize. Systems are evaluated on a wide variety of subjective metrics by volunteers and paid listeners. In the last two years, the Blizzard Challenge has included an Indian-language synthesis task, which drove our work on the Indic frontend.

The metrics in the Blizzard Challenge did not test the quality of frontends explicitly. The closest metric that tested pronunciation quality was Word Error Rate, in which our systems did well for Tamil, Telugu, Malayalam, and Marathi.

The data for the Blizzard 2015 tasks consisted of six Indian languages, with four hours of data each in Hindi, Tamil, and Telugu, and two hours of data each in Marathi, Bengali, and Malayalam. The databases were recorded by professional speakers in recording studios. Each database had corresponding transcripts in UTF-8. We used the data from Blizzard 2015, as well as Assamese and Rajasthani data from Blizzard 2014 (Prahallad et al., 2014), which also consisted of two hours of data in each language.

In order to compare the knowledge-based Indic frontend to previous grapheme-based approaches, we used an objective metric of speech synthesis quality. We varied only the frontends of the systems and kept everything else the same. We compared the synthetic speech with held-out reference recorded speech by computing the Mean Mel-Cepstral Distortion (Mashimo et al., 2001) (MCD) of the predicted cepstra. Since this is a distance measure, a lower value suggests better synthesis. Kominek (Kominek, 2009) has suggested that MCD is linked to perceptual improvement in the intelligibility of synthesis, and that an improvement of about 0.08 is perceptually significant and an improvement of 0.12 is equivalent to doubling the data. The MCD is a database-specific metric which cannot be compared across databases.

Table 1 shows the MCD for Hindi, Tamil and Telugu built with the two grapheme-based frontends described earlier and the Indic frontend. We performed this comparison only on these languages, although we built systems using the Indic frontend for all the languages mentioned above.

Table 1: *MCD for languages built with Raw Graphemes, UniTran, and the Indic frontend*

Language	Raw	UniTran	Indic Frontend
$Hindi$	5.10	5.05	**4.94**
$Tamil$	5.10	5.04	**4.90**
$Telugu$	5.54	5.85	**5.12**

From the results above, we can see that the MCD of the voices built with the Indic frontend are significantly better than the voices built with UniTran, which are in turn better than voices built with the raw graphemes frontend

except in the case of Telugu, where UniTran is significantly worse.

5. Availability

The Indic frontend has been released as part of the standard Festvox distribution. Documentation is provided in the Festvox manual (Black and Lenzo, 2003) for building voices using the Indic frontend and for adding support for new voices. Our current version of Flite also has support for Indic voices created using Festvox.

The Flite TTS for Android application is built with support for Indian languages. Voices in Hindi, Gujarati, Marathi, Tamil, and Telugu are available for download.

6. Conclusion

In this paper, we described the design and development of the Indic frontend, a common frontend for Grapheme-to-Phoneme conversion in Indian languages. The Indic frontend has been designed to provide a common framework to implement various Letter-to-Sound rules for Indian languages, allowing easy extensibility to new Indian languages.

We used an objective metric of speech synthesis quality to compare the Indic frontend with previous grapheme-based frontends used for low-resource languages, and found that voices built with the Indic frontend were significantly better. We also described preliminary work on synthesizing numbers in Indian languages and handling English words written in the Latin script.

We have recently begun work on synthesizing Code-Mixed text using the Indic frontend, in which Indian languages may be mixed with languages such as English, and in which the entire sentence may be written in the Latin script. Synthesizing words that are written in the "wrong" script can also apply to foreign words and Named Entities. An additional challenge that such text poses is that spellings are not standardized, particularly if the text is from Social Media. Our approach to solving this problem is to identify the language the word is in, normalize spellings and then transliterate Indian language words into their native scripts so that the Indic frontend can be used to synthesize them.

Text processing of non-standard words for Indian languages is an area where very little work has been done. Text-processing modules are usually implemented on a language-by-language basis, so creating a common text-processing frontend for Indian languages would be an interesting future direction. Likewise, there has been very little work done on prosody in Indian languages for TTS systems.

