COMPUTATIONAL SANSKRIT & DIGITAL HUMANITIES

Selected Papers Presented at the 17th World Sanskrit Conference, July 9-13, 2018

University of British Columbia

Vancouver, Canada

Edited by

Gérard Huet and
Amba Kulkarni
Computational Sanskrit & Digital Humanities: 
Selected Papers Presented at the 17th World Sanskrit Conference, 
July 9-13, 2018, Vancouver, Canada.

DOI: 10.14288/1.0391834. 
URI: http://hdl.handle.net/2429/74653.

Edited by Gérard Huet and Amba Kulkarni 
General Editor: Adheesh Sathaye

Electronic edition published (2020) by the Department of Asian Studies, University of British Columbia, for the International Association for Sanskrit Studies.


© Individual authors, 2020. Content is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 4.0 International license (CC BY-NC-ND 4.0). http://creativecommons.org/licenses/by-nc-nd/4.0/

All papers in this collection have received double-blind peer review.
Computational Sanskrit &
Digital Humanities

Selected papers presented
at
the 17th World Sanskrit Conference

University of British Columbia, Vancouver
9–13 July 2018

Edited by
GÉRARD HUET & AMBA KULKARNI
Preface

This volume contains edited versions of papers accepted for presentation at the 17th World Sanskrit Conference in July 2018 in Vancouver, Canada. A special track of the conference was reserved for the topic “Computational Sanskrit & Digital Humanities”, with the intent to cover not only recent advances in each of the now mature fields of Sanskrit Computational Linguistics and Sanskrit Digital Libraries, but to encourage cooperative efforts between scholars of the two communities, and prepare the emergence of grammatically informed digital Sanskrit corpus. Due to its rather technical nature, the contributions were not judged on mere abstracts, but on submitted full papers reviewed by a Program Committee.

We would like to thank the Program Committee of our track for their work:

- Dr Tanuja Ajotikar, Belgavi, Karnataka
- Pr Stefén Baums, University of Munich
- Pr Yigal Bronner, Hebrew University of Jerusalem
- Pr Pawan Goyal, IIT Kharagpur
- Dr Oliver Hellwig, Düsseldorf University
- Dr Gérard Huet, Inria Paris (co-chair)
- Pr Girish Nath Jha, JNU, Delhi
- Pr Amba Kulkarni, University of Hyderabad (co-chair)
- Dr Pawan Kumar, Chinmaya Vishwavidyapeeth, Veliyanad
- Pr Andrew Ollett, Harvard University
- Dr Dhaval Patel, I.A.S. Officer, Gujarat
- Pr Srinivasa Varakhedi, KSU, Bengaluru

14 contributions were accepted, revised along referees’ recommendations, and finely edited to form this collection.

The first two papers concern the problem of proper mechanical simulation of Pāṇini’s Aṣṭadyāyī. In “A Functional Core for the Computational
Aṣṭadyāyī”, Samir Janardan Sohoni and Malhar A. Kulkarni present an original architecture for such a simulator, based on concepts from functional programming. In their model, each Pāṇinian sūtra translates as a Haskell module, an elegant effective formalization. They explain an algorithm for sūtra-conflict assessment and resolution, discuss visibility and termination, and exhibit a formal derivation of word bhavati as a showcase.

A different computational framework for the same problem is offered by Sarada Susarla, Tilak M. Rao and Sai Susarla in their paper “PAIAS: Pāṇini Aṣṭadyāyī Interpreter As a Service”. They explain their development of a Web service usable as a Sanskrit grammatical assistant, implementing directly the Pāninian mechanisms. Here sūtras are records in a database in the JSON format, managed by a Python library. They pay particular attention to the meta-rules of the grammar, and specially to defining sūtras. They refrain from expanding such definitions in operative sūtras, but insist on their emulation along the grammatical processing.

These two papers conceptualize two computational models of Pāninian grammar that are strikingly different in their software architecture. However, when one examines examples of sūtra representations in both systems, the information content looks very similar, which may suggest some future inter-operability of these two interesting tools.

In the general area of mechanical analysis of Sanskrit text, we have several contributions at various levels. At the level of semantic roles analysis, at the heart of dependency parsing, Sanjeev Panchal and Amba Kulkarni present possible solutions to the complementary problem of ambiguity. In their paper “Yogyatā as an absence of non-congruity” they explain various definitions used by Sanskrit grammarians to express compatibility, and how to use these definitions to reduce ambiguity in dependency analysis.

The next paper, “An ‘Ekalavya’ Approach to Learning Context Free Grammar Rules for Sanskrit Using Adaptor Grammar”, by Amrith Krishna, Bodhisattwa Prasad Majumder, Anil Kumar Boga, and Pawan Goyal, presents an innovative use of adaptor grammars to learn patterns in Sanskrit text definable as context-free languages. They present applications of their techniques to word reordering tasks in Sanskrit, a preliminary step towards recovering prose ordering from poetry, a crucial problem in Sanskrit.

Concerning meter recognition, we have a contribution of Shreevatsa Rajagopalan on “A user-friendly tool for metrical analysis of Sanskrit verse”. The main feature of this new metrical analysis tool, available either as a
Web service or as a software library, is its robustness and its guidance in error-correction.

Two more contributions use statistical techniques (Big Data) for improving various Sanskrit-related tasks.

For instance, in the field of optical character recognition, the contribution by Devaraja Adiga, Rohit Saluja, Vaibhav Agrawal, Ganesh Ramakrishnan, Parag Chaudhuri, K. Ramasubramanian and Malhar Kulkarni on “Improving the learnability of classifiers for Sanskrit OCR corrections”.

In the same vein of statistical techniques, Nikhil Chaturvedi and Rahul Garg present “A Tool for Transliteration of Bilingual Texts Involving Sanskrit”, which accommodates smoothly text mixing various encodings.

While much of the work in Sanskrit computational linguistics is on Classical Sanskrit, researchers are also applying computational techniques to Vedic Sanskrit. One such effort is a detailed formalization of Vedic recitation phonology by Balasubramanian Ramakrishnan: “Modeling the Phonology of Consonant Duplication and Allied Changes in the Recitation of Tamil Taittiryaka-s”.

On a more theoretical perspective on Sanskrit syntax, Brendan Gillon presents a formalization of Sanskrit complements in terms of the categorial grammar framework. His paper “Word complementation in Sanskrit treated by a modest generalization of categorial grammar” explains modified versions of the cancellation rules that aim at accommodating free word order. This raises the theoretical problem of the distinction between complements and modifiers in Sanskrit.

Turning to the Digital Humanities theme, we have a number of contributions. In “TEITagger: Raising the standard for digital texts to facilitate interchange with linguistic software”, Peter Scharf discusses how fine-grain XML representation of corpus within the Text Encoding Initiative standard allows the inter-communication between digital Sanskrit libraries and grammatical tools such as parsers as well as meter analysis tools.

A complementary proposal is discussed in the paper “Preliminary Design of a Sanskrit Corpus Manager” by Gérard Huet and Idir Lankri. They propose a scheme for a fine-grained representation of Sanskrit corpus allowing inter-textuality phenomena such as sharing of sections of text, but also a variance of readings. They propose to use grammatical analysis tools to help annotators feeding digital libraries with grammatical information using modern cooperative work software. They demonstrate a prototype of such a tool, in the framework of the Sanskrit Heritage platform.
Moving towards philological concerns such as critical editions, the paper “Enriching the digital edition of the Kāśikāvṛtti by adding variants from the Nyāsa and Padamañjarī”, by Tanuja P. Ajotikar, Anuja P. Ajotikar, and Peter M. Scharf discusses the problem of managing complex information from recensions and variants. It argues for a disciplined method of using TEI structure to represent this information in machine-manipulable ways, and demonstrates its use on processing variants of the Kāśikāvṛtti, the major commentary of the Aṣṭadyāyī.

In the same area of software-aided philology, the contribution “From the web to the desktop: IIIF-Pack, a document format for manuscripts using Linked Data standards”, by Timothy Bellefleur, presents a proposal for a common format fit to manage complex information about corpus recensions in various formats, including images of manuscripts. This is in view of facilitating the interchange of such data by various teams using this common format. His proposal uses state-of-the-art standards of hypertext. It has already been put to use in an interactive software platform to manage recensions for the critical edition of the Vetālapañcaviṃśati by Pr. Adheesh Sathaye.

The volume concludes with the contribution “New Vistas to study Bhartṛhari: Cognitive NLP” by Jayashree Aanand Gajjam, Diptesh Kanojia, and Malhar Kulkarni which presents highly original research on cognitive linguistics in Sanskrit, by comparing the results of experiments with eye-tracking equipment with theories of linguistic cognition by Bhartṛhari.

We thank the numerous experts who helped us in the review process and all our authors who responded positively to the reviewer’s comments and improved their manuscripts accordingly. We thank the entire 17th WSC organizing committee, led by Pr. Adheesh Sathaye, which provided us the necessary logistic support for the organization of this section.
Contributors

Devaraja Adiga
Department of Humanities and Social Sciences,
Indian Institute of Technology Bombay,
Powai, Mumbai, India.
pdadiga@iitb.ac.in

Vaibhav Agrawal
Indian Institute of Technology,
Kharagpur,
India.
vaibhav@iitkgp.ac.in

Anuja Ajotikar
Shan State Buddhist University,
Myanmar
anujaajotikar@gmail.com

Tanuja Ajotikar
KAHER’s Shri B. M. Kankanwadi Ayurveda Mahavidyalaya,
Belagavi,
India.
gtanu30@gmail.com

Timothy Bellefleur
Department of Asian Studies,
University of British Columbia, Vancouver
tbelle@alumni.ubc.ca
Anil Kumar Boga
Department of Computer Science and Engineering,
Indian Institute of Technology,
Kharagpur, India.
bogaanil.009@gmail.com

Nikhil Chaturvedi
Department of Computer Science and Engineering,
Indian Institute of Technology,
New Delhi
cs5130291@cse.iitd.ac.in

Parag Chaudhuri
Indian Institute of Technology Bombay,
Powai, Mumbai, India.
paragc@cse.iitb.ac.in

Jayashree Aanand Gajjam
Department of Humanities and Social Sciences,
Indian Institute of Technology Bombay,
Powai, Mumbai, India.
jayashree_aanand@iitb.ac.in

Rahul Garg
Department of Computer Science and Engineering,
Indian Institute of Technology,
New Delhi
rahulgarg@cse.iitd.ac.in

Brendan S. Gillon
McGill University
Montreal, Quebec
H3A 1T7 Canada
brendan.gillon@mcgill.ca
Pawan Goyal  
Department of Computer Science and Engineering,  
Indian Institute of Technology,  
Kharagpur, India. 
pawang@cse.iitkgp.ernet.in

Gérard Huet  
Inria Paris Center,  
France.  
Gerard.Huet@inria.fr

Diptesh Kanojia  
IITB-Monash Research Academy, Powai, Mumbai, India  
diptesh@iitb.ac.in

Amrith Krishna  
Department of Computer Science and Engineering,  
Indian Institute of Technology,  
Kharagpur, India.  
amrith.krishna@cse.iitkgp.ernet.in

Amba Kulkarni  
Department of Sanskrit Studies,  
University of Hyderabad,  
Hyderabad, India.  
apksh@uohyd.ernet.in

Malhar Kulkarni  
Department of Humanities and Social Sciences,  
Indian Institute of Technology Bombay,  
Powai, Mumbai, India.  
malhar@hss.iitb.ac.in
IDIR LANKRI
Université Paris Diderot, 
Paris
lankri.idir@gmail.com

BODHISATTWA PRASAD MAJUMDER
Walmart Labs
India.
bodhisattwam2017@email.iimcal.ac.in

SANJEEV PANCHAL
Department of Sanskrit Studies, 
University of Hyderabad, 
Hyderabad, India.
snjvpnch1@gmail.com

SHREEVATSA RAJAGOPALAN
Independent Scholar
1035 Aster Ave 1107
Sunnyvale, CA 94086, USA
shreevatsa.public@gmail.com

BALASUBRAMANIAN RAMAKRISHNAN
Independent Scholar
145 Littleton Rd
Harvard, MA 01451, USA
balasr@acm.org

GANESH RAMAKRISHNAN
Indian Institute of Technology Bombay, 
Powai, Mumbai, India.
ganesh@cse.iitb.ac.in
Contributors

K. Ramasubramanian
Department of Humanities and Social Sciences,
Indian Institute of Technology Bombay,
Powai, Mumbai, India.
ram@hss.iitb.ac.in

Tilak M Rao
School of Vedic Sciences,
MIT-ADT University,
Pune, India.
rao.tilak@gmail.com

Rohit Saluja
IITB-Monash Research Academy,
Powai, Mumbai, India.
rohitsaluja@cse.iitb.ac.in

Peter Scharf
The Sanskrit Library,
Providence, Rhode Island, U.S.A.
and
Language Technologies Research Center,
Indian Institute of Information Technology,
Hyderabad, India.
scharf@sanskritlibrary.org

Samir Janardan Sohoni
Department of Humanities and Social Sciences,
Indian Institute of Technology Bombay,
Powai, Mumbai, India.
sohoni@hotmail.com

Sarada Susarla
Karnataka Sanskrit University,
Bangalore, India.
sarada.susarla@gmail.com
Sai Susarla
School of Vedic Sciences,
MIT-ADT University,
Pune, India.
sai.susarla@gmail.com
Contents

Preface i

Contributors v

A Functional Core for the Computational Aṣṭādhyāyī 1

Samir Sohoni and Malhar A. Kulkarni

PAIAS: Pāṇini Aṣṭādhyāyī Interpreter as a Service 31

Sarada Susarla, Tilak M. Rao and Sai Susarla

Yogyatā as an Absence of Non-congruity 59

Sanjeev Panchal and Amba Kulkarni


Amrith Krishna, Bodhisattwa Prasad Majumder, Anil Kumar Boga, and Pawan Goyal

A User-friendly Tool for Metrical Analysis of Sanskrit Verse 113

Shreevatsa Rajagopalan
Improving the Learnability of Classifiers for Sanskrit OCR Corrections 143

Devaraja Adiga, Rohit Saluja, Vaibhav Agrawal, Ganesh Ramakrishnan, Parag Chaudhuri, K. Ramasubramanian and Malhar Kulkarni

A Tool for Transliteration of Bilingual Texts Involving Sanskrit 163

Nikhil Caturvedi and Rahul Garg

Modeling the Phonology of Consonant Duplication and Allied Changes in the Recitation of Tamil Taittirīyaka-s 181

Balasubramanian Ramakrishnan

Word Complementation in Classical Sanskrit 217

Brendan Gillon

TEITagger
Raising the Standard for Digital Texts to Facilitate Interchange with Linguistic Software 229

Peter M. Scharf

Preliminary Design of a Sanskrit Corpus Manager 259

Gérard Huet and Idir Lankri

Enriching the Digital Edition of the Kāśikāvṛtti by Adding Variants from the Nyāsa and Padamañjarī 277

Tanuja P. Ajotikar, Anuja P. Ajotikar, and Peter M. Scharf
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>From the Web to the desktop: IIIF-Pack, a document format for manuscripts using Linked Data standards</td>
<td>295</td>
</tr>
<tr>
<td>Timothy Bellefleur</td>
<td></td>
</tr>
<tr>
<td>New Vistas to study Bhartṛhari: Cognitive NLP</td>
<td>311</td>
</tr>
<tr>
<td>Jayashree Aanand Gajjam, Diptesh Kanojia and Malhar Kulkarni</td>
<td></td>
</tr>
</tbody>
</table>
A Functional Core for the Computational Aṣṭādhyāyī

SAMIR JANARDAN SOHONI and MALHAR A. KULKARNI

Abstract: There have been several efforts to produce computational models of concepts from Panini’s Aṣṭādhyāyī. These implementations targeted certain subsections of the Aṣṭādhyāyī such as the visibility of rules, resolving rule conflict, producing *sandhi*, etc. Extrapolating such efforts extremely will give us a much-coveted computational Aṣṭādhyāyī. A computational Aṣṭādhyāyī must produce an acceptable derivation of words showing the order in which *sūtra* s are applied.

We have developed a mini computational Aṣṭādhyāyī which purports to derive accented verb forms of the root *bhū* in the *laṭ lakāra*. An engine repeatedly chooses, prioritizes and applies *sutra* s to an input, given in the form of a *vivakṣā*, until an utterance is derived. Among other things, this paper describes the structure of *sūtra* s, the visibility of *sūtra* s in the *sapādasaptādhyāyī* and *tripādī* sections, phasing of the *sutra* s and the conflict resolution mechanisms.

We found that the *saṃjñā* and *vidhi sūtra* s are relatively simple to implement due to overt conditional clues. The *adhikāra* and *paribhāṣā* sutras are too general to be implemented on their own, but can be bootstrapped into the vidhi *sūtra* s. The *para-nitya-antaraṅga-apavāda* method of resolving *sūtra* conflicts was extended to suit the computational Aṣṭādhyāyī. Phasing can be used as a device to defer certain *sūtra* s to a later stage in the derivation.

This paper is the first part of a series. We intend to write more as we implement more from the Aṣṭādhyāyī.

Keywords: computational Ashtadhyayi, derivation, conflict resolution, sutra, visibility, phase
1 Introduction

An accent is a key feature of the Sanskrit language. While the ubiquitous accent of Vedic has fallen out of use in Classical Sanskrit, Pāṇini’s grammatical mechanisms are capable of producing accented speech. We aim to derive an accented instance by using computer implementation of Pāṇinian methods. Our system can produce the output shown in Listing 1.

Our implementation uses a representation of vivakṣā (See Listing 11) to drive the derivation. We model Aṣṭādhyāyī sūtras as requiring a set of preconditions to produce an effect. The sūtras look for their preconditions in an input environment. The effects produced by sūtras become part of an ever-evolving environment which may trigger other sūtras. To resolve rule conflicts, we have made a provision for a harness which is based on the paribhāṣā.

We have used Haskell, a lazy functional programming language, to build the prototype. Our implementation uses a phonetic encoding of characters for accentuation and Pāṇinian operations (See Sohoni and M. A. Kulkarni (2016)). The phonetic encoding allows for faster phonetic modifications and testing operation, something that seems to happen very frequently across most of the sūtras.

The following is an outline of the rest of this paper. Previous work related to the computational Aṣṭādhyāyī is reviewed in Section 2. Section 3 discusses how phonetic components, produced and consumed by sūtras, are represented. It also discusses tracing the antecedents of components. Implementation of sūtras is discussed in Section 4. Intermediate steps of a derivation, known as frames, are discussed in 5.1. Section 5.2 also discusses the environment which is used to check triggering conditions of sūtras. Deferring application of sūtras by arranging them into phases is discussed in 6. The process of derivation is explained in Section 7. Prioritization and conflict resolution of sūtras is discussed in Section 8. Section 9 discusses how visible frames in a derivation are made available to a sūtra. Some conclusions and future work are discussed in Section 10.
Wiwakshaa --->["gana",Just "1"],("purusha",Just "1"),
("wachana",Just "1"),("lakaara",Just "wartamaana"),
("prayoga",Just "kartari")]

Initial ---> भू

[***>(6.1.162) ---> भू
[***>(3.2.123) ---> भूलँद
[(1.4.13) wins (1.4.13) vs (1.3.9) by SCARE
]***>(1.4.13) ---> भूलँद
[***>(1.3.9) ---> भूल
[***>(3.4.78) ---> भूितप
[(1.4.13) wins (1.4.13) vs (1.4.104) by SCARE
,(1.4.13) wins (1.4.13) vs (1.3.9) by SCARE
,(1.4.13) wins (1.4.13) vs (3.1.68) by SCARE
,(1.4.13) wins (1.4.13) vs (7.3.84) by SCARE
]***>(1.4.13) ---> भूितप
[(1.4.104) wins (1.4.104) vs (1.3.9) by SCARE
,(1.4.104) wins (1.4.104) vs (3.1.68) by SCARE
,(1.4.104) wins (1.4.104) vs (7.3.84) by SCARE
]***>(1.4.104) ---> भूितप
[(3.1.68) wins (1.3.9) vs (3.1.68) by paratwa
,(7.3.84) wins (3.1.68) vs (7.3.84) by paratwa
]***>(7.3.84) ---> भोितप
[(3.1.68) wins (1.3.9) vs (3.1.68) by paratwa
]***>(3.1.68) ---> भोश॒िĢतप
[(1.4.13) wins (1.4.13) vs (1.3.9) by SCARE
]***>(1.4.13) ---> भोश॒िĢतप
[***>(1.3.9) ---> भोअति
[***>(1.4.14) ---> भोअति
[***>(1.4.109) ---> भोअति
[(8.4.66) wins (6.1.78) vs (8.4.66) by paratwa
]***>(8.4.66) ---> भोअति
[***>(6.1.78) ---> भवति
[***>(8.4.66) ---> भवति

Listing 1
A derivation of the pada भवति
2 Review of Literature

Formal foundations of a computational Aṣṭādhyāyī can be seen in Mishra (2008, 2009, 2010). The general approach in Mishra’s work is to take a linguistic form such as bhavati and apply heuristics to carve out some grammatical decompositions. The decompositions are used to drive analytical processes that may yield more decompositions along the boundaries of sandhis to produce seed-forms.1 This part is an analysis done in a top-down manner. The second phase is a bottom-up synthesis, wherein, each of the seed-forms is processed by a synthesizer to produce finalized linguistic expressions that must match the original input. To support analysis, Mishra’s implementation relies upon a database which contains partial orders of morphological entities, mutually exclusive morphemes, and other such artifacts.2 In the synthesis phase, Mishra (2010) also implements a conflict resolver using the siddha principle.3

Goyal, A. P. Kulkarni, and Behera (2008) have also created a computational Aṣṭādhyāyī which focuses on ordering sūtras in the regions governed by पूर्वनिर्देशः (A. 8.2.1), पत्तुकोरिसः (A. 6.1.86) and असिद्धद्विभावत (A. 6.4.22). Input, in the form of prakṛti along with attributes, is passed through a set of modules that have thematically grouped rules. The implementation embodies the notion of data spaces. Rules are able to see various data spaces in order to take input. Results produced by the rules are put back into the appropriate data spaces. Conflict resolution is based on paribhāṣā-driven concepts such as principle of apavāda as well as ad-hoc techniques.4

Goyal, A. P. Kulkarni, and Behera (2008) §4 mention the features of a computational Aṣṭādhyāyī. Also, a computational Aṣṭādhyāyī should seamlessly glue together grammatical concepts just like the traditional Aṣṭādhyāyī. It should not add any side effects, neither should it be lacking any part of the traditional Aṣṭādhyāyī. Above all, a computational Aṣṭādhyāyī must produce an acceptable derivation of words.

The system described in Mishra (2010) does not use traditional building blocks such as the Māheśvara Sūtras or the Dhātupātha, but can be made

---

1See Mishra (2009), Section 4.2 for a description of the general process.
2See Mishra (2009), Section 6.1 for details of the database.
3Mishra (2010):255
4Goyal, A. P. Kulkarni, and Behera (2008), cf. ‘Module for Conflict Resolution’ in §4.4
to do so.\textsuperscript{5} We believe that canonical building blocks such as Māheśvara Sūtras and Aṣṭādhyāyī śūtrapāṭha should strongly influence the computational Aṣṭādhyāyī.

Peter M. Scharf (2016) talks about the need for faithfully translating Pāṇinian rules in the realm of computation and shows elaborate XMLization of Pāṇini’s rules. Bringing Pāṇini’s rules into the area of computation uncovers some problems that need to be solved. T. Ajotikar, A. Ajotikar, and Peter M. Scharf (2016) discuss some of those issues.

XML is useful in describing structured data and therefore XMLization of the Aṣṭādhyāyī is a step in the right direction. However, processing Pāṇinian derivations in XML will be fraught with performance issues. XML is good for the specification of data but it cannot be used as a programming language. A good deal of designing a computational Aṣṭādhyāyī will have to focus on questions such as “How to implement (not specify) the notion $X$”. $X$ may refer to things such as run-time evaluation of apavāda, or dynamically creating any pratyāhāra from the Māheśvara Sūtras or dealing with an atideśasūtra so that a proper target rule is triggered. The power of a real, feature-rich programming language will be indispensable in such work.

Patel and Katuri (2016) have demonstrated the use of programming languages to derive subanta forms. Patel and Katuri have discovered a manual way to order rules (NLP ordering) for producing subantas according to Bhattōji Dikṣita’s Vaiyākaraṇa Siddhāntakaumudi. It is conceivable that as more rules are added to the system to derive other types of words, the NLP ordering may undergo a lot of change and it may ultimately approach the order that comes about due to Pāṇinian paribhāṣās and those compiled by Nagojibhaṭṭa (See Kielhorn (1985)).

In the present paper we describe the construction of rules, the progress of a derivation, the resolution of conflicts by modeling competitions between the rules in the ambit of paribhāṣā पूर्वपरिनित्यात्मकमुदत्रस्यवादानाम उत्तरोत्तरं बलीयं and other such concepts.

\textsuperscript{5}Mishra (2010):256, §4, “There is, however, a possibility to make the system aware of these divisions.”
type Attribute v = (String, Maybe v)

```haskell
data Component = Component {cmpWarnas :: [Warna], cmpAttrs :: [Tag], cmpOrigin :: [Component]}

type State = [Component]
```

Listing 2

State

3 Phonetic Components

The phonetic payload, which comprises of phonemes, is known as a Component. Listing 2 shows the implementation.\(^6\) A Component is made up of phonetically encoded Warnas. Some name-value pairs known as Tags give meta information about the Components. Usually, the tags contain samjñas. Over the course of a derivation, Components can undergo changes. At times, sūtras are required to test previous incarnations of a sthānī (substituend), so a list of previous forms of the Components is also retained. The current yield of the derivation at any step is in the State which is a list of Components.

Listing 3 shows how भू + तिप can be represented as an intermediate phonetic State. The Devanāgarī representations of भू and तिप are converted into an internal representation of Warnas using the encode function. Suitable tags are applied to the components bhu and tip and they are strung together in a list to create a State.

3.1 Tracing Components to Their Origins

The sūtra बर्तमाने रूट्र (A. 3.2.123) inserts a लृट्र pratyaya after a dhātu. This pratyaya will undergo changes and ultimately become द्र due to application

---

\(^6\)Excerpts of implementation details are shown in Haskell. References to variable names in computer code are shown in bold teletype font. In code listings the letter ‘w’ is used for व. In other places the Roman transliteration is used which prefers ‘v’ instead of ‘w’.
egState = let bhu = Component (encode "भू") -- phonemes
    [(["dhaatu",Nothing])] -- tags
    [] -- no previous history

tip = Component (encode "ित्") -- phonemes
    [(["wibhakti",Nothing]) -- tags
    ,[(["parasmaipada",Nothing])
    ,[(["ekawachana",Nothing])
    ,[(["saarwadhaatuka",Nothing])
    ,[(["pratyaya",Nothing])]]
    [] -- no previous history

in [bhu, tip]

Listing 3
An example of State

of *it*-sutras A. 1.3.2-9. तृणकर्तिः (A. 1.3.8) will mark the लृ as an इत, causing its removal. A. 1.3.8 should not mark the लृ of a तृण तृण pratyaya as an इत. The लृ in ten lakāras is not an इत. These lakāras should figure into A. 1.3.8 as an exception list so that A. 1.3.8 does not apply to them. However, other sutras like A. 1.3.2, A. 1.3.3 and A. 1.3.9 may still apply leaving back only लृ. If a list of ten lakāras was kept as an exception list in A. 1.3.8, लृ will not match any one of those and will be liable to be dropped. Somehow, the लृ which remains from lakāras, needs to be traced back to the original lakāra.

As shown in Listing 2, the datatype Component recursively contains a list of Components. The purpose of this list is to keep around previous forms of a Component. As a Component changes, its previous form along with all attributes is stored at the head of the list. This makes it easy to recover any previous form of a component and examine it. Listing 4 shows the traceOrigin function. In case of 1.3.8, if calling traceOrigin on a लृ produces one of the 10 lakāras, 1.3.8 does not mark such a लृ an इत. This way of tracing back Components to their previous forms can help in determination of a sthānī and its attributes under the influence of atideśa sūtra like स्थानिवददेशोऽनल्लियो (A. (1.1.56).
traceOrigin :: Component -> [Component]
traceOrigin (Component ws as []) = []
traceOrigin (Component ws as os) =
    nub \$ (concat.foldr getOrig [os]) os
    where getOrig c os = (traceOrigin c) : os

Listing 4

Tracing origin of a Component

4 Sūtras

According to one opinion in the Pāṇinian tradition, there are six different types of sūtras. The following verse enumerates them;\(^7\)

संज्ञा च परिभाषा च विधिनियम एव च ।
अतिदेशोपविकारवश पद्विंच सूऽलक्षणम् ॥

The saṃjñā sūtras apply specific saṃjñās to grammatical entities based on certain indicatory marks found in the input. They help the vidhi sūtras bring about changes.

The real executive power of the Aṣṭādhyāyī lies in the vidhi, niṣedha and niyama sūtras. The vidhi sūtras bring about changes in the state of the derivation. The niṣedha and niyama sūtras are devices that prevent over-generation of vidhi sūtras. They are strongly associated with specific vidhi sūtras and also share some of their conditioning information.

The paribhāṣā sūtras are subservient to vidhisūtras. They can be thought of as algorithmic helper functions which are called from many places in a computer program. In the spirit of the kāryakālapakṣa, the paribhāṣā sūtras are supposed to unite with vidhi sūtras to create a complete sūtra which produces a certain effect. The paribhāṣā sūtras need not be explicitly implemented because their logic can be embedded into the vidhi sūtras.

The adhikāra sūtras create a context for vidhi sūtras to operate. From an implementation perspective, the context of the adhikāra can be built into the body of vidhi, niṣedha or niyama sūtras and therefore adhikāra sūtras need not be explicitly implemented.

\(^7\)Vedantakeshari (2001):9-11. According to other opinions there are 7 or even as many as 8 types of sūtras if niṣedha and upavidhi types are considered.
The atideśa sūtras create situational analogies. By forming analogies, atideśa sūtras cause other vidhi sūtras to trigger. In this implementation we implement saṃjñā, vidhi and niyama sūtras. We have not implemented atideśa sūtras.

In traditional learning, every paribhāṣā sūtra is expected to be known in the place it is taught. The effective meaning of a vidhi sūtra is known by resorting to the methods of yathoddeśapakṣa or kāryakālapakṣa. In one opinion, in the yathoddeśapakṣa the boundary of the sapādasaptādhyāyī and the tripādi presents an ideological barrier which cannot be crossed over by the paribhāṣā sūtras for reasons of being invisible. The kāryakālapakṣa has no such problem.\(^8\)

We are inclined towards an implementation based on kāryakālapakṣa as it allows us to escape having to implement each and every paribhāṣā sūtra explicitly and yet enlist the necessary paribhāṣā sūtras’ numbers which go into creating ekavākyatā (full expanded meaning). This choice allows for swift development with less clutter. Therefore, the paribhāṣā and adhikāra sūtras are not explicitly implemented.

Sūtras are defined as shown in Listing 5. In the derivation of the word भव॑ित॒ no niyama sūtras were encountered, so they are not implemented in this effort but could be implemented by adding a Niyama value constructor. The Widhi value constructor is used to represent all types of sūtras other than saṃjñā sūtras. The Samjnyaa value constructor is used to make saṃjñā sūtras. Both the value constructors appear to be same in terms of their parameters, only the name of the constructor differentiates them. This is useful in pattern matching on sūtra values in the SCARE model (Section 8.4) which treats saṃjñā sūtras specially.

### 4.1 Testing the Conditions for Application of sūtras

In a computational Aṣṭādhyāyī, a sūtra must be able to sense certain conditions that exist in the input and it should also be able to produce an effect. These are the two basic requirements any implementable sūtra must satisfy. The Testing field in Listing 5 refers to a datatype that has testing functions slfTest and condTest. A sūtra will be able to produce its effect provided that slfTest returns True and condTest returns a function which can produce effects. More on this is explained in Section 7.1. Listing 6 shows

\(^8\)See Kielhorn (1985), paribhāṣās 2 & 3 – कार्पकारणपक्षे तु विनािामयपरिशिष्टिरिति विशेष:
\(^9\)Rgvedic convention is used to show accent marks.
Listing 5

Definition of sūtra

datatype Testing. Function slfTest is used to prevent a sūtra from applying ad infinitum. Some sūtras produce an effect without any conditions. For example, परः सत्तिकर्षे संसिहता (A. 1.4.109) defines the samjñā samhitā (the mode of continuous speech) which is not pre-conditioned by anything which can be sensed in the input. This sūtra can get applied and reapplied continuously had it not been for function slfTest. The slfTest function in sūtra A. 1.4.109 allows its application only if it was not applied earlier. Unlike A. 1.4.109, some sūtras produce effects which are conditioned upon things found in the input. For example, उदात्तादनुदात्तातः (A. 8.4.66) will look for an udātta syllable followed by an anudātta one in samhitā and convert the anudātta into a svarita syllable. As long as there is no udātta followed by anudātta in the input, A. 8.4.66 will not apply. A. 8.4.66 does not run the risk of being applied ad infinitum because it is conditioned on things which can be sensed in the input. Therefore, function slfTest in A. 8.4.66 always returns True. Function condTest should test for the condition that an udātta is followed by an anudātta in samhitā, in which case it should return True. Listing 6 shows the functions which each sūtra is expected to implement.

If a sūtra is inapplicable, condTest returns Nothing, which means no effects can be produced. In case a sūtra is applicable condTest returns an effect function. Simply calling the effect function with the correct Environment parameter will produce an effect as part of a new Environment. Effect is simply a function that takes an Environment and produces a newer Environment. Each Sutra is expected to implement the functions slfTest and condTest.
type Effect = TheEnv -> TheEnv

data Testing = TestFuncs {
    slfTest :: Environment Trace -> Bool
    , condTest :: Environment Trace ->
                ([Attribute String], Maybe Effect)

    }

Listing 6

Definition of testing functions

4.2 Organization of sūtras

The sūtras are implemented as Haskell modules. Every sūtra module exports a details function. The details function gives access to the definition of the sūtra and also the slfTest and condTest functions which are required in other parts of the code. Listing 7 shows a rough sketch of sūtra A. 1.3.9. In addition, every sūtra will have to implement its own effects function.
module S1_3_9 where

-- imports omitted for brevity

details = Widhi (SutraNumber 1 3 9)
    (TestFuncs selfTest condTest)

selfTest :: TheEnv -> Bool
selfTest _ = True

condTest :: TheEnv -> ([Attribute String], Maybe Effect)
condTest env = -- details omitted for brevity

effects :: Effect
effects env = -- details omitted for brevity

Listing 7
Sūtra module
5 The Ecosystem for Execution of Sūtras

The next sūtra applicable in a derivation takes its input from the result produced by the previous one. Due to the siddha/asiddhā notion between certain sūtras, it can be generally said that the input for the next sūtra may come from any of the previously generated results. This section discusses the ecosystem in which sūtras get their inputs.

5.1 Frames

As each sūtra applies in a derivation, some information about it is captured in a record known as a Frame. Some of the information, such as all the Conflicts, is captured for reporting. The two important constituents of Frame are the Sutra which was applied and the State it produced. As a derivation progresses, the output State from one sūtra becomes the input State of another.

Listing 8 shows Frame as an abstract datatype which expects two types, conflict and sutra, to create a concrete datatype. Frame (Conflict Sutra) Sutra is the realization of a concrete type which is, for convenience, called TheFrame. The Trace is merely a list of TheFrames. It is meant to show a step-by-step account of the derivation.

5.2 Environment

To produce an effect, some sūtras look at indicators in what is fed as input. A sūtra such as सावधातुकाधातुकयोः (A. 7.3.84) is expected to convert an इक्क letter at the end of the aṅga into a guṇa letter, provided that a sārvadhātuka

```haskell
data Frame conflict sutra = Frame {frConflicts :: [conflict]
  ,frSutra :: (Maybe sutra)
  ,frOutput :: State
}

type TheFrame = Frame (Competition Sutra) Sutra

type Trace = [TheFrame]
```

Listing 8

The Trace
or ārdhadhātuka pratyaya follows. If this sūtra is fed an input such as the one shown in Listing 9, all necessary conditions can be found in this input viz. there is an इक् at the end of the aṅga, followed by the sārvadhātuka pratyaya ितप्.

Now that its required conditions have been fulfilled, A. 7.3.84 produces an effect such as the one shown in Listing 10. Thus, the input becomes an environment that is looked at by the sūtras to check for any triggering conditions. A sūtra may need to look past its input into the input of some previously triggered sūtras. Generalizing this, an environment consists of outputs produced by all sūtras in the derivation thus far. The Trace data structure (see Section 5.1) becomes a very important constituent of Environment.

There are sūtras which produce an effect conditioned by what the speaker intends to say. वर्तमाने लटू (A. 3.2.123), for example, will be fed an input which may contain entities like a dhātu along with other samjñās associated with it. However, the specific lakāra, which the dhātu must be cast into, can be known only from the intention of the speaker. Unless it can be sensed
type Wiwakshaa = [Tag]
eggWiwakshaa = [attr gana "1",
                 ,attr purusha "1",
                 ,attr wachana "1",
                 ,attr lakaara "wartamaana",
                 ,attr prayoga "kartari",
                 ,attr samhitaa "yes"
               ]

Listing 11
Tags to describe vivakṣā

that the speaker wishes to express a vartamāna form, A. 3.2.123 cannot be applied. Thus, some sūtras are conditioned on what is in the vivakṣā. As shown in Listing 11, Wiwakshaa is modelled as a list of name-value Tags. The attr function is a helper which creates a Tag. This listing shows a vivakṣā for creating a 3rd person, singular, present tense, active voice form of some dhātu in samhitā mode.

