A rule-based transcription of Sanskrit to IPA

Presented by the Pandas in Pandemic

- Khushee A Namdeo
- Vamshi Krishna Bonagiri
- Vanshpreet Singh Kohli

Overview


Why Sanskrit?
Why new transcription method?
Sanskrit Phonology
The Algorithm
The software
Conclusion

Why Sanskrit?

- One of the oldest classical languages, shares close common ancestry with most modern Indo-Aryan languages
- Has vast amounts of significant literature, which makes it a matter of linguistics' interest.
- Panini, the creator of Sanskrit formulated 3,949 rules.
- Recognized by inclusion into Constitution's Eighth Schedule lanuages, one of the official languages of India
- Around 14k Indians report Sanskrit as mother tongue

Why do we need a new rule-based transcription?

- Sanskrit an important topic of study, but suffers from lack of an all-encompassing system of phonetic transcription
- IAST* and ITRANS** used widely, but considered inadequate in capturing the phonological and prosodic features
- Students of Sanskrit worldwide have varying pronunciations of the same few sounds, especially new learners
- Some recent tools try to improve upon existing solutions, but are often inadequate
- Hence the authors of the software in question proposed a new algorithmic approach to rule-based transcription

*International Alphabet of Sanskrit Transliteration
**Indian languages TRANSliteration

An overview of Sanskrit phonology

Simple vowels

	Front	Central	Back
High	इ ([i]), ई ([i:])		उ ([u]), ऊ ([u:])
Mid	ϕ, ए ([e:])	अ ([ə]), ϕ	ϕ, ओ ([o:])
Low		ϕ, आ ([a:])	

Compound vowels (diphthongs)

X	X+इ,ए	X+उ,ओ
अ,आ	ऐ ([a:i])	औ ([a:u])

Special vowels (sonorants)

ऋ ([r̩]),	ॠ ([r̩:])
ऌ ([l̩]),	ॡ ([l̩:])

	Vl. plosive	Vl. aspirated plosive	Vd. plosive	Vd. aspirated plosive	Nasal	Approximant	Fricative
Glottal							ह [ɦə]**
Velar	क [kə]	ख [kʰə]	ग [gə]	घ [gʰə]	ङ [ŋə]		
Palatal	च [t͡ɕə]	छ [t͡ɕʰə]	ज [d͡ʑə]	झ [d͡ʑʰə]	ञ [ɲə]	य [jə]	श [ɕə]
Alveolar						र [ɹə]	
						ल [lə]*	स [sə]
Retroflex	ट [ʈə]	ठ [ʈʰə]	ड [ɖə]	ढ [ɖʰə]	ण [ɳə]	ळ [ɭə]*	ष [ʂə]
Dental	त [t̪ə]	थ [t̪ʰə]	द [d̪ə]	ध [d̪ʰə]	न [nə]	व [ʋə]	
Labial	प [pə]	फ [pʰə]	ब [bə]	भ [bʰə]	म [mə]		

Consonants

Merged cells represent shared place of articulation

*Lateral approximants

**Voiced fricatives

Base	Diacritic	IPA	Base	Diacritic	IPA
अ		ə	आ	◌ा	aː
इ	◌ि	i	ई	◌ी	iː
उ	◌ु	u	ऊ	◌ू	uː
ऋ	◌ृ	ɻ	ॠ	◌ॄ	ɻː
ऌ	◌ॢ	l̩	ॡ	◌ॣ	l̩ː
ए	◌े	eː	ऐ	◌ै	aːi
ओ	◌ो	oː	औ	◌ौ	aːu
अं	◌ं	əm	अः	◌ः	əh
ॐ		oːm			

Sanskrit speech sounds

- Sanskrit has some complex phonological processes (such as sandhi) whose rules we need not encode into the algorithm, as their phonetic interaction is implied by the orthography itself
- We have taken the interpretation of sounds to be as close as possible to what is believed to have been the pronunciation in the classical Sanskrit era
- One such consideration is the pronunciations of a visarga, or the ‘◌◌:’-terminal sound. Present interpretation of the pronunciation of this sound is shifting towards a new trend: duplicating the vowel sound of the previous syllable after ([h]). For instance, रिवः would end as [-ihi] according this rule as opposed to [-h]. While this seems to be a rising trend, it did not always use to be so, and the sound was supposed to be simply a [h]-terminating one, without vowel duplication

The Algorithm

Tackling nasalization and schwa

- In case the nasal sound is not explicitly shown, an anusvara is shown on the character preceding it, and the actual sound corresponding to it is inferred from the forthcoming sound at the time of reading The specific nasal sound to be used is inferred based on the next sound, if one exists, or is taken to be [m], the bilabial nasal sound, by default. Eg. संबित = [səm.bi̪t]
- A consonant character in Devanagari Sanskrit, unless explicitly marked halant, has an implied schwa. Removal of schwa is required when either explicitly marking a character halant, or when combining it with another vowel, in which case, the vowel combination overrides the schwa eg. गो = ग + ◌ो + ओ ([goː])