Finally, although the Indic frontend is a general g2p converter, we have only performed experiments on Speech Synthesis. Using the Indic frontend to generate or bootstrap lexicons for Automatic Speech Recognition (ASR) in Indian languages would be an interesting future direction.

7. Bibliographical References

Albert, D. et al. (1985). *Tolkāppiyam Phonology and Morphology: An English Translation*, volume 115. International Institute of Tamil Studies.

Bali, K., Talukdar, P. P., Krishna, N. S., and Ramakrishnan, A. (2004). Tools for the development of a Hindi speech synthesis system. In *Fifth ISCA Workshop on Speech Synthesis*.

Black, A. W. and Lenzo, K. A. (2001). Flite: a small fast run-time synthesis engine. In *4th ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis*.

Black, A. W. and Lenzo, K. (2002). Building Voices in the Festival Speech Synthesis System, http://festvox.org/bsv.

Black, A. W. and Lenzo, K. A. (2003). Building synthetic voices. *Language Technologies Institute, Carnegie Mellon University and Cepstral LLC*.

Black, A. W. and Tokuda, K. (2005). The Blizzard Challenge 2005: Evaluating corpus-based speech synthesis on common datasets. In *in Proceedings of Interspeech 2005*.

Choudhury, M., Basu, A., and Sarkar, S. (2004). A diachronic approach for schwa deletion in Indo Aryan languages. In *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology*.

Choudhury, M. (2003). Rule-based grapheme to phoneme mapping for Hindi speech synthesis. In *90th Indian Science Congress of the International Science Congress Association (ISCA), Bangalore, India*.

Hussain, S. (1997). Phonetic correlates of lexical stress in Urdu. *Unpublished dissertation*.

Kominek, J. (2009). *TTS From Zero: Building Synthetic Voices for New Languages*. Ph.D. thesis, Carnegie Mellon University.

Laxmi Narayana, M. and Ramakrishnan, A. (2007). Defining syllables and their stress labels in MILE Tamil TTS corpus. In *Proc. Workshop in Image and Signal Processing (WISP-2007), IIT Guwahati*.

Mashimo, M., Toda, T., Shikano, K., and Campbell, N. (2001). Evaluation of cross-language voice conversion based on GMM and STRAIGHT.

Naim R, T. and Nagar, I. (2009). Prosodic rules for schwa-deletion in Hindi text-to-speech synthesis. *International Journal of Speech Technology*, 12(1):15–25.

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

Pandey, P. (2014). Akshara-to-sound rules for hindi. *Writing Systems Research*, 6(1):54–72.

Parlikar, A. (2014). Flite TTS engine for Android. *Open-source Software*.

Prahallad, K., Vadapalli, A., Kesiraju, S., Murthy, H. A., Lata, S., Nagarajan, T., Prasanna, M., Patil, H., Sao, A. K., King, S., Black, A. W., and Tokuda, K. (2014). The Blizzard Challenge 2014.

Qian, T., Hollingshead, K., Yoon, S. y., Kim, K. y., and Sproat, R. (2010). A python toolkit for universal transliteration. In *LREC 2010*.

Ramakrishnan, A. and Laxmi Narayana, M. (2007). Grapheme to phoneme conversion for Tamil speech synthesis. In *Proc. Workshop in Image and Signal Processing (WISP-2007), IIT Guwahati*.

Sitaram, S., Jeblee, S., and Black, A. W. (2015a). Using acoustics to improve pronunciation for synthesis of low

resource languages. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Sitaram, S., Parlikar, A., Anumanchipalli, G. K., and Black, A. W. (2015b). Universal grapheme-based speech synthesis. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Taylor, P., Black, A. W., and Caley, R. (1998). The architecture of the Festival speech synthesis system. In *The Third ESCA/COCOSDA Workshop (ETRW) on Speech Synthesis*.

Unicode Staff, C. (1991). *The Unicode standard: worldwide character encoding*. Addison-Wesley Longman Publishing Co., Inc.