In the case of certain sūtras the triggering conditions remain intact forever. Such sūtras tend to get applied repeatedly. To allow application only once, housekeeping attributes have to be maintained. The housekeeping attributes may be checked by the slfTest function in the sūtras and reapplication can be prevented. Consider वर्तमाने छट् (A. 3.2.123) once again. The vivakṣā will continue to have vartamāna in it. As such, A. 3.2.123 can get reapplied continuously. While producing the effect of inserting laṭ, A. 3.2.123 could create a housekeeping tag, say “(3.2.123)”. If A. 3.2.123 were to apply only in the absence of housekeeping attribute “(3.2.123)”, the reapplication could be controlled using a suitably coded slfTest function. Such Housekeeping is also part of the environment. Just like Wiwakshaa, it is also represented as a list of Tags. The entire representation of Environment is shown in Listing 12. It is a parameterized type that expects a type t to create an environment from. Environment Trace is a concrete type which is given an alias of TheEnv.
data Environment t = Env { envWiwaksha :: Wiwaksha,
              , envHsekapg :: Housekeeping,
              , envTrace :: t
            }

type TheEnv = Environment Trace

Listing 12
The Environment

6 Phasing

Samuel Johnson said that language is the dress of thought. Indeed, Pāṇini’s
generative grammar derives a correct utterance from an initial thought pattern. The seeds of the finished linguistic forms are sowed very early in the
process of derivation. Morphemes are gradually introduced depending on
certain conditions and are ultimately transformed into final speech forms.
It seems that linguistic forms pass through definite stages. This is a crude
approximation of the derivation process: laying down the seed form from
semantic information in the *vivakṣā*, producing *aṅgas*, producing *padas* and
finally making some adjustments for the *samhitā* mode of utterance. At each
step along the way, there could be several *sūtras* that may apply. Grammar-
ians call this situation a *prasaṅga*. However, only one *sūtra* can be applied
in a *prasaṅga*. When the most suitable *sūtra* gets applied it is said to have
become *pravṛtta*. To make the resolution of a *prasaṅga* relatively simple, *sū-
tras* apparently belonging to the latter stages should not get applied earlier
in the derivation, even if they have scope to apply.

Phasing is a method to minimize the number of *sūtras* that participate in
a *prasaṅga*. Those *saṃjñā sūtras*, which form the basis of certain *adhikāra sūtras*, are deferred until later in the derivation process. For instance, परः
सत्रिकर्ष: सहित (A. 1.4.109) creates a basis for the *samhitāyām adhikāra* which
begins from तयोय्यावचि सहितायय्य (A. 8.2.108). Similarly, यम्मान्त प्रत्ययविधत्स्वतदिधि
प्रत्येकम (A. 1.4.13) applies the *saṃjñā aṅga* to something when there is a
pratyaya after it. The *saṃjñā aṅga* creates a basis for the *adhikāra sūtra
अझय (A. 6.4.1). If the *sūtras*, which apply certain *saṃjñās*, are suppressed

See Abhyankar and Shukla (1961) pages 271 and 273
in the beginning of the derivation and are released subsequently, other *vidhi* *sūtras* operating within certain *adhikāras* will not participate in an untimely prasaṅga. For example, phasing परः सन्निकृष्ट: संहिता (A. 1.4.109) will defer *sūtras* like उद्दात्तदनुदात्तस्य स्वरितः (A. 8.4.66) till a later time.

A *Phase* has a name and contains a list of *sūtras* which make up that phase. Listing 13 defines a *Phase* and creates a list called *phases* containing two phases—“pada” and “samhitāa”. The way *phases* is defined, *pada* formation phase (due to A. 1.4.14) and *samhitā* formation phase (due to A. 1.4.109) will be deferred till a later time.

### 7 The Process of Derivation

The details of *Sutras* are collected in a list called the *ashtadhyayi*. For brevity, a small representation is shown in Listing 14.
-- remove phases from the ashtadhyayi
ashtWithoutPhases = filterSutras
  (predByPhases allPhases)
  ashtadhyayi

-- generate the word form using phases
generateUsingPhases :: TheEnv -> [Sutra] -> [Phase Sutra]
  -> TheEnv

generateUsingPhases env sutras phases =
  foldl’ (gen sutras) newEnv phases
  where
  gen sutras env phase = generate env (phsNums phase ++ sutras)
  newEnv = generate env sutras

-- the returned environment will contain the derivation
finalEnv = generateUsingPhases env ashtWithoutPhases allPhases

Listing 15
Generation using phases

Before the process of derivation begins, sūtras which are part of some phase, are removed from the ashtadhyayi. The generation of derived forms will continue as long as applicable sūtras are found in the ashtadhyayi. When sūtras are no longer applicable, sūtras from a phase are added to the ashtadhyayi. Adding one phase back to the ashtadhyayi holds the possibility of new sūtras becoming applicable. The process of derivation continues once again until no more sūtras are applicable. The process of adding back sūtras from other phases continues until there are no more phases left to add. Listing 15 shows this way of using phases to generate the derived form.

Function generateUsingPhases uses the generate function to actually advance the derivation. Given an Environment and a set of Sutras, the process of generating a linguistic form will consist of picking out all the applicable Sutras that have prasaṅga. The Sutras should get prioritized so that only one Sutra can become pravṛtta. The chosen Sutra should be invoked to produce a new Environment. This process can continue until
generate :: TheEnv -> [Sutra] -> TheEnv
generate env sutras
  | null (envWiwakshaa env) || null (envTrace env)
  | null sutras = env
  | otherwise =
    if (isJust chosen)
    then generate newEnv sutras
    else traceShow sutras env
where list = choose env sutras
  chosen = prioritize env (testBattery env) list
  newEnv = invoke env chosen

Listing 16
Generation of a derived form

no more Sutras apply in which case the derivation steps are shown using traceShow. The function generate embodies this logic as shown in Listing 16.

7.1 Choosing the Applicable sūtras

Given the list ashtadhyayi as defined in Section 7 and a starter Environment, each and every Sutra in the ashtadhyayi is tested for applicability. The applicability test is shown in Listing 17. A Sutra will be chosen if it clears two-stage condition checking. In the initial stage, slfTest checks if any Housekeeping attributes in the Environment prevent the Sutra from applying. If the initial stage is cleared, the second stage invokes the condTest function of the Sutra. condTest checks the Trace in the Environment for existence of conditions specific to the sūtra. In case the conditions exist, condTest returns a collection of Tags and a function, say eff, to produce the effects. See Section 4.1 to read more about slfTest and condTest.

The function choose, shown in Listing 17, uses the test described above. All Sutras in the ashtadhyayi, for which test returns an effects function eff, are collected and returned as a list. All sūtras in the list are applicable and have a prasaṅga. This list of sūtras has to be prioritized so that only one sūtra can be invoked.
choose :: TheEnv -> [Sutra] -> [(Sutra, [Tag], Effect)]
choose env ss =
  [fromJust r | r <- res, isJust (r) == True]
  where
    res = map appDetails ss
    t = envTrace env
appDetails :: Sutra -> Maybe (Sutra, [Tag], Effect)
appDetails sut = case (test env sut) of
  (_, Nothing) -> Nothing
  (conds, Just eff) -> Just (sut, conds, eff)

test :: TheEnv -> Sutra -> ([Tag],Maybe Effect)
test e s | null (envTrace e) = ([], Nothing)
  | otherwise = if slfTest testIfc e
    then condTest testIfc e
    else ([], Nothing)
  where testIfc = testing s

Listing 17
Choosing the applicable sūtras

8 Prioritizing sūtras

As the derivation progresses, many sūtras can form a prasaṅga, for their triggering conditions are satisfied in the environment. It is the responsibility of the grammar to decide which sūtra becomes pravṛttta by ensuring that the derivation does not loop continuously.

8.1 Avoiding Cycles

It may so happen that of all the sūtras in a prasaṅga, the one that has been chosen to become pravṛttta, say $S_c$, produces an Environment, say $E_i$, that already exists in the Trace. In case $E_i$ is reproduced, a cycle will be introduced which will cause the derivation to not terminate. While prioritizing, such sūtras, as producing an already produced Environment, must be filtered out.
8.2 Competitions for Conflict Resolution

The chosen sūtras can be thought to compete with one another. If there are $n$ sūtras there will be $(n-1)$ competitions. The first and the second sūtra will compete against one another. The winner among the two will compete against the third and so on until we are left with only one sūtra. The sūtra which triumphs becomes pravṛtta for it is the strongest among all those which had a prasaṅga. This view of conflict resolution is shown in Figure 1. S$_1$ to S$_n$ are competing sūtras.

![Figure 1](image)

*Competitions among sūtras*

We model competition as a match between two entities. The match can end in a draw or produce a winner. As shown in Listing 18, **Competition** is defined as an abstract type which contains a **Resolution**. The **Resolution** gives the **Result** and has a provision to note a **Reason** for that specific outcome of the match.

The actual competition is represented as a function which takes two Sutras and produces a **Resolution** as shown in Listing 19.

8.3 Competitions and Biases

A sūtra $S_i$ is eliminated as soon as it looses out to another sūtra $S_j$ and never participates in any other competition in the prasaṅga. One might object to this methodology of conducting competitions by suggesting that
Following are abstract types. Concrete realizations such as 'Conflict Sutra' and 'Resolution Sutra' are used in code.

```haskell
{-
    Following are abstract types. Concrete realizations such as 'Conflict Sutra' and 'Resolution Sutra' are used in code.
-}
data Conflict a = Conflict a a (Resolution a)
data Resolution a = Resolution (Result a) Reason
data Result a = Draw | Winner a
type Reason = String
```

**Listing 18**

*The Conflict between sūtras*

```haskell
type Competition a = Sutra -> Sutra -> Resolution Sutra
```

**Listing 19**

*Match between sūtras*

$S_i$ could have debarred another sūtra $S_k$ later on, therefore it is important to keep $S_i$ in the fray. In fact, the objector could claim that all sūtras must compete with one another before a sūtra can become pravṛtta. We note that the objector’s method of holding competitions would have been useful if the competition between $S_i$ and $S_k$ would produce a random winner every time. In fact, the competitions in this grammar are biased. They don’t give both the sūtras equal chance of winning and this is intentional in the design of Pāṇini’s grammar. In the presence of biases, what is the use of conducting fair competitions? Therefore the proposed method of holding competitions should be acceptable.

The biases are introduced by the maxim पूवपरि́त्यान्तरञ्जपवादानाम उत्तरोत्तरे बलीयः.\footnote{See Kielhorn (1985):१९, paribhāṣā 38.} One sūtra is stronger than another one by way of four tests, namely, paratva, nityatva, antarangatva and apavādatva. The paratva test says that, among any two sūtras, the one which is placed later in the Aṣṭādhyāyī wins. According to the nityatva test, among two competing sūtras, one that has prasaṅga inspite of the other being applied first, is the winner. The
A battery of tests to choose a winning sūtra

principle of antaraṅgatva dictates that of two conflicting sūtras, the one that wins, relies on relatively fewer nīmītta (conditions) or nīmīttas which are internal to something. Finally, the test of apavādatva teaches that, among two conflicting sūtras, a special sūtra wins over a general one to prevent niravakāśatva (total inapplicability) of the special sūtra. The four tests are such that a latter one is stronger determiner of the winning sūtra than the prior ones. Test of apavādatva has highest priority and paratva has the lowest. If the test of apavādatva produces a winner the other three tests need not be applied. If antaraṅgatva produces a winner, the other two tests need not be administered. If nityatva produces a winner we need not check paratva. In the worst-case scenario all the four tests have to be applied one after another to a pair of conflicting sūtras.

To seek a winning sūtra among two that compete with each other, a prioritization function administers a battery of tests, beginning with apavādatva (See Listing 20).
8.4 Sūtra-Conflict Assessment and Resolution Extension (SCARE)

The maxim पूर्वपरिनित्यान्तरझापवादानाम उत्तरोत्तरं वर्तीयः introduces four methods of conflict resolution as explained in Section 8.3. In tradition, these methods expect that samjñās have already been applied by resorting to either kāryakālapakṣa or yathoddeśapakṣa. However, computation differs from tradition, in that the assumptions made in traditional approach need to be explicitly executed in computation. In a computational Aṣṭādhyāyī, methods introduced by the maxim will work provided that all samjñās have already applied. Somehow samjñās need to apply before any of the methods in the maxim have applied. SCARE prioritizes samjñā sūtras higher than any other type of sūtra. When any other type of sūtra competes with a samjñā sūtra, the latter wins. In case samjñā sūtras compete, all of them will eventually get a chance to apply. Since SCARE is required to differentiate between samjñā and non-samjñā sūtras, the Sutra datatype has an explicit value constructor called Samjnyaa. Listing 21 shows the implementation of ScareTest.

8.5 Determination of apavāda sūtras

An apavāda relationship holds between a pair of sūtras when the general provision of one sūtra is overridden by the special provision of another sūtra. Arbitrary pairs of sūtras may not always have an apavāda relation between them. There has to be a reason for an apavāda relationship to exist between two sūtras. The niṣedha and niyama sūtras suggest something which can
run counter to what other sūtras say. Therefore they are called apavādas of other sūtras.

A niṣedha sūtra, say न विभक्तिः सुमा: (A. 1.3.4), is considered an apavāda of the general one such as हल्ध अन्त्यम (A. 1.3.3). A niṣedha sūtra directly advises against taking an action suggested by another sūtra. Another type of rule, the niyama sūtra, does the work of regulating some operation laid down by another rule. A niyama sūtra such as धातोः तििमि समव (A. 6.1.80) is considered an apavāda of a more general rule such as वान्तो यि प्रत्यये (A. 6.1.79).

Wherever an apavāda relationship holds between two sūtras, it is static and unidirectional. Also, there can be apavādas of apavādas. Since the corpus of Pāṇini’s rules is well known, the apavāda relationships can be worked out manually to build a mapping of the apavādas. Thus, A. 1.3.4 is considered an apavāda of A. 1.3.3 and A. 6.1.80 is considered an apavāda of A. 6.1.79.

Listing 22 shows how apavādas are setup. The datatype Apawaada records a main sūtra and notes all the apavādas of the main sūtra. allApawaadas is a list of Apawaadas which is converted into a map apawaadaMap for faster access. Function isApawaada is used to check if sūtra s1 is an apavāda of sūtra s2. The apawaada function in Listing 20 uses the isApawaada function to return an apavāda or returns a draw if apavāda relationship does not exist between sūtras.

9 Visibility

The canonical term siddha means that the output from one sūtra A is visible and can possibly trigger another sūtra B. The effects of A are visible to B. The sūtras in the tripādī are not siddha in the sapādasaptādhyāyī. This means that even if sūtras from the tripādī have applied in a derivation, the results produced are not visible to the sūtras in the sapādasaptādhyāyī. In other words, the State produced by certain sūtras, cannot trigger sūtras in a specific region of the Aṣṭādhyāyī.

Listing 23 shows the visibility as dictated by पूविसद्यासिद्धम (A. 8.2.1). The entire derivation thus far is captured as a Trace containing frames Fn thru F1. The problem is this: Given a sūtra, say Si, the latest Frame Fj is

---

12 पूविसद्यासिद्धम

13 सिद्ध सिद्ध आयन्तमाणो विद्याय निषेधसमृ।
data Apawaada = Apawaada { apwOf :: SutraNumber,
                        , apwApawaadas :: [SutraNumber] }

allApawaadas = [Apawaada (SutraNumber 1 3 3)
                , ((SutraNumber 1 3 4))
               ]

apawaadaMap = M.fromList [(apwOf a, apwApawaadas a)
                          | a <- allApawaadas]

-- | Is sutra s1 an apavāda of s2?
isApawaada :: SutraNumber -> SutraNumber -> Bool
isApawaada s1 s2 = case M.lookup s2 apawaadaMap of
                      Just sns -> s1 `elem` sns
                      _         -> False

Listing 22
Setting up apavādas
visibleFrame :: Sutra -> Trace -> Maybe TheFrame
visibleFrame _ [] = Nothing
visibleFrame s (f@(Frame _ (Just s1) _ _):fs) =
  if curr < s8_2_1
    then if top < s8_2_1
        then Just f
        else visibleFrame s fs
    else visibleFrame s fs
  else if top > curr
    then visibleFrame s fs
    else Just f

where s8_2_1 = SutraNumber 8 2 1
curr = number s
top = number s1

Listing 23
Visibility

required such that $F_j$ contains $sutra$ $S_k$ whose output State is visible to $S_i$. If $S_i$ is from the sapādasaptaśādyāyī, frames from the head of the Trace are skipped as long as they contain $sūtras$ from tripādī. Such frames will be asiddha for $S_i$. If, however, $S_i$ is from tripādī, frames from the trace are skipped so long as the $sūtra$ in the frame has a number higher than $S_i$. This is because in the tripādī latter $sūtras$ are asiddha to prior ones. The foregoing logic is implemented in function visibleFrame.

10 Conclusion and Future Work

A computational Aṣṭādhyāyī can potentially become a good pedagogical resource to teach grammatical aspects of Sanskrit. As a building block, a computational Aṣṭādhyāyī can be used to build other systems like morphological analyzers and dependency parsers.

From an implementation perspective, resorting to kāryakālapakṣa allows paribhāṣā and adhikāra $sūtras$ to be merged into the logic of vidhi or niyama $sūtras$. While displaying the derivation after it is completed, the concerned vidhi and niyama $sūtras$ can always enlist numbers of the paribhāṣā and adhikāra $sūtras$ which they have united with. This allows for faster de-
development of the computational Aṣṭādhyāyī without having to implement seemingly trivial sūtras in the paradigm used to implement sūtras as noted in Section 4.2.

It could be suboptimal to represent all grammatical entities as **Components** having **Warnas** and **Tags**. To a certain extent, it increases the number of **Tags** applied to **Components**. For example, since pratyayayās are expressed as a **Component**, a ‘pratyaya’ **Tag** has to be applied. Functional languages have extremely powerful type systems. To leverage the type system, **Components** can be implemented as typed grammatical entities. For instance, a **Component** can have more value constructors for **Upasarga**, **Dhaatu** and **Pratyaya**, to name a few.

Abstracting the input as vivakṣā does away with the need of applying heuristics to determine what needs to be derived. However, our choice of representing Wiwakshaa as a simple list of **Tags** is an oversimplification. The vivakṣā could be a complex psycholinguistic artifact which may contain elements such as the kārakas, hints for using specific dhātu, argument structure of dhātus, etc. It may have a sophisticated data structure. A thorough study of semantic aspects of Aṣṭādhyāyī is necessary to know what vivakṣā may look like in its entirety.

In the बाधबीजप्रकरणम्, Kielhorn (1985) discusses many variations under each of the four methods introduced in the para-nitya paribhāṣā. Those variations should be plugged into the framework discussed in Section 8. Yet, there may be instances of derivations where the maxim पूर्वपरिन्यात्नरञ्जापवादायाम् उत्तरोत्तरं बलीयः may not be honoured and a better way is required to resolve sūtra conflicts in totality. Effects such as vipratiṣedha and pūrvavipratiṣedha also need to be included in the SCARE.
References


PAIAS: Pāṇini Aṣṭādhyāyī Interpreter As a Service

SARADA SUSARLA, TILAK M. RAO and SAI SUSARLA

Abstract: It is widely believed that Pāṇini’s Aṣṭādhyāyī is the most accurate grammar and word-generation scheme for a natural language there is. Several researchers attempted to validate this hypothesis by analyzing Aṣṭādhyāyī’s sūtra system from a computational / algorithmic angle. Many have attempted to emulate Aṣṭādhyāyī’s word generation scheme. However, prior work has succeeded in taking only small subsets of the Aṣṭādhyāyī pertaining to specific constructs and manually coding their logic for linguistic analysis.

However, there is another school of thought that Aṣṭādhyāyī itself (along with its associated corrective texts) constitutes a complete, unified, self-describing solution for word generation (kṛt, taddhita), compounding (samāsa) and conjugation (sandhī). In this paper, we describe our ongoing effort to directly compile and interpret Aṣṭādhyāyī’s sūtra corpus (with its associated data sets) to automate its prakṛti-pratyaya-based word transformation methodology, leaving out kārakas. We have created a custom machine-interpretable language in JSON for Aṣṭādhyāyī, a Python-based compiler to automatically convert Aṣṭādhyāyī sūtras into that language, and an interpreter to reproduce Aṣṭādhyāyī’s prakriyā for term definitions, meta-rules and vidhis. Such an interpreter has great value in analyzing the generative capability of Pāṇinian grammar, assessing its completeness or anomalies and the contributions of various commentaries to the original methodology. We avoid manually supplying any data derivable directly from Aṣṭādhyāyī. Unlike existing work that aimed at fast interpretation of rules, we focus initially on fidelity to Aṣṭādhyāyī.

We have started with a well-annotated online Aṣṭādhyāyī resource. We are able to automatically enumerate the character sequences denoted by saṃj nās defined in Aṣṭādhyāyī, and determine which paribhāṣā sūtras apply to which vidhī sūtras. We are in the process of developing a generic rūpa-siddhi engine starting from a prakṛti-pratyaya
sequence. Our service named PAIAS\(^1\) provides programmatic access to \textit{Aṣṭādhyāyī}, its data sets and their interpretation via open RESTful API for third-party tool development.

1 Introduction

There is growing interest and activity in applying computing technology to unearth the knowledge content of India’s heritage literature, especially in Śaṁskṛt language. This has led to several research efforts to produce analysis tools for Śaṁskṛt language content at various levels - text, syntax, semantics and meaning Goyal, Huet, et al. (2012), Oilver Hellwig (2009), Huet (2002), Kulkarni (2016), and Kumar (2012). The word-generating flexibility and modular nature of the Śaṁskṛt grammar makes it at once both simpler and difficult to produce a comprehensive dictionary for the language: simpler because it allows auto-generation of numerous variants of words, and difficult because unbounded nature of Śaṁskṛt vocabulary makes a comprehensive static dictionary impractical. Yet, a dictionary is essential for linguistic analysis of Śaṁskṛt documents. \textit{Pāṇini’s Aṣṭādhyāyī} comes to the rescue for Śaṁskṛt linguistic analysis by offering a procedural basis for word generation and compounding to produce a dynamic, semi-automated dictionary. \textit{Aṣṭādhyāyī} is considered a monumental work in terms of its ability to codify the conventions governing the usage of a natural language into precise, self-contained generative rules. Ever since the advent of computing, researchers have been trying to automate the rule engine of \textit{Aṣṭādhyāyī} to realize its potential. However, due to the sheer size of the rule corpus and its complexity, to date, only specific subsets of its rule base have been digested manually to produce word-generation tools pertaining to specific grammar constructs Goyal, Huet, et al. (2012), Krishna and Goyal (2015), Patel and Katuri (2016), and Scharf and Hyman (2009).

However, this approach limits the tools’ coverage of numerous word forms and hence their usefulness for syntax analysis of the vast Śaṁskṛt corpus. Interpreting \textit{Pāṇini’s Aṣṭādhyāyī} as separate subsets is complex and unnatural due to intricate interdependencies among rules and their triggering conditions. \textit{Pāṇini’s Aṣṭādhyāyī} has a more modular, unified mechanism (\textit{prakriyā})\(^2\) for word generation via rules for joining \textit{prakṛti} (stems) with

\(^{1}\)Pronounced like ‘payas’ meaning milk.

\(^{2}\)In this paper, we use the IAST convention for Sanskrit words.
numerous pratyayas based on the word sense required. Most aspects of the joining mechanism are common across conjugation (sandhi), compounding (samāsa) and new word derivation (e.g., kṛt and taddhita forms). However, commentaries such as Siddhānta Kaumudī (SK) have arranged the rules for the purpose of human understanding of specific grammatical constructs. For the purpose of computational tools, we believe the direct interpretation of Pāṇini’s Aṣṭādhyāyī offers a more natural and automatable approach than SK-based approaches.

With this view, we have taken up the problem of relying solely on Aṣṭādhyāyī and its associated sūtras for deriving all Saṃskṛt grammatical operations of word transformation (or rūpa-siddhi). Our approach is to compile the Aṣṭādhyāyī sūtra text into executable rules automatically (incorporating interpretations made by commentaries), and to minimize the manual coding work to be done per sūtra. We have built a web-based service called PAIAS with a RESTful API for programmatic access to the Aṣṭādhyāyī engine (i.e., the sūtra corpus and its associated data sets) and to enable its execution for word transformation and other purposes. We adopted a service-oriented architecture to cleanly isolate functionality from end-user presentation, so numerous tools and presentation interfaces can evolve for Saṃskṛt grammar employing appropriate programming languages.

In this paper, we describe our ongoing work and its approach, and the specific results obtained so far. In section 2, we set the context by contrasting our approach to relevant earlier work. In section 3, we give an overview of the project’s goals and guiding design principles. In section 4, we describe how we prepared the source Aṣṭādhyāyī for use in PAIAS. In section 5, we explain our methodology for Aṣṭādhyāyī interpretation including the high-level workflow, sūtra compilation scheme and rule interpreter. In section 6, we outline how PAIAS enumerates several entity sets referred throughout Aṣṭādhyāyī. In section 7.3, we describe our methodology to interpret paribhāṣa sūtras. In section 8, we provide details of our implementation and its status. In section 9, we illustrate the operation of the interpreter by showing how our engine automatically expands pratyāhāras. Finally we conclude and outline future work in Section 10.
2 Related Work

Aṣṭādhyāyī and its interpretation for Saṃskṛt grammatical analysis and word synthesis has been studied extensively Goyal, Huet, et al. (2012), Goyal, Kulkarni, and Behera (2008), Krishna and Goyal (2015), Patel and Katuri (2016), Satuluri and Kulkarni (2013), Scharf and Hyman (2009), and Subbanna and Varakhedi (2010). For the purpose of this paper, we assume the reader is familiar with Pāṇini’s Aṣṭādhyāyī and its various concepts relevant to computational modeling. For a good overview of those concepts, the reader is referred to earlier publications Goyal, Kulkarni, and Behera (2008) and Petersen and Oliver Hellwig (2016). In their Aṣṭādhyāyī 2.0 project, Petersen and Oliver Hellwig (2016) have developed a richly annotated electronic representation of Aṣṭādhyāyī that makes it amenable to research and machine-processing. We have achieved a similar objective via manual splitting of sandhis and word-separation within compounds, and by developing a custom vibhakti analyzer for detecting word recurrence across vibhakti and vacana variations.

Petersen and Soubusta (2013) have developed a digital edition of the Aṣṭādhyāyī. They have created a relational database schema and web-interface to support custom views and sophisticated queries. We opted for a hierarchical key-value structure (JSON) to represent Aṣṭādhyāyī as it enables a more natural navigational exploration of the text unlike a relational model. We feel that the size of the Aṣṭādhyāyī is small enough to fit in DRAM of modern computers making efficiency benefits of the relational model less relevant. We used a document database (MongoDB) due to the schema flexibility and extensibility it offers along with powerful navigational queries. Scharf (2016) developed perhaps the most comprehensive formalization to date of Pāṇini’s grammar system including the Aṣṭādhyāyī in XML format with the express purpose of assisting the development of automated interpreters. In his formalization, the rules are manually encoded in a pre-interpreted form via spelling out the conditions, contexts, and actions of each rule. In contrast, our attempt is to derive those from the rule’s text itself. Scharf’s encoded rule information enables validation of our rule interpretation. We could also leverage its other databases that form part of Pāṇini’s grammar ecosystem.

The first step to interpret the Aṣṭādhyāyī is to understand the terms and metarules that Pāṇini defines in the text itself. T. Ajotikar, A. Ajotikar, and Scharf (2015) explains some of Pāṇini’s techniques that an interpreter
needs to incorporate, and illustrated how Scharf (2016) captures them. Unlike earlier efforts at processing Aṣṭādhyāyī that have manually enumerated the terms and their definitions including pratyāhāras Mishra (2008), our approach is to extract them from the text itself automatically.

Several earlier efforts attempted to highlight and emulate various techniques used in Aṣṭādhyāyī for specific grammatical purposes. They typically select a particular subset of Aṣṭādhyāyī’s engine and code its semantics manually to reproduce a specific prakriyā. For brevity, we only discuss the most recent work that comes close to ours. Krishna and Goyal (2015) have built an object-oriented class hierarchy to mimic the inheritance structure of Aṣṭādhyāyī rules. They have demonstrated this approach for generating derivative nouns. Our goal and hence approach differ in two ways, namely, to interpret Aṣṭādhyāyī sūtras faithfully as opposed to achieving specific noun and verb forms, and to mechanize the process of converting sūtras into executable code to the extent possible. However, the learnings and insights from earlier work on interpreting Aṣṭādhyāyī Mishra (2008) will apply to our work as well, and hence can be incorporated into our engine.

Patel and Katuri (2016) have built a subanta generator that imitates the method given by siddhānta kaumudī. Their unique contribution is a way to order the sūtras for more efficient interpretation. However, they also encode the semantic meaning of individual sūtras manually, and do not suggest a method to mechanize sūtra interpretation from its text directly. Satuluri and Kulkarni (2013) have attempted to generate samāsa compounds by emulating the relevant subset of Aṣṭādhyāyī. Subbanna and Varakhedi (2010) have emulated the exception model used in Aṣṭādhyāyī. For this, they have emulated a small subset of its sūtras relevant to that aspect.

3 Design Goals and Scope

The objective of our project is to develop a working interpreter for Pāṇini’s Aṣṭādhyāyī that emulates its methodology faithfully by mechanizing the interpretation of sūtra text as much as possible. To guide our design, we set the following principles:

Fidelity: We focus on reproducing the prakriyā of Aṣṭādhyāyī sūtra corpus (by taking the semantic adjustments from relevant vyākhyānas as appropriate). We do not focus on optimizing the interpretation engine for speedy execution as of now.
Reuse: We would like to provide a powerful query interface to the *Aṣṭādhyāyī* and its data sets to enable sophisticated analytics and learning aids.

Extensibility: We would also like to promote the development of an extensible and interoperable framework for *Aṣṭādhyāyī* by providing a programmatic interface to its interpreter engine. This framework should support plugging in functionality developed by third parties in multiple programming languages and methodologies.

The specific contributions of this paper include

- A programmatic interface to *Aṣṭādhyāyī* with a powerful query language for sophisticated search,
- A mechanism to automatically extract definitions of *saṃj nās* (both statically and dynamically defined) by interpreting their *sūtra* text,
- A machine-processable language and its interpreter to transform the bulk of *Aṣṭādhyāyī sūtra* text into executable code, and a mechanism of interpretation that tracks word transformation state persistently, and
- An extensible framework that supports interoperability among techniques for *Aṣṭādhyāyī sūtra* interpretation developed by multiple researchers to accelerate tool development.

4 Preparing the *Aṣṭādhyāyī* for machine-processing

We have started with a well-annotated and curated online resource for *Aṣṭādhyāyī* Sarada Susarla and Sai Susarla (2012) available as a spreadsheet due to its amenability to augmentation and scripted manipulation. Table 1 outlines its schema. The spreadsheet assigns each *sūtra* a canonical ID string in the format APSSS (e.g., 11076 to denote the 76th *sūtra* in 1st *pāda* of 1st *adhyāya*). To enable machine-processing, each *sūtra* is provided with its words split by *sandhī* and the individual words in a *samāsa* separated by hyphens and tagged with simple morphological attributes such as type (*subanta*, *tiṅanta* or *avyaya*), *vibhakti* and *vacana* to enable auto-extraction. The *adhikāra sūtras* are also explicitly tagged with their influence given as
a sūtra range. For each sūtra, the padas that are inherited from earlier sūtras through anuvṛtti are listed along with their source sūtra id and vibhakti modification in the anuvṛtta form if any.

We auto-convert this spreadsheet into a JSON JSON (2000) dictionary and use it as the basis for the PAIAS service. Table 2 shows an example JSON description of sūtra 1.2.10 with all the abovementioned features illustrated.

```json
{
    "Adhyaaya" : "Adhyaaya # adhyaAyaH",
    "Paada" : "Paada # pAdaH",
    "sutra_num" : "sutra_num sUt. saM.",
    "sutra_krama" : "sutra_krama sUt. kra. saM",
    "Akaaraadi_krama" : "Akaaraadi_krama akArAdi kra. saM",
    "Kaumudi_krama" : "Kaumudi_krama kaumudI kra. saM",
    "sutra_id" : "sutra_id pUrNa sUt. saM.",
    "sutra_type" : "sutra_type sutralakShaNam",
    "Term" : "Term saMj~nA",
    "Metarule" : "Metarule paribhAShA",
    "Special_case" : "Special_case atideshaH",
    "Influence" : "Influence adhikAraH",
    "Commentary" : "Commentary vyAkhyAnam",
    "sutra_text" : "sutra_text sutram",
    "PadacCheda" : "PadacCheda padchChedaH",
    "SamasacCheda" : "SamasacCheda samAsachChedaH",
    "Anuvrtti" : "Anuvrtti pada sutra #
                anuvRRitti–padam sutra–sa~NkhyA",
    "PadacCheda_notes" : "PadacCheda_notes"
}
```

Table 1

Aṣṭādhyāyī Database Schema
"12010" : {
  "Adhyaaya" : 1,
  "Paada" : 2, "sutra_num" : 10,
  "sutra_krama" : 12010, "Akaaraadi_krama" : 3913,
  "Kaumudi_krama" : 2613,
  "sutra_id" : "1.2.10",
  "sutra_type" : [ "atideshaH" ], "Commentary" : "...",
  "sutra_text" : "halantAchcha |",
  "PadacCheda" : [
    { "pada" : "halantAt", "pada_split" : "hal–antAt",
      "type" : "subanta", "vachana" : 1,
      "vibhakti" : 5 },
    { "pada" : "cha", "type" : "avyaya",
      "vachana" : 0, "vibhakti" : 0 }
  ],
  "Anuvrtti" : [
    { "sutra": 12005, "padas": [ "kit"] },
    { "sutra": 12008, "padas": [ "san" ] },
    { "sutra": 12009, "padas": [ "ikaH", "jhal"] }
  ]
},

Table 2
Aṣṭādhyāyī Database Schema
5 Aṣṭādhyāyī Interpreter: High-level Workflow

The input to our Aṣṭādhyāyī engine is a sequence of tagged lexemes that we call pada descriptions or pada_descs, and its output is one or more alternate sequences of tagged lexemes denoting possible word transformations. A pada_desc is a dictionary of tag-value pairs in JSON format. The tag values can be user-supplied (in case of human-assisted analysis), system-inferred or user-endorsed. Table 2 shows a sūtra description where the padacCheda section represents the sūtra as a sequence of pada_descs. An example tag is a pada ‘type’ such as subanta, tiṅanta, nīpāta, avyaya, pratyaya, sanj nā etc. Each application of an Aṣṭādhyāyī sūtra, referred to in this paper as ‘prakriyā’, modifies the input pada_desc sequence by adding/editing/removing pada_descs to denote word-splitting, morphing or merging operations based on the semantics of the sūtra.

For instance, when applying the sanj nā sūtra for the sanj nā ‘it’, we tag a given input word with a tag called ‘it_varnas’ whose value is the offset of the ‘it’ varṇas found in the word. Such tags can also be used to store intermediary states of grammar transformations for reference by subsequent operations. This persistent tracking of the transformation state of words offers the power required for interpreting Aṣṭādhyāyī sūtras faithfully. The need for such facility to carry over internal state from one sūtra to another has been identified by Patel and Katuri (2016) for their subanta generator tool.

In order to transform tagged lexemes, the first step is to identify the occurrence of pre-determined patterns in input lexemes which are denoted by explicit terms (sanj nās) in Aṣṭādhyāyī sūtras. Instead of handcoding those pattern definitions into the interpreter, our approach is to automatically extract them from the sūtras themselves and interpret them at prakriyā time. To accomplish this, we have devised a machine-processable representation scheme for various sūtras, which we elucidate in Section 6. Likewise, Aṣṭādhyāyī provides a set of 23 paribhāṣā sūtras or metarules (augmented with approx. 100 more metarules in paribhāṣendu-śekhara treatise). The purpose of these metarules is to modify the operation of the vidhi sūtras. In Section 7.3, we describe how we manually encode metarules as (condition, action) pairs such that we can mechanically determine which paribhāṣās apply to a given Aṣṭādhyāyī sūtra. Since paribhāṣās operate on vidhi sūtra texts, their applicability can be pre-determined a priori instead of at prakriyā time.
At a high-level, our approach to Aṣṭādhyāyī interpretation involves the following manual steps:

- Splitting of sandhis and samāsa in the sūtra text to facilitate detection of word recurrences.
- Enumerating the anuvṛtta padas of each sūtra (from earlier sūtras).
- Coding of each of the 23 paribhāṣā sūtras into condition-action pairs.
- Preparation of a vibhakti suffix table that covers subantas of Aṣṭādhyāyī for use in morphological analysis of sūtra words.
- Coding of custom functions to interpret the meaning of some technical words used in Aṣṭādhyāyī but not defined therein (e.g., adarśanam, ādiḥ, antyam, etc.).
- Adding special case interpretation of the sūtra ‘halantyam’ as ‘hali antyam’ to break the cyclic dependency for pratyāhāra generation (as explained in Section 9).

In the next section, we outline the preprocessing steps needed for Aṣṭādhyāyī interpretation.