Syllabification

- The software we are covering adapts the WWG* algorithm (for Sinhalese) to Sanskrit and for the syllabification of the Devanagri text using vowel-consonant-vowel clusters. *Weerasinghe-Wasala-Gamage
- We apply rules based on the number of consonants in the middle consonant cluster, i.e., n. Based on this length and prosodic syllabification conventions, we mark the boundaries of the syllables
- For instance, संस्कृत ([sə̃s.kɻ.t̪ə]) is broken down into the syllables sə̃s , kɻ and t̪ə .
- We reuse boundary vowels, so a vowel that was processed while considering the current cluster will be included again to spot the next cluster

The WWG Algorithm

The algorithm works with a consonant cluster V_BCV_A, where C is a consonant cluster of length (say) n. Here are the several conditions under which syllabification is carried out:

- If n = 1, mark syllable break after V_B eg. कृतम् ([kɾ̩.t̪əm]) is split just before [t̪].
- If n = 2, mark syllable break after the first consonant from the left, for example वल्कलानि (['ʋəl.kə.lɑː.ni]) is split just before [k].
- If n = 3:
  - if third consonant from left = र् or य् then mark syllable break after first consonant from the left, eg. मत्स्यः (['mət̪.sjəh]) is split just before [s].

- If n = 3:
  - If the first two consonants are stops, then mark syllable break after first consonant from left, for instance उक्त्वा (['uk.ʈ̪ʋɑː]) is split just after [k].
  - If neither of the above two situations is satisfied, mark syllable break before first consonant from right, eg. कृत्स्नम् (['kɾɪt̪s.nəm]) Is split just before [n].
- If n > 3:
  - If first consonant from right = र् or य, then mark syllable break just before the consonant preceding the र् or य. For example, कार्त्स्न्यम् ([kɑːɾʈ̪s.njəm]) is split right before [n].
  - Else, mark syllable break after least sonorous C. Sonority is loosely defined as the loudness of speech sounds. Voiced sounds are more sonorous than unvoiced sounds.

[a] > [e o] > [i u j w] > [r] > [l] > [m n ŋ] > [z v ð] > [f θ s] > [b d g] > [p t k]

A *pseudo code* of the WWG Algorithm

Input: Sanskrit text to be syllabified
`initialize` scope at the beginning of text;
while *end of text not reached* **do**
    `move` to next $V_B\mathbb{C}V_A$, where $\mathbb{C}$ is a consonant cluster;
    if *length of cluster* $\mathbb{C} = 1$ **then**
        `mark` syllable break after V_B;
    else if *length of cluster* $\mathbb{C} = 2$ **then**
        `mark` syllable break after first C from left;
    else if *length of cluster* $\mathbb{C} = 3$ **then**
        if *third consonant from left* = र् or य् **or** *first and second consonants are stops* **then**
            `mark` syllable break after first C from left;
        else
            `mark` syllable break before first C from right;
    else
        if *first consonant from right* = र् or य् **then**
            `mark` syllable break before second C from right;
        else
            `mark` syllable break after least sonorous C;
end
Result: Syllabified Sanskrit text

Assigning stress

In Sanskrit, a syllable is either heavy (H) or light (L). The “base case” for syllables is being light, but they become heavy subject to meeting one or more of these:

1. Syllable contains long vowel or diphthong
2. Syllable is nasal-terminated or has a nasalized vowel
3. Syllable is stressed

If a syllable already satisfies at least one of the first two conditions above, it is already heavy. However, if not, we must use condition 3 and add stress to make it into a heavy one.

Why bother assigning stress?

The goal here is to ensure that any syllable that results in a cluster of consonants because of the adjoining consonants in the next syllable, is heavy.

For example, let us consider the syllables of the word

([ku.ˈɹuk.ʂeː.ʈɹə]) कुरुक्षेत्र

When taken independently, the four syllables are L,L,H,L respectively but when we consider the word, the four syllables are L,H,H,L. It thus becomes heavy as a result of receiving stress. The third result, heavy already, does not receive any extra stress.

The software itself

- The prototype software built as a part of the work we are discussing has been developed, and is free and open source as well. The code is available in a public repository at GitHub.
- It has been written using Python3, and is quite simple for anyone to try out. Text can be transcribed via the command line interface, and a large text file can be transcribed with a single command as well.
- The authors were kind enough to observe a lack of permissively licensed software to encourage scrutiny, improvement and further development.
- In fact, since Sanskrit is written in Devanagari, the tool can even be used to transcribe Hindi to IPA with decent accuracy.

Testing the software out!

```
albus@Zenbook:~$ cd sanskrit_IPA/
albus@Zenbook:~/sanskrit_IPA$ python3 init.py


> transcribe थैंक् यू
tʰɑːiŋk juː
> transcribe दैट्स् इट् फ्रम् अस्
ɖɑːiʈs iʈ pʰɹəm. əs
> The End :)
```