5.1 Preparing the Aṣṭādhyāyī Interpreter

To prepare the Aṣṭādhyāyī engine for rule interpretation, we automatically preprocess the Aṣṭādhyāyī database as follows.

1. We first perform morphological analysis of each word of every sūtra to extract its prātipadikam. This is required to identify recurrence of a word in the Aṣṭādhyāyī regardless of vibhakti and vacana variations. We describe this step in Section 5.2.

2. For each sūtra, we generate a canonical sūtra text that we refer to as its ‘mahāvākya’ as follows. We expand the sūtra’s text to include all anuvṛtta-padas inherited from earlier sūtras. We represent a mahāvākya as a list of pada descriptions, each with its morphological analysis output.

3. We auto-extract the definitions of all terms (saṃj nās) used in the Aṣṭādhyāyī. These come in different forms and need to be handled differently. We describe this step in Section 6.
4. We compile samjānā and vidhi sūtras into rules to be interpreted at prakrīyā time.

5. We determine the vidhi sūtras where each of the paribhasā sūtras apply, by checking their preconditions. Then we modify the vidhi sūtras.

6. Finally, we create an optimized condition hierarchy for rule-checking by factoring the preconditions for all the Aṣṭādhyāyī sūtras into a decision tree. This step is still work in progress and is out of the scope of this paper.

5.2 Morphological Analysis of Aṣṭādhyāyī Words

To detect recurrences of a sūtra word at different locations in Aṣṭādhyāyī (e.g. through anuvṛtti or embedded references) despite their vibhakti and vacana variations, we need the prātipadikam of each word. Since most Aṣṭādhyāyī words are subantas specific to the treatise and not found in typical Saṃskṛt dictionaries, we developed a simple suffix-based vibhakti analyzer for this purpose. Since our Aṣṭādhyāyī spreadsheet already has words tagged by their vibhakti and vacana, our vibhakti analyzer takes them as hints and finds possible matches in predefined vibhakti tables based on various common word-endings. Once a match is found, it emits an analysis that includes possible alternative prātipadikas along with their liṅga and word-ending. We store the subanta analysis for each Aṣṭādhyāyī word and store it in the padacCheda section of the sūtra JSON entry for ready reference.

With this technique, we are able to determine the prātipadikam accurately for all the technical terms used in Aṣṭādhyāyī and use it for detecting word recurrences. Though the tool generated multiple options for liṅga, that ambiguity doesn’t hurt for our purpose of detecting word recurrence since the prātipadikam is unique.

Then we extract term (saṃjānā) definitions from the saṃjānā sūtras as described in Section 6.

6 Extracting Saṃjñā Definitions from Aṣṭādhyāyī

Aṣṭādhyāyī’s word transformation method consists of detecting pre-defined patterns denoted by saṃjānās or terms and performing associated transformations. These terms denote either a set of explicitly enumerated member elements or conditional expressions to be dynamically checked at prakrīyā
time. Hence during preprocessing stage, we create a term definition database where each term is defined as a list of member elements or as a compiled rule. The Aṣṭādhyāyī itself defines four types of terms (in increasing order of extraction complexity):

1. Terms defined in an *adhikāra* cum *saṃj nā sutra* denoting a set of elements enumerated explicitly in subsequent *vidhi sutras* (e.g., *pratyaya*, *taddhita*, *nipāta*). The term itself becomes an *anuvṛtta pada* in all *vidhi sutras* in its *adhikāra*. Moreover, those *sūtras* refer to both the term and its member elements in *prathamā vibhakti*. Hence, to extract the definition, we pick *prathamā vibhakti* terms excluding (a) terms defined in *saṃj nā sutras* of Aṣṭādhyāyī and (b) a manually prohibited list of words meant to convey colloquial meaning. With this method, we were able to successfully extract all the *pratyayas*, *taddhitas*, *nipātas*, *samāsas* from Aṣṭādhyāyī automatically, and verify their authenticity with those identified in Dikshita (2010).

2. Terms with explicit name defined in *saṃj nā sutras* denoting a set of elements enumerated explicitly (e.g., *vṛddhi*). In this case, the elements are listed in *prathamā vibhakti* and hence can be extracted directly from the *sūtra*.

3. Terms with an explicit name defined in *saṃj nā sutras*, and denoting a condition to be computed at *prayoga* time (e.g., ‘it’).

4. Terms whose name (*saṃj nā*) and its members (*saṃj ni*) are both dynamically computed quantities (e.g., *pratyāhāras* such as ‘ac’ and ‘hal’)

The last two variants require interpreting the *sūtra* text in different ways as described in Section 7. During Aṣṭādhyāyī *prakriyā*, when an input *pada_desc* needs to be checked for match with a *saṃj nā*, we have two options. If the *saṃj nā* is represented as a list of member elements, then all *padas* in the *pada_desc* that appear as members of a list will be annotated with the *saṃj nā* name. For instance, when checking the word ‘rāma’ against ‘gunā’ *saṃj nā*, the *pada_desc* of the ‘rāma’ word will be augmented with a property named ‘gunā’ whose value is the index of the last ‘a’ alphabet in the ‘rāma’ word, i.e, 3.
7 Compiling Rules from sūtras

In this section, we describe a mechanism we have devised for transforming Aṣṭādhyāyī sūtras into machine-interpretable rules. This is a core contribution of our work as it enables direct interpretation of sūtras. We have implemented this mechanism for saṃj nā and paribhāṣā sūtras first because (i) they form a crucial prerequisite to the rest of the engine, and (ii) because they have not been studied by earlier work as systematically as the interpretation of vidhi sūtras. Moreover Aṣṭādhyāyī’s paribhāṣā sūtras state the mechanism for interpreting vidhi sūtras explicitly. Our vidhi sūtra interpretation is a work in progress and will not be discussed further.

Our sūtra interpretation scheme is based on some grammatical conventions we have observed in the sūtra text. First, the bulk of Aṣṭādhyāyī sūtras employ subanta padas and avyayas, and use tiñanta padas sparingly. Second, saptamī vibhakti is used to indicate the context/condition in which a sūtra applies. Third, each sūtra word either denotes a saṃj nā (or its negation), a predefined function (e.g. ādiḥ, antyam, etc), a set of terms or characters (e.g., cuṭū), or joining avyayas (e.g., saha, ca, vā etc.). Finally, whenever multiple words of the same vibhakti occur, one of them is a višeṣya and others are its višeṣaṇas.

Hence we compile each Aṣṭādhyāyī sūtra into a hierarchical expression via specially defined operators, called a rule. The rule is either atomic, i.e., a pada_desc describing a sūtra word, or composite, coded as a JSON list. If pada_desc, it is interpreted via a special operator called INTERPRET described below. If list, its first element is a predefined operator, and the rest are its arguments. The arguments can in turn be pada_descs or sub-rules.

7.1 Special Operators

This section describes several special operators that we have defined to form rules.

INTERPRET: This operator interprets a single sūtra word on the input tagged lexemes. It checks whether the pattern it denotes (e.g., ‘it’) applies to any of the input lexemes (e.g., ‘hal’). If the sūtra word is a saṃj nā (e.g., ‘it’), the interpreter interprets its rule recursively and tags the lexemes with the result (e.g., locations of ‘it’ varṇas). If the sūtra word is one of a predefined set of words with special meaning, the interpreter invokes its detector function. For instance, ādiḥ of the
lexeme ‘hal’ is ‘h’. Otherwise, the sūtra word denotes a set of terms or characters, in which case the interpreter returns whether the lexeme is a member of that set. For instance, if the sūtra word ‘cuṭū’ is interpreted against input lexeme ‘c’, it returns True because cuṭū denotes consonants in the ca-varga and ṭa-varga, i.e., {ca, cha, ja, jha, na, ṭa, ṭha, a, ḍha, ṇa}. If the sūtra word is a negation such as ataddhīta or apratīyayāḥ, INTERPRET applies the negation before returning the result.

The following conjunct operators are used to compose larger rules:

PIPE: This operator takes a sequence of rules and invokes them by feeding the output of a rule invocation as input to the subsequent rule. The pipe exits when one of the stages return empty, and returns the output of the last rule. This is used to process all sūtra padas of the same vibhakti. For instance, when interpreting the pipe [‘PIPE’, ‘ādīḥ’, ‘cuṭū’] against the lexeme ‘hal’, the output of ‘ādīḥ’ namely ‘h’ is compared against ‘cuṭū’ membership, which returns None.

IF: This operator takes a list of rules. If all of them evaluate to something other than None, it returns the input tagged lexeme set as is, otherwise None. This is used to encapsulate saptamī vibhakti padas that indicate the enabling context for a sūtra to apply (e.g., upadeśe). It is also used to encapsulate a ṣaṣṭhī vibhakti padam in a samj nā sūtra which indicates the samj ni (definition of a samj nā).

PAIR: This operator represents a pair of elements mentioned in a sūtra along with the avyaya ‘saha’. The prathamā vibhakti pada sequence describes the first element and the tṛtiyā vibhakti pada sequence denotes the last element. An example is shown in Figure 3 for the sūtra ‘ādirantyena sahetā’. If the pair denotes a sequence, then it describes the first and last elements.

GEN_SAMJNA: This operator handles a samj nā defined as a computed expression such as ‘ak’, ‘hal’, ‘sup’ etc. It matches the input tagged lexeme against the rule for the samj nā. Upon a match, it invokes the rule for the ‘samj ni’ by passing the samj nā as a parameter. For instance, the sūtra ‘ādirantyena sahetā’ gets compiled into the following rule:
• [GEN_SAMJNA, {‘saṃj nī’ : None, ‘saṃj nā’ : [PAIR, ‘ādiḥ’, [PIPE, ‘antyam’, ‘it’]]}]

Since there is no explicit saṃj nī in this sūtra, we apply a special pratyāhāra expander function to generate the character sequence from the input pair. Figure 3 shows the hierarchical representation of the sūtra text that leads to the above rule.

PROHIBIT: This function prohibits applying a sūtra under a matched sub-condition. It is not the same as negation of a match condition. This is used to process the sūtra word ‘na’ in a sūtra. For instance, when processing the ‘it’ saṃj nā sūtra ‘na vibhaktau tusmāh’, as shown in Figure 1, this function removes any ‘it’ varṇa tagging done while processing its sub-conditions denoted by the words ‘hal’, ‘antyam’ and ‘tusmāh’.

![Rule Hierarchy for sūtra ‘na vibhaktau tusmāh’](image)

**Figure 1**

*Rule Hierarchy for sūtra ‘na vibhaktau tusmāh’.*

7.2 Compiling Rules from Saṃj nā sūtras

Saṃj nā sūtras come in two flavors:

1. those that explicitly list a term and its definition in prathamā vibhakti with some other conditions e.g., (upadeśe pratyayasya ādiḥ it) cuṭū, and
Figure 2
Rule Hierarchy for sūtra ‘cuṭū’.

Figure 3
Rule Hierarchy for sūtra ‘ādirantyena sahetā’.
2. those that describe the samj nā name as a computed expression and its denoted items in ṣaṣṭhīi vibhakti, e.g., ādiḥ antyena itā saha (svasya rūpasya).

In the above representation of the sūtra texts, we denote words that are inherited by anuvṛtti in parentheses.

Figure 2 shows a tree representation for a sūtra of the first flavor. A samj nā sūtra has three components: the term denoted by the edge labeled ‘kA’, its definition denoted by ‘kasya’, and the context in which the definition applies, denoted by ‘kutra’. saptaṁ vibhakti padas in the sūtra denote the context. The samj nā term, if explicitly present in the sūtra will be in prathamā vibhakti with its defining words also in prathamā. In that case, an executable version of the sūtra is a representation of the tree as a hierarchical list. All words in the same vibhakti in the sūtra have viśeṣaṇa-viśeṣya relation.

Figure 3 shows the tree representation for a sūtra of the second flavor. In this, the ṣaṣṭhīi vibhakti word should be interpreted as a filter or qualifier for the prathamā vibhakti words, not as the samj ni (the definition). This sūtra also has tṛtīyā vibhakti padas joined by ‘saha’, which can be interpreted as sequence generation operator. This operator takes prathamā vibhakti padas to indicate start of the sequence and tṛtīyā vibhakti padas to indicate end of sequence.

Figure 4 shows another sūtra of the second flavor, where the samj nā and samj ni definition are both parameterized. It has a pada that is a negation of pratyaya. Matching this sūtra requires a pre-defined function that checks if given word is a pratyaya. The samj ni in this case is the set of savarṇas of x.

7.3 Interpreting Paribhāṣā sūtras

A paribhāṣā sūtra describes how to interpret sūtras whose text matches a given condition. It can be represented as a set of actions guarded by conditions. It is applied to transform vidhi sūtras prior to compiling them into rules. The condition indicates the sūtra to which the paribhāṣā applies, expressed in terms of properties of the words in the sūtra text. The actions indicate how the matching sūtra should be transformed prior to interpretation.

For instance, consider the sūtra ‘ādyantau ṭakitau’. It describes that if a vidhi sūtra contains ‘ṭit’ or ‘kit’ pada (i.e., which has varṇa ‘ṭ’ or ‘k’ as
‘it’), then the sūtra should be expanded to add extra words ‘ṣaṣṭhyantasya ādiḥ’ or ‘ṣaṣṭhyantasya antaḥ’ respectively to the sūtra text. We express this logic in our interpreter by coding the paribhāṣā as shown in Algorithm 1. Here, the condition is expressed as a rule-matching query with new operators SAMJNA, PRATYAYA, IT_ENDING, AND and NOT. It applies if a vidhi sūtra has an individual pada (in its padacCheda) which is not a saṃj nā or pratyaya word, but has ‘ṭ’ as its ‘it’ varṇa. In that case, the sūtra’s text must be augmented with two additional words ‘ṣaṣṭhyantasya ādiḥ’. That matching vidhi sūtra will then be compiled into a rule and then interpreted during word transformation prakriyā time. Similarly, if the sūtra word has ‘k’ as its ‘it’ varṇa, then the additional words will be ‘ṣaṣṭhyantasya antaḥ’.

As another example, the paribhāṣā sūtra ‘midaco’ntyāt paraḥ’ has the following effect. If a vidhi sūtra has ‘m’ as ‘it’ (other than in a pratyaya or saṃj nā word), then the words ‘ṣaṣṭhyantasya antyāt acaḥ paraḥ’ should be added to the sūtra text.

We manually define the condition action pairs for each of the 23 paribhāṣā sūtras as shown in Algorithm 1. At initiation time, the Aṣṭādhyāyī engine checks these conditions on each of the vidhi sūtras of Aṣṭādhyāyī and transforms them accordingly prior to the rule compilation step.

In our current implementation, we have handcoded the condition-action pairs for about half of the paribhāṣās of Aṣṭādhyāyī, and are able to successfully identify the sūtras to which they apply. This is because of our ability to identify the various listable saṃj nās, which are needed in formulating the conditions. However, processing of vidhi sūtras is future work.
Algorithm 1 Codifying \textit{paribhāṣā sūtra} ‘ādayantau ṭakitau’.

\begin{verbatim}
paribhasa_defs = {
    str(11046) : [
        {'cond' : {
            'PadacCheda' : [AND, [[NOT, SAMJNA], [NOT, PRATYAYA],
                                [IT_ENDING, {'varna' : 'T'}]]],
            'sutra_type' : ['vidhiH']
        },
        {'action' : [
            sutra_add_pada, {'pada' : 'ShaShThyantasya',
                              'vibhakti' : 6, 'type' : 'subanta'},
            sutra_add_pada, {'pada' : 'AdiH',
                              'vibhakti' : 1, 'type' : 'subanta'}
        ]
    },
    {  
        'cond' : {
            'PadacCheda' : [AND, [[NOT, SAMJNA], [NOT, PRATYAYA],
                                [IT_ENDING, {'varna' : 'k'}]]],
            'sutra_type' : ['vidhiH']
        },
        {'action' : [
            sutra_add_pada, {'pada' : 'ShaShThyantasya',
                              'vibhakti' : 6, 'type' : 'subanta'},
            sutra_add_pada, {'pada' : 'antaH',
                              'vibhakti' : 1, 'type' : 'subanta'}
        ]
    }
}
\end{verbatim}
8 Implementation

We have implemented PAIAS as a Python library and a Flask web microservice that provides RESTful API access to its functionality. The API-based interface provides a flexible, reliable and reusable foundation for open collaborative development of higher-level tools and user interfaces in multiple programming languages to accelerate research on Aṣṭādhyāyī, while ensuring interoperability of those tools. The code is available on GitHub at https://github.com/vedavaapi/ashtadhyayi and will soon be available as a pip installable module.

The module comes bundled with the Aṣṭādhyāyī spreadsheet along with dhātu pātha and other associated data sets. Upon first invocation after a clean install, the Aṣṭādhyāyī module computes mahāvākyas for all sūtras, compiles sūtras into machine-executable rules, builds saṃjñāna definitions, extracts listable terms such as Pratyayas etc, and transforms vidhi sūtras by applying the matching paribhāṣā sūtras. It then stores all this derived state persistently in JSON format in a MongoDB database. This enables fast access to the Aṣṭādhyāyī engine subsequently. Our current implementation does not handle the transformation and interpretation of vidhi sūtras yet.

Figure 2 shows an example Python script using the Aṣṭādhyāyī library. We have also devised a powerful query interface to Aṣṭādhyāyī for sophisticated search. Figure 3 shows a Python script to find unique words that occur in the Aṣṭādhyāyī grouped by vibhakti. The query condition can be specified as a JSON dictionary supporting a hierarchical specification of desired attributes as shown in this example.
Algorithm 2 Example usage of \textit{Aṣṭādhyāyī} Service.

```python
from ashtadhyayi.utils import *
from ashtadhyayi import *

def a():
    return ashtadhyayi()

# Provide mahaavaakya of given sutra as individual words
def mahavakya(sutra_id):
    s = a().sutra(sutra_id)
    return s['mahavakya_padacCheda']

# Show all vidhi sutras where given paribhasha sutra applies
def paribhasha(sutra_id):
    p = get_paribhasha(sutra_id)
    if not p:
        print('Error: Paribhasha description not found for', sutra_id)
        return []
    matches = []
    for s_id in p.matching_sutras():
        s = a().sutra(s_id)
        out = dict((k, s[k]) for k in ('sutra_krama', 'sutra_text', 'sutra_type'))
        matches.append(out)
    return matches

# Return praatipadikam of given pada taking vibhakti and vachana hints.
def praatipadika(pada, vibhakti=1, vachana=1):
    pada = sanscript.transliterate(pada, sanscript.SLP1, sanscript.DEVANAGARI)
    return Subanta.analyze({'pada': pada, 'vibhakti': vibhakti, 'vachana': vachana})
```
Algorithm 3 Example script to extract unique words of various vibhaktis in Aṣṭādhyāyī.

```python
from ashtadhyayi.cmdline import *

a = ashtadhyayi()
myfilter = {'PadacCheda': {'vibhakti': 1}}
result = {}
for v in [0, 1, 2, 3, 4, 5, 6, 7]:
    myfilter['PadacCheda']['vibhakti'] = v
    v_padas = []
    for s_id in a.sutras(myfilter):
        s = a.sutra(s_id)
        for p in s['PadacCheda']:
            if 'vibhakti' not in p:
                continue
            if p['vibhakti'] != v:
                continue
            v_padas.append(p['pada'])
    result[v] = sorted(set(v_padas))
print_dict(result)
```
9 Evaluation: Putting it all together

In this section, we illustrate the automated operation of the PAIAS engine via by showing the expansion of a pratyāhāra ‘ac’ into its denoted varṇas. pratyāhāra expansion is an essential step to enable the rest of Aṣṭādhyāyī interpretation.

<table>
<thead>
<tr>
<th>sūtra # (APSSS)</th>
<th>Mahāvākya Representation</th>
<th>Generated rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>13002</td>
<td>it = upadeśe(7) ac(1) anunāsikah(1)</td>
<td>[PIPE, [IF, “upadeśa”], [PIPE, “ac”, “anunāsikah”]]</td>
</tr>
<tr>
<td>13003.1</td>
<td>it = upadeśe(7) hal(7) antyam(1)</td>
<td>[PIPE, [IF, “upadeśa”, “hal”], [PIPE, “antyam”]]</td>
</tr>
<tr>
<td>13003.2</td>
<td>it = upadeśe(7) hal(1) antyam(1)</td>
<td>[PIPE, [IF, “upadeśa”], [PIPE, “hal”, “antyam”]]</td>
</tr>
<tr>
<td>13004</td>
<td>it = upadeśe(7) na(0) vibhaktau(7) tusmāḥ(1)</td>
<td>[PIPE, [IF, “upadeśa”, “vibhakti”], [PROHIBIT, “tu-s-ma”]]</td>
</tr>
<tr>
<td>13005</td>
<td>it = upadeśe(7) ādiḥ(1) niṭudavah(1)</td>
<td>[PIPE, [IF, “upadeśa”], [PIPE, “ādiḥ”, “ni-ṭu-du”]]</td>
</tr>
<tr>
<td>11071</td>
<td>pratyāhāra = svasya(6) rūpasya(6) ādiḥ(1) antyena(3) saha(0) itā(3)</td>
<td>[GEN_SAMJNA, {‘śamj ni’ : None, ‘śamj nā’ : [PAIR, ‘ādiḥ’, [PIPE, ‘antyam’, ‘it’ ] } } ]</td>
</tr>
<tr>
<td>11060</td>
<td>lopa = iti(0) adarśanam(1)</td>
<td>[PIPE, “adarśanam”]</td>
</tr>
</tbody>
</table>

Table 3

Mahāvākya representations produced by PAIAS engine from specific saṃj nā sūtras relevant to pratyāhāra expansion.

Our engine accomplishes ‘ac’ expansion as follows. First, it compiles all saṃj nā sūtras into mahāvākyas and then into machine-interpretable rules.
Table 3 shows the mahāvākya representations and corresponding rules generated by our engine for specific sūtras relevant to pratyāhāra expansion, namely ‘it’ and dynamically computed saṃj nā names. The engine maintains a terms_db database that caches the set of lexemes denoted by each saṃj nā. At start time, the engine resets this database and populates it with the lexemes of all saṃj nās that are listed explicitly in Aṣṭādhyāyī. During pada_desc tagging, whenever the occurrence of a saṃj nā needs to be detected, the engine checks the terms_db cache first before attempting to interpret the saṃj nā’s rules.

9.1 Expansion of pratyāhāra ‘hal’

Pratyāhāras are computed saṃj nā names. To expand them, the engine should be able to interpret the saṃj nā sūtras for ‘it’, especially, the sūtra ‘halantyam’. However, to break its cyclic dependency on the expansion of pratyāhāra ‘hal’, this sūtra should be interpreted twice - first as a samāsa with vigraha ‘hali antyam’, where ‘hali’ is the saptamī vibhakti form of the māheśvara sūtra ‘hal’, and second as ‘hal antyam’. As a result of the first interpretation (sūtra 13003.1), the engine adds the varṇa ‘l’ as the definition of the ‘it’ saṃj nā in terms_db cache. During the second interpretation of ‘it’ as ‘upadeśe hal antyam’, the engine recursively checks if ‘hal’ is a saṃj nā. This in turn matches with sūtra 11071 because ‘l’ in ‘hal’ gets tagged as ‘it_varṇa’. Hence ‘hal’ gets detected as a computed saṃj nā, which denotes the set of varṇas from ‘ādi’ of ‘hal’ i.e., ‘h’ upto the but not including the last ‘l’ in ‘upadeśa’ i.e., the māheśvara sūtra character sequence. Hence the saṃj nā ‘hal’ and its denoted character sequence i.e., all consonants of the Śāṃskṛt alphabet get added to the terms_db cache. When unwinding the recursion back to continue the second interpretation of sūtra 13003, this time, the ending hal varṇa in each māheśvara sūtra gets an ‘it_varṇa’ tag.

9.2 Expansion of pratyāhāra ‘ac’

Next, when trying to interpret the input lexeme ‘ac’, the engine looks to tag the lexeme’s constituent parts by matching them with the definitions of known saṃj nās. This time, the sūtra 13003.2 applies, causing the ‘c’ to be tagged as an ‘it_varṇa’ because ‘c’ is a member of the ‘hal’ set in the terms_db cache. Since there is no explicit saṃj nā called ‘ac’, the engine
checks to see if ‘ac’ is a dynamically computed saṃj nā name by applying sūtra 11071 ‘ādiḥ antyena saha itā’.

This time, when computing the varṇa set as part of sūtra 11071, the last ‘ha’ varṇa in each māheśvara sūtra needs to be suppressed. To accomplish this, we had to manually code the interpretation of a single vidhi sūtra ‘tasya lopah’. To do so, we had to manually rewrite it as ‘itā lopaḥ’ because our engine does not yet have the logic to interpret vidhi sūtras automatically. The engine reduces the definition of ‘lopah’ to be ‘adarśanam’ from sūtra 11060. We manually wrote a function to interpret the Aṣṭādhyāyī word ‘adarśanam’ to suppress the emission of its referrent lexeme - here the one with the ‘it_varṇa’ tag.

Thus the engine is able to generate the varṇa sequence for ‘ac’ pratyāhāra as ‘a i u e o ai au’. The terms_db cache serves two purposes: i) to break infinite recursions in expansion of saṃj nā definitions that are possible in Aṣṭādhyāyī, and ii) to speedup subsequent processing of a saṃj nā once it has been expanded.

10 Future Directions

We recognize that generating an automated interpretation engine for Aṣṭādhyāyī is a complex and long-term task due to the need to validate and adjust the methodology manually, and the thousands of sūtras involved. However, our attempt is to rely on the precision of Aṣṭādhyāyī’s exposition to mechanise large parts of the work. Our second objective is to provide a robust foundational platform so multiple researchers can work collaboratively and leverage each other’s innovations to accelerate the task. To this end, we would like to work closely with other researchers to incorporate existing approaches to Aṣṭādhyāyī interpretation and its validation. This is especially true for vidhi sūtras which constitute the bulk of Aṣṭādhyāyī.

We hope that our programmatic interface to Aṣṭādhyāyī and its semantic functionality enables interoperable applications and deeper exploration of the grammatical structure of Saṃskṛt literature by the larger computer science community. Certain directions include data-driven analysis of the relative usage of Saṃskṛt grammar constructs in Saṃskṛt literature, vocabulary and its evolution over time, kāraka analysis via a mix of data-driven and first-principles approaches. A robust grammar engine provides a sound
basis for such projects. Another area of future research would be to explore the engine’s applicability for modeling other natural languages.

11 Conclusion

In this paper, we have presented a programmatic interface to the celebrated Saṃskṛt grammar treatise Aṣṭādhyāyī with the goal to evolve a direct interpreter of its sūtras for Saṃskṛt word generation and transformation in all its variations. Our initial experience indicates that the consistent structure and conventions of Aṣṭādhyāyī’s sūtras make them amenable to mechanized sūtra interpretation with fidelity. However, much more work needs to be done to fully validate the hypothesis. Having a flexible, reusable and extendible interface to Aṣṭādhyāyī provides a sound basis for collaborative research and application development.
References


Yogyatā as an absence of non-congruity

Sanjeev Panchal and Amba Kulkarni

Abstract: Yogyatā or mutual congruity between the meanings of the related word is an important factor in the process of verbal cognition. In this paper, we present the computational modeling of yogyatā for automatic parsing of Sanskrit sentences. Among the several definitions of yogyatā we modeled it as an absence of non-congruity. We discuss the reasons behind our modeling.

Due to lack of any syntactic criterion for viśeṣaṇa (adjectives) in Sanskrit, parsing Sanskrit texts with adjectives resulted in a high number of false positives. Hints from the vyākaraṇa texts helped us in the formulation of a criterion for viśeṣaṇa with syntactic and ontological constraints, which provided us a clue to decide the absence of non-congruity between two words with respect to the adjectival relation. A simple two-way classification of nouns into dravya and guṇa with further sub-classification of guṇas into guṇavacanas was found to be necessary for handling adjectives. The same criterion was also necessary to handle the ambiguities between a kāraka and non-kāraka relations. These criteria together with modeling yogyatā as an absence of non-congruity resulted in 81% improvement in precision.

1 Introduction

Three factors viz. ākāṅkṣā (expectancy), yogyatā (congruity) and sannidhi (proximity) play a crucial role in the process of śābdabodha (verbal cognition). These factors have been found to be useful in the development of a Sanskrit parser as well. The concept of subcategorisation of modern Linguistics comes close to the concept of ākāṅkṣā. Subcategorization structures provide syntactic frames to capture different syntactic behaviors of verbs. Sanskrit being an inflectional language, the information of various relations is encoded in suffixes rather than in positions. These suffixes express the expectancy, termed as ākāṅkṣā in the Sanskrit literature. Kulkarni, Pokar,
Sanjeev Panchal and Amba Kulkarni (2010) describe how the ākāṅkṣā was found to be useful in the proposition of possible relations between words. Sannidhi has been found to be equivalent to the weak non-projectivity principle (Kulkarni, P. Shukla, et al. 2013c). In this paper, we will discuss the role of the third factor viz. yogyatā, in building a Sanskrit parser.

The concept of selection restriction is similar to the concept of yogyatā. The expectancy, or the ākāṅkṣā, proposes a possible relation between the words in a sentence. Such a relation would hold between two words only if they are meaning-wise compatible. It is the selection restriction or yogyatā which then comes into force to prune out incongruent relations, keeping only the congruent ones. Katz and Fodor (1963) proposed a model of selection restrictions as necessary and sufficient conditions for semantic acceptability of the arguments to a predicate. Identifying a selection restriction that is both necessary and sufficient is a very difficult task. Hence there were attempts to propose alternatives. One such alternative was proposed by Wilks (1975) who viewed these restrictions as preferences rather than necessary and sufficient conditions. After the development of WordNet, Resnik (1993) modeled the problem of induction of selectional preferences using the semantic class hierarchy of WordNet. Since then there is an upsurge in the field of computational models for the automated treatment of selectional preferences with a variety of statistical models and Machine learning techniques. In recent times, one of the ambitious projects to represent World Knowledge was taken up under the banner of Cyc. This knowledgebase contains over five hundred thousand terms, including about seventeen thousand types of relations, and about seven million assertions relating these terms.¹ In spite of the availability of such a huge knowledge base, we rarely find Cyc being used in NLP applications.

The first attempt to use the concept of yogyatā in the field of Machine Translation was by the Akshar Bharati group (Bhanumati 1989) in the Telugu-Hindi Machine Translation system. Selectional restrictions were used in defining the Kāraka Charts that provided a subcategorization frame as well as semantic constraints over the arguments of the verbs. On similar lines Noun Lakṣaṇa Charts and Verb Lakṣaṇa Charts were also used for disambiguation of noun and verb meanings. These charts expressed selectional restrictions using both ontological concepts as well as semantic

¹http://www.cyc.com/kb, accessed on 30th August, 2017
properties. An example Kāraka chart for the Hindi verb jānā (to go) is given in table 1.

<table>
<thead>
<tr>
<th>case relation</th>
<th>necessity</th>
<th>case marker</th>
<th>semantic constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>apādānam (source)</td>
<td>desirable</td>
<td>se</td>
<td>not (upādhi:vehicle)</td>
</tr>
<tr>
<td>karaṇam (instrument)</td>
<td>desirable</td>
<td>se</td>
<td>(upadhi:vehicle)</td>
</tr>
<tr>
<td>karma (object)</td>
<td>mandatory</td>
<td>0/ko</td>
<td>-</td>
</tr>
<tr>
<td>kartā (agent)</td>
<td>mandatory</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1
Kāraka Chart for the verb jānā (to go)

Here upādhi is an imposed property. The first row in Table 1 states a constraint that a noun with case marker se has a kāraka role of apādānam (source) provided it is not a vehicle. The ontological classification was inspired by the ontology originated from the vaiśeṣika school of philosophy. The parsers for Indian languages were further improved. Bharati, Chaitanya, and Sangal (1995) mentions the importance of two semantic factors viz. animacy and humanity, in parsing, that removes the ambiguity among the kartā and karma (roughly subject and object). This hypothesis was further strengthened with experimental verification by Bharati, Husain, et al. (2008).

In the next section, we first state the importance of yogyatā in parsing, as a filter to prune out meaningless parses. Since yogyatā deals with the compatibility between meanings, and a word expresses meanings at different levels, we also discuss the mutual hierarchy among these various meanings. In the third section, we look at various definitions of yogyatā offered in the tradition, and decide the one that is suitable for implementation. In the same section, we evolve strategies to disambiguate relations based on yogyatā. Finally, the criteria evolved for disambiguation are evaluated. The evaluation results are discussed in section four, followed by the conclusion.

2 Yogyatā as a filter

Necessary condition for understanding a sentence is that a word having an expectancy for another word should become nirākāṅkṣa (having no further
expectancy) once a relation is established between them. Further, such related words should also have mutual compatibility from the point of view of the proposed relation. If they are not, then the expectancy of such words will not be put to rest and there would not be any verbal cognition. Therefore the role of yogyatā in verbal cognition is very important. The purpose of using yogyatā in parsing is not to make a computer ‘understand’ the text, but to rule out incompatible solutions from among the solutions that fulfill the ākāṁkṣās. For example, in the sentence

Skt: *yānam vanam gacchati.*
Gloss: vehicle{neut., sg., nom./acc.} forest{neut., sg., nom./acc.} go{present, 3rd per., sg.}

There are 6 possible analyses, based on the ākāṁkṣā. They are

1. *yānam* is the kartā and *vanam* is the karma of the verb *gam,*
2. *yānam* is the karma and *vanam* is the kartā of the verb *gam,*
3. *yānam* is the kartā of the verb *gam* and *vanam* is the viśeṣaṇa of *yānam,*
4. *yānam* is the karma of the verb *gam* and *vanam* is the viśeṣaṇa of *yānam,*
5. *yānam* is the viśeṣaṇa of *vanam* which is the kartā of the verb *gam,*
6. *yānam* is the viśeṣaṇa of *vanam* which is the karma of the verb *gam.*

If the machine knows that the kartā of an action of going should be movable, and that the designation of *yāna* is movable, but that of *vana* is not movable, then mechanically it can rule out the second analysis. The words *yānam* and *vanam* on account of the agreement between them have the potential to be viśeṣaṇas of each other. But the semantic incompatibility between the meanings of these words rules out the last four possibilities, leaving only the first correct analysis.

As another example, look at the sentence

Skt: *Rāmeṇa bāṇena Vālī hanyate.*
Gloss: Rama{ins.} arrow{ins.} Vali{nom.} is_killed.

*Rāma* and *bāṇa,* both being in instrumental case, can potentially be a kartā as well as a karaṇam of the verb *han* (to kill). If the machine knows that *bāṇa* can be used as an instrument in the act of killing, while *Rāma* being the name of a person, can not be a potential instrument in the act of
killing, it can then filter out the incompatible solution: Rāma as a karaṇam and bāna as a kartā.

Look at another sentence payasā siṅcati (He wets with water). Here payas (water) is in instrumental case, and is a liquid, and hence is compatible with the action of siṅc (to wet). But in the sentence vahṅinā siṅcati (He wets with fire), vahṅi (fire) is not fit to be an instrument of the action of wetting, and as such it fails to satisfy the yogyatā. But now imagine a situation where a person is in a bad mood, and his friend without knowing it starts accusing him further for some fault of his, instead of uttering some soothing words of the console. Third-person watching this utters kim vahṅinā siṅcasī (Why are you pouring fire?) - a perfect verbalization of the situation. The words, here, are like a fire to the person who is already in a bad mood. This meaning of vahṅi is its extended meaning. Thus, even if a relation between primary meanings does not make sense, if the relation between extended meanings makes sense, we need to produce the parse. Therefore, in addition to the primary meanings, the machine also, sometimes, needs access to the secondary/extended meanings of the words.

2.1 Word and its Meanings

Every word has a significative power that denotes its meaning. In Indian theories of meaning, this significative power is classified into three types viz. abhidhā (the primary meaning), lakṣaṇā (the secondary or metaphoric meaning) and vyañjanā (the suggestive meaning). In order to use the concept of yogyatā in designing a parser, we should know what is the role of each of these meanings in the process of interpretation.

The secondary meaning comes into play when the primary meaning is incompatible with the meanings of other words in a sentence. The absence of yogyatā is the basic cause for this signification. Indian rhetoricians accept three conditions as necessary for a word to denote this extended or metaphoric sense. These three conditions are²

1. inapplicability / unsuitability of the primary meaning,
2. some relation between the primary meaning and the extended meaning, and

² mukhyārtha bhadhe tad yogyo 'ṛūḍhitato pravojanāt |
anyo'rtho lakṣyate yat sā lakṣaṇāropitā kriyā ||
(KP II 9)
3. definite motive justifying the extension.

In addition to these two meanings, there is one more meaning, called vyañjanā or the suggestive meaning. This corresponds to the inner meaning of any text/speaker’s intention. In order to understand this meaning, consider a sentence *gato’stam arkaḥ* which literally means ‘the sun has set’. Every listener gets this meaning. In addition to this meaning, it may also convey different signals to different listeners. For a child playing in the ground, it may mean ‘now it is getting dark and it is time to stop playing and go home’, for a Brahmin, it may mean ‘it is time to do the sandhyā-vandana’, and for a young man it may mean ‘it is time to meet his lover’. This extra meaning co-exists with the primary meaning. It does not block the primary meaning. Therefore vyaṅgārthā (suggestive meaning) exists in parallel with the primary/secondary meaning.

Since the suggestive meaning is in addition to the primary/secondary meaning, and is optional, and also is different for different listeners, it involves subjectivity for processing. Hence it is not possible to objectively process this meaning for any utterance.\(^3\) This also puts an upper limit on the meaning one can get from a linguistic utterance without the interference of subjective judgments. In summary, we observe that these three meanings are not in the same plane. Lakṣaṇā comes into play only when abhidhā fails to provide a suitable meaning for congruent interpretation. And the suggestive meaning can co-exist with the abhidhā as well as the lakṣaṇā, and as such, is outside the scope of automatic processing.

### 3 Modeling Yogyatā

Yogyatā is the compatibility between the meanings of related words. This meaning, as we saw above, can be either a primary or a metaphoric one. The absence of any hindrance in the understanding of a sentence implies there is yogyatā or congruity among the meanings. There have been different views among scholars about what yogyatā is. According to one definition, yogyatā is *artha-abādhaḥ*\(^4\) (that which is not a hindrance to meaning). It

---

\(^3\)One of the reviewers commented that taking into account the advents in Big Data and Machine Learning techniques, it may even be possible to process such meanings by machines in the future. However, we are of the opinion that machine would need semantically annotated corpus for learning, which does not yet exist.

\(^4\)All the meanings we will be discussing below are found in NK p. 675.
is further elaborated as $bādhaka-pramā-virahāḥ$ or $bādhaka-niścaya-abhāvaḥ$ (absence of the decisive knowledge of incompatibility). There are other attempts to define it as an existing qualifying property. One such definition is $sambandha-arhatvam$ (eligibility for mutual association), and the other one is $paraspara-anvaya-prayojaka-dharmavattvam$ (a property of promoting mutual association). The first set of definitions presents yogyatā as an absence of incompatibility whereas the second set of definitions present it as the presence of compatibility between the meanings.

Let us see the implications of modeling yogyatā through these two lenses.

1. We establish a relation only if the two morphemes are mutually congruous.
   In this case, we need to take care of not only the congruity between primary meanings but even between the metaphoric/secondary meanings.

2. We establish a relation if there is no incongruity between the two meanings.

The first possibility ensures that the precision is high and there is less chance of Type-1 error, i.e. of allowing wrong solutions. The second possibility, on the other hand, ensures that the recall is high and there is less chance of Type-2 error, viz. the possibility of missing any correct solution. But there is a chance that we allow some unmeaningful solutions as well. If we decide to go for the first possibility, we need to handle both the primary as well as secondary meanings, and we need to state precisely under what conditions the meanings are congruous. And this means modeling congruity for each verb and for each relation. This is a gigantic task, and there is a possibility of missing correct solutions if we do not take into account all the possible extensions of meanings. Therefore, we decided to go for the second choice allowing a machine to do some mistakes of choosing incongruous solutions but we did not want to throw away correct solutions even by mistake. This decision is in favor of our philosophy of sharing the load between man and machine. Our aim is to provide access to the original text by reducing the language learning load. So we can not afford to miss a possible solution. Thus at the risk of providing more solutions than the actual possible solutions, we decided to pass on some load to the reader of pruning out irrelevant solutions manually.
In the first step, we decided to use yogyatā only in those cases where a case marker is ambiguous between more than one relation. We noticed the following three cases of ambiguities with reference to the relations.

1. viśeṣya-viśeṣaṇa-bhāva (adjectival relation)
   Here both the viśeṣya and viśeṣaṇa agree in gender, number and case, and hence only on the basis of the word form, we can not tell which one is viśeṣya and which one is viśeṣaṇa.

2. a kāraka and a non-kāraka relation as in
   a. karaṇam (instrument) and hetu (cause), with an instrumental case marker,
   b. sampradānam (beneficiary), prayojanam (purpose) and tādarthya (being intended for), with a dative case marker,
   c. apādānam (source) and hetu (cause), with an ablative case marker.

3. śaṣṭhī sambandha (a genitive relation) and a viśeṣaṇa (an adjective)
   When two words are in the genitive case, it is not clear whether there is an adjectival relation between them, or a genitive relation.

We now discuss each of these three cases below.

3.1 Viśeṣya-viśeṣaṇa-bhāva (Adjectival relation)

We come across a term samānādhikaraṇa (co-reference) in Pāṇini to denote an adjective (Joshi and Roodbergen 1998, p. 6). One of the contexts in which the term samānādhikaraṇa is used is the context of an agreement between an adjective and a noun. For example, dhāvantaṁ mṛgaṁ (a running deer), or sundaraḥ aśvaḥ (a beautiful horse). Pāṇini has not defined the term samānādhikaraṇa, either. The term samānādhikaraṇa (co-reference) literally means ‘having the same locus’. Patañjali in the Samartha-āhnika discusses the term sāmānādhikaraṇya (co-referential) (literally a property of being in the same locus). In the example, sundaraḥ aśvaḥ (a beautiful horse), both the qualities of saundarya (beauty) and aśvatva (horse-ness) reside in an aśva (horse), which is the common locus. Similarly, in the case of ācāryaḥ droṇaḥ, or agne gṛhapate (O Agni! house-holder), both the words ācārya as well as droṇa refer to the same individual, so do agni

\[sāmānādhikaraṇyam ekavibhaktitvam ca. dvayoḥ sañjñinorvā (MBh 1.1.1)\]
and grhapatī. This is true of various other relation-denoting terms such as guru, śiṣya, pitā, putra, etc. and upādhis (imposed / acquired properties) such as rājā, mantrī, vaidya, etc. From all this discussion, we may say sāmānādhikaraṇya (the property of having the same locus) is the semantic characterisation of a viśeṣaṇa.

In Sanskrit, there is no syntactic / morphological category as a viśeṣaṇa (an adjective). The gender, number and case of a viśeṣaṇa follows that of a viśeṣya (the head). From the point of view of analysis this provides a syntactic clue for a possible viśeṣya-viśeṣaṇa-bhāva between two words such as in śuklaḥ paṭaḥ (a white cloth). This agreement is just a necessary condition, and not sufficient. Because, a viśeṣaṇa, in addition to agreeing with the viśeṣya should also be semantically fit to be a qualifier of the viśeṣya. For example, there can be two words say yānam (a vehicle) and vanam (a forest), that match perfectly in gender, number and case, but we can not imagine a viśeṣya-viśeṣaṇa-bhāva between yāna and vana. Is it only the semantics that rules out such a relation or are there any clues, especially syntactic ones, that help us to rule out a viśeṣya-viśeṣaṇa-bhāva between such words?

**In search of clues:**

Pāṇini has not defined the terms viśeṣya and viśeṣaṇa. Patañjali uses two terms dravya (substance) and guṇa (quality) while commenting on the agreement between a viśeṣya and a viśeṣaṇa.

\[ \text{yad asau dravyam śrito bhavati guṇah tasya yat liṅgam vacanam ca tad guṇasya api bhavati. (MBh under A4.1.3 Vt VI.)} \]

A quality assumes the gender and number of the substance in which it resides.

**But then what is this guṇa?**

We come across the description of guṇa by Kaiyyaṭa.

\[ \text{sattve niviṣate apaiti prthag jātiṣu dṛṣyate} \]
\[ \text{ādheyaḥ -ca-akriyājah-ca saḥ asattva-prakṛti-guṇah} \]

(MBh A4.1.44)

Guṇa is something which is found in things / substances (sattve
niviśate), which can cease to be there (apaiti), which is found in different kinds of substances (pṛthag jātiṣu), which is sometimes an effect of an action and sometimes not so (ādheyah-ca-akriyājah-ca), and whose nature is not that of a substance (asattva-prakṛti).

Thus guṇa is something which is not a substance since it resides in other things. It is not universal since it is found in different kinds of substances. It is not an action, since guṇa is sometimes an effect of an action, as in the case of the color of a jar and sometimes not, as in the case of the magnitude of a substance. This characterisation of guṇais very close to the vaiśeṣika’s concept of guṇa (Raja 1963).

Then, is this vaiśeṣika guṇa a viśeṣaṇa?

Patañjali commenting on the word guṇa under A2.2.11 provides an example contrasting two types of guṇas. While both śukla and gandha are qualities (guṇa) according to the vaiśeṣika ontology, the usage śuklah paṭah (a white cloth) is possible, while gandham candanam (fragrance sandal-wood) is not. Thus, only some of the vaiśeṣika guṇas have a potential to be a viśeṣaṇa, and not all.

If viśeṣaṇa is not a vaiśeṣika guṇa, what is it?

The characterisation of guṇa by Bhartṛhari in Guṇa-samuddeśa includes bhedakam as one of the characteristics of guṇa. But, in addition, guṇa, according to him, is also capable of expressing the degree of quality in a substance through a suffix. He defines guṇa as

\[
\text{saṁsargi bhedakam yad yad savvyāpāram pratiyate}
\]
\[
guṇatvam paratantratvāt tasya śāstra udāḥṛtam.\text{VP III.5.1}
\]

Whatever rests on something else (saṁsargi), differentiates it (bhedaka), and is understood in that function (savyāpāra) is, being dependent, called quality in the śāstra. (Iyer 1971)

According to Bhartṛhari, apart from being a differentiator, a guṇa has another important characteristic, viz. that such a distinguishing quality can
also express the degree of excellence through some suffix (such as a comparative suffix tarap, or a superlative suffix tamap). This concept of guṇa of Bhartṛhari, thus is different from the concept of the guṇa of a vaiśeṣika. This definitely rules out the case of gandha, since we can not have gandhatara but we can have śuklatara to distinguish the white-ness between two white cloths.

**Another clue from Pāṇini**

We have another hint from Pāṇini through Patañjali. While in A4.1.3, Patañjali has used the terms dravya and guṇa in connection with agreement, in A1.2.52, he uses the term guṇavacana while describing a višeṣaṇa

\[\text{guṇavacananāmī śabdānām-āśrayataḥ liṅgavacananāṁ bhavanti-īti}\]

(A1.2.52).

The words which are guṇavacanas take the gender and number of the substance in which they reside.

The term guṇavacana is used for those words which designate quality and then a substance in which this quality resides (Cardona 2009). In the example, śuklaḥ pataḥ, since śukla in addition to being a quality (white color), can also designate a substance, such as a paṭa (cloth), which is (white) in color, it is a guṇavacana word. But gandha (fragrance) designates only quality, and can not be used to designate a substance that has a fragrance, and hence is not a guṇavacana.

**Is guṇavacana necessary and sufficient to describe a višeṣaṇa?**

Let us look at the examples above. It definitely rules out yānaṁ and vanaṁ to be qualifiers of each other, since neither of them is quality. But then what about dhāvan (the one who is running) in dhāvan bālakah (a running boy)? Is dhāvan a guṇavacana?

Guṇavacana is a technical term, used by Pāṇini to define an operation of elision of matup suffix in certain quality denoting words such as ’sukla etc. So technically, a word such as dhāvan, though it designates a substance, is not a guṇavacana. This is clear from Patañjali’s commentary on A1.4.16 where he

6The Vārtika guṇavacanam ca is followed by several other vārttikas, of which the following two are relevant. samāsa-kṛt-taddhita-avyaya-sarvanāma-asarvaliṅgā jātiḥ ||41 || sanīkhyā ca ||42 ||
states that compounds (samāsa), primary derivatives (kṛdantas), secondary derivatives (taddhitāntas), indeclinables (avyaya), pronouns (sarvanāma), words referring to universals (jāti), numerals (saṁkhyā) can not get the designation guṇavacana, since the latter sanjñās (technical terms) supersede the previous ones.\(^7\)

The very fact that Kātyāyana had to mention that words belonging to all the latter categories are not guṇavacana, indicates that all these categories of words have the potential to get the guṇavacana designation, but Pāṇini did not intend to assign this sanjñā to these words. Whatever may be the reason, but this list of various categories, in fact, provides us a morphological clue for a word to be a viśeṣaṇa.

Here are some examples of viśeṣaṇas belonging to these different grammatical categories.

1. Samāsa (a compound)
   Bahuvrīhi (exo-centric) compounds refer to an object different from the components of the compound, and thus typically act as adjectives. For example, pītāmbaraḥ is made up of two components pīta (yellow) and ambara (cloth), but it refers to the ‘one wearing a yellow-cloth’ (and is conventionally restricted to Viṣṇu). An example of tat-puruṣ (endo-centric) compound as a viśeṣaṇa is parama-udāraḥ (extremely noble).

2. Kṛdanta (an adjectival participle)
   Nouns derived from verbs act as qualifiers of a noun. For example, in the expression dhāvantam mṛgam (a running deer), dhāvantam, a verbal noun, is a viśeṣaṇa. Only certain kṛdanta suffixes such as śatṛ, śānac, kta, etc. produce nouns that can be viśeṣaṇas, and not all.

3. Taddhita (a secondary derivative)
   Taddhitas with certain suffixes derive new nouns such as bhāratīya (Indian), dhanavān (wealthy), guṇin (possessing good qualities), etc. that denote a substance, as against certain other taddhita words such as manusyatā (humanity), vārddhakya (senility) etc. which derive new words designating qualities.

4. Sarvanāma (a pronoun)
   Pronouns also act as qualifiers. For example, in the expression idam pustakam (this book), idam is a viśeṣaṇa.

\(^7\)gaṇavacanaśaṁjñāyāḥ ca etābhiḥ bādhanāṁ yathā syāt iti
5. Jāti (a universal)
In an expression āmraḥ vṛkṣaḥ (a mango tree), both the words āmraḥ and vṛkṣaḥ are common nouns. But one is a special and the other one is a general one. So the designation of āmra is a subset of the designation of vṛkṣa. Only in such cases, where there is a parājāti-aparājāti (hypernymy-hyponymy) relation, the one denoting an aparājāti (hyponymy) qualifies to be a viśeṣaṇa of the other one.

6. Saṁkhya (a numeral)
In an expression ekaḥ puruṣaḥ (a man), the word ekaḥ designates a number, which is a viśeṣaṇa of puruṣa.

There are still two more classes of words that are not covered in the above list, but which can be viśeṣaṇas. They are: words denoting an acquired property or an imposed property, and the relation-denoting terms. For example, ācāryaḥ in ācāryaḥ droṇaḥ, is an imposed property and putraḥ in Daśarathasya putraḥ rāmaḥ is a relation denoting term.

In summary, samastapada, certain kṛdantas, certain taddhitāntas, saṁkhya, sarvanāma, ontological categories such as parā-aparā jātis, semantico-syntactic property such as guṇavacana and finally semantic properties such as relation denoting terms and upādhis, all these serve as characterisations of a viśeṣaṇa. This characterization is only a necessary condition, and not sufficient since it does not involve any mutual compatibility between the words. However, it brings in more precision in the necessary conditions for two words to be in viśeṣya-viśeṣaṇa-bhāva.

3.1.1 Deciding a Viśeṣya

Once we have identified the words that are mutually compatible with regard to an adjectival relation, the next thing is to decide the viśeṣya (head) among them. The commentary on A2.1.57 is useful in deciding the viśeṣya. This sūtra deals with the compound formation of two words that are in viśeṣya-viśeṣaṇa-bhāva. In Sanskrit compound formation, the one which is subordinate gets a designation of upasarjana. This provides us a clue about which word classes are subordinate to which ones. A noun may refer to a substance through an expression expressing the class character (jāti) such as utpalam (a flower), or through an action associated with it (kriyāvacana), as in dhāvan (running), or through a guṇavācaka such as nīlam. If there are two words designating common nouns, one denoting a special and the other one general, then the one which denotes a special type of common noun is
subordinate. For example, in āmraḥ vṛkṣaḥ, āmra is a special kind of tree, and hence is a višeṣaṇa and vṛkṣa is its višeṣya. If one word designates a common noun and the other one either a guṇavacana or a kriyāvacana, then the word denoting the common noun becomes the višeṣya. Thus in nilam utpalam, utpalam is the višeṣya. In pācakaḥ brāhmaṇaḥ (cook Brahmin), brāhmaṇaḥ is the višeṣya. When one of the words designate a guṇavacana and the other a kriyāvacana, or both the words designate either guṇavacanas or kriyāvacanas, then either of them can be a višeṣya, as in khañjaḥ kubjaḥ (a hump-backed who is limping) or kubjaḥ khañjaḥ (a limping person with hump-back), similarly as in khañjaḥ pācakaḥ (a limping cook) or pācakaḥ khañjaḥ (a limping person who is a cook), etc.

On the basis of the above discussion, we have the following preferential order for the višeṣya.

\[
\text{jātivācaka} > \{\text{guṇvacana, kṛdanta}\}.
\]

We saw earlier that a višeṣaṇa can be any one of the following: a pronoun, a numeral, a kṛdanta, a taddhitānta, a samasta-pada, guṇavācaka, jāti, relation denoting terms, and an upādhi. So adding all these categories to the above preferential order, we get,

\[
\text{jātivācaka} > \text{upādhi} > \text{taddhitānta} > \text{guṇvacana} > \text{numeral} > \text{kṛdanta} > \text{pronoun}.
\]

### 3.1.2 Flat or Hierarchical Structure?

After we identify all the words that have a samānādhikaraṇa relation between them, and mark the višeṣya (the head) among them, the next task is to know whether a višeṣaṇa is related to this višeṣya directly, or through other višeṣaṇas.

If there are \(n\) višeṣaṇas, and all of them are related to the višeṣya directly, then it results in a flat structure. But if a višeṣaṇa is related to the višeṣya...
through other viśeśaṇas, then there are exponentially large number of ways in which $n$ viśeśaṇas can relate to the viśeṣya. For example, if there are three words say $a$, $b$ and $c$, of which $c$ is the viśeṣya. Then computationally, there are three ways in which the other two words may relate to $c$.

1. Both $a$ and $b$ are the viśeṣaṇa of $c$. (This results in a flat structure.)
2. $a$ is a viśeṣaṇa of $b$ and $b$ that of $c$.
3. $b$ is a viśeṣaṇa of $a$ and $a$ that of $c$.

In positional languages like English, only the first two cases are possible. For example, consider the phrase ‘light red car’, which may either mean a car which is red in color and is light in weight, or a car which is light-red in color. In the second case, light-red is a compound.

Sanskrit being a free word order language, one can imagine, computationally, a possibility for the third type as well. The relation between the adjectival terms being that of sāmānādhikaraṇyā (co-referential), semantically, only a flat structure is possible with adjectives. The other two cases of hierarchical structures result in compound formation in Sanskrit.

This is also supported by Jaimini’s Mīmāṁsā sūtra

\[ \text{guṇānām ca parārthatvāt asambandhaḥ samatvāt syāt. (MS 3.1.22)} \]

In as much as all subsidiaries are subservient to something else and are equal in that respect, there can be no connection among themselves.
(Jha 1933)

Thus, a viśeṣaṇa is not connected to another viśeṣaṇa. The associated structure is a flat one, with all the viśeṣaṇas being connected to the viśeṣya.

3.2 Distinguishing a kāraka from a non-kāraka:

In Sanskrit, some case markers denote both a kāraka relation as well as a non-kāraka relation, as we saw earlier. In a sentence, if a verb denotes an action, then nouns denote the participants in such an action. These participants, which are classified into 6 types, viz. kartā, karma, karaṇam, sam-pradānam, apādānam, and adhikaraṇam are collectively called as kārakas. Other nouns in the sentence, which do not participate directly in the action,
express non-kāraka relations such as hetu (cause), prayojanam (purpose), etc. We get a clue to distinguish between the nouns which are related by a kāraka relation and those which are related by a non-kāraka one in the Aruṇādhikāra of the Śābara bhāṣya. There it is mentioned that

\[ na \ ca \ amūrta-arthaḥ \ kriyātāḥ \ sādhanam \ bhavatīti \ (SB; \ p \ 654) \]

No unsubstantial object can ever be the means of accomplishing an act.

Thus anything other than dravya can not be a kāraka. As we saw earlier, the guṇavacanas also can designate a dravya. And thus, all the dravyas and the guṇavacanas are qualified to be a kāraka. And the rest, i.e. nouns which denote either a guṇa which is not a guṇavacana or a kriyā (verbal nouns), may have a non-kāraka relation with a verb.

Let us see some examples.

Skt: \( rāmaḥ \ daśarathasya \ ājñayā \ rathena \ vanam \ gacchati \).
Gloss: Rama {nom.} Dasharatha{gen.} order{ins.} ratha{ins.} for- est{acc.} goes.
Eng: On Dasharatha’s order, Rama goes to the forest by a chariot.

Skt: \( rāmaḥ \ adhyayanena \ atra \ vasati \).
Gloss: Rama {nom.} study{ins.} here lives.
Eng: Rama lives here in order to study.

In the first sentence ājñā (order) is the cause for Rama’s going to forest, ratha (chariot) is the instrument (or vehicle) for his going and in the second sentence adhyayana is the cause of Rāma’s stay.

Since both hetu as well as karaṇam demand a 3\textsuperscript{rd} case suffix, ākāṅkṣā would establish a relation of karaṇam between ājñāyā and gacchati,\textsuperscript{11} between rathena and gacchati and also between adhyayana and gacchati. Now with the above definition of a kāraka, adhyayana, being a verbal noun (a krdanta) in the sense of bhāva, represents an abstract concept and therefore it does not designate a dravya (a substance). Hence it can not be a karaṇam. Similarly ājñā, which is a guṇa (according to Vaiśeṣika ontology, being a

\textsuperscript{11}To be precise, the relation is between the meaning denoted by the nominal stem ājñā and the one denoted by the verbal root gam.
śabda), can not be a karaṇa. Thus the use of congruity helps in pruning out impossible relations.

On the same grounds, establishment of apādānam and sampradānam relations between a non-dravya\textsuperscript{12} denoting noun and a verb can also be prevented.

3.3 Congruous substantive for a Śaṣṭhī (genitive)

Pāṇini has not given any semantic criterion for the use of the genitive relation. His rule is śaṣṭhī seṣe (A2.3.50) which means, in all other cases that are not covered so far, the genitive case suffix is to be used. The relation marked by the śaṣṭhī (genitive) case marker falls under the utthāpya (aroused) ākāṁkṣā. This is a case of uni-directional expectancy. Thus, there is no syntactic clue to which noun the word in genitive case would get attached. All other nouns in the sentence are potential candidates for a genitive relation to join with. The clue is, however, semantic. Patañjali in the Mahābhāṣya on A2.3.50 provides some semantic clues. He says there are hundreds of meanings of śaṣṭhī. Some of them are sva-svāmi-bhāva as in rājñaḥ puruṣah (a king’s man), avayava-avayavī-bhāva as in vykṣasya sākhā (branch of a tree) etc. So in order to establish a genitive relation, we need the semantic inputs. However, there are certain constraints. They are

1. A genitive connecting a verbal noun expressing bhāva such as luyt etc. expresses a kāraka\textsuperscript{13} relation and not the genitive one, as in rāmasya gamanam.

2. A genitive always connects with a viśeṣya, and never with a viśeṣaṇa, since there is a samānādhikaraṇa relation between the viśeṣya and viśeṣaṇa. For example, in the expression rāmasya vīreṇa putreṇa, the genitive relation of rāmasya is with putreṇa and not with vīreṇa.

Lexical resources such as Sanskrit WordNet\textsuperscript{14} and Amarakośa\textsuperscript{15} that are marked with the semantic information of part-whole relation, janya-janaka-bhāva, ājīvikā relation etc. help in identifying the genitive relations with confidence. When both the words refer to dravyas (substantives), then also there is a possibility of a genitive relation. So note that, while for other

\textsuperscript{12}To be precise, a non-dravya and non-guṇavacana.
\textsuperscript{13}kartṛkarmaṇoḥ kṛti (A2.3.65)
\textsuperscript{14}http://www.cfilt.iitb.ac.in/wordnet/webswn/english_version.php
\textsuperscript{15}http://scl.samsaadhanii.in/amarakosha/index.html
relations, we look for the absence of non-congruity for ruling out the relations, in the case of genitives, instead, we look for the presence of congruity, to prune out impossible relations. We took this decision, since we found it difficult to describe the non-congruity in the case of genitive relations.

**Ambiguity between a genitive and an adjectival relation**

Further, we come across an ambiguity in the genitive relation, in the presence of adjectives. Look at the following two examples.

Skt: \( \text{vīrasya Rāmasya bāṇam} \)
Gloss: brave\{gen.\} Rama\{gen.\} arrow
Eng: An arrow of brave Rama

and

Skt: \( \text{Rāmasya putrasya pustakam} \)
Gloss: Rama\{gen.\} son\{gen.\} book
Eng: A book of Rama’s son

In the first example, \( \text{vīra} \) being a \( \text{guṇavacana} \), with the earlier characterisation of an adjective, \( \text{vīra} \) would be marked an adjective. while in the second one there is a kinship relation.

4 Evaluation

As stated earlier, ākāṅkṣā states the possibility of relations between two words. The mutual compatibility between the meanings further helps in pruning out the incompatible relations. We classified the content nouns into two classes: dravya and guṇa. Guṇas being further marked if they are guṇavacanas. We tested the mutual compatibility only when the suffix is ambiguous. To be precise, the yogyatā is used only to disambiguate between a kāraka versus non-kāraka relation, to establish the višeṣya-višeṣaṇa-bhāva, and to establish a genitive relation. This ensured that we do not miss the metaphoric meanings. In the case of kāraka relations, if the noun denotes a guṇavacana, then the possible kāraka relation, on the basis of expectancy is
pruned out. Similarly, in the case of adjectival relations, the relations with a non-
guṇavācaka guṇa is pruned out.

The performance of the system with and without yogyatā was measured to evaluate the impact of yogyatā. The corpus for evaluation of sentences consists of around 2300 sentences. It includes sentences with various grammatical constructions, a few passages from school text book, Bhagavadgītā, and a sample from Māgha’s Śiśupālavadham. The ślokas in Bhagavadgītā as well as in Śiśupālavadham were converted to a canonical form.\textsuperscript{16} The sentences with conjunction were not considered for the evaluation, since the nouns in conjunction conflict with the adjectives, and the criteria for handling conjunction are under development. The statistics showing the size of various texts, the average word length and the average sentence length is given in Table 2.

<table>
<thead>
<tr>
<th>Type</th>
<th>Sents</th>
<th>Words</th>
<th>characters</th>
<th>avg sntlen</th>
<th>avg wrd len</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text books</td>
<td>260</td>
<td>1,295</td>
<td>9,591</td>
<td>4.98</td>
<td>7.40</td>
</tr>
<tr>
<td>Syntax</td>
<td>937</td>
<td>3,339</td>
<td>25,410</td>
<td>3.56</td>
<td>7.61</td>
</tr>
<tr>
<td>Māgha’s SPV</td>
<td>66</td>
<td>623</td>
<td>5,851</td>
<td>9.40</td>
<td>9.39</td>
</tr>
<tr>
<td>Bhagavadgītā</td>
<td>940</td>
<td>5,698</td>
<td>42,251</td>
<td>6.06</td>
<td>7.41</td>
</tr>
<tr>
<td>Total</td>
<td>2,203</td>
<td>10,955</td>
<td>83,103</td>
<td>3.77</td>
<td>7.58</td>
</tr>
</tbody>
</table>

\textbf{Table 2}

\textit{Corpus Characteristics}

All these sentences were run through a parser, first without using the conditions of yogyatā and second times using the conditions of yogyatā. In both cases, the parser produced all possible parses. We also ensured that the correct parse is present among the produced solutions. Table 3 shows the statistics providing the number of solutions with and without using the filter of yogyatā. The number of parses produced was reduced drastically. This improved the precision by 63\% in textbook stories, by 67\% in the grammatical constructs, and by 81\% in case of the text from Bhagavadgītā and Māgha’s kāvya. Better results in the case of these texts pertains to the fact that these texts have more usage of adjectives and non-kāraka relations as against the textbook sentences, and artificial grammatical constructs.

\textsuperscript{16}All the ślokas were presented in their anvita form, following the traditional Daṇḍān-
vaya method, where the verb typically is at the end, and višeṣaṇas precede the viśeṣyas.
<table>
<thead>
<tr>
<th>Corpus type</th>
<th>Sents</th>
<th>avg sols without yogyata</th>
<th>avg sols with yogyata</th>
<th>improvement in precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text books</td>
<td>260</td>
<td>39.76</td>
<td>14.56</td>
<td>63%</td>
</tr>
<tr>
<td>Syntax</td>
<td>937</td>
<td>19.5</td>
<td>6.33</td>
<td>67%</td>
</tr>
<tr>
<td>Literary</td>
<td>66</td>
<td>11,199</td>
<td>2,107</td>
<td>81%</td>
</tr>
<tr>
<td>BhG</td>
<td>940</td>
<td>2,557</td>
<td>478</td>
<td>81%</td>
</tr>
<tr>
<td>Total</td>
<td>2203</td>
<td>1439.54</td>
<td>268.85</td>
<td>81%</td>
</tr>
</tbody>
</table>

Table 3

*Improvement*

### 5 Conclusion

Yogyatā or mutual congruity between the meanings of the related words is an important factor in the process of verbal cognition. In this paper, we presented the computational modeling of yogyatā for automatic parsing of Sanskrit sentences. Among the several definitions of *yogyatā*, we modeled it as an absence of non-congruity.

Due to lack of any syntactic criterion for viśeṣaṇa (adjectives) in Sanskrit, parsing Sanskrit texts with adjectives resulted in a high number of false positives. Hints from the vyākaraṇa texts helped us in the formulation of a criterion for viśeṣaṇa with syntactic and ontological constraints, which provided us a hint to decide the absence of non-congruity between two words with respect to the adjectival relation. A simple two-way classification of nouns into dravya (substance) and guṇa (quality) with further classifications of guṇas into guṇavacanas was found to be necessary for handling adjectives. The same criterion was also found useful to handle the ambiguities between a *kāraka* and *non-kāraka* relations. These criteria together with modeling *yogyatā* as an absence of non-congruity resulted in 81% improvement in precision.

Finally, the fact that there can not be an adjective of an adjective, having identified a višeṣya, there is only one way all the višeṣaṇas can connect with the višeṣya. This theoretical input provided much relief from a practical point of view, in the absence of which possible solutions would have been exponential.
6 Abbreviations

A: Pāṇini’s Aṣṭādhyāyī, See Pande, 2004
Aa.b.c : adhyāya(chapter), pāda(quarter), sūtra number in Aṣṭādhyāyī
BM: Bālamanoramā, see Pande, 2012
MBh: Patañjali’s Mahābhāṣya, see Mīmāṁsaka
KP: Kāvyaprakāśa, see Jhalakikar
MS: Mīmāṁsā sūtra, through SB
NK: Nyāyakośa, see Jhalkaikar
PM: Padamañjarī, see Mishra
SB: Śābara Bhāṣya, see Mīmāṁsaka, 1990
VP: Vākyapadīyam, see Sharma, 1974
References


Yogyatā AS AN ABSENCE OF NON-CONGRUITY


Amrith Krishna, Bodhisattwa Prasad Majumder, Anil Kumar Boga, and Pawan Goyal

Abstract: This work presents the use of Adaptor Grammar, a non-parametric Bayesian approach for learning (Probabilistic) Context-Free Grammar productions from data. In Adaptor Grammar, we provide the set of non-terminals followed by a skeletal grammar that establishes the relations between the non-terminals in the grammar. The productions and the associated probability for the productions are automatically learnt by the system from the usages of words or sentences, i.e., the dataset. This facilitates the encoding of prior linguistic knowledge through the skeletal grammar and yet the tiresome task of finding the productions is delegated to the system. The system completely learns the grammar structure by observing the data. We call this approach the ‘Ekalavya’ approach. In this work, we discuss the effect of using Adaptor grammars for Sanskrit at word-level supervised tasks such as compound type identification and also in identifying the source and derived words from corpora for derivational nouns. In both of the works, we show the use of sub-word patterns learned using Adaptor grammar as effective features for their corresponding supervised tasks. We also present our novel approach of using Adaptor Grammars for handling Structured Prediction tasks in Sanskrit. We present the preliminary results for the word reordering task in Sanskrit. We also outline our plan for the use of Adaptor grammars for Dependency Parsing and Poetry to Prose Conversion tasks.
1 Introduction

The recent trends in Natural Language Processing (NLP) community suggest an increased application of black-box statistical approaches such as deep learning. In fact, such systems are preferred as there has been an increase in the performance of several NLP tasks such as machine translation, sentiment analysis, word sense disambiguation, etc. (Manning 2016). In fact, MIT Technology Review reported the following regarding Noam Chomsky’s opinion about the extensive use of ‘purely statistical methods’ in AI. The report says that “derided researchers in machine learning who use purely statistical methods to produce behavior that mimics something in the world, but who don’t try to understand the meaning of that behavior.” (Cass 2011).

Chomsky quotes, “It’s true there’s been a lot of work on trying to apply statistical models to various linguistic problems. I think there have been some successes, but a lot of failures. There is a notion of success ... which I think is novel in the history of science. It interprets success as approximating un-analyzed data.” (Pinker et al. 2011). Norvig (2011), in his reply to Chomsky, comes in defense of statistical approaches used in the community. Norvig lays emphasis on the engineering aspects of the problems that the community deals with and the performance gains achieved in using such approaches. He rightly attributes that, while the generative aspects of a language can be deterministic, the analysis of a language construct can lead to ambiguity. As probabilistic models are tolerant to noise in the data, the use of such approaches is often necessary for engineering success. It is often the case that the speakers of a language deviate from the laid out linguistic rules in usage. This can be seen as noise in the dataset, and yet the system we intend to build should be tolerant to such issues as well. The use of statistical approaches provides a convenient means of achieving the same. But, the use of statistical approaches does not imply discarding of the linguistic knowledge that we possess. Manning (2016) quotes the work of Paul Smolensky, “Work by Paul Smolensky on how basically categorical systems can emerge and be represented in a neural substrate (Smolensky and Legendre 2006). Indeed, Paul Smolensky arguably went too far down the rabbit hole, devoting a large part of his career to developing a new categorical model of phonology, Optimality Theory (Prince and Smolensky 1993).” This is an example where the linguistics and the statistical computational models had a successful synergy, fruitful for both the domains.
The Probabilistic Context-Free Grammars (PCFGs) provide a convenient platform for expressing linguistic structures with probabilistic prioritization of the structures they accept. It has been shown that PCFGs can be learned automatically using statistical approaches (Horning 1969). In this work, we look into Adaptor grammar (Johnson, T. L. Griffiths, and Goldwater 2007), a non-parametric Bayesian approach for learning grammar from the observations, say, sentences or word usages in the language. When given a skeletal grammar along with the fixed set of non-terminals, Adaptor grammar learns the right-hand side of the productions and the probabilities associated with them. The grammar does so just by observing the dataset provided to it, and hence the name ‘Ekalavya’ approach.

The use of Adaptor grammars for linguistic tasks provides the following advantages for a learning task.

1. Adaptor grammars in effect output valid PCFGs, which in turn are context-free grammars, and thus are valid for linguistic representations.

2. It helps to encode linguistic information which is already described in various formalisms via the skeletal grammars. Thus domain knowledge can effectively be used. The only restriction here might be that the expressive power of the grammar is limited to that of a Context-Free Grammar.

3. By leveraging the power of statistics, we can obtain the likelihood of various possible parses, in case of structural ambiguity during an analysis of a sentence.

4. While the proposed structures might not be as competitive in performance as with the black-box statistical approaches such as the deep learning approaches, the interpretability of the Adaptor grammar-based systems is a big plus. Grammar experts can look into the individual production rules learned by the system. This frees the experts from coming up with the rules in the first place. Additionally, by looking into the production rules, understandable to any domain expert with the knowledge of context-free grammars, it can be validated whether the system has learned patterns that are relevant to the task or not.
In Section 2, we discuss the preliminaries regarding Context-Free Grammars, Probabilistic CFGs, and Adaptor Grammar. In Section 3, we discuss the use of Adaptor grammars in various NLP tasks for different languages. We then describe the work performed in Sanskrit with Adaptor grammars in Section 4. We then discuss future directions in Sanskrit tasks, specifically for multiple structured prediction tasks.

2 Preliminaries - CFG and Probabilistic CFG

Context-Free Grammar was proposed by Noam Chomsky who initially termed it as phrase structure grammar. Formally, a Context-Free Grammar $G$ is a 4-tuple $(V, \Sigma, R, S)$, where $V$ is a set of non-terminals, $\Sigma$ is a finite set of terminals, $R$ is the set of productions from $V$ to $(V \cup \Sigma)^*$, where $*$ is the ‘Kleene Star’ operation. $S$ is an element of $V$ which is treated as the start symbol, which forms the root of the parse trees for every string accepted by the grammar. Using the notation $L_X$ for the language generated by non-terminal $X$, the language generated by the grammar $G$ is $L_S$.

\[
\begin{align*}
S & \rightarrow \text{HB} | \text{BC} \\
H & \rightarrow \text{‘Word1’} | \text{BD} \\
B & \rightarrow \text{‘Word2’} | \text{‘Word3’} | \text{‘Word4’} \\
C & \rightarrow \text{‘Word3’} | \text{‘Word5’} \\
D & \rightarrow \text{‘Word5’} | \text{‘Word2’}
\end{align*}
\]

$V = \{S,H,B,C,D\}$  
$\Sigma = \{\text{‘Word1’}, \text{‘Word2’}, \text{‘Word3’}, \text{‘Word4’}, \text{‘Word5’}\}$

\textbf{Figure 1}

\textit{An example of a Context Free Grammar}

The productions in Context-Free Grammars are often handcrafted by expert linguists. It is common to have large CFGs for many of the real-life NLP tasks. It is common that a given string can have multiple possible parses for the given grammar. This is due to the fact that a Context-Free Grammar contains all possible choices that can be produced from a given Non-terminal (O’Donnell 2015). The grammar neither provides a deterministic parse nor prioritizes the parses. This leads to structural ambiguity in the grammar. Probabilistic Context-Free Grammars (PCFGs) have been introduced to weigh the probable trees when the ambiguity arises, and thus provide a means for prioritizing the desired rules. A PCFG is a 5-tuple
(V, Σ, R, S, θ), where θ, denotes a vector of real numbers in the range of [0, 1] indexed by productions of R, subject to noting RX for the set of productions of X in R, for all X in V we require

$$\sum_{\rho \in RX} \theta_\rho = 1$$

Figure 2

Example of a Probabilistic Context Free Grammar corresponding to CFG shown in Figure 1

The probabilities associated with all the productions of a given non-terminal should add up to 1. The probability of a given tree is nothing but the product of the probabilities associated with the rules which are used to construct the tree. A given vector X denotes the parameters of a multinomial distribution that have the non-terminal X on their left-hand side (LHS) (O’Donnell 2015).

Note that PCFGs make two strong conditional independence assumptions (O’Donnell 2015):

1. The decision about expanding a non-terminal depends only on the non-terminal and the given distribution for that non-terminal. No other assumptions can be made.

2. Following from the first assumption, a generated expression is independent of other expressions.

There are numerous techniques suggested for the estimation of weights for the productions in PCFG. The Inside-Outside algorithm is a maximum likelihood estimation approach based on the unsupervised Expectation maximization parameter estimation method. Summarily, the algorithm starts by initializing the parameters with a random set of values and then iteratively
modifies the parameter values such that the likelihood of the training corpus is increased. The process continues until the parameter values converge, i.e., no more improvement of the likelihood over the corpus is possible.

Another way of estimating parameters is through the Bayesian Inference approach (Johnson, T. Griffiths, and Goldwater 2007). Given a corpus of strings \( s = s_1, s_2, \ldots, s_n \), we assume a CFG \( \mathcal{G} \) generates all the strings in the corpus. We take the dataset \( s \) and infer the parameters \( \theta \) using Bayes’ theorem

\[
P(\theta|s) \propto P_{\mathcal{G}}(s|\theta)P(\theta)
\]

where,

\[
P_{\mathcal{G}}(s|\theta) = \prod_{i=1}^{n} P_{\mathcal{G}}(s_i|\theta)
\]

Now, the joint posterior distribution for the set of possible trees \( t \) and the parameters \( \theta \) can be obtained by

\[
P(t, \theta|s) \propto P(s|t)P(t|\theta)P(\theta) = \prod_{i=1}^{n} P(s_i|t_i)P(t_i|\theta))P(\theta)
\]

To calculate the posterior distribution, we assume that the parameters in \( \theta \) are drawn from a known distribution termed as the prior. We assume that each non-terminal in the grammar has a given distribution which need not be the same for all. For a non-terminal, the multinomial distribution is indexed by the respective productions and since we use Dirichlet prior over here, each production probability \( \theta_{X \rightarrow \beta} \) has a corresponding Dirichlet parameter \( \alpha_{X \rightarrow \beta} \). Now, either through Markov Chain Monte Carlo Sampling approaches (Johnson, T. Griffiths, and Goldwater 2007) or through variational inference or a hybrid approach, the parameters are learnt (Zhai, Boyd-Graber, and Cohen 2014).

However, this approach as well does not deal with the real bottleneck, which is to come up with relevant rules which can solve a task for a given corpus. For large datasets, the CFGs could have a large set of rules and it is often cumbersome to come up with rules by experts alone. Non-Parametric Bayesian Approaches have been proposed as modifications for PCFGs. Roughly, the Non-parametric Bayesian approaches can be seen as learning a single model that can adapt its complexity to the data (Gershman and David M Blei 2012). The term non-parametric does not imply that
there are no parameters associated with the learning algorithm, but rather it implies that the number of parameters is not fixed, and increases with an increase in data or observations.

The most general version of learning PCFGs goes by the name of Infinite HMM or Infinite PCFG (Johnson 2010). In infinite PCFG, say for the model described in Liang et al. (2007), we are provided with a set of atomic categories and a combination of these categories as rules. Now, depending on the data, the learning algorithm learns the productions and the number of possible non-terminals along with the probabilities associated with them (Johnson 2010). Another variation that is popular with the Non-Parametric Grammar induction models is the Adaptor grammar (Johnson, T. L. Griffiths, and Goldwater 2007). Here, the number of non-terminals remains fixed and is set manually. But, the production rules and their corresponding probabilities are obtained by inference. The productions are obtained for a subset of non-terminals which are ‘adapted’, and it uses a skeletal grammar to obtain the linguistic structures.

An Adaptor Grammar is a 7-tuple $G = (V, \Sigma, R, S, \theta, A, C)$. Here $A \subseteq V$ denotes non-terminals which are adapted, i.e., productions for the non-terminals in $A$ will automatically be learnt from data. $C$ is the Adaptor set, where $C_X$ is a function that maps a distribution over trees $T_X$ to a distribution over distributions over $T_X$ (Johnson 2010).

![Figure 3](image)

**Figure 3**

*Example of an Adaptor Grammar. The non-terminals marked with an ‘@’ show that they are adapted. The productions will be learnt from data, where each production is a variable length permutation of subset of the elements in the alphabet set*

The independence assumptions that exist for PCFGs are not anymore valid in the case of Adaptor Grammars (Zhai, Boyd-Graber, and Cohen 2014). Here the non-terminal $X$ is defined in terms of another distribution $H_X$. Now the adaptors for each of the non-terminal $X$, $C_X$, can be based on Dirichlet Process or a generalisation of the same, termed as Pitman-Yor
Process. Here $TD_X(G_{Y_1}, G_{Y_2}, \ldots, G_{Y_m})$ is a distribution over all the trees rooted in the non-terminal $X$.

$$H_X = \sum_{X \rightarrow Y_1 \ldots Y_m \in R_X} \theta_{X \rightarrow Y_1 \ldots Y_m} TD_X(G_{Y_1}, G_{Y_2}, \ldots, G_{Y_m})$$

$$G_X \sim C_X(H_X)$$

### 3 Adaptor Grammar in Computational Linguistics

Adaptor Grammar has been widely used in multiple morphological and syntactic tasks for various languages. Adaptor Grammar has been initially shown for word segmentation task in English (Johnson, T. L. Griffiths, and Goldwater 2007). A sentence with no explicit word boundaries was given as observations and the task was to predict the actual words in the sentence. The task is similar to tasks for variable-length motif identification.

Adaptor Grammars has been introduced by Johnson, T. L. Griffiths, and Goldwater (2007) as a non-parametric Bayesian framework for performing inference of syntactic grammar of a language over parse trees. A PCFG (Probabilistic Context-Free Grammar) and an adaptor function jointly define an Adaptor grammar. The PCFG learns the grammar rules behind the data generation process and the adaptor function maps the probabilities of the generated parse trees to substantially larger values than of the same under the conditionally independent PCFG model.

Adaptor grammars have been very effectively used in numerous NLP related tasks. Johnson (2010) has drawn connections between topic models and PCFGs and then proposed a model with combined insights from adaptor grammars and topic models. While LDA defines topics projecting documents to lower-dimensional space, Adaptor grammar defines the distribution over trees. The author also projects a hybrid model to identify topical collocations using the power of PCFG encoded topic models. Adaptor grammars are also used in named entity structure learning. Zhai, Kozareva, et al. (2016) has used adaptor grammars for identifying entities from shopping-related queries in an unsupervised manner.

The word segmentation task is essentially identifying the individual words from a continuous sequence of characters. This is seen as a challenging task in computational cognitive science as well. Johnson (2008a) used Adaptor Grammar for word segmentation on the Bantu Language,
‘Sesotho’. The author specifically showed how the grammar with additional syllable structure yields a better F-score for word segmentation task than the usual collocation grammar. A similar study has been carried out by Kumar, Padró, and Oliver González (2015). The authors present the mechanism to learn complex agglutinative morphology with specific examples of three of four Dravidian languages, Tamil, Malayalam, and Kannada. Furthermore, the authors specifically have stressed upon the task of dealing with sandhi using finite-state transducers after producing morphological segment generation using Adaptor grammars. Adaptor grammar succeeds in leveraging the knowledge about the agglutinative nature of the Dravidian language but refrains from modeling the specific morphotactic regularities of the particular language. Johnson also demonstrates the effect of syllabification on word segmentation task using PCFGs (Johnson 2008b). Johnson further motivates the usability of the aforementioned unsupervised approaches for word segmentation and grammar induction tasks by extracting the collocalational dependencies between words (Johnson and Demuth 2010).

Due to their generalizability, Adaptor grammars have been used extensively in NLP. Hardisty, Boyd-Graber, and Resnik (2010) achieves state-of-the-art accuracy in perspective classification using adaptive Naïve Bayes model – the adaptor grammar-based non-parametric Bayesian model. Besides this, adaptor grammar has been proven to be effective in grammar induction (Cohen, David M Blei, and Smith 2010). Grammar induction is an unsupervised syntax learning task. The authors achieved considerable results along with the finding that the variational inference algorithm (David M. Blei, Kucukelbir, and McAuliffe 2017) can be extended to the logistic normal prior instead of the Dirichlet prior. Neubig et al. (2011) proposed an unsupervised model for phrase alignment and extraction where they claimed that their method can be thought of as an adaptor grammar over two languages. Zhai, Kozareva, et al. (2016) has presented a work, where the authors attempted to identify relevant suggestive keywords to a typed query so as to improve the results for search in an e-commerce site. The authors previously presented a new variational inference approach through a hybrid of Markov chain Monte Carlo and variational inference. It has been reported that the hybrid scheme has improved scalability without compromising the performance on typical common tasks of grammar induction.

Botha and Blunsom (2013) presented a new probabilistic model that extends Adaptor grammar to make it learn word segmentation and mor-
pheme lexicons in an unsupervised manner. Stem derivation in Semitic languages such as Arabic achieves better performance using this mildly context-sensitive grammar formalism. Again, Eskander, Rambow, and Yang (2016) recently investigated with Adaptor Grammars for unsupervised morphological segmentation to establish a claim of language-independence. Keeping aside other baselines such as morphological knowledge input from external sources and other cascaded architectures, adaptor grammar proved to be outperforming in a majority of the cases.

Another use of Adaptor grammar has been seen in the identification of native language (Wong, Dras, and Johnson 2012). Authors used adaptor grammar in identifying n-gram collocations of an arbitrary length over a mix of Parts of Speech tags and words to feed them as a feature in the classifier. By modeling the task with syntactic language models, the authors showed that extracted collocations efficiently represent the native language. Besides grammar induction, Huang, Zhang, and Tan (2011) further uses Adaptor grammar for machine transliteration. The PCFG framework helps to learn syllable equivalent in both languages and hence aids in the automatic phonetic translation. Furthermore, Feldman et al. (2013) recently explored a Bayesian model to understand how feedback from segmented words can alter the phonetic category learning of infants due to access to the knowledge of the joint occurrence of word-pairs.

As an extension to the standard Adaptor Grammar, O’Donnell (2015) presented Fragment Grammars which were built as a generalization of Adaptor Grammars. They generalize Adaptor Grammars by scoping the productivity and abstraction to occur at any point within individual stored structures. The specific model has adopted ‘stochastic memoization’ as an efficient substructure storing mechanism from the Adaptor grammar framework. It further memoizes partial internal computations via a lazy evaluation version of the original storage mechanism given by Adaptor Grammar.

4 Adaptor Grammar for Sanskrit

Adaptor Grammars have also been used for Sanskrit as well, mainly as a means of obtaining variable-length character n-grams to be used as features for classification tasks. Below, we describe two different applications, compound type identification, as well as identifying the Taddhita suffix for derivational nouns.
4.1 Variable Length Character n-grams for compound type identification

Krishna, Satuluri, Sharma, et al. (2016) used adaptor grammars for identifying patterns present in different types of compound words. The underlying task was, given a compound word in Sanskrit, to identify the type of the compound. The problem was a multi-class classification problem. The classifier needed to classify a given compound into one of the four broad classes, namely, *Avyayībhāva*, *Dvandva*, *Bahuvrīhi*, *Tatpuruṣa*.

The system is developed as an ensemble-based supervised classifier. We used the Random Forests classifier with an easy ensemble approach to handle the class imbalance problem persisting in the data. The classifier had a majority of its labels in *Tatpuruṣa*. The presence of *Avyayībhāva* was the least. The classifier incorporated rich features from multiple sources. The rules from *Aṣṭādhyayī* pertaining to compounds that are of conditional nature i.e. contains those containing selectional constraints were encoded as a feature. This was encoded by applying those selectional restrictions over the input compounds. Variable-length character n-grams for each class of compounds were obtained from adaptor grammar. Each filtered production from the compound class-specific grammar was used as a feature. We also incorporated noun pairs that follow the knowledge structure in *Amarakośa* as mentioned in Nair and A. Kulkarni (2010). We used a selected subset of relations from Nair and A. Kulkarni (2010).

We capture semantic class-specific linguistic regularities present in our dataset using variable-length character n-grams and character n-gram collocations shared between compounds using adaptor grammars.

We learn 3 separate grammars namely, G1, G2, and G3, with the same skeletal structure as Figure 4a, but with different data samples belonging to *Tatpuruṣa*, *Bahuvrīhi* and *Dvandva* respectively. We did not learn grammar for *Avyayībhāva*, due to insufficient data samples for learning the patterns. We use a ‘$’ marker to indicate the word boundary between the components, where the components were in sandhi split form. A ‘#’ symbol was added to mark the beginning and end of the first and the final components, respectively. We also learn a grammar G4, where the entire dataset is taken together along with additional 4000 random pair of words

---

1The work has been done as part of the compound type identification work published in Krishna, Satuluri, Sharma, et al. (2016). Please refer to the aforementioned work for a detailed explanation of the concepts described here.
from the Digital Corpus of Sanskrit, where none of the words appeared as a compound component in the corpus. The co-occurrence or the absence of it was taken as the proxy for compatibility between the components. The skeletal grammar in Figure 4b has two adapted non-terminals, both marked by ‘@’. Also, the adapted non-terminal ‘Word’ is a non-terminal appearing as a production to the adapted non-terminal ‘Collocation’. The ‘+’ symbol indicates the notion of one or more occurrence of ‘Word’, as used in regular expressions. This is not standard to use the notation in productions as per context-free grammar. This is ideally achieved using recursive grammars in CFGs with additional non-terminals. But, in order to present a simpler representation of skeletal grammar, we followed this scheme. In subsequent representations, we will be using recursiveness instead of the ‘+’ notation.

\[
\begin{align*}
@Collocation & \rightarrow \text{Word}^+ \\
@Word & \rightarrow \text{Phoneme}^+ \\
\text{Phoneme} & \rightarrow \{\text{Sanskrit Alphabet, S, #}\}
\end{align*}
\]

(a) Skeletal grammar for the adaptor grammar b) Derivation tree for an instance of a production ‘#sa$ śa’ for the non-terminal @Collocation

Every production in the learned grammars has a probability to be invoked, where the likelihood of all the productions of a non-terminal, sums to one. To obtain discriminative productions from G1, G2, and G3, we find conditional entropy of the productions with that of G4 and filter only those productions above a threshold. We also consider all the unique productions in each of the Grammars in G1 to G3. We further restrict the productions based on the frequency of the production in the data and the length of the sub-string produced by the production, both of them were kept at the value of three.

We show an instance of one such production for a variable-length character n-gram collocation. Here, for the adapted non-terminal @Collocation, we find that one of the production finally derives ‘#sa$ śa’, which actually is derived as two @Word derivations as shown in the Figure 4b. We use this as a regular expression, which captures some properties that need to be satisfied by the concatenated components. The particular production
mandates that the first component must be exactly $sa$, as it is sandwiched between the symbols # and $. Now, since $sa$ occurs after the previous substring which contains $ $ the boundary for both the components, $sa$ should belong to the second component. Now, since as per the grammar both the substrings are independent word productions, we relax the constraint that both the substrings should occur immediately one after the other. We treat the same as a regular expression, such that $sa$ should occur after $sa$, and any number of characters can come in between both the substrings. For this particular pattern, we had 22 compounds, all of those belonging to $Bahuvrihi$, which satisfied the criteria. Now, compounds where the first component is ‘$sa’ are mostly $Bahuvrihi$ compounds, and this is obvious to Sanskrit linguists. But here, the system was not provided with any such prior information or possible patterns. The system learned the pattern from the data. Incidentally, our dataset consisted of a few compound samples belonging to different classes as well where the first component was ‘$sa’.

4.1.1 Experiments

**Dataset** - We obtained a labeled dataset of compounds and the decomposed pairs of components from the Sanskrit studies department, UoHyd\(^2\). The dataset contains more than 32,000 unique compounds. The compounds were obtained from ancient digitised texts including Śrīmad Bhagavat Gītā, Caraka saṃhitā among others. The dataset contains the sandhi split components along with the compounds. With more than 75% of the dataset containing $Tatpurusa$ compounds, we down-sample the $Tatpurusa$ compounds to a count of 4000, to match with the second-highest class, $Bahuvrihi$. We find that the $Avyayībhāva$ compounds are severely under-represented in the data-set, with about 5% of the $Bahuvrihi$ class. From the dataset, we filtered 9,952 different data-points split into 7,957 data points for training and the remaining as a held-out dataset.

**Result** - To measure the impact of different types of features we incorporated, we train the classifier incrementally with different feature types. We report the results over the held-out data. At first, we train the system with only $Aṣṭādhyāyī$ rules and some additional hand-crafted rules. We find that the overall accuracy of the system is about 59.34%. Then we augmented the classifier by adding features from $Amarakośa$. We find that the overall accuracy of the system has increased to 63.81%. We then finally add the adaptor

\(^2\)http://sanskrit.uohyd.ac.in/scl/
<table>
<thead>
<tr>
<th>Class</th>
<th>P</th>
<th>R</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.92</td>
<td>0.43</td>
<td>0.58</td>
</tr>
<tr>
<td>B</td>
<td>0.85</td>
<td>0.74</td>
<td>0.79</td>
</tr>
<tr>
<td>D</td>
<td>0.69</td>
<td>0.39</td>
<td>0.49</td>
</tr>
<tr>
<td>T</td>
<td>0.68</td>
<td>0.88</td>
<td>0.77</td>
</tr>
</tbody>
</table>

Table 1

*Classwise performance of the Random Forests Classifier.*

grammar-based features which have increased the performance of the system to an accuracy of 74.98%. The effect of adding adaptor grammar features was more visible for the improvement in the performance of *Dvandva* and *Bahuvrīhi*. Notably, the precision for *Dvandva* and *Bahuvrīhi* increased by absolute values 0.15 and 0.06 respectively, when compared to the results before adding adaptor grammar-based features. Table 1 presents the result of the system with the entire feature set per Compound class. The addition of adaptor grammar features has resulted in an overall increase in the performance of the system from 63.81% to 74.91%. The patterns for adaptor grammar were learned only using the data from the training set and the held-out data was not used. This was done so as to ensure no over-fitting of data takes place. Also, we filtered the productions with length less than 3 and which do not occur many times in the grammar.

### 4.2 Distinctive Patterns in Derivational Nouns in Taddhita

Derivational nouns are a means of vocabulary expansion in a language. A new word is created in a language where an existing word is modified by an affix. Taddhita is a category of such derivational affixes which are used to derive a *prātipadika* from another *prātipadika*. The challenge here is to identify Taddhita *prātipadikas* from corpora in Sanskrit and also to identify their source words.

Pattern-based approaches often result in false positives. The edit distance, a popular distance metric to compare the similarity of two given strings, between the source and derived words due to the patterns tends to vary from 1 to 6. For example, consider the word ‘rāvaṇi’ derived from

---

3The work has been done as part of the Derivational noun word pair identification work published in Krishna, Satuluri, Ponnada, et al. (2017). Please refer to the aforementioned work for a detailed explanation of the concepts described here.
‘Ekala vyārāvaṇa’, where the edit distance between the words is just 1. But, ‘Āś-valāyana’ derived from ‘aśvala’ has an edit distance of 6. Also, the word ‘kālaśa’ is derived from the word ‘kalaśa’, but ‘kāraṇa’ is not derived from ‘karaṇa’. Similarly ‘stutya’ is derived from ‘stu’ but using a kṛt affix. But, dakṣinā (South direction) is used to derive dākṣhiṇātya (Southern) with a taddhita affix. If we have to use vṛddhi as an indicator, which is the only difference between both the examples, then there are cases such as kāraka derived from kṛ for kṛt and aṣvaka is derived from aṣva using taddhita. All these instances show the level of ambiguity that can arise in deciding the pairs of source and derived words using taddhita. All the aforementioned examples show the need for knowledge of Aṣṭādhyāyī (or the knowledge of affixes), semantic relation between the word pairs or a combination of these to resolve the right set of word pairs.

The approach proposed in Krishna, Satuluri, Ponnada, et al. (2017) first identifies a high recall low precision set of word pairs from multiple Sanskrit Corpora based on pattern similarities as exhibited by the 137 affixes in Taddhita. Once the patterns are obtained, we look for various similarities between the word pairs to group them together. We use rules from Aṣṭādhyāyī, especially from the Taddhita section. But since we could not incorporate rules of semantic and pragmatic nature, to compensate for the missing rules, we tried to identify patterns from the word pairs, specifically the source words, to be used. We use Adaptor Grammar for the purpose.

Currently, we do not identify the exact affix that leads to the derivation of the word. Also, since the affixes are distinguished not just by the visible pattern, but also by the ‘it’ markers, it is challenging to identify the exact affix. So, we group all those affixes that result in similar patterns into a single group. All the word pairs that follow the same pattern belong to one group. To further increase the group size, we group all those entries that differ by vṛddhi and guṇa also into the same group. Such distinctions are not considered while forming a group. Effectively we only look into the pattern at the end of the ‘derived word’. We call all such collection of groups based on the patterns as our ‘candidate set’.

For every distinct pattern in our candidate set, we first identify the word pairs and then create a graph with the given word pairs. A word pair is a node and edges are formed between nodes where they match different set of similarities. The first set of similarities are based on rules directly from Aṣṭādhyāyī, while the second set of node similarities were using character n-grams using Adaptor grammars. Once the similarities were found, we
apply the Modified Adsorption approach (Talukdar and Crammer 2009) on the graph. The modified adsorption is a semi-supervised label proration approach where labels are provided to a subset of nodes and then propagated to the remaining nodes based on the similarity they share with other nodes.

Figure 5 shows a sample construction of the graph for the word pairs, where words differ by a pattern ‘ya’. Here every pair obtained by pattern matching is a node. Now, Modified Adsorption is a semi-supervised approach. So, we need a limited number of labeled nodes. The nodes marked in grey are labeled nodes. They are called as seed nodes. The label here is just binary, i.e. a word pair can either be a true Taddhita pair or not. Now, edges are formed between the word pairs. Modified Adsorption provides a means of designing the graph explicitly, while many of its predecessors relied more on nearest-neighbor based approaches (Zhu and Ghahramani 2002). Also, the edges can be weighted based on the closeness between different nodes. Once the graph structure is defined, we perform the modified adsorption. In this approach, the labels from the seed nodes are propagated through the edges, such that the labels from seed nodes are propagated to other unlabelled nodes as well. The highly similar nodes should be given similar labels or else the optimization function penalizes any other label assignments. We use three different means of obtaining similarities between the nodes. The first such set of similarity is the rules in Aṣṭādhyāyī that the pair of nodes have a match with. The second set of similarity is the sum of probabilities of productions from adaptor grammar, which are matched for a pair of nodes. The third is the word vector similarity between the source words in the node pairs. For a detailed working of the system and a detailed explanation of each set of features please refer to Krishna, Satuluri, Ponnada, et al. (2017). Here, we republish the working of the second set of features obtained using Adaptor grammar and the results of the model thereafter.

Character n-grams similarity by Adaptor Grammar - Pāṇini had an obligation to maintain brevity, as his grammar treatise was supposed to be memorized and recited orally by humans (Kiparsky 1994). In Aṣṭādhyāyī, Pāṇini uses character sub-strings of varying lengths as conditional rules for checking the suitability of the application of an affix. We examine if there are more such regularities in the form of variable-length character n-grams that can be observed from the data, as brevity is not a concern for us. Also, we assume this would compensate for the loss of some of the information which Pāṇini originally encoded using pragmatic rules. In order to identify
‘Ekalavya’ Approach

Figure 5

Graph structure for the group of words where derived words end in ‘ya’. Nodes in grey denote seed nodes, where they are marked with their class label. The Nodes in white are unlabelled nodes.

the regularities in the pattern in the words, we use Adaptor grammar.

In Listing 1, ‘Word’ and ‘Stem’ are non-terminals, which are adapted. The non-terminal ‘Suffix’ consists of the set of various end-patterns.

| Word → Stem Suffix |
| Word → Stem |
| Stem → Chars |
| Suffix → a|ya|......|Ayana |

Listing 1: Skeletal CFG for the Adaptor grammar

The set $A_2$ captures all the variable-length character n-grams learned as the productions by the grammar along with the probability score associated with the production. We form an edge between two nodes in $G_{i2}$, if there exists an entry in $A_2$, which are present in both the nodes. We sum the probability value associated with all such character n-grams common to the pair of nodes $v_j, v_k \in V_i$, and calculate the edge score $\tau_{j,k}$. If the edge score
is greater than zero, we find the sigmoid of the value so obtained to assign the weight to the edge. The expression for calculating $\tau_{j,k}$ in the equation given below uses the Iverson bracket (Knuth 1992) to show the conditional sum operation. The equation essentially makes sure that the probabilities associated with only those character n-grams get summed, which are present in both the nodes. We define the edge score $\tau_{j,k}$, weight set $W_{i2}$ and Edge set $E_{i2}$ as follows.

$$\tau_{j,k} = \sum_{l=1}^{\mid A_2 \mid} a_{k2,l} [a_{k2,l} = a_{j2,l}]$$

$$E_{i2}^{v_k,v_j} = \begin{cases} 
1 & \tau_{j,k} > 0 \\
0 & \tau_{j,k} = 0
\end{cases}$$

$$W_{i2}^{v_k,v_j} = \begin{cases} 
\sigma(\tau_{j,k}) & \tau_{j,k} > 0 \\
0 & \tau_{j,k} = 0
\end{cases}$$

As mentioned, we use the label distribution per node obtained from phase 1 as the seed labels in this setting.

### 4.2.1 Experiments

As we mentioned, we use three different set of similarity sets for weighting the edges. But, in Modified Adsorption (MAD) the edge weight requires to be a scalar. This implies a similarity score between a pair of nodes using one similarity function can be used at a time. hence, we chose to apply the similarity weights sequentially on the graph. An alternative would have been to obtain a weighted average of the different similarity scores. But, our pipeline approach can be seen as a means of bootstrapping our seeds set. In Modified Adsorption, we need to provide seed labels, which are labels for some of the nodes. In reality, the seed nodes do not have a binary assignment of the labels, rather a distribution of the labels (Talukdar and Crammer 2009). So after the run of each similarity set, we get a label distribution for each of the nodes in the graph. This label distribution is used to generate seed nodes in the subsequent run of the modified adsorption. The seed nodes also get modified during the run of the algorithm.

**Dataset** - We use multiple lexicons and corpora to obtain our vocabulary $C$. We use IndoWordNet (M. Kulkarni et al. 2010), the Digital Corpus of
‘Ekalavya’ Approach

Sanskrit⁴, a digitized version of the Monier Williams⁵ Sanskrit-English dictionary, a digitized version of the Apte Sanskrit-Sanskrit Dictionary (Goyal, G. P. Huet, et al. 2012) and we also utilize the lexicon employed in the Sanskrit Heritage Engine (Goyal and G. Huet 2016). We obtained close to 170,000 unique word lemmas from the combined resources.

Results - In Krishna, Satuluri, Ponnada, et al. (2017), we report results from 11 of the patterns from a total of more than 80 patterns we initially obtained. Due to the lack of enough evidence in the form of data-points we did not attempt the procedure for others. Here, we only show results for 5 of the patterns, which were selected based on the size of evidence from the corpora we obtain. Since we use each of the similarity set sequentially, we have outputs at each of the phase of the sequences. The result of the system after incorporating Aṣṭādhyāyī rules is MAD₁, while that after incorporating Adaptor grammar ngrams is MAD₂ and the final result after the word vector similarity is MAD. Now, since we have 5 different patterns, we have an index i sub-scripted to the systems to denote the corresponding patterns. We additionally use a baseline called as Label Propagation (LP), based on the algorithm by Zhu and Ghahramani (2002). We can find that the systems which incorporates adaptor grammar are the MAD and MAD₂. Both the systems are the best and second best performing systems respectively.

Table 2 shows the results of our system. We compare the performance of 5 different patterns, selected based on the number of candidate word pairs available for the pattern. The system proposed in the work MADᵢ performs the best for all the 5 patterns. Interestingly, MAD₂ᵢ is the second best-performing system in all cases. The system uses 3 kinds of similarity measures in a sequential pipeline of which adaptor grammar comes as the second feature set. To understand the impact of adding adaptor grammar-based features, we can compare the results with that of MAD₁ᵢ. The system shows the result for each of the patterns before using adaptor grammar-based features.

A baseline using the label propagation algorithm was also used. The motive behind the label propagation baseline was to measure the effect of Modified adsorption on the task. In Label Propagation, we experimented with the parameter K with different values, K ∈ {10, 20, 30, 40, 50, 60}, and found that K = 40, provides the best results for 3 of the 5 end-patterns.

⁴http://kjc-sv013.kjc.uni-heidelberg.de/dcs/
⁵http://www.sanskrit-lexicon.uni-koeln.de/monier/
The values for $K$ are set by empirical observations. We find that for those 3 patterns ('a', 'in', 'i'), the entire vertex set has $vṛddhi$ attribute set to the same value. For the other two ('ya', 'aka'), $K = 50$ gave the best results. Here, the vertex set has nodes where the $vṛddhi$ attribute is set to either of the values. For a better insight towards this finding, the notion of the pattern that we use in the design of the system needs to be elaborated. A pattern is effectively the substrings that remain in both the source word and derived word after removing the portions which are common in both. This pattern is the visible change that happens in the derivation of a word. To reduce the number of distinct patterns we did not consider the pattern changes that occur due to $vṛddhi$ and $guṇa$ as distinct patterns, rather we abstracted them out. Now, multiple affixes may lead to the generation of the same set of patterns. In the case of pattern, rather end-pattern, (Krishna, Satuluri, Ponnada, et al. 2017), ‘a’, the effect may be the result of application of one of the following affixes such as $aṇ̱ aṇ$ etc. Here, all the affixes of pattern ‘a’
lead to vrddhi. But for the pattern ‘ya’, the affixes may or may not lead to a vrddhi. We report the best result for each of the system in Table 2.

5 Inference of Syntactic Structure in Sanskrit

In this section, we are reporting an ongoing work, where we investigate the effectiveness of using Adaptor grammar for inference of syntactic structures in Sanskrit. We experiment with the effect of Adaptor Grammar in capturing the ‘natural order’ or the word order followed in prose. For this task, we use a dataset of Sanskrit sentences which are in prose order. The dataset consists of 2000 sentences from Pañcākhyānaka and more than 600 sentences from Mahābhārata,. For this experiment, we only consider the morphological classes of the words involved in the sentences. Currently, we use the morphological tags as used in the Sanskrit Library. We keep 500 of the sentences for testing and the remaining 2000 are used for identifying the patterns. Some of the constructs had one or two words, which we ignore for the experiment.

We learn the necessary productions in grammar and then evaluate the grammar on the 500 test sentences. We calculate the likelihood of generating each of the sentences. In order to test the likelihood of the correct sentence, we also generate all possible permutations of the morphological tags in each of the test sentences. For sentences of length > 5, we break them into sub-sequences of 5 and find the permutations of the sub-sequences and concatenate them again. This is used as a means of sampling the possible combinations as the explicit enumeration of all the permutations are computationally costly. From the generated candidate set we find the likelihood of the ground truth sentence and rank them. We report our results based on two measures.

1. **Edit Distance (ED)** - The edit distance of the top-ranked sentence among the candidate set for a given sentence with that of the ground truth. Edit distance is roughly described as the minimum number of operations required to convert one string to another based on a fixed set of operations with predefined costs. We use the standard Levenshtein distance (Levenshtein 1966), where the three operations are ‘insert’, ‘delete’ and ‘substitution’. All the 3 operations have a

http://sanskritlibrary.org/helpmorphids.html
cost of 1. We compare the ground truth sentence with the predicted sentence that has the highest likelihood to obtain the measure. The predicted sentences with a lower Edit Distance implies a better result.

2. **Mean Reciprocal Rank (MRR)** - Mean Reciprocal Rank is the average of reciprocal ranks for each of the queries. Here a test sentence is treated as a query. The different permutations are the retrieved results for the query. So from the ranked retrieved list, we find the inverse of the rank of the gold standard sentence. The better the MRR Score, the better the result.

\[
\frac{1}{|Q|} \sum_{i=1}^{\left|Q\right|} \frac{rel_i}{rank_i}
\]

We first attempt the same skeletal grammars as proposed by Johnson, T. L. Griffiths, and Goldwater (2007) for capturing the syntactic regularities. We used both the ‘unigram’ and ‘collocation’ grammar as mentioned in the work. Figures 6 and 7 show the first two grammars that we have used for the task.

![Figure 6](image)

*Unigram grammar as used in Johnson, T. L. Griffiths, and Goldwater (2007)*

With these grammars, we experimented with various hyper-parameter settings. Since both the grammars are right recursive grammars, the length of the productions so learnt from the grammar varied greatly. Though this is beneficial for identifying the word lengths, the association with the morphological tags cannot be much longer. Secondly, the number of productions to be learnt is user-defined hyper-parameter. We find that due to the possible varying length size of strings and fewer observations, the main morphological patterns that were learnt as the productions were not repeated enough in the observations to be statistically significant.
We modified both the grammars to restrict the length of the productions to a maximum of 4 and limited the number of productions to be learnt. We show the modification done to the adapted non-terminal ‘word’ in both the grammars. This restricts the number of productions that ‘word’ can learn. The modified portion can be seen in Figure 8.

The results for all the four grammars are shown in Table 3. It can be seen that there is considerable improvement in the Mean Reciprocal Rank and the edit distance measures for the task with the restricted grammar. On our manual inspection of the patterns learnt from all the grammars, it was observed that the initial skeletal grammars were essentially over-fitting the training instances due to longer lengths. The modified grammars could reduce the Edit distance to almost half and double the Mean Reciprocal Rank.
Table 3

Results for the word reordering task.

<table>
<thead>
<tr>
<th>Grammar</th>
<th>MRR</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unigram</td>
<td>0.2923</td>
<td>4.87</td>
</tr>
<tr>
<td>Collocation</td>
<td>0.3016</td>
<td>4.66</td>
</tr>
<tr>
<td>Modified Unigram</td>
<td>0.4025</td>
<td>3.21</td>
</tr>
<tr>
<td>Modified Collocation</td>
<td>0.5671</td>
<td>2.20</td>
</tr>
</tbody>
</table>

Rank for the task.

For example, consider the sentence ‘tatra budhaḥ vrata caryā samāptau āgacchat (ā agacchat)’ from Mahābhārata. Consider the corresponding sequence of morphological tags as shown, ‘i m1s iic f3s f7s i ipf[1]_a3s’. We filter out the ‘iic’ tags as the ‘iic’ tag stands for the compound component. It can be seen as part of the immediate next noun tag following it. We do not filter out the ‘i’ tags as of now, where ‘i’, stands for the indeclinable. So in effect the tag sequence is ‘i m1s f3s f7s i ipf[1]_a3s’. The ‘Collocation’ Grammar had the following sequence as the most likely output ‘i f7s i m1s f3s ipf[1]_a3s’ with an edit distance of 4. In the ‘Modified Collocation’ Grammar the predicted sequence is ‘i m1s f3s i f7s ipf[1]_a3s’. The edit distance of the sentence is 2. Here, it can be seen that just 2 tags have swapped their position. The tags ‘i’ and ‘f7s’ have changed their positions, but are still at adjacent positions to each other. The fourth and fifth words in the original sentence have changed to become the fifth and fourth words in the predicted sentence.

The results shown here are preliminary in nature. What excites us the most is the provision this framework provides to incorporate the syntactic knowledge which is explicitly defined in our grammar formalisms. With this work, we plan to extend the work to two immediate tasks. First, we plan to extend the word-reordering task to the poetry to prose conversion task. Currently, the task is to convert a bag of words into its corresponding prose or the ‘natural order’. But we will investigate the regularities involved in poetry apart from the aspects of meter and incorporate the regularities to guide the grammar in picking up those patterns. We can also attempt to learn the conditional probabilities for the syntactic patterns in both poetry

\[\text{http://sanskritlibrary.org/helpmorphids.html}\]
and prose. Second, we will be performing the Dependency parse analysis of given sentences at a morphological level. Goyal and A. Kulkarni (2014) presents a scheme for converting Sanskrit constructs in constituency parse structure to Dependency parse structure. Headden III, Johnson, and McClosky (2009) provides some insights into the use of PCFGs and lexical evidence for unsupervised dependency parsing. Currently, we will be working only on the projective dependency parsing. We will be relying on the Dependency Model with Valence to define our PCFG formalism for dependency parsing.

6 Conclusion

The primary goal of this work was to look into the applicability of the Adaptor Grammars, a non-parametric Bayesian approach for learning syntactic structures from observations. In this work, we introduced the basic concepts of the Adaptor grammars, various applications in which the grammar is used in NLP tasks. We provide detailed descriptions of how adaptor grammar is used in word-level vocabulary expansion tasks in Sanskrit. The adaptor grammars were used as effective sub-word n-gram features for both Compound type identification and Derivational noun pair identification. We further showed the feasibility of using adaptor grammar for syntactic level analysis of sentences in Sanskrit. We plan to investigate the feasibility of using the Adaptor grammars for dependency parsing and poetry to prose conversion tasks at the sentence level.

Acknowledgements

The authors acknowledge the use of the morphologically tagged database of the Pañcākhyānaka and Mahābhārata produced under the direction of Professor Peter M. Scharf while laureate of a Blaise Pascal Research Chair at the Université Paris Diderot 2012–2013 and maintained by The Sanskrit Library.
References


Feldman, Naomi H, Thomas L Griffiths, Sharon Goldwater, and James L Morgan. 2013. “A role for the developing lexicon in phonetic category acquisition.” *Psychological review* 120.4p. 751.


Zhai, Ke, Zornitsa Kozareva, Yuening Hu, Qi Li, and Weiwei Guo. 2016. “Query to Knowledge: Unsupervised Entity Extraction from Shopping

A user-friendly tool for metrical analysis of Sanskrit verse

SHREEVATSA RAJAGOPALAN

Abstract: This paper describes the design and implementation of a tool that assists readers of metrical verse in Sanskrit (and other languages/literatures with similar prosody). It is open-source, and available online as a web application, as a command-line tool and as a software library. It handles both varṇavṛtta and mātrāvṛtta metres. It has many features for usability without placing strict demands on its users. These include allowing input in a wide variety of transliteration schemes, being fairly robust against typographical or metrical errors in the input, and “aligning” the given verse in light of the recognized metre.

This paper describes the various components of the system and its user interface, and details of interest such as the heuristics used in the identifier and the dynamic-programming algorithm used for displaying results. Although originally and primarily designed to help readers, the tool can also be used for additional applications such as detecting metrical errors in digital texts (its very first version identified 23 errors in a Sanskrit text from an online corpus), and generating statistics about metres found in a larger text or corpus. These applications are illustrated here, along with plans for future improvements.

1 Introduction

1.1 Demo

As a software tool is being discussed, it seems best to start with a demonstration of a potential user interaction with the tool. Suppose I wish to learn about the metre of the following subhāṣita (which occurs in the Pratijnāyaugandharāyaṇa attributed to Bhāsa):
Then I can visit the tool’s website, http://sanskritmetres.appspot.com, enter the above verse (exactly as above), and correctly learn that it is in the metre Śālinī. More interestingly, suppose I do not have the verse correctly: perhaps I am quoting it from memory (possibly having misheard it, and unaware of the line breaks), or I have found the text on a (not very reliable) blog, or some errors have crept into the digital text, or possibly I just make some mistakes while typing. In such a case, possibly even with an unreasonable number of mistakes present, I can still use the tool in the same way. Thus, I can enter the following error-ridden input (which, for illustration, is encoded this time in the ITRANS convention):

\[
\text{kāṣṭhād agnir jāyate mathya-mānād-bhūmis toyaṃ khanya-mānā dadāti} \\
\text{sotsāhānāṃ nāstyasādhyam narāṇāṃ mārgārabdhāḥ sarva-yatnāḥ phalanti} ||
\]

Here, some syllables have the wrong prosodic weight (laghu instead of guru and vice-versa), some syllables are missing, some have been introduced extraneously, not a single line of the input is correct, and even the total number of syllables is wrong. Despite this, the tool identifies the metre as Śālinī. The output from the tool, indicating the identified metre, and highlighting the extent to which the given verse corresponds to that metre, is shown in Figure 1. The rest of this paper explains how this is done, among other things.

1.2 Background

A large part of Sanskrit literature, in kāvya, śāstra and other genres, is in verse (padya) rather than prose (gadya). A verse in Sanskrit (not counting some modern Sanskrit poets’ experimentation with “free verse” and the like) is invariably in metre.

Computer tools to recognize the metre of a Sanskrit verse are not new. A script in the Perl programming language, called sscan, written by John Smith, is distributed among other utilities at the http://bombay.
Metrical analysis of Sanskrit verse

Figure 1
A screenshot of the output from the tool for highly erroneous input. Despite the errors, the metre is correctly identified as Śālinī. The guru syllables are marked in bold, and the deviations from the expected metrical pattern (syllables with the wrong weight, or missing or superfluous syllables) are underlined (and highlighted in red).
indology.info website, and although the exact date is unknown, the timestamp in the ZIP file suggests a date of 1998 or earlier for this file (Smith 1998?). This script, only 61 lines long (38 source lines not including comments and description) was the spark of inspiration that initiated the writing of the tool being described in the current paper, in 2013. Other software or programs include those by Murthy (2003?), by A. Mishra (2007) and by Melnad, Goyal, and P. M. Scharf (2015). A general introduction to metre and Sanskrit prosody is omitted in this paper for reasons of space, as the last of these papers (Melnad, Goyal, and P. M. Scharf 2015) quite excellently covers the topic.

Like these other tools, the tool being discussed in this paper recognizes the metre given a Sanskrit verse. It is available in several forms: as a web application hosted online at http://sanskrit-metres.appspot.com, as a commandline tool, and as a Python library; all are available in the source-code form at https://github.com/shreevatsa/sanskrit. It is being described here for two reasons:

1. It has some new features that I think will be interesting (see section 1.4), some of which distinguish it from other tools. The development of this tool has thrown up a few insights (see Section 4) which may be useful to others who would like to develop better tools in the future.

2. A question was raised about this tool (P. Scharf 2016), namely:

   “An open-source web archive of metrically related software and data can be found at https://github.com/shreevatsa/sanskrit with an interface at http://sanskritmetres.appspot.com/. The author and contributors to this archive and data were unknown at the time and not included in our literature review. No description of the extent, comprehensiveness, and effectiveness of the software has been found.”

I took this as encouragement that such a description may be desirable / of interest to others.

1.3 The intended user

The tool can be useful for someone trying to read or compose Sanskrit verses, and for someone checking a text for metrical errors. In other words, the tool
can be used by different kinds of users: a curious learner, an editor working with a text (checking verses for metrical correctness), a scholar investigating the metrical signature of a text, or an aspiring poet. To make these concrete, consider the following “user stories” as motivating examples.

- **Devadatta** is learning Sanskrit. He knows that Sanskrit verse is written in metre and that this is supposed to make it easier to chant or recite. But he knows very little about various metres, so that when he looks at a verse, especially one in a longer metre like Śārdūla-vikrīḍitam or Sragdharā, he cannot quickly recognize the metre. All he sees is a string of syllables, and he has no idea where to pause (yati), how to recite, or even where to break at pādās if they are not indicated clearly in the text he is reading. With this tool, these problems are solved, and he can focus on understanding and appreciating the poetry, now that he can read it aloud rhythmically and melodically and savor its sounds.

- **Chitralekha** is a scholar. She works with digital texts that, though useful to have, are sometimes of questionable provenance and do not always meet her standards of critical editions. Errors might have crept into the texts, and she has the idea that some instances of scribal emendation or typographical errors (such as omitted letters, extraneous letters, or transposed letters) are likely to cause metrical errors as well. With this tool, she can catch a great many of them (see Section 3). Sometimes, she is interested in questions about prosody itself, such as: what are all the metres used in this text? Which ones are the most common? How frequently does the poet X use a particular “poetic license” of scansion? What are the rules governing Anuṣṭubh (Śloka), typically? This tool can help her with such questions too.

- **Kamban** would like to write poetry, like his famous namesake. He has a good command of vocabulary and grammar and has some poetic imagination, but when he writes a verse, especially in an “exotic” (to him) metre, he is sometimes unsure whether he has got all the syllables right. With this tool, he enters his tentative attempt and sees whether anything is off. He knows that the metres will soon become second-nature to him and he will not need the tool anymore, but still he
wishes he could have more help—such as choosing his desired metre, and knowing what he needs to add to his partially composed verse.

With the names of these users as mnemonics, we can say that the tool can be used to discover, check, and compose metrical verse and facts about them.

1.4 User-friendly features

As mentioned earlier, the tool has several features for easing the user’s job:

1. It accepts a wide variety of input scripts (transliteration schemes). Unlike most tools, it does not enforce the input to be in any particular input scheme or system of transliteration. Instead, it accepts IAST, Harvard-Kyoto, and ITRANS transliteration, Unicode Devanāgarī, and Unicode Kannada scripts, without the user having to indicate which input scheme is used. The tool is agnostic to the input method used, as it converts all input to an internal representation based on SLP1 (P. M. Scharf and Hyman 2011). It is straightforward to extend to other scripts or transliteration methods, such as SLP1 or other Indic scripts.

2. It is highly robust against typographical errors or metrical errors in the verse that is input. This is perhaps the most interesting feature of the tool and is useful because the text in the “wild” is not always error-free.

3. It can detect the metre even from partial verses—even if the user is not aware that the verse one is looking up is incomplete.

4. Informative “display” of a verse in relation to the identified metre, by aligning the verse to the metre using a dynamic programming algorithm to find the best alignment.

5. Supports learning more about a metre, by pointing to other examples of the metre, and audio recordings of the metre being recited in several styles (where available).

6. Quick link to provide feedback (by creating an issue on GitHub), specific to the input verse being processed on the page.
Figure 2

A “data flow diagram” of the system’s operation. The rectangles denote different forms taken by the data; the ovals denote code that transforms (or uses, or generates) the data.

2 How it works

This section describes how the system works. At a high level, there are the following steps/components:

1. Metrical data, about many known metres. This has been entered into the system.

2. Building the Index (Pre-processing): from the metrical data, various indices are generated.

3. Detection and Transliteration: The input supplied by the user is examined, the input scheme detected, and transliterated into SLP1.

4. Scansion: The SLP1 text (denoting a set of phonemes) is translated into a metrical signature (a pattern of laghus and gurus).

5. Matching: The metrical signature is compared against the index, to identify the metre (or metres).

6. Display: The user’s input is displayed to the user, appropriately reformatted to fit the identified metre(s), and with highlighting of any deviations from the metrical ideal.

These steps are depicted in Figure 2, and described in more detail in the following subsections.
2.1 Metrical data

This is the raw data about all the metres known to the system. They are stored in the JSON format, so that they could be used by other programs too. In what follows, a metrical **pattern** is defined as string over the alphabet \{L, G\}, i.e., a sequence of symbols each of which is either L (denoting laghu or a light syllable) or G (denoting guru or a heavy syllable).

As described elsewhere (Melnad, Goyal, and P. M. Scharf 2015), there are two main types of metres, *varṇavṛtta* and *mātrāvṛtta* (note that (Murthy 2003) points out that the Śloka metre constitutes a third type by itself), with the former having three subtypes:

1. **samavṛtta** metres, in which all four pādas of a verse have the same metrical pattern,

2. **ardhasamavṛtta** metres, in which odd pādas have one pattern and even pādas another (so that the two halves of the verse have the same metrical pattern),

3. **viṣamavṛtta** metres, in which potentially all four pādas have different metrical patterns.

Correspondingly each metre’s characteristics are indicated in this system with the minimal amount of data necessary:

1. **samavṛtta** metres are represented by a list of length one (or for convenience, simply a string), containing the pattern of each of their pādas,

2. **ardhasamavṛtta** metres are represented by a list of length two, containing the pattern of the odd pādas followed by the pattern of the even pādas,

3. **viṣamavṛtta** metres are represented by a list of length four, containing the pattern for each of the four pās.

Additionally, with the pattern, *yati* can be indicated; also spaces can be added, which are ignored. The yati is ignored for identification, but used later for displaying information about the metre.

Here are some lines, as examples:
For mātrāvṛtta metres (those based on the number of morae: mātrās), the constraints are more subtle, and as not every syllable’s weight is fixed, there are so many patterns that fit each metre that it may not be efficient to generate and store each pattern separately. Instead, the system represents them by using a certain conventional notation, which expands to regular expressions. This notation is inspired by the elegant notation described in another paper (Melnad, Goyal, and P. M. Scharf 2015), and uses a particularly useful description of the Āryā and related metres available in a paper by Ollent Ollett (2012).

Here, 2 will be interpreted as the regular expression (G|LL) and 4 as the regular expression (GG|LG|GL|LLLL|LGL) – all possible sequences of laghus and gurus that are exactly 4 mātrās long. Note that with this notation, the frequently mentioned rule of “any combination of 4 mātrās except LGL (ja-gaṇa)” is simply denoted as 22, expanding to the regular expression (G|LL)(G|LL) which covers precisely the 4 sequences of laghus and gurus of total duration 4, other than LGL.
The number of metres “known” to the current system. Not too much should be read into the raw numbers as a larger number isn’t necessarily better; see Section 4.1.3 for why.

<table>
<thead>
<tr>
<th>Type of metre</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>samavṛtta</td>
<td>1242</td>
</tr>
<tr>
<td>ardhasamavṛtta</td>
<td>132</td>
</tr>
<tr>
<td>viṣamavṛtta</td>
<td>19</td>
</tr>
<tr>
<td>mātrāvṛtta</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1398</strong></td>
</tr>
</tbody>
</table>

Table 1

The data in the system was started with a hand-curated list of popular metres (Ganesh 2013). It was greatly extended with the contributions of Dhaval Patel, which drew from the Vṛttaratnākara and the work of Mishra (A. Mishra 2007). A few metres from these contributions are yet to be incorporated, because of reasons described in section 4.1.3. Overall, as a result of all this, at the moment we have a large number of known metres, shown in Table 1.

2.2 Metrical index

The data described in the previous section is not used directly by the rest of the program. Instead, it is first processed into data structures (which we can consider a sort of “index”) that allow for efficient lookup, even when the number of metres is huge. These enable the robustness to errors that is one of the most important features of the system. The indices are called PĀDA1, PĀDA2, PĀDA3, PĀDA4, ARDHA1, ARDHA2, and FULL. Each of these indices consists of an associative array (a Python dict) that maps a pattern (a “pattern” is a string over the alphabet \{L, G\}) to a list\(^1\) of metres that contain that pattern (at the position indicated by the name of the index), and similarly an array that maps a regular expression to the list of metres that contain it. For instance, ardha2 maps the second half of each known metre to that metre’s name. It is at this point that we also introduce laghu-

---

\(^1\)Why a list? Because different metres can share the same pāda, for instance. And there can even be multiple names for the same metre. See Section 4.1.3 later.
ending variants for many metres (see more in 4.1.2). Section 2.5 describes how these indices are used.

Although this index is generated automatically and not written down in code, the following hypothetical code illustrates some sample entries in the ardha2 index:

```python
ardha2_patterns = {
    # ...
    'GGGGGLGGLGGGGGLGGLGG': ['Śālinī'],
    # laghu variants for illustration.
    # In reality we don't add for Śālinī...
    'GGGGGLGGLGLGGGGGLGGLGG': ['Śālinī'],
    'GGGGGLGGLGGLGGLGGLG': ['Śālinī'],
    'GGGGGLGGLGLGGGGLGGLGL': ['Śālinī'],
    # ...
}
ardha2_regexes = {
    # ...
    "22 4 22" + "4 22 L 22 .": ['Āryā', 'Upagīti'],
    # ...
}
```

### 2.3 Transliteration

The first step that happens after users enter their input is automatic transliteration. Detecting the input scheme is based on a few heuristics. Among the input schemes initially supported (Devanāgarī, Kannada, IAST, ITRANS, and Harvard-Kyoto), the detection is done as follows:

- If the input contains any Kannada consonants and vowels, treat it as Kannada.

- If the input contains (m)any Devanāgarī consonants and vowels, treat it as Devanāgarī. Note that this should not be applied to other characters from the Devanāgarī Unicode block, such as the daṇḍa symbol, which are often used with other scripts too, as encouraged in the Unicode standard.

- If the input contains any of the characters आउरङ्गम्यम्भून्त्यद्धेश, treat it as IAST.
• If the input matches the regular expression

    aa|ii|uu|[RrLl]\^[Ii]|RR[Ii]|LL[Ii]|~N|Ch|~n|N\^|Sh|sh

treat it as ITRANS. Here, the Sh and sh might seem dangerous, but the consonant cluster स्ह is unlikely in Sanskrit.

• Else, treat the input as Harvard-Kyoto.

An option to explicitly indicate the input scheme (bypassing the automatic inference) could be added but has not seemed necessary so far. The input is transliterated into (a subset of) the encoding SLP1 (P. M. Scharf and Hyman 2011), which is used internally, as it has many properties suitable for computer representation of Sanskrit text. While the input is being transliterated according to the detected scheme, known punctuation marks (and line breaks) are retained, while all “unknown” characters that have not been programmed into the transliterator (such as control characters and accent marks in Devanāgarī) are ignored.

The exact details of how the transliteration is done are omitted here, as transliteration may be regarded as a reasonably well-solved problem by now. One point worth mentioning is that there are no strict input conventions. In other work (Melnad, Goyal, and P. M. Scharf 2015), a convention is adopted like:

    If the input text lacks line-end markers, it is assumed to be a single pāda and to belong to the samavṛtta type of metre

Such a scheme may be interesting to explore. For now, as much as possible, the system tries to assume an untrained user and therefore infer all such things, or try all possibilities.

2.4 Scan

The transliteration into SLP1 can be thought of as having generated a set of Sanskrit phonemes (this close relationship between phonemes and the textual representation is the primary strength of the SLP1 encoding). From these phonemes, scansion into a pattern of laghus and gurus can proceed directly, without bothering with syllabification (however, syllabification is still done, for the sake of the “alignment” described later in section 2.6). The rule for scansion is mechanical: initial consonants are dropped, and
each vowel is considered as a set along with all the non-vowels that follow it before the next vowel (or end of text) is found. If the vowel is long or if there are multiple consonants (treating anusvāra and visarga as consonants here, for scansion only) in this set, then we have a guru, else we have a laghu.

The validity of this method of scansion, with reference to the traditional Sanskrit grammatical and metrical texts, is skipped in this paper, as something similar has been treated elsewhere (Melnad, Goyal, and P. M. Scharf 2015). However, note that this is the “purist” version of Sanskrit scansion. There is an issue of śithila-dvitva or poetic licence, which is treated in more detail in Section 4.3.

2.5 Identification

The core of the tool’s robust metre identification is an algorithm for trying many possibilities for identifying the metre of the input text. Identifying the metre given a metrical pattern (the result of scansion) is done in two steps: (1) first the input is broken into several “parts” in various ways, and then (2) each of these parts is matched against the appropriate indices.

2.5.1 Parts

Given the metrical pattern corresponding to the input text, which may be either a full verse, a half-verse, or a single quarter-verse (pāda), we try to break it into parts in multiple ways. One way of breaking the input, which should not be ignored, is already given by the user, in the form of line breaks in the input. If there are 4 lines, for example, it is a strong possibility that these are the 4 pādas of the verse. If there are 2 lines, each line may contain two pādas. But what if there are 3 lines, or 5? Another way of breaking the input is by counting syllables. If the number of syllables is a multiple of 4 (say 4n), it is possible that every n syllables constitute a pāda of a samavṛtta metre. But what if the number of syllables is not a multiple of 4?

The solution adopted here is to consider all ways of breaking a pattern into k parts even when its length (say l) may not be a multiple of k. Although this would apply to any positive k, we only care about k = 4 and k = 2, so let’s focus on the k = 4 case for illustration. In that case, suppose that the length l leaves a remainder r when divided by 4, that is,

\[ l \equiv r \pmod{4}, \quad 0 \leq r < 4 \]
or in other words $l$ can be written as $l = 4n + r$ for some integer $n$, where $0 \leq r < 4$. Then, as $\lfloor l/4 \rfloor = n$ (here $\lfloor \cdot \rfloor$ denotes the “floor function”, or integer division with rounding down), we can consider all the ways of breaking the string of length $l$ into 4 parts of lengths $(n+a, n+b, n+c, n+d)$ where $a + b + c + d = r$ (in words: we consider all ways of distributing the remainder $r$ among the 4 parts). For example, when $r = 2$, we say that a string of length $4n + 2$ can be broken into 4 parts in 10 ways:

$$(n, n, n, n + 2)$$

$$(n, n, n + 1, n + 1)$$

$$(n, n, n + 2, n)$$

$$(n, n + 1, n, n + 1)$$

$$(n, n + 1, n + 1, n)$$

$$(n, n + 2, n, n)$$

$$(n + 1, n, n, n + 1)$$

$$(n + 1, n, n + 1, n)$$

$$(n + 1, n + 1, n, n)$$

$$(n + 2, n, n, n)$$

Similarly, there are 4 ways when $r = 1$, 20 ways when $r = 3$, and of course there is exactly one way $(n, n, n, n)$ when $r = 0$.

In this way, we can break the given string into 4 parts (in 1, 4, 10, or 20 ways) or into 2 parts (in 1 or 2 ways), either by lines or by syllables. For instance, if we are given an input of 5 lines, then there are 4 ways we can break it into 4 parts, by lines. What we do with these parts is explained next.

### 2.5.2 Lookup/match

Once we have the input broken into the appropriate number of parts (based on whether we’re treating it as a full verse, a half verse, or a pāda), we look up each part in the appropriate index. For a particular index, to match against patterns is a direct lookup (we do not have to loop through all patterns in the index). To match against regexes, we do indeed loop through all regexes, which are fewer in number compared to the number of patterns. If needed, we can trade-off time and memory here; for instance, we could have indexed a large number of instantiated patterns instead of regexes even for mātrā
Metrical analysis of Sanskrit verse

<table>
<thead>
<tr>
<th>kind of index</th>
<th>treating input as</th>
<th>full verse</th>
<th>half verse</th>
<th>single pāda</th>
</tr>
</thead>
<tbody>
<tr>
<td>pāda1</td>
<td>first part of 4</td>
<td>first part of 2</td>
<td>the full input</td>
<td></td>
</tr>
<tr>
<td>pāda2</td>
<td>second part of 4</td>
<td>second part of 2</td>
<td>the full input</td>
<td></td>
</tr>
<tr>
<td>pāda3</td>
<td>third part of 4</td>
<td>first part of 2</td>
<td>the full input</td>
<td></td>
</tr>
<tr>
<td>pāda4</td>
<td>fourth part of 4</td>
<td>second part of 2</td>
<td>the full input</td>
<td></td>
</tr>
<tr>
<td>ardha1</td>
<td>first part of 2</td>
<td>the full input</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ardha2</td>
<td>second part of 2</td>
<td>the full input</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>full</td>
<td>the full input</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Table 2

What to match or look up, depending on how the input is being treated. Everywhere in the table above, phrases like “first part of 4” mean both by lines and by syllables. For instance, when treating the input as a full verse, the first \( \frac{1}{4} \) part by lines and the first \( \frac{1}{4} \) part by syllables are both matched against the pāda1 index.

metres. Note that in this way, to match an ardhasamavṛtta or a viṣamavṛtta that has been input perfectly, we search directly for the full pattern (of the entire verse) in the index. We do not have to run a loop for breaking a line into pādas in all possible ways, as in (Melnad, Goyal, and P. M. Scharf 2015). Details of which indices are looked up are in Table 2.

2.6 Align/Display

The metre identifier, from the previous section, results in a list of metres that are potential matches to the input text. Not all of them may match the input verse perfectly; some may have been detected based on partial matches. Whatever the reason for this imperfect match (an over-eager matching on the part of the metre identifier, or errors in the input text), it would be useful for the user to see how closely their input matches a given metre. And even when the match is perfect, aligning the verse to the metre can help highlight the pāda breaks, the location of yati, and so on. This is done by the tool, using a simple dynamic-programming algorithm very similar to the standard algorithm for the longest common subsequence problem: in effect, we simply align both the strings (the metrical pattern of
the input verse, and that of the known metre) along their longest common subsequence.

What this means is that given two strings \( s \) and \( t \), we use a dynamic programming algorithm to find the minimal set of “gap” characters to insert in each string, such that the resulting strings match wherever both have a non-gap character (and never have a gap character in both). For example:

\[
\begin{align*}
('abcab', 'bca'), & \Rightarrow ('abcab', '-bca-') \\
('hello', 'hello'), & \Rightarrow ('hello', 'hello') \\
('hello', 'hell'), & \Rightarrow ('hello', 'hell-') \\
('hello', 'ohell'), & \Rightarrow ('-hello', 'ohell-') \\
('abcdabcd', 'abcd'), & \Rightarrow ('abcdabcd', 'abcd----') \\
('abcab', 'acb'), & \Rightarrow ('abcab', 'a-c-b') \\
('abcab', 'acbd'), & \Rightarrow ('abcab\-', 'a-c-bd')
\end{align*}
\]

We use this algorithm on the verse pattern and the metre’s pattern, to decide how to align them. Then, using this alignment, we display the user’s input verse in its display version (transliterated into IAST, and with some recognized punctuation retained). Here, laghu and guru syllables are styled differently in the web application (styling customizable with CSS). This also highlights each location of yati or caesura (if known and stored for the metre), so that the user can see if their verse violates any of the subtler rules, such as words straddling yati boundaries.

This algorithm could also be used for ranking the results, based on the degree of match between the input and each result (metre identified).

3 Text analysis and results

As part of testing the tool (and as part of pursuing the interest in literature and prosody that led to the tool in the first place), a large number of texts such as from GRETIL\(^2\) were examined. Although primarily designed to help readers, the tool can also be used to analyze a metrical text, to catch errors or generate statistics about the metres used. In the very first version of the tool, the first metre added was Mandākrāntā, and the tool was run on a text of the Meghadūta from GRETIL, the online corpus of Sanskrit texts. This text was chosen because the Meghadūta is well-known to be entirely

\(^2\)Göttingen Register of Electronic Texts in Indian Languages: and related Indological materials from Central and Southeast Asia, \url{http://gretil.sub.uni-goettingen.de}
in the Mandākrāntā metre, so the “gold standard” to use as a reference to compare against was straightforward. Surprisingly, this tool successfully identified 23 errors in the 122 verses!\(^3\) These were communicated to the GRETIL maintainer.

Similarly, testing of the tool on other texts highlighted many errors. Errors identified in the GRETIL text of Bhartṛhari’s Śatakaṭravaya were carefully compared against the critical edition by D. D. Kosambi.\(^4\) In this text, as in Nilakaṇṭha’s Kalī-vidambana,\(^5\) in Bhallaṭa’s Bhallaṭa-śataka,\(^6\) and in almost all cases, the testing highlighted errors in the text, rather than any in the metre recognizer. This constitutes evidence that the recognizer has a high accuracy approaching 100%, though the lack of a reliable (and nontrivial) “gold standard” hinders attaching a numeric value to the accuracy. In the terminology of “precision and recall”, the recognizer has a recall of 100% in the examples tested (for example, no verse that is properly in Śārdūla-vikrīḍitam is failed to be recognized as that metre), while the precision was lower and harder to measure because of errors in the input (sufficiently many errors can make the verse partially match a different metre).

After sufficiently fixing the tool and the text so that Meghadūta was recognized as being 100% in the Mandākrāntā metre, other texts were examined. These statistics\(^7\) confirmed that, for example, the most common metres in the Amaruśataka are Śārdūlavikrīḍitam (57%), Hariṇī (13%) and Śikhariṇī (10%), while those in Kālidāsa’s Raghuvamśa are Śloka, Upajāti and Rathoddhatā. And so on. Once errors in the texts are fixed, this sort of analysis can give insights into the way different poets use metre. It can also be used for students to know which are the most common metres to focus on learning, at least for a given corpus. Other sources of online texts, like

\(^{3}\)See a list of 23 errors and 3 instances of broken sandhi detected in one of the GRETIL texts of the Meghadūta, at https://github.com/shreevatsa/sanskrit/blob/f2ef7364/meghdk_u_errors.txt (October 2013).

\(^{4}\)See https://github.com/shreevatsa/sanskrit/blob/7c42546/texts/gretil_stats/diff-bharst_u.htm-old.patch for a list of errors found, in diff format, with comments referring to the location of the verse in Kosambi

\(^{5}\)https://github.com/shreevatsa/sanskrit/blob/08ccb91/texts/gretil_stats/diff-nkalivpu.htm.patch

\(^{6}\)https://github.com/shreevatsa/sanskrit/blob/67251bc/texts/gretil_stats/diff-bhall_pu.htm.patch

\(^{7}\)http://sanskritmetres.appspot.com/statistics
TITUS, SARIT\textsuperscript{8} or The Sanskrit Library\textsuperscript{9} could also be used for testing the system.

4 Interesting issues and computational experience

Some insights and lessons learned as a result of this project are worth highlighting, as are some of the design decisions that were made either intentionally or unconsciously.

4.1 Metrical data

4.1.1 The gaṇa-s

For representing the characteristics of a given metre, a popular scheme used by all Sanskrit authors of works on prosody is the use of the 8 gaṇas. Each possible laghu-guru combination of three syllables (trīka), namely each of the $2^3$ possibilities LLL, LLG, LGL, LGG, GLL, GLG, GGL, GGG, is given a distinct name (na, sa, ja, ya, bha, ra, ta, ma respectively), so that a long pattern of laghus and gurus can be concisely stated in groups of three. This is an excellent mnemonic and space-saving device, akin to writing in octal instead of binary. For instance, the binary number 110110010101\textsubscript{2} can be written more concisely as the octal number 6625\textsubscript{8} and the translation between them is immediately apparent (110110010101\textsubscript{2} corresponds to 6625\textsubscript{8} and vice-versa, by simply treating each group of three binary digits (bits) as an octal digit, or conversely expanding each octal digit to a three-bit representation). Similarly, the pattern GGLGLLGLGLG of Indravamśa can be more concisely expressed by the description as “ta ta ja ra”. Moreover, another mnemonic device of unknown origin uses a string “yamātārājabhānasalagaṃ” that traverses all the 8 gaṇas (and the names la and ga used for any “leftover” laghus and gurus respectively), assigning them syllable weights (via vowel lengths) such that the three syllables starting at any of the 8 consonants are itself in the gaṇa named by that consonant.\textsuperscript{10}

Thus we can see that the gaṇa names are a useful mnemonic and space-saving device, and yet at the same time, from an information-theoretic point of view, they contain absolutely no information that is not present

\textsuperscript{8}http://sarit.indology.info
\textsuperscript{9}http://sanskritlibrary.org
\textsuperscript{10}In the modern terminology of combinatorics, this is a de Bruijn sequence.
in the expanded string (the pattern of Ls and Gs). Moreover, for a typical reader who is not trying to memorize the definitions of metres (either in the GGLGGLLGLGLG form or the “ta ta ja ra”’ form), the ganas add no value and serve only to further mystify and obscure the topic. Moreover, they can be misleading as to the nature of yati breaks in the metre, as the metre being described is rarely grouped into threes, except for certain specific metres (popularly used in stotras) such as भुजक्रप्रयाताम्, तोटकम्, and सन्त्रिणिः. One can as easily (and more enjoyably) learn the pattern of a metre by committing a representative example (a good verse in that metre) to memory, rather than the definition using ganas, as the author and others know from personal experience. For these reasons, the gana information is de-emphasized in the tool described in this paper.

4.1.2 pādānta-laghu

Sanskrit poetic convention is that the very last syllable in a verse can be laghu even if the metre requires it to be guru. Consider for instance, the very first verse of Kālidāsa’s Meghadūta, in the Mandākrāntā metre:

kaścit kāntā-viraha-guruṇā svādhikārāt pramattaḥ
śāpenāstaṃgamita-mahimā varṣa-bhogyena bhartuḥ
yakṣaś cakre janaka-tanayā-snāna-puṇyodakeṣu
snigdhacchāya-taruṣu vasatiṃ rāmagiryāśrameṣu

Even though the Mandākrāntā requires in each pāda a final syllable that is guru, the final syllable of the verse above is allowed to be śu which if it occurred in another position (and not followed by a consonant cluster) would be treated as a laghu syllable. A similar convention, though not always stated as clearly in texts in prosody, more or less applies at the end of each half (ardha or pair of pādas) of the verse (for an example, see the kāṣṭhād agnir... verse in Śālinī from Section 1.1).

The question of such a laghu at the end of odd pādas (viṣama-pādānta-laghu) is a thorny one, with no clear answers. Even the word of someone like Kedarabhaṭṭa cannot be taken as final on this matter, as it needs to hold up to actual usage and what is pleasing to the trained ear. Certainly we see such laghus being used liberally in metres like Śloka, Upajāti and Vasantatilakā. At the same time, there are metres like Śālinī where this would be unusual. The summary from those well-versed in the metrical
tradition\textsuperscript{11} is that such \textit{laghus} are best avoided (and are therefore unusual, the works of the master poets) in \textit{yati-prabala} metres, those where the \textit{yati} is prominent. This is why, \textit{Śālinī} with 11 syllables to a \textit{pāda} requires a stricter observance of \textit{guru} at the end of odd \textit{pādas} than a metre like \textit{Vasantatilakā} with 14. As a general rule of thumb, though, such \textit{viṣama-pādānta-laghus} can be regarded as incorrect in metres longer than \textit{Vasantatilakā}. It is not clear how a computer could automatically make such subjective decisions, so something like the idea (Melnad, Goyal, and P. M. Scharf 2015) of storing a boolean parameter about which metres allow this option, seems desirable. Still, the question of how that boolean parameter is to be chosen remains open.

4.1.3 Is more data always better?

It seems natural that having data about more metres would lead to better decisions and better results, but in practice, some care is needed. A common problem is that when there are too many metres in our database, the likelihood of false positives increases. To see this more clearly, imagine a hypothetical case in which every possible combination of \textit{laghu} and \textit{guru} syllables was given its own name as a metre: in that case, a verse intended to be in the metre \textit{Śārdūlavikrīḍtam}, say, with even a single error, would perfectly match some other named metre, and we would be misled as to the truth. A specific case where this happens easily is when a user inputs a single \textit{pāda} but the system tries to treats it as a full verse. In this case, the quarters of the input, as they are much shorter, are more likely to match some metre accidentally. The solution of returning multiple results (a \textit{list} of results rather than a single result) alleviates this problem (cf. the idea of \textit{list decoding} from computer science).

A related problem is the over-precise naming of metres. We know that \textit{Indravajrā} and \textit{Upendravajrā} differ only in the weight of the first syllable, and that the \textit{Upajāti} metre consists of free alternation between them for the four \textit{pādas} in a verse, as for this particular metre, the weight of the first syllable does not matter too much. However, there exist theorists of prosody who have, to each of the $2^4 = 16$ possibilities (all the ways of combining \textit{Indravajrā} and \textit{Upendravajrā}), given names like \textit{Māyā}, \textit{Premā}, \textit{Mālā}, \textit{Ṛddhiḥ} and so on (A. Mishra 2007). This is not very useful to a reader, as in such cases, the metre in question is, in essence, really more

\textsuperscript{11}Śatāvadhānī R. Ganesh, personal communication
common than such precise naming would make it seem. Velankar (Velankar 1949) even considers the name *Upajāti* as arising from the “displeasure” of the “methodically inclined prosodist”.

Another issue is that data compiled from multiple works on prosody (or sometimes even from the same source) can have inconsistencies. It can happen that the same metre is given different names in different sources (Velankar 1949, p. 59). This is very common with noun endings that mark gender, such as -ā versus -aṃ, but we also see cases where completely different names are used. It can also happen that the same name is used for entirely different metres (see also the confusion about *Upajāti* mentioned below in Section 4.4). For these reasons, instead of storing each metre as a (name, pattern) pair as mentioned earlier, or as the (better) (name, pattern, bool) triple (Melnad, Goyal, and P. M. Scharf 2015), it seems best to store a (pattern, bool, name, source for name) tuple. I started naively, thinking the name of metres is objective truth, and as a result of this project I realized that names are assigned with some degree of arbitrariness.

Finally, a couple more points: (1) There exist metres that end with laghu syllables, and the code should be capable of handling them. (2) It is better to keep metrical data as data files, rather than code. This was a mistake made in the initial design of the system. Although it did not deter helpful contributors like Dhaval Patel from contributing code-like definitions for each metre, it is still a hindrance that is best avoided. Keeping data in data files is language-agnostic and would allow it to be used by other tools.

Overall, however, despite these issues, on the whole, the situation is not too bad, because it is mostly a small set of metres that is used by most poets. Although the repertoire of Sanskrit metres is vast (Deo 2007), and even the set of commonly used metres is larger in Sanskrit than in other languages, nevertheless, as with rāgas in music, although names can and have been given to a great many combinations, not every mathematical possibility is an aesthetic possibility.\(^\text{12}\)

### 4.2 Transliteration

It appears that accepting input in various input schemes is one of the features of the tool that users enjoy. Although the differences between various input schemes are mostly superficial and easily learned, it appears that many

\(^{12}\)This remark comes from Śatāvadhānī Ganesh who has pointed this out multiple times.
people have their preferred scheme that they would like to employ wherever possible. These are fortunately easy for computers to handle.

As pointed out elsewhere in detail (P. M. Scharf and Hyman 2011), the set of graphemes or phonemes one might encounter in putatively Sanskrit input is larger than that supported by common systems of transliteration like Harvard-Kyoto or IAST. Characters like chandrabindu and ळ will occur in the input especially with modern poetry or verse from other languages. The system must be capable of doing something reasonable in such cases.

A perhaps unusual choice is that the system does not currently accept input in SLP1, even though SLP1 is used internally. The simple reason is that no one has asked for it, and it does not seem that many people type in SLP1. SLP1 is a great internal format and can be a good choice for interoperability between separate tools, but it seems that the average user does not prefer typing křzRaH for कृष्ण. Nevertheless, this is a minor point as this input method can easily be added if anyone wants it.

In an earlier paper (Melnad, Goyal, and P. M. Scharf 2015), two of the deficiencies stated about the tool by Mishra (A. Mishra 2007) are that:

1. By supporting only Harvard-Kyoto input, that tool requires special treatment of words with consecutive a-i or a-u vowels (such as the word “प्रउग”). In this tool, as Devanāgarī input is accepted, such words can be input (besides of course by simply inserting a space).

2. That tool does not support accented input, which (Melnad, Goyal, and P. M. Scharf 2015) do because they accept input in SLP1. In this tool, accented input is accepted if input as Devanāgarī. However, as neither this tool nor the one by (Melnad, Goyal, and P. M. Scharf 2015) supports Vedic metre, this point seems moot: Sanskrit poetry in the classical (non-Vedic) metre is not often accompanied by accent markers! In this tool, accent marks in Devanāgarī are accepted but ignored.

4.3 Scansion

As a coding shortcut when the program was first being written, I decided to treat anusvāra and visarga as consonants too for scansion, instead of especially handling them. To my surprise, I have not had to revise this and eliminate the shortcut, because, in every instance, the result of scansion is the same. I am not aware of any text on prosody treating anusvāra and
visarga as consonants, but their identical treatment is valid for Sanskrit prosody. This is a curious insight that the technological constraints (or laziness) have given us!

As mentioned in earlier work (Melnad, Goyal, and P. M. Scharf 2015), in later centuries of the Sanskrit tradition, there evolved an option of considering certain guru syllables as laghu, as a sort of poetic license, in certain cases. Specifically, certain consonant clusters, especially those containing $r$ like $pr$ and $hr$, were allowed to be treated as if they were single consonants, at the start of a word. This rule is stated by Kedārabhaṭṭa too and seems to be freely used in the Telugu tradition even today. A further trend is to allow this option everywhere, based on how “effortlessly” or “quickly” certain consonant clusters can be pronounced, compared with others. A nuanced understanding of this matter comes from a practising poet and scholar of Sanskrit literature, Śatāvadhānī R. Ganesh:¹³  

13personal communication, but see also corroboration at https://groups.google.com/d/msg/bvparishat/ya1cGLuhc14/EkIqH9NbgawJ

14See another summary here: https://github.com/shrevatsa/sanskrit/issues/1#issuecomment-68502605

With this understanding,¹⁴ the question arises whether the system needs to encode this exception, especially for dealing with later or modern poetry. This could be done, but as a result of the system’s robustness to errors, in practice, this turns out to be less necessary. Any single verse is unlikely to exploit this poetic license in every single pāda, so the occasional usage of this exception does not prevent the metre from being detected. The only caveat is that this already counts as an error, so verses that exploit this exception would have slightly lower robustness to further additional errors.
4.4 Identification

It is not enough for a verse to have the correct scansion (the correct pattern of *laghu* and *guru* syllables), for it to be a perfect specimen of a given metre. There are additional constraints, such as *yati*: because a pause is indicated at each *yati-sthāna* (caesura), a word must not cross such a boundary, although separate lexical components of a compound word (*samāsa*) may. Previously (Melnad, Goyal, and P. M. Scharf 2015), an approach has been suggested of using a text segmentation tool such as the Sanskrit Heritage Reader (Huet 2005; Huet and Goyal 2013) for detecting when such a constraint is violated. This would indeed be ideal, but the tool being described in this paper alleviates the problem by displaying the user’s input verse aligned to the metre, with each *yati-sthāna* indicated. Thus, any instance of a word crossing a *yati* boundary will be apparent in the display.

Note that we can provide information on all kinds of *Upajāti*, even if they are not explicitly added to our database, a problem mentioned previously (Melnad, Goyal, and P. M. Scharf 2015). *Upajāti* just means “mixture”; the common *upajāti* of *Indravajrā* and *Upendravajrā*, as a metre, has nothing to do with the *upajāti* of *Vaṃśastha* and *Indravaṃśa* (Velankar 1949). In fact, the latter is sometimes known by the more specific name of *Karambajāti*, among other names. Whenever an *Upajāti* of two different metres is used and input correctly, each of the two metres will be recognized and shown to the user, because different *pāda*s will match different patterns in our index. So without us doing any special work of adding all the kinds of Upajāti to the data, the user can see in any given instance that their verse contains elements of both metres, and in exactly what way. Of course, adding the “mixed” metre explicitly to the data would be more informative to the user, if the mixture is a common one.

4.5 Display

Once a metre is identified, for some users, telling the user the name of the metre may be enough. However, if we envision this tool being used by anyone reading any Sanskrit verse (such as Devadatta from Section 1.3), then for many users, being told the name of the metre (or even the metre’s pattern) carries mainly the information that the verse is in some metre, but does not substantially improve the reader’s enjoyment of the verse. Seeing the verse

---

15Śatāvadhān R. Ganesh, personal communication
aligned to the metre, with line breaks introduced in the appropriate places and \textit{yati} locations highlighted, helps a great amount. What would help the most, however, is a further introduction to the metre, along with familiar examples that happen to be in the same metre, and audio recordings of typical ways of reciting the metre.

The tool does this, for popular metres (see Figure 1), drawing on another resource (Ganesh 2013). In these audio recordings made in 2013, Śatāvadhānī R. Ganesh describes several popular metres, with well-chosen examples (most recited from memory and some composed extempore for the sake of the recordings). Some interesting background such as its usage in the tradition—a brief “biography” of the metre—is also added for some metres. Although they were not created for the sake of this tool, it was the same interest in Sanskrit prosody that led both to the creation of this tool and to my request for these recordings. Showing the user’s verse accompanied by examples of recitation of other verses in the same metre helps the user read aloud and savor the verse they input.

Incidentally, an introduction to metres via popular examples and accompanying audio recordings is also the approach taken by the book \textit{Chandovallarī} (S. Mishra 1999). The examples chosen are mostly from the \textit{stotra} literature, which are most likely to be familiar to an Indian audience. In this way, it can complement the recordings mentioned in the previous paragraph, in which the examples were often chosen for their literary quality or illustrative features.

4.6 Getting feedback from users

The main lesson I learned from building this system was the value of making things accessible to as many users as possible, by removing as many barriers as possible. Write systems that are “liberal” in what they accept, but are nevertheless conservative enough to avoid making errors (Postel’s law).

There exist users who may not have much computer science or programming knowledge, but are nevertheless scholars who are experts in a specific subject. For example, India’s tech penetration is low; even many Sanskrit scholars aren’t trained or inclined to enter verse in standard transliteration formats. The very fact that they are visiting your tool and using it means that they constitute a self-selecting sample. It would be a shame not to use their expertise. Their contributions and suggestions can help improve the system. In the case of this tool, the link to GitHub
discussion pages, and making it easy with a quick link to report issues encountered during any particular interaction, have generated a lot of improvements, both in terms of usability and correctness. A minor example of a usability improvement is setting things up so that the text area is automatically focused when a user visits the web page—this is trivial to set up, but not something that had occurred as something desirable to do. In this case, a user asked for it.

Though user feedback guided many design decisions, gathering and acting on more of the user feedback would lead to further improvements.

5 Conclusions and future work

This paper has described a tool for metre recognition that takes various measures to be useful to users as much as possible. In this section, we list the current limitations of the tool and improvements that can be (and are planned to be) made.

In terms of transliteration, though there are many transliteration schemes supported, even the requirement to be in a specific transliteration scheme is too onerous—instead, the tool must let the user type, and in real-time display its understanding of the user’s input, while offering convenient input methods (such as a character picker\textsuperscript{16}) that do not require prior knowledge of how to produce specific characters. Similarly, on the output side, a user’s preferred script for reading Sanskrit (which may not be the same as their input script) should be used and remembered for future sessions, so that for instance a user can completely use the tool and see all Sanskrit text in the Kannada script. There may even exist users who prefer to read everything in SLP1!

Very few mātrā metres are currently supported (only members of the Āryā family have been added). There are many simple mātrā metres used in stotras, such a metre consisting of alternating groups of 3 and 4 mātrās. More examples for each metre, such as from Chandovallarī (S. Mishra 1999), would help.

The program is a monolithic application. It should be made more modular and packaged into libraries for distribution so that other software can easily incorporate the same user-friendly features. Similarly, in addition to the human interface, providing an API would make this code usable from

\textsuperscript{16}For instance, \url{https://r12a.github.io/pickers/devanagari}
another website or application. Another limitation is that the program requires a dedicated server to run; if rewritten to run entirely in the browser it could be packaged as a browser extension so that any Sanskrit verse on any web page can be quickly queried about and reformatted in a metrically clear form. The automatic inference of the transliteration scheme and other aspects of the user’s intention, though a user-friendly feature, might have errors occasionally, so the program would be improved by allowing them to be indicated manually when desired.

Finally, the most promising avenue for future work is running this tool on large texts rather than for one verse at a time, which can uncover many insights about prosody. For instance, the most common Anuṣṭubh (Śloka) metre, the work-horse of Sanskrit literature and beloved of the epic poets of the Rāmāyaṇa and the Mahābhārata, is still difficult to define clearly. The naive definition, that the odd pāda match the regular expression “....LGG.” and the even pādā match “....LGL.”, is found insufficient: there are both more and fewer constraints in practice. It is not the case that all $2^{16}$ choices for the first four syllables are acceptable, nor is it the case that every acceptable śloka satisfies even these constraints. G. S. S. Murthy (Murthy 2003) surveys and summarizes the literature on this metre and concludes with some perceptive remarks:

It is indeed surprising that anuṣṭup has remained ill-defined for so long. [...] If anuṣṭup is being used for thousands of years in saṃskṛt literature without a precise definition having been spelled out till date, it must be simply because the internal rhythm of anuṣṭup becomes ingrained in the mind of a student of saṃskṛt at an early age due to constant and continuous encounter with anuṣṭup and when one wants to compose a verse in anuṣṭup, one is guided by that rhythm intuitively.

It is now almost within reach, by running a tool like this on a large corpus consisting of the Mahābhārata, Rāmāyaṇa, and other large works, to arrive at a descriptive definition of śloka based on the verses found in the literature so that we can make explicit the rules that have been implicitly adhered to by the natural poets.
Acknowledgements

I am indebted to the poet and scholar Śatāvadhānī R. Ganesh for encouraging my interest in Sanskrit (and other Indian) prosody. It is his intimate love of metres (reminding me of the story of the mathematician Ramanujan for whom every positive integer was a personal friend), that led me to the realization that an understanding of metre greatly enriches the joy of poetry. Dhaval Patel contributed metrical data, and raised points about nuances, from Vṛttaratnākara (some still unresolved). Sridatta A pointed out some more. I thank Vishvas Vasuki for being a heavy user and pointing out many bugs and suggestions, and for initiating the sanskrit-programmers mailing list where this project began. Finally, I thank my wife Chitra Muthukrishnan for supporting me during this work, both technically and otherwise, and for reviewing drafts of this article.

Ganesh, Shatavadhani R. 2013. *Sanskrit Metres (A playlist with a series of audio recordings containing recitation and information about popular metres).* URL: https://www.youtube.com/playlist?list=PLABJEFgj0PWVXr2ERGu2xtoSXrNdBs5xS.


141


Improving the learnability of classifiers for Sanskrit OCR corrections

Devaraja Adiga, Rohit Saluja, Vaibhav Agrawal, Ganesh Ramakrishnan, Parag Chaudhuri, K. Ramasubramanian and Malhar Kulkarni

Abstract: Sanskrit OCR documents have a lot of errors. Correcting those errors using conventional spell-checking approaches breaks down due to the limited vocabulary. This is because of high inflections of Sanskrit, where words are dynamically formed by Sandhi rules, Samāsa rules, Taddhita affixes, etc. Therefore, correcting OCR documents require huge efforts. In this paper, we present different machine learning approaches and various ways to improve features for ameliorating the error corrections in Sanskrit OCR documents. We simulated Subanta Prakaraṇam of VaiyākaraṇaSiddhāntaKaumudī for synthesizing off-the-shelf dictionary. Most of the methods we propose can also work for general Sanskrit word corrections.

1 Introduction

Optical character recognition (OCR) is the process of identifying characters in document images for creating editable electronic texts. SanskritOCR by Indsenz, Google OCR and Tesseract are major OCRs available for Sanskrit. Word level error analysis for 6 books printed at various places of India having different fonts scanned with 300 DPI are listed in Table 1. Correcting the errors manually becomes cumbrous even with the OCR accuracy as high as above 90% unless complemented by a mechanism for correcting the errors. User feedback based OCR correcting mechanisms can improve through correcting a contiguous text having a uniform font. We discuss different approaches for correcting Sanskrit OCR based on available system resources.
<table>
<thead>
<tr>
<th>Book Name</th>
<th>Publisher Details</th>
<th>Year of Publication</th>
<th>No. of Pages OCRed</th>
<th>WER - Indsenz</th>
<th>WER - Google</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raghuvamanśam Sanjīvinīsametam</td>
<td>Nirṇaya Sāgara Press, Mumbai</td>
<td>1929</td>
<td>200</td>
<td>19%</td>
<td>35%</td>
</tr>
<tr>
<td>Nṛśimhapūr-vottaratā-panīyopaniṣat</td>
<td>Ānandāśrama, Pune</td>
<td>1929</td>
<td>160</td>
<td>34%</td>
<td>41%</td>
</tr>
<tr>
<td>Siddhānta Śekhara-1</td>
<td>Calcutta University Press</td>
<td>1932</td>
<td>390</td>
<td>38%*</td>
<td>66%</td>
</tr>
<tr>
<td>Gaṇaka-Tarangini</td>
<td>Jyotish Prakash Press, Varanasi</td>
<td>1933</td>
<td>150</td>
<td>34%</td>
<td>46%</td>
</tr>
<tr>
<td>Siddhānta Śkhara-2</td>
<td>Calcutta University Press</td>
<td>1947</td>
<td>241</td>
<td>55%*</td>
<td>53%</td>
</tr>
<tr>
<td>Siddhānta Śiromaṇī</td>
<td>Sampuranananda University, Varanasi</td>
<td>1981</td>
<td>596</td>
<td>18%</td>
<td>29%</td>
</tr>
</tbody>
</table>

Table 1

Word Error Rates for Indsenz’s SanskritOCR and Google OCR (*After training 5 pages)

Conventional approaches for spell checking uses Levenshtein-Damerrau edit distance to a known dictionary and auto-corrects the errors using a language model (Whitelaw et al. 2009). For post-OCR corrections of languages highly rich in inflections, this naive approach results in poor accuracy (Sankaran and Jawahar 2013). It primarily depends upon lookups into a fixed vocabulary. Such vocabulary for Sanskrit is always incomplete because of the complexity arising due to its inflectional nature, tendency to do Sandhi, and highly productive in derivative morphology such as Samāsa, Taddhita, and Kṛdanta.

In recent works, encoder-decoder Recurrent Neural Networks (RNNs) with character-based attention have shown state-of-the-art results in Neural Language Correction (Xie et al. 2016). Saluja et al. (2017a) proposed a logistic regression-based machine learning framework for correcting In-
dic OCRs using dual-engine OCR. For correcting OCRs across four Indic languages (Sanskrit, Hindi, Kannada, and Malayalam) in the single-engine environment, Saluja et al. (2017b) have succeeded in reaching the state of art using a special type of RNNs, called Long Short Term Memory Networks (LSTM).

<table>
<thead>
<tr>
<th>OCR Word</th>
<th>Corrected Word</th>
<th>Ground Truth</th>
</tr>
</thead>
<tbody>
<tr>
<td>विश्रीआआआआर्िनि</td>
<td>विश्रीणानि</td>
<td>विश्रीणानि</td>
</tr>
<tr>
<td>षष्ठे</td>
<td>पाणिपतिक्</td>
<td>षष्ठोपणिष्ठि</td>
</tr>
<tr>
<td>भतुमुनिरा</td>
<td>आस्थूष्णितविष्ण्णः</td>
<td>भतुमुनिरास्थूष्णितविष्ण्णः</td>
</tr>
<tr>
<td>महल्टसनिरस्वना:</td>
<td>महल्टूर्�年报निरस्वना:</td>
<td>महल्टूर्�年报निरस्वना:</td>
</tr>
<tr>
<td>मातापकङ्क</td>
<td>महाचबङ्क</td>
<td>महाचबङ्क</td>
</tr>
<tr>
<td>नै९वाशिङ्चिङ्च</td>
<td>हेवातंशिङ्चिङ्च</td>
<td>हेवातंशिङ्चिङ्च</td>
</tr>
<tr>
<td>सूञउगव्यङ्च नैथस्तेनूञ</td>
<td>भगवा</td>
<td>भगवा</td>
</tr>
</tbody>
</table>

**Table 2**

Examples of Sanskrit OCR words corrected by our framework. OCRRed data of over 5k Sanskrit document images and 12k in different languages were corrected using our framework - OpenOCRCorrect.

The basic dictionary lookup approach requires fewer system resources whereas Neural language correcting models demand higher system specifications. So we propose and evaluate different models for correcting Sanskrit OCR1, starting from simple dictionary lookup to Neural attention models. For building the vocabulary for Sanskrit, we developed a Subanta-generator. We will be using various other auxiliary sources which we will discuss in the next section. Then in section 4 we discuss the results for various error detecting approaches in detail. Suggestion generation will be explained in the following section. Table 2 shows the OCR errors corrected by our framework and Figure 1 is a screen-shot of our framework. We are using multicolor coding to depict compound words, out-of-vocabulary words, auto-corrected words and correct words.

Our contributions in this paper are i) Suggesting different models for error detection based upon the amount of training data (if the range is 10k and GPU is not available use Plug-in classifier, for a range of 100k with GPU use LSTM and for a range of 1000k with GPU use attention models)

---

1The source code of Sanskrit OCR corrector, OpenOCRCorrect is available at https://goo.gl/WqoVi2
ii) increasing the learnability of ML classifiers by increasing auxiliary sources and iii) Comparing the different ML-based and Deep learning-based methods for the task of Error detection. We have improved the results of plug-in-classifier by introducing more auxiliary sources and synthesized words. Further, we use attention model to compare the results with LSTM based error detection.

2 Auxiliary Sources

Figure 2 depicts the functionality of human-interactive framework for OCR corrections. We will be using various auxiliary sources that are helpful in verifying the correct words and curating the word-level errors. Our system is leveraged by OCR data from different systems, dynamically updated OCR confusions, and domain-specific vocabulary. We are also using a synthesized off-the-shelf dictionary. These features are used for supervised learning by training a plug-in-classifier for achieving a better F-score. Erroneous words are corrected using suggestions through human interaction to keep the confidence level high. Later on, words having similar errors are auto-corrected.
In the following sections, we discuss various auxiliary sources used by the framework.

### 2.1 OCR documents from different systems

Since different OCR systems are using different models they are likely to make different kinds of errors and are likely to be correct on the OCR words that they agree upon. This observation is especially leveraged by the ensemble-based ML approach (Polikar 2006). Therefore OCR documents from different systems can become a powerful auxiliary source.

### 2.2 Off-the-shelf dictionary

Since the vocabulary is incomplete for Sanskrit due to rich inflections, we developed a Subanta generator for synthesizing noun variants. A databank of noun variants is available through Huet (2017) which has around 6.5 lakh unique Subanta words. Among the different declension generators, Patel and
Shivakumari Katuri (2015) is an open-source Subanta generator for Sanskrit. We developed a new Subanta generator for the following reasons

- For ease of integration into the OCR framework
- For overcoming the errors produced by the existing Subanta generator. Examples from Patel and Sivakumari Katuri (2015) -
  - प्रथमा एकवचनम् for words ending with ऋ.
  - द्वितीया द्विवचनम् for many of the Sarvanāmaās
  - Declensions for words ending with वसु affix are wrong in case of भसंद्
- To have the provision for future enhancements

Aṣṭādhyāyā rules corresponding to Subanta Prakaraṇam and required Sandhi rules are coded in accordance with the rules explanations as given in (B. Dīkṣita, V. Dīkṣita, and Sarasvatī 2006). For resolving the conflicts we chose the order of applicability of rules as per the Paribhāṣā -परिनिर्णयपदवादानामुरौरंबलीयः. Context dependencies of many rules are resolved by collecting the context information. For example, for the rule एकाचो वशो भष ठिपनतर्य स्योः (अ.8-2-37), the roots collected are गाध्, गुध्, गृध्, दध्, द्रम, द्राघ्, द्राघ्, वध्, बध्, बीभ्, बुध्. And also the roots गाह्, गुह्, गृह्, मह्, दह्, दिद्, दिद्, दढ्, द्राह्, द्रढ्, बाह्, ब्रह् are considered after applying 'ददेधा' (अ.8-2-32) or 'होढः' (अ.8-2-31).An example word where this rule is applied - कामधुक् (हुाितिदकम्-कामूह). For the rules नाम्नस्ताच्छुदुः (अ.7-1-78), आच्छीनभोज्ञुम् (अ.7-1-80) and शान्तयोनिन्त्यम् (अ.7-1-81), we grouped the participles of roots belonging to different conjugations accordingly. Similar way we tried to completely/partially solve the context dependencies of many rules.

We have processed XML file of Monier-Williams Sanskrit Dictionary available in the Cologne Digital Sanskrit Dictionary collections, Institute for Indology and Tamilistics, University of Cologne (http://www.sanskrit-lexicon.uni-koeln.de/download.html) and extracted more than 1.8 lakh words with the gender information from the XML file. Vibhakti variants for these words are generated using the Subanta generator and around 3.2 million unique words are generated. We also used the verbs which are listed in the क्रियाहृतपनिपादिका (Verb-forms-Generator) of ILTP-DC (Indian language technology proliferation and deployment center), which are around 3 lakh unique words. These 3.5 million words are used as an off-the-shelf dictionary for the OCR corrector.
2.3 Domain specific vocabulary

In Sanskrit literature frequency of commonly used words changes from one Śāstra to another. So the domain-specific vocabulary is the most powerful auxiliary resource which will fill the words not found in the off-the-shelf dictionary. Domain-specific vocabulary is created by extracting unique strings from the various books available in Göttingen Register of Electronic Texts in Indian Languages (GREtil 15.11.2001 - 16.02.2018). This auxiliary source is also dynamically updated as the user corrects the document, which helps in correcting the rest of the document.

2.4 Sandhi Rules

Due to Sandhi rules and Samāsa, words can change dynamically in Sanskrit documents. We are using basic Sandhi rules to find the subwords of a compound word and to match with the words from the vocabulary for detecting its correctness. A greedy approach is used for this splitting with a minimum set of words of maximum length and minimum edit distance as the criteria. For example, the OCR word जागिरतावायाभेवावाऽयमुं will be split into जागिरत, अवायाभ् (this word is matched with अवायाम्), एव, अवाऽयम् and उं. This helps in detecting out-of-vocabulary words and generating suggestions for them.

2.5 Document and OCR specific n-gram confusions

Since different OCR systems use different preprocessing techniques, different classifier models, error confusions for a word varies from one OCR engine to another (Abdulkader and Casey 2009). Thus, the OCR specific confusions can be helpful in deciding whether the part of the erroneous word should be changed or not and also in deciding the tie while changing the part of the erroneous word. For example, while changing the erroneous word िनवः, if the dictionary lookup suggests िनबः and िनरः as nearest possible words, having higher n-gram confusion to व->ब biases the selection towards िनबः.
3 Methodologies Followed

3.1 Learning by Optimizing Performance Measures though Plug-in Approach

We rephrase our basic problem of error detection as that of continuously evolving a classifier that labels the OCR of a word as correct or incorrect. The classifier should be trained to optimize a performance measure that is not necessarily the conventional likelihood function or sum of squares error. An example performance measure to be maximized and that is coherent with our needs of maximizing recall (coverage) in detecting erroneous words while also being precise in this detection is the $F$—score, which, unfortunately, does not decompose over the training examples and can be hard to optimize. We adapt a plug-in approach (Narasimhan, Vaish, and Agarwal 2014) to train our binary classifier over such non-decomposable objectives while also being efficient for incremental re-training.

Consider a simple binary classification problem where the task is to assign every data point $x \in \mathcal{X}$, a binary label $y \in \{-1, +1\}$. Plug-in-classifiers achieve this by first learning to predict Class Probability Estimate (CPE) scores. A function $g : \mathcal{X} \to [0, 1]$ is learned such that $g(x) \approx \text{Probability}(y = 1)$. Various tools such as logistic regression may be used to learn this CPE model $g$. The final classifier is of the form $\text{sign}(g(x) - \eta)$ where $\eta$ is a threshold that is tuned to maximize the performance measure being considered, e.g. F-measure, G-mean etc.

In Saluja et al. (2017a), various features based on dictionary n-grams and language rules have been used in Sanskrit, Hindi, and Marathi. Our major work in this paper is to improve features for such a classifier and verify their effect in three different domains in Sanskrit. We use train:val:test ratio as 48:12:40 for all our experiments that use plug-in-classifier since we wanted to explore the possibility of using the classifier to correct the last 40% of the book, once initial 60% of the book is corrected.

3.2 LSTM with fixed delay

The basic RNN (Recurrent Neural Network) can be represented by Equations .1 and .2.

$$h_t = g(W_{hh}h_{t-1} + W_{hx}x_t + b_h)$$  \hspace{1cm} (1)
$y_t = W_{yh} h_t$  \hspace{1cm} (2)

g can be sigmoid($\sigma(x_i) = \frac{exp(x_i)}{\sum_j[exp(x_j)]}$) or tanh ($tanh(x) = 2\sigma(2x) - 1$) or Rectified Linear Unit (ReLU) ($f(x) = \max(0,x)$) (Talathi and Vartak. 2014). The matrices $W_{hx}$ and $W_{yh}$ connect the input to the hidden layer and hidden layer to output respectively. These matrices are common for instance in the sequence. The matrix $W_{hh}$ is the feedback from past input and is responsible for remembrance and forgetfulness of the past sequence based on context.

Equation .2 at each time $t$ can be unfolded back in time, to time $t = 1$ for the 1st character of the word sequence, using Equation .1 and the network can be trained using back-propagation through time (BPTT) (Mike and Paliwal. 1997).

Since we have taken care to ensure equal byte length per letter with ASCII transliteration scheme, for the loss function we used negative log-likelihood of Log SoftMax (multi-class) function. The Log SoftMax function is given in .3, where $y_{ti}$ is the value at $i^{th}$ index of output vector $y_t$.

$$f(y_{ti}) = \log\left(\frac{exp(y_{ti})}{\sum_j[exp(y_{tj})]}\right)$$ \hspace{1cm} (3)

The equations are similar for the LSTM with each unit as a memory unit, instead of a neuron. Such memory unit remembers, forgets, and transfers cell state to the output(or next state) based on input history. The cell state at time $t$ is given by equation .4 where the forget gate $f_t$ and the input gate $i_t$ fire according to equations .5 and .6 respectively.

$$c_t = f_t c_{t-1} + i_t g_1(W_{hc} h_{t-1} + W_{xc} x_t + b_c)$$ \hspace{1cm} (4)

$$f_t = g_2(W_{xf} x_t + W_{hf} h_{t-1} + W_{cf} c_{t-1} + b_f)$$ \hspace{1cm} (5)

$$i_t = g_2(W_{xi} x_t + W_{hi} h_{t-1} + W_{ci} c_{t-1} + b_i)$$ \hspace{1cm} (6)

The data is selectively transferred from the cell to hidden state $h_t$ according to equation .7 where the selection is done by the firing of output gate $o_t$ as per equation .8.

$$h_t = o_t g_3(c_t)$$ \hspace{1cm} (7)
$g_1$ and $g_3$ are generally tanh and $g_2$ is generally sigmoid.

$$o_t = g_2(W_{zo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o)$$

(8)

Saluja et al. (2017b) uses 512 X 2 LSTM with the fixed delay (between the input sequence and output sequence) trained and tested on characters from 86k OCR word correction pairs with train:val:test split as 64:16:20. The model, when trained on large data, is able to learn OCR specific error patterns as well as a language model. The model abstains from changing the correct word. Thus, for error detection, the word changed by such a model is marked as incorrect whereas the word unchanged by the model is marked as correct. Such a model over-fits the OCR system and domain. It works well with the dataset range of a hundred thousand OCR word correction pairs.

3.3 Attention Model

Here, we use the model with more number of layers than LSTM based model discussed in the previous section. Attention models are the models with a separate encoder as well as a decoder as compared to LSTM based model wherein the same layers encode as well as decode a sequence. Attention models contain RNN layers as an encoder that can take characters from OCR words, and similar RNN based decoders decode the encoder’s output to correct words when trained with a large amount of data. The attention layers, which are applied on encoder’s output to help the decoder, learn to give attention to different contexts around the character being corrected based on the input. We train and test such a model with the 86k OCR word correction pairs used in (Saluja et al. 2017b). We use open-source library OpenNMT (http://opennmt.net/) for this purpose with the default model that includes 500 X 2 LSTM encoder as well as 500 X 2 LSTM decoder. Such a model is able to learn a dataset of an order of millions of OCR word correction pairs as per our experiences for French and English in ICDAR Post-OCR Competition 2017. Here again, we mark words changed by the model as incorrect for error detection and the words that remained unchanged as correct. As we will see later, even when trained on 86k pairs, such a model is able to perform close to the LSTM based model for the error detection task.
4 Error Detection Methods and Results

4.1 Unsupervised approach

<table>
<thead>
<tr>
<th>Approach</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>Prec.</th>
<th>Recall</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gen. Dict. Lookup</td>
<td>89.18</td>
<td>40.12</td>
<td>59.87</td>
<td>10.82</td>
<td>29.75</td>
<td>89.18</td>
<td>44.61</td>
</tr>
<tr>
<td>Sandhi Rules</td>
<td>54.34</td>
<td>13.23</td>
<td>86.77</td>
<td>45.66</td>
<td>43.89</td>
<td>54.34</td>
<td>48.56</td>
</tr>
<tr>
<td>Sec. OCR Lookup</td>
<td>90.68</td>
<td>23.59</td>
<td>76.40</td>
<td>9.31</td>
<td>42.79</td>
<td>90.68</td>
<td>58.14</td>
</tr>
</tbody>
</table>

Table 3

Error Detection Results with unsupervised methods. Using Sandhi rules while dictionary lookup increase true detections(TN) but increase false detections(FN) as well which is balanced by secondary OCR lookup.

We applied various methods for detecting errors in the OCR text. To start with, we used the book named “Āryabhaṭīyabhāṣya of Nīlakaṇṭha III Golapāda(AnantaśayanaSaṃskṛtaGranthāvaliḥ, 1957)” for which we had the OCR text (OCRed from indsenz) and the ground truth data available.

Using unsupervised methods, commonly used dictionary lookup based approach gave poor F-Scores due to a lot of correct words marked as errors, i.e. lower True Negative percentage as shown in Table 3. Marking the words that are formed by applying Sandhi rules on dictionary lexicons as correct increased detection of correct words(True Negatives) but not the detection of errors (True Positives) as compared to the previous approach. For this book, lookup into OCR output of other engine (Google Docs) for the same document images improved the F-Score to a decent value.

4.2 Single Engine Environment

For supervised learning using the plug-in-classifier as explained in section 3.1, we are splitting the data with train:val:test ratio as 48:12:40, we train the plug-in-classifier with various features. We are able to improve the row 1 results in Table 3 by including frequency of n-grams (upto 8) in the general dictionary as features. We also include the binary feature based on lookup in a general dictionary. The results are shown in the first row of Table 4.
<table>
<thead>
<tr>
<th>Approach</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>Prec.</th>
<th>Recall</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifier with ngrams frequency + word lookup in General Dict. as features</td>
<td>73.38</td>
<td>22.86</td>
<td>77.13</td>
<td>26.61</td>
<td>38.88</td>
<td>73.38</td>
<td>50.83</td>
</tr>
<tr>
<td>Classifier with ngrams frequency + word lookup in Synthesised Dict. (superset of gen. dict.) as features</td>
<td>74.06</td>
<td>21.02</td>
<td>78.98</td>
<td>25.94</td>
<td>41.14</td>
<td>74.06</td>
<td>52.89</td>
</tr>
<tr>
<td>Classifier with ngrams frequency + word lookup in Synthesised Dict. as well as Domain Dict. as features</td>
<td>66.37</td>
<td>13.08</td>
<td>86.92</td>
<td>33.63</td>
<td>50.38</td>
<td>66.37</td>
<td>57.28</td>
</tr>
<tr>
<td>Classifier with features in row 3 + no. of Sandhi components in OCR word as features</td>
<td>68.50</td>
<td>13.53</td>
<td>86.47</td>
<td>31.50</td>
<td>50.10</td>
<td>68.50</td>
<td>57.87</td>
</tr>
</tbody>
</table>

**Table 4**

*ML Classifier’s Error Detection Results in Single Engine Environment.*

*Here we achieved the F-score close to that of Secondary OCR lookup using Unsupervised approach.*

We further include more words in the dictionary by synthesizing nouns
and collecting the verbs as explained in 2.2. This helps us to achieve the results shown in row 2 of Table 4.

Adding frequencies of n-grams from OCR word as features from domain dictionary generated as explained in 2.3 along with synthesized dictionary improved the results as shown in row 3 of Table 4.

For improving the results further as shown in row 3 of Table 4, we used three splitting based features. i) Split the OCR words using commonly used Sandhi rules and used the no. of lexicon components obtained from the general dictionary as features. ii) We also used no. of lexicon components obtained by splitting the OCR word as lexicons of domain dictionary (for Jyotiṣa) as a feature. Herein, the no. of characters from unknown sub-strings in the OCR word are added to the feature. iii) The product of features obtained in (i) and (ii) is also used as the feature. We normalized all these features about the mean and standard deviation of training data.

The results are shown in row 4 of Table 4. It is important to note that here in single-engine environment we are able to reach closer to the dual engine environment based Secondary OCR Lookup approach given in row 3 of Table 3.

### 4.3 Multi Engine Environment

We further include the dual engine OCR agreement as a feature in addition to the features used in previous sections and achieve the results obtained in Table 5. Here we have used Indsenz as primary OCR engine and Google docs as secondary OCR engine.

We improve the results further by using the feature of dual OCR agreement between Indsenz and Tesseract in addition to previous features to obtain the results shown in row 4 of Table 5.

We present the results of Plug-in Classifier trained and tested on the dataset of books with different domains in Table 6 for proving its consistency over various domains. Row 1 in this table shows the baseline for the book ‘Nṛśimhapūrvottaratāpanīyopaniṣat’ (ĀnandāśramaSaṃskṛtaGranthāvaliḥ, 1929) and row 2 shows the results achieved using all the features (obtained using triple engine environment, off-the-shelf dictionary, domain vocabulary and n-gram frequency from general, synthesized and domain vocabularies). It is important to note that the TP (Errors detected as errors) is high for the baseline in this case as compared to TP for baseline in other domains. However, TN (Correct words detected as correct) for the dictionary lookup
<table>
<thead>
<tr>
<th>Approach</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>Prec.</th>
<th>Recall</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifier with features in table 4 row 2 along with dual engine agreement*</td>
<td>85.13</td>
<td>17.84</td>
<td>82.16</td>
<td>14.87</td>
<td>48.62</td>
<td>85.13</td>
<td>61.89</td>
</tr>
<tr>
<td>Classifier with features in table 4 row 3 along with dual engine agreement</td>
<td>78.04</td>
<td>13.67</td>
<td>86.33</td>
<td>21.96</td>
<td>53.11</td>
<td>78.04</td>
<td>63.20</td>
</tr>
<tr>
<td>Classifier with features in table 4 row 4 along with dual engine agreement</td>
<td>83.49</td>
<td>15.26</td>
<td>84.74</td>
<td>16.51</td>
<td>52.25</td>
<td>83.49</td>
<td>64.28</td>
</tr>
<tr>
<td>Classifier with features in table 4 row 4 along with triple engine agreements</td>
<td>83.43</td>
<td>14.95</td>
<td>85.04</td>
<td>16.56</td>
<td>52.74</td>
<td>83.43</td>
<td>64.63</td>
</tr>
</tbody>
</table>

**Table 5**

*ML Classifier’s Error Detection Results in Multi Engine Environments.* (*state of the art (Saluja et al. 2017a)). Here TP is significantly increased when compared to single engine environment.

baselines are however close to each other for all domains as shown in row 1 of Table 3 and row 1 and row 3 of Table 6. The reason for high TN could be less ambiguity (as compared to other domains) in incorrect words since TP (unlike TN) does not depend on the presence of correct OCR words in a dictionary. Hence we are getting F-score as high as 62.87 for the baseline in this case. We also evaluated the system for Sāhitya domain. For this
Table 6

**ML Classifier’s Error Detection Results for other domains.** Above results shows the generality of the model for different domains of Sanskrit literature.

<table>
<thead>
<tr>
<th>Approach</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>Prec.</th>
<th>Recall</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vedānta gen. dict. lookup baseline</td>
<td>85.52</td>
<td>34.35</td>
<td>65.65</td>
<td>14.48</td>
<td>49.71</td>
<td>85.52</td>
<td>62.87</td>
</tr>
<tr>
<td>Vedānta Plug-in Classifier</td>
<td>79.95</td>
<td>9.80</td>
<td>90.20</td>
<td>20.05</td>
<td>77.95</td>
<td>79.95</td>
<td>78.94</td>
</tr>
<tr>
<td>Sāhitya gen. dict. lookup baseline</td>
<td>64.24</td>
<td>35.36</td>
<td>64.64</td>
<td>35.76</td>
<td>32.86</td>
<td>64.24</td>
<td>43.49</td>
</tr>
<tr>
<td>Sāhitya Plug-in Classifier</td>
<td>87.88</td>
<td>13.37</td>
<td>86.62</td>
<td>12.12</td>
<td>66.52</td>
<td>87.88</td>
<td>75.72</td>
</tr>
</tbody>
</table>

Table 7

**Neural Network’s Error Detection Results.** (*state of the art (Saluja et al. 2017b))

<table>
<thead>
<tr>
<th>Approach</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>Prec.</th>
<th>Recall</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM with fixed delay*</td>
<td>92.64</td>
<td>5.45</td>
<td>94.54</td>
<td>7.36</td>
<td>94.84</td>
<td>92.64</td>
<td>93.72</td>
</tr>
<tr>
<td>Char. level Attention model</td>
<td>81.53</td>
<td>7.74</td>
<td>92.26</td>
<td>18.47</td>
<td>91.92</td>
<td>81.53</td>
<td>86.41</td>
</tr>
</tbody>
</table>

Here, in Table 7, we present the results for the approaches described in Sections 3.2 and 3.3 for 86k pairs used in (Saluja et al. 2017b) with 64:16:20 as train:val:test split. The first row shows the Sanskrit results from (Saluja et al. 2017b). The second row presents the results for the character level attention model. For the attention model, we use characters from OCR word
and its preceding OCR word (as context) at input and characters from the correct word at the output. We tried other contexts at the input as well. Using the context of characters from one word gave optimized F-Score.

F-scores show that using these approaches we can outperform all other ML techniques, but requires a large amount of training data for generic adaptations. Since these models learn error patterns and language based on the dataset, if the test data differs (in terms of OCR confusions/system and/or domain from training data), we can make use of approaches mentioned in the previous sections. Since plug-in-classifier uses general auxiliary sources, we recommend to use it for practical purposes.

5 Suggestion Generation

The results for various ways of exploiting auxiliary sources, to generate appropriate suggestions, are given in (Saluja et al. 2017a) for “Āryabhaṭīyabhaṣya of Nilakaṇṭha III Golapāda(1957)”.

Here, in Table 8, we show the improvement in results due to adaptations of domain dictionary and OCR Confusions on-the-fly for “Āryabhaṭīyabhaṣya of Nilakaṇṭha III Kālakriyāpāda(AnantaśayanaSaṃskṛtaGranthāvaliḥ, 1931)”.

We synthetically generated word images for the words in Sanskrit dictionaries, and OCR-ed them using ind.senz (ind.senz 2014) and extracted around 0.5 million erroneous-correct word pairs. We used the longest common subsequence algorithm (Hirschberg 1977) for generating around 0.78 million OCR character confusions. The row 1 of Table 8 shows the total percentage of correct suggestions obtained using various auxiliary sources with i) words common to dual OCR systems as Domain Vocabulary throughout the document and ii) obtained synthesized confusions. As shown in row 2, we further improved the quality of suggestions by uploading the corrected domain words on-the-fly after the user corrects the page. Adapting the confusions on-the-fly page by page further improved results as shown in row 3. Using real confusions from the primary OCR text and ground truth from other books further helped in improving results as shown in row 4 of Table 8.
Table 8

<table>
<thead>
<tr>
<th>Sources Included</th>
<th>Percentage of Correct Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain words with dual OCR agreement + Synthesized Confusions</td>
<td>36.26</td>
</tr>
<tr>
<td>Prev. + adapting Domain Words/Page</td>
<td>36.38</td>
</tr>
<tr>
<td>Prev. + adapting Confusions/Page</td>
<td>37.14</td>
</tr>
<tr>
<td>Prev. - Synthesized + Real Confusions</td>
<td>39.40</td>
</tr>
</tbody>
</table>

 Improvement in Suggestions with Adaptive sources for “Āryabhaṭīyabhāṣya of Nilakaṇṭha III Kālakriyāpāda (AnantaśayanaSaṃskṛtaGranthāvaliḥ, 1931)”.

6 Conclusions

In this paper, we demonstrate different ML approaches for Sanskrit OCR corrections. Our framework leverages synthesized dictionary, n-gram error confusions and domain vocabularies. Error confusions and domain-specific vocabularies grow on-the-fly with user corrections. We have presented a multi-engine environment which is useful in detecting potential errors. Using various auxiliary sources along with plug-in-classifier we succeed in achieving F-Scores better than (Saluja et al. 2017a). LSTM with fixed delay is outperforming other approaches. Deep neural network-based approaches, however, require higher-level resources like GPU and a large amount of training data. Our system is able to generate correct suggestions for the errors having edit distance as high as 15. As shown in (Saluja et al. 2017a), our GUI is able to reduce the overall cognitive load of the user by providing adequate color coding, generating suggestions, and auto-correcting similar erroneous words. As a future enhancement to the framework, Sandhi splitting using a greedy approach can be improved with better algorithms.
References


A Tool for Transliteration of Bilingual Texts Involving Sanskrit

Nikhil Chaturvedi and Rahul Garg

Abstract: Sanskrit texts are increasingly being written in bilingual and trilingual formats, with Sanskrit paragraphs or shlokas followed by their corresponding English commentary. Sanskrit can also be written in many ways, including multiple ramanized encodings such as SLP-1, Velthuis etc. The need to handle code-switching in such texts is exacerbated due to the requirement of rendering web pages with multilingual Sanskrit content. These need to automatically detect whether a given text fragment is in Sanskrit, followed by the identification of the form/encoding, further selectively performing transliteration to a user specified script. The Brahmi-derived writing systems of Indian languages are mostly rather similar in structure, but have different letter shapes. These scripts are based on similar phonetic values which allows for easy transliteration. This correspondence forms the basis of the motivation behind deriving a uniform encoding schema that is based on the underlying phonetic value rather than the symbolic representation. The open-source tool developed by us performs this end-to-end detection and transliteration, and achieves an accuracy of 99.1% between SLP-1 and English on a Wikipedia corpus using simple machine learning techniques.

1 Introduction

Sanskrit is one of the most ancient languages in India and forms the basis of numerous Indian languages. It is the only known language which has a built-in scheme for pronunciation, word formation and grammar (Maheshwari 2011). It is one of the most used languages of its time and hence encompasses a rich tradition of poetry and drama as well as scientific, technical, philosophical and religious texts. Unfortunately, Sanskrit is now spoken by only a small number of people. The aforementioned literature, though available, remains inaccessible to most of the world. However, in recent years, Sanskrit has shown a resurgence
through various media, with people reviving the language over the internet (Nair and Devi 2011) and through bilingual and trilingual texts.

There exist numerous web-based application tools that provide age-old Sanskrit content to users and assist them with getting an insight into the language. Cologne Sanskrit Dictionary Project (Kapp and Malten 1997) aims to digitize the major bilingual Sanskrit dictionaries. Sanskrit Reader Companion (Goyal and Huet 2013) by INRIA has tools for declension, conjugation, Sandhi splitting and merging along with word stemming. Samsadhani (A. Kulkarni 2017) by University of Hyderabad supports transliteration, morphological analysis and Sandhi. Sanskrit language processing tools developed at the Jawaharlal Nehru University (Jha 2017) provide a number of tools with the final aim of constructing a Sanskrit-Hindi translator. In this paper, we attempt to construct a transliteration tool to render the web pages of the above tools in multiple scripts and encodings at the backend. Through this, we aim to expand the reach of Sanskrit to a wider community, along with the standardization of an open-source tool for transliteration.

The number of bilingual and trilingual content involving Sanskrit has been on a steady rise. For example, the Gita Supersite (Prabhakar 2005) maintained by IIT Kanpur serves as a huge bilingual database of the Bhagavad Gita, the Ramacharitmanas and Upanishads. Traditional texts such as Srisa Chandra Vasu’s translation of the Ashtadhyayi in English (Vasu 1897) exist in a similar format. These works broadly follow a commentary structure with Sanskrit hymns, verses and words followed by their translation in popular modern day languages like English or Hindi. Code-switching (Auer 2013) is the practice of moving back and forth between two languages, or between two dialects/registers of the same language. Due to their commentarial nature, multilingual Sanskrit works constitute massive amounts of code-switching. For example, an excerpt of the Valmiki Ramayana from Gita Supersite: "तपस्वी ascetic, वाल्मिकीकः Valmiki, तपः स्वाध्यायनिरतम् highly delighted in the practice of religious austerities and study of vedas, वांविनदिं वर्षम् eloquent among the knowledgeable, मुनिपुरुषम् preeminent among sages, नारदम् Narada, परिप्रच्छच्छ enquired." This motivates the need for a word-level transliteration tool that tackles areas of code-switching and performs transliteration through an automatic detection of the relevant sections.

Romanisation is another phenomenon that has led to the resurgence of Sanskrit on the Internet. In linguistics, romanisation is the conversion of writing from a different writing system to the Roman (Latin) script. Multiple methods of this transliteration have emerged, although none has emerged as the clear standard. These methods include SLP1, Velthuis, Harvard-Kyoto, ISO15919, WX, IAST and National Library at Kolkata romanisation. Such romanisation makes it easy for
large parts of the world population to pronounce and appreciate Sanskrit verses. Therefore, any standardized transliteration tool for Sanskrit needs to support all the above romanisation encodings

A property of the Sanskrit language and other major Indian languages like Hindi, Marathi, Tamil, Gujarati etc. that forms the basis of our transliteration, is that these languages are written using different letter shapes (scripts) but are rather similar structurally. The same sounds are duplicated across these languages, allowing for easy transliteration. The phonetic sound [ki] (IPA) will be rendered as कि in Devanagari, as ਕਿ in Gurmukhi, and as கி in Tamil. Each having different code-points in Unicode and ISCII1. This enabled us to formulate a mediating encoding schema that encodes the sound of a syllable rather than any syntactical aspect, thus allowing seamless transliteration between any 2 given scripts.

Romanised Sanskrit however exacerbates the problem of code-switching. The requirement for a general-purpose transliteration tool is now to differentiate between two words of the same script, which turns out to be a non-trivial problem. We again use the intuition of phonetics to overcome this problem. Certain sounds (or sequence of sounds) occur more frequently in some languages than in others. This allows us to formulate the classifier using a simple Naive Bayes model that functions on all possible substrings of a given word. We manage to achieve a classification accuracy of 99.1% between English and Sanskrit written using SLP1.

The rest of the paper is organized as follows. In section 2 we briefly describe presently used encoding and romanisation schemes for writing Sanskrit texts. Section 3 describes the prevalent transliteration tools available. Sections 4 and 5 respectively describe our transliterator and script detector. In section 6, we present our results and discuss possible future work.

2 Sanskrit Alphabet and Encodings

The Sanskrit alphabet comprises 5 short (हर्ष्य)2 vowels, 8 long (दीघ) vowels and 9 prolated (शुन्त) vowels. Each of these vowels can be pronounced in three different

---

1Indian Script Code for Information Interchange (ISCII) is an 8-bit coding scheme for representing the main Indic scripts. Unicode is based on ISCII, and with Unicode being the standard now, ISCII has taken a back seat.

2In this paper, we use Unicode Devanagari enclosed within round brackets for better understanding through popular Sanskrit terms.
ways: acute accent (उद्दात्त), grave accent (अनुदात्त) and circumflex (स्वरित). Vowels in acute accent are written as before (अ), in grave accent, a horizontal line is drawn under them (आ) and circumflex vowels are written with a vertical line drawn above them (ऋ). There are 33 consonants including 4 semi-vowels, 3 sibilants and 1 aspirate (ः).

There are several methods of transliteration from Devanagari to the Roman script (a process known as romanization) which share similarities, although no single system of transliteration has emerged as the standard. SLP1 (P. Scharf 2011) and WX (Bharati et al. 1995) map each Devanagari letter to exactly one ASCII symbol. Velthuis (Velthuis 1981) is based on using the ISO 646 repertoire to represent mnemonically the accents used in standard scholarly transliteration. IAST (Trevelyan, Jones, and Williams 1894) incorporates diacritics to represent letters. Harvard-Kyoto (Nakatani 1996) largely resembles SLP1 in terms of using capital letters in its mapping. ITRANS (Chopde 1991) exists as a pre-processing package and hence is widely used for electronic documents. ISO15919 (ISO/TC-46 2001) like IAST uses diacritics. A comparison of some of the above schemes was first presented in (Huet 2009). A more detailed comparison is also given under Appendix A of this paper. Reader may refer to (P. M. Scharf and Hyman 2012) for a thorough analysis and discussion.

Unicode has designated code blocks for almost all major Indian scripts. The supported scripts are: Assamese, Bengali (Bangla), Devanagari, Gujarati, Gurmukhi, Kannada, Malayalam, Oriya, Tamil, and Telugu among others. Across scripts, Unicode respects alphabet correspondence and letters with similar phonetic values are assigned the same code-points. As a result, transliteration can be done easily with a mere offsetting. In Unicode, the Devanagari symbol (अ) is coded as U+0905, whereas its representation in Gurmukhi script is (ਅ) which is coded as U+0A05. In comparison, the symbol (क) in Unicode Devanagari has its code as U+0915 while in Gurmukhi is (ਕ) with the code as U+0A15. Therefore, transliteration of Sanskrit texts written using Unicode in Indian scripts can be easily done by simply changing the offset value.

However, the Unicode encoding doesn’t represent the language in its true essence. Hindi, Sanskrit and most other Indian languages are centred around phonetic values. Hence the encoded token should ideally represent the entire sound rather than it being split into different symbols for consonants and vowels. Since Unicode is based on the display of fonts and not the underlying phonetic structure, it requires significant parsing to figure out anything about the letter from its corresponding encoding, which section of consonants it belongs to, whether it is voiced or unvoiced etc. For example, the symbol (ॐ) stands for the consonants
(स) and (र) followed by the vowel (ई). Its Unicode representation will consist of (स + ् + र + ् + ई). Phonetically 3 units (स and र and ई), but represented in Unicode through 5 Unicode characters. Our tool fixes this issue by creating a new representation that encapsulates the consonants and the vowels (or lack of it) in a single encoding.

A comprehensive phonetic encoding (PE) based on the Ashtadhyayi rules for computational processing of Sanskrit language has been described in (Sohoni and M. Kulkarni 2015). In order to implement this encoding in a general purpose programming language, a 20-bit encoding scheme was proposed. Although this encoding is phonetically rich, it is possible to compact it into fewer bits without compromising on the essential phonetic information present in the language. Our proposed internal encoding described in the following sections aims to achieve this goal.

### Table 1

<table>
<thead>
<tr>
<th>Tool</th>
<th>UniDev</th>
<th>SLP1</th>
<th>ITRANS</th>
<th>IAST</th>
<th>ISO</th>
<th>IAST</th>
<th>Harv. Kyoto</th>
<th>WX</th>
<th>Sohoni PE</th>
<th>Bilingual Support</th>
<th>Open Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITRANS</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Sanscript</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Aksharamukha</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Samsaar-dhanii</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Google Input</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Proposed Tool</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 3 Existing Transliteration Tools

A number of tools exist as of today for Sanskrit transliteration to other scripts and encodings. We present a brief survey of the same. Aksharamukha (Rajan 2001), Sanscript (Prasad 2015) and ITRANS (Chopde 1991) are some of the tools currently used for transliteration in Sanskrit. Google Input is another tool that is used to transliterating Devanagari to English. Though Aksharamukha and ITRANS support the romanised forms of Sanskrit, none of the aforementioned tools manage to handle bilingual scenarios. Most of these (except Sanscript) are also not open source and hence cannot be utilized by Sanskrit Developers. These tools have been summarised in Table 1.
International Phonetic Alphabet (IPA) is an internationally accepted scheme for encoding phonetic sounds. However, it has a number of representational and backward transliteration issues because of being completely sound based. The imported sounds (नुक्त) don’t share any correspondence to their roots. The sounds of (ऋ) and (िर) have the same representation in IPA, making it impossible to differentiate them while translating back. Anuswar (अनुस्वार) has multiple representations based on context, but none is unique to it (m, n, chandra). Visarga (विसर्ग) has the same representation as (ऋ).

WX and SLP encoding schemes are also phonetic in nature. However, the Sanskrit language alphabet system has a rich structure that categorizes the phonemes according to the place of pronunciation (Gutturals, Palatals, Retroflex, Dentals, Labials), the amount of air exhaled (aspirated or unaspirated) and whether the consonents are voiced and unvoiced. These attributes of the phonemes are very useful while carrying out phonological or morphological processing in the language. It is desirable to have an encoding that represents these attributes of the language in a natural manner.

Due to these inefficacies of existing tools and phonetic schemes, we created our own unified encoding schema which naturally encodes the sounds in the Sanskrit Varnamala (as described in the next section).
4 Design of the Transliterator

4.1 Internal Representation

We created an internal encoding that represents simple syllables (single consonant followed by single vowel) using 16-bits. Initial 5 bits in this encoding represent the script (hence can support 32 scripts). Next 6 bits represent the consonants (व्यंजन) while the last 5 bits represent the vowel (स्वर/मात्र). Each 16-bit code represents a specific simple syllable sound, which can further be reverse mapped to a specified destination script. In contrast, the Unicode representation for a simple Sanskrit syllable would require 32-bits under the UTF-16 encoding, and 48-bits under the UTF-8 encoding.

With 33 consonants and 14 vowels, we can encode their permutations using just 9-bits versus the 11-bits that we currently are using. But, we preferred to use some extra bits so as to keep our representation clean and allow for the bits within themselves to represent certain nuances of the Sanskrit language. Our encoding respects the phonetic structure of the language as described by Panini in a manner very similar to the phonetic encoding (PE) of (Sohoni and M. Kulkarni 2015). Just by using the bit patterns of this encoding, it is possible to figure out important phonetic characteristics of the letters.

For the 5 bits of the vowels, the second-last bit represents whether the vowel is a simple vowel (अ, इ, उ, ऋ, ल) or a diphthong/compound vowel (पू, पे, ओ, ओ). The last bit of the vowels represent the length of the vowel. Long (दीघ) vowels (आ, इ, उ, ऋ, ल, ए, ऐ, ओ, औ) will have their last bit as 1, while short (हस्त) vowels (अ, इ, उ, ऋ, ल) will have their last bit as 0.

In the case of consonants, the first 3 bits represent the place of pronunciation of the letter. Thus, the sequence 000 refers to the throat as the source and the letters are called Gutturals (क, ख, ग, घ, ङ, ह), 001 refers to the palate and letters are called Palatals (च, छ, ज, झ, ञ, य, श). 010 refers to the murdha and are called Retroflex letters (ट, ठ, ड, ढ, ण, र, ष). Lastly, 101 contains letters with source of origin as the teeth and are called Dentals (त, थ, द, ध, न, ल, स). Lastly, 100 refers to the lips and the letters are called Labials (प, फ, ब, भ, म, व), while 101, 110 and 111 are reserved for special symbols and accents.

As for the last 3 bits of consonants, the first of these is 0 for stop-consonants (स्पर्श) which means non-nasal, non-semivowel and non-sibilant consonants. The second of these bits represents voicing (whether or not the vocal chords vibrate in pronunciation). It is 1 for voiced (घोष) consonants like (ग, घ) while 0 for unvoiced
(अघोष) consonants like (क, ख). The last of these bits represents aspiration (a puff of air at the end of the pronunciation). It is 1 for aspirated (महाप्राण) consonants (ख, घ) while 0 for unaspirated (अन्त्यप्राण) consonants (क, ग). A table describing the proposed encoding is given in Appendix B.

4.2 Transliterator Pipeline

The transliterator takes a bilingual (or trilingual) document as its input and produces an output document in the same format where the Sanskrit text is transcribed into the specified script. It consists of 5 stages, namely fragmentation, script detection, tokenisation, universalisation and specification, explained below.

Fragmentation refers to splitting the given text into smaller fragments (words, sentences, paragraphs etc). The assumption shall be that the script and encoding remain same through these fragments if not through the entire text. In order to make it most general, currently fragmentation is done at the word level.

Script Detection refers to identification of the language, scripts and encodings for the various fragments through a Naive Bayes model described in section 5.

Tokenisation refers to splitting the fragment further into tokens, each of which represent a single sound. It is similar to the concept of English syllables. So the sound [ki] will be seen as one single token under this model.

Universalisation refers to the conversion of the token to the universal 16-bit encoding designed by us. This is done through pre-populated hash maps for different script tokens.
**Specification** refers to the conversion of the universal encoding to the specified script using pre-populated hash maps.

### 4.3 Use Cases

<table>
<thead>
<tr>
<th></th>
<th>Pred English</th>
<th>Pred SLP-1</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual English</td>
<td>72294</td>
<td>1605</td>
<td>97.8%</td>
</tr>
<tr>
<td>Actual SLP-1</td>
<td>213</td>
<td>25004</td>
<td>99.2%</td>
</tr>
<tr>
<td>Precision</td>
<td>99.7%</td>
<td>93.9%</td>
<td>98.2%</td>
</tr>
</tbody>
</table>

**Table 2**

*Confusion matrix of English vs Sanskrit-SLP-1 without proper noun correction*

#### 4.3.1 Web-based Applications

One of the foremost uses of our transliteration tool is its utility for web-based applications. A number of websites nowadays serve the historical epics like the Gita and the Ramayana that were originally written in Sanskrit. Along with this, many websites also provide an avenue for people to learn Sanskrit grammar, understand conjugation and splitting of words, along with explaining the various forms of Sanskrit verb roots. Such websites are as of now available only in the Devanagari script. Our tool can be used to transliterate these pages to a user defined script/encoding at the backend. The model for this use case has been depicted in Figure 1. We insert our tool as a middle-ware between the backend and the frontend. The user specifies his required script/encoding on the frontend and all outgoing pages from the server pass through our tool while getting converted to that required script. The frontend then renders the converted HTML to the user for a seamless experience.

#### 4.3.2 Bilingual Texts

Numerous Sanskrit texts have been modified to bilingual and trilingual texts through their translation to popular modern languages like English and Hindi.
These works exist in a commentary form and incorporate massive amounts of code-switching. To represent any such text in a script different to that of its origin turns out to be an ordeal because the tool needs to conditionally perform the transliteration at a micro-level. This problem gets exacerbated when the Sanskrit verses are written using their Romanised form while the translation language is English. Figure 2 depicts the model for this use case.

### 4.3.3 User Driven

The third use for our tool is on the lines of Google input tools. Our tool can allow a user to enter a line of Sanskrit (in any script) intertwined with English and will output the resulting sentence to the user after transliteration. This not only provides an unmatched amount of flexibility to the user, but also has abundant relevance in the growing age of multi-lingual social media.

<table>
<thead>
<tr>
<th>Baseline 67.2%</th>
<th>Pred English</th>
<th>Pred SLP-1</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual English</td>
<td>73178</td>
<td>721</td>
<td>99.0%</td>
</tr>
<tr>
<td>Actual SLP-1</td>
<td>205</td>
<td>25012</td>
<td>99.2%</td>
</tr>
<tr>
<td>Precision</td>
<td>99.7%</td>
<td>97.2%</td>
<td>99.1%</td>
</tr>
</tbody>
</table>

(a) English vs SLP-1

<table>
<thead>
<tr>
<th>Baseline 58.6%</th>
<th>Pred English</th>
<th>Pred Velthuis</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual English</td>
<td>72649</td>
<td>1250</td>
<td>98.3%</td>
</tr>
<tr>
<td>Actual Velthuis</td>
<td>860</td>
<td>24357</td>
<td>96.6%</td>
</tr>
<tr>
<td>Precision</td>
<td>98.8%</td>
<td>95.1%</td>
<td>97.9%</td>
</tr>
</tbody>
</table>

(b) English vs Velthuis

| Confusion matrix of English vs Sanskrit using different Romanisation schemata Part-1 |

Table 3
Table 4
Confusion matrix of English vs Sanskrit using different Romanisation schemata Part-2

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Pred</th>
<th>Pred</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>English</td>
<td>ITRANS</td>
<td></td>
</tr>
<tr>
<td>Actual English</td>
<td>72778</td>
<td>1121</td>
<td>98.5%</td>
</tr>
<tr>
<td>Actual ITRANS</td>
<td>645</td>
<td>24572</td>
<td>97.4%</td>
</tr>
<tr>
<td>Precision</td>
<td>99.1%</td>
<td>95.6%</td>
<td>98.2%</td>
</tr>
</tbody>
</table>

(a) English vs ITRANS

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Pred</th>
<th>Pred</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>English</td>
<td>HK</td>
<td></td>
</tr>
<tr>
<td>Actual English</td>
<td>73269</td>
<td>630</td>
<td>99.1%</td>
</tr>
<tr>
<td>Actual HK</td>
<td>199</td>
<td>25018</td>
<td>99.2%</td>
</tr>
<tr>
<td>Precision</td>
<td>99.7%</td>
<td>97.5%</td>
<td>99.2%</td>
</tr>
</tbody>
</table>

(b) English vs Harvard-Kyoto

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Pred</th>
<th>Pred</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>English</td>
<td>ISO</td>
<td></td>
</tr>
<tr>
<td>Actual English</td>
<td>73576</td>
<td>323</td>
<td>99.6%</td>
</tr>
<tr>
<td>Actual ISO</td>
<td>94</td>
<td>25123</td>
<td>99.6%</td>
</tr>
<tr>
<td>Precision</td>
<td>99.9%</td>
<td>98.7%</td>
<td>99.6%</td>
</tr>
</tbody>
</table>

(c) English vs ISO15919

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Pred</th>
<th>Pred</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>English</td>
<td>IAST</td>
<td></td>
</tr>
<tr>
<td>Actual English</td>
<td>73368</td>
<td>531</td>
<td>99.3%</td>
</tr>
<tr>
<td>Actual IAST</td>
<td>111</td>
<td>25106</td>
<td>99.6%</td>
</tr>
<tr>
<td>Precision</td>
<td>99.8%</td>
<td>97.9%</td>
<td>99.4%</td>
</tr>
</tbody>
</table>

(d) English vs IAST
5 Design of the Script Detector

Differentiating English from Indian scripts, or differentiating different Indian scripts is easy as each uses a different alphabet with a different Unicode range. Hence, one can easily achieve a Word-level classifier with 100% accuracy. However, differentiating English text from Romanized Sanskrit/Hindi texts requires learning, specially to be able to do such classification at word-level. For this we designed a modified Naive Bayes classifier described next.

5.1 Modified Naive-Bayes Classifier

While learning, two dictionaries are maintained. The first dictionary compiles all seen complete words, while the other forms an occurrence database of all possible substrings of length \(<=10\). The intuition is that certain sounds (or sequence of sounds) occur more frequently in some languages then the others.

For a word, define the absolute frequency of a word as the actual number of occurrences for that word for a given language in the training dataset. On the other hand, the relative frequency of a given word is defined as its fraction of occurrences in the given language versus all other languages under consideration. While classifying, if the word is seen and the absolute as well as relative frequency is above a pre-set threshold for a particular language in training data, we classify it as that language. We use the relative frequency metric to account for mixed language nature of Wikipedia pages used as our dataset.

If the classifier encounters an unseen word, it is broken into all possible substrings of length \(>=2\) and length \(<=10\). Subsequently, the probability of seeing a substring given a language, \(p(\text{substr} \mid \text{lang})\), over all substrings of word using the trained substring dictionary is computed. This is a simplified version of the Naive Bayes model for the problem at hand. The word is classified to the language for which this metric turns out to be the maximum.

5.2 Training and Test Data

Training Data: One thousand random Wikipedia pages for both English and Sanskrit were used as the training data. The Sanskrit pages were converted to different Romanised Sanskrit encodings (such as SLP-1) using our universal encoder. We then parse out the irrelevant HTML meta-data and tags, stripping it down to just plain text content.

Test Data: One hundred more such random pages for both languages were used as the test data.
6 Results and Future Work

We tested our word-level language detection model on 100 random Sanskrit Wikipedia pages (after converting them to the 6 most popular romanisation schemes of SLP1, Velthuis, ITRANS, Harvard-Kyoto, ISO15919 and IAST). During our testing, we discovered that multiple English proper nouns like 'Bopanna' or 'Kannada' were getting classified as SLP-1 leading to a lower recall for English. In our opinion, such a misclassification aligns with the intention of the tool as it classifies the origin based on the prevalent sounds in the word. For Indian proper nouns appropriated to English these sounds still remain similar to those of their Sanskrit roots, and hence rather should be classified as that. These earlier results are presented in Table 2.

The final confusion matrices, obtained after manually removing proper nouns from the training and test dataset, are shown in Table 4. Each scheme shown has a corresponding baseline to compare our results with, shown in the top left cell. For SLP1, this baseline was the existence of a capital letter in the middle of a word. For Velthuis, it was the existence of a full stop in the middle of a word or the existence of doubly repeated vowels. For ITRANS, the baseline was similar to Velthuis, with repeated 'L' and 'R' instead of full stop. For Harvard-Kyoto, we selected the baseline as capital in the middle of the word alongside repeated 'L' and 'R'. Lastly, for ISO15919 and IAST, it was kept as the existence of a letter beyond the simple English letters and punctuation within a word.

As one can notice in Table 4, we in general attain a high precision for English and a high recall for the romanised words. A large number of misclassified words in both the English and SLP1 cases are 2-3 letter words. 'ati', 'ca' etc. are examples of SLP1 words misclassified as English, while 'Raj', 'Jan', 'are' etc. are examples of English words misclassified as SLP1. For these words, the modified Naive-Bayes model does not end up having enough information for correct classification.

We also tested our tool on a bilingual text test case by converting a extract from an English commentary on Ramayana from Gita-supersite (Prabhakar 2005) to an mixture of SLP-1 and English. Subsequently, we converted the previous result back to Unicode Devanagari and English to see its differences with the original text. As can be seen in Figure 3, the transliteration from Devanagari-English to SLP1-English has a 100% accuracy due to our tool exploiting the difference in Unicode for the two scripts.

Our tool is available at https://github.com/709nikhil/sanskrit-transliteration. This tool can be further improved in several ways. The primary one being heuristically
breaking down word into syllables rather than substrings, to provide a stronger basis for the phoneme intuition. One could also use machine learning approaches other than Naive Bayes, such as deep learning methods or conditional random fields (CRFs) (Lafferty, McCallum, and Pereira 2001). One could also incorporate contextual history into the transliteration to deal with the problem of incorrect classification of proper nouns, thereby aiming at a near perfect accuracy.

Acknowledgments

We thank the anonymous referees and the editors for their meticulous comments on the manuscript which helped in significantly improving the quality of the final paper.
References


# Appendix A: Comparison of various Devanagari Romanisations

<table>
<thead>
<tr>
<th>Devanagari</th>
<th>Unicode</th>
<th>Velthuis</th>
<th>SLP-1</th>
<th>WX</th>
<th>ITRANS</th>
<th>Harvard-Kyoto</th>
<th>IAST</th>
<th>ISO-15919</th>
</tr>
</thead>
<tbody>
<tr>
<td>अ</td>
<td>U+0905</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>आ</td>
<td>U+0906</td>
<td>aa</td>
<td>A</td>
<td>A</td>
<td>a/a</td>
<td>A</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>इ</td>
<td>U+0907</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>ई</td>
<td>U+0908</td>
<td>I</td>
<td>I</td>
<td>i/i</td>
<td>I</td>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>उ</td>
<td>U+0909</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
<tr>
<td>ऊ</td>
<td>U+090A</td>
<td>uu</td>
<td>U</td>
<td>U</td>
<td>a/u</td>
<td>U</td>
<td>u</td>
<td>u</td>
</tr>
<tr>
<td>ऋ</td>
<td>U+090F</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>ऌ</td>
<td>U+0910</td>
<td>al</td>
<td>E</td>
<td>E</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>ऍ</td>
<td>U+0913</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>ऎ</td>
<td>U+0914</td>
<td>au</td>
<td>O</td>
<td>O</td>
<td>a/u</td>
<td>au</td>
<td>au</td>
<td>au</td>
</tr>
<tr>
<td>ऊ</td>
<td>U+090B</td>
<td>.r</td>
<td>r</td>
<td>q</td>
<td>RRI/Ri</td>
<td>R</td>
<td>ṭ</td>
<td>r</td>
</tr>
<tr>
<td>ॠ</td>
<td>U+090C</td>
<td>.l</td>
<td>L</td>
<td>L</td>
<td>LLI/Li</td>
<td>Ṭ</td>
<td>l</td>
<td>l</td>
</tr>
<tr>
<td>ऌ</td>
<td>U+090D</td>
<td>.m</td>
<td>M</td>
<td>M</td>
<td>M/N/m</td>
<td>M</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>ॡ</td>
<td>U+090E</td>
<td>.n</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>अं</td>
<td>U+0915</td>
<td>ka</td>
<td>ka</td>
<td>ka</td>
<td>ka</td>
<td>ka</td>
<td>ka</td>
<td>ka</td>
</tr>
<tr>
<td>अः</td>
<td>U+0916</td>
<td>kha</td>
<td>kha</td>
<td>kha</td>
<td>kha</td>
<td>kha</td>
<td>kha</td>
<td>kha</td>
</tr>
<tr>
<td>अँ</td>
<td>U+0917</td>
<td>ga</td>
<td>ga</td>
<td>ga</td>
<td>ga</td>
<td>ga</td>
<td>ga</td>
<td>ga</td>
</tr>
<tr>
<td>गः</td>
<td>U+0918</td>
<td>gha</td>
<td>gha</td>
<td>gha</td>
<td>gha</td>
<td>gha</td>
<td>gha</td>
<td>gha</td>
</tr>
<tr>
<td>अऄ</td>
<td>U+0919</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
</tr>
<tr>
<td>अऎ</td>
<td>U+091A</td>
<td>ca</td>
<td>ca</td>
<td>ca</td>
<td>ca</td>
<td>ca</td>
<td>ca</td>
<td>ca</td>
</tr>
<tr>
<td>अए</td>
<td>U+091B</td>
<td>cha</td>
<td>Cha</td>
<td>Cha</td>
<td>cha</td>
<td>cha</td>
<td>cha</td>
<td>cha</td>
</tr>
<tr>
<td>अऐ</td>
<td>U+091C</td>
<td>ja</td>
<td>ja</td>
<td>ja</td>
<td>ja</td>
<td>ja</td>
<td>ja</td>
<td>ja</td>
</tr>
<tr>
<td>अऒ</td>
<td>U+091D</td>
<td>jha</td>
<td>Jha</td>
<td>Jha</td>
<td>Jha</td>
<td>Jha</td>
<td>Jha</td>
<td>Jha</td>
</tr>
<tr>
<td>अओ</td>
<td>U+091E</td>
<td>ta</td>
<td>Ta</td>
<td>Ta</td>
<td>Ta</td>
<td>Ta</td>
<td>Ta</td>
<td>Ta</td>
</tr>
<tr>
<td>अऔ</td>
<td>U+091F</td>
<td>.tha</td>
<td>Wa</td>
<td>Tha</td>
<td>Tha</td>
<td>Tha</td>
<td>Tha</td>
<td>Tha</td>
</tr>
<tr>
<td>अक</td>
<td>U+0920</td>
<td>.tha</td>
<td>Wa</td>
<td>Ta</td>
<td>Ta</td>
<td>Ta</td>
<td>Ta</td>
<td>Ta</td>
</tr>
<tr>
<td>अञ</td>
<td>U+0921</td>
<td>.da</td>
<td>Da</td>
<td>Da</td>
<td>Da</td>
<td>Da</td>
<td>Da</td>
<td>Da</td>
</tr>
<tr>
<td>अट</td>
<td>U+0922</td>
<td>.dha</td>
<td>Dha</td>
<td>Dha</td>
<td>Dha</td>
<td>Dha</td>
<td>Dha</td>
<td>Dha</td>
</tr>
<tr>
<td>अठ</td>
<td>U+0923</td>
<td>.na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
</tr>
<tr>
<td>अड</td>
<td>U+0924</td>
<td>.na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
<td>Na</td>
</tr>
<tr>
<td>अढ</td>
<td>U+0925</td>
<td>tha</td>
<td>Tha</td>
<td>Tha</td>
<td>Tha</td>
<td>Tha</td>
<td>Tha</td>
<td>Tha</td>
</tr>
<tr>
<td>अण</td>
<td>U+0926</td>
<td>.da</td>
<td>Da</td>
<td>Da</td>
<td>Da</td>
<td>Da</td>
<td>Da</td>
<td>Da</td>
</tr>
<tr>
<td>अत</td>
<td>U+0927</td>
<td>.dha</td>
<td>Dha</td>
<td>Dha</td>
<td>Dha</td>
<td>Dha</td>
<td>Dha</td>
<td>Dha</td>
</tr>
<tr>
<td>अथ</td>
<td>U+0928</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
</tr>
<tr>
<td>अद</td>
<td>U+0929</td>
<td>pa</td>
<td>pa</td>
<td>pa</td>
<td>pa</td>
<td>pa</td>
<td>pa</td>
<td>pa</td>
</tr>
<tr>
<td>अध</td>
<td>U+092A</td>
<td>.pha</td>
<td>Pa</td>
<td>pha</td>
<td>pha</td>
<td>pha</td>
<td>pha</td>
<td>pha</td>
</tr>
<tr>
<td>अन</td>
<td>U+092B</td>
<td>.pha</td>
<td>Pa</td>
<td>pha</td>
<td>pha</td>
<td>pha</td>
<td>pha</td>
<td>pha</td>
</tr>
<tr>
<td>अऩ</td>
<td>U+092C</td>
<td>ba</td>
<td>ba</td>
<td>ba</td>
<td>ba</td>
<td>ba</td>
<td>ba</td>
<td>ba</td>
</tr>
<tr>
<td>अप</td>
<td>U+092D</td>
<td>.bha</td>
<td>Ba</td>
<td>bha</td>
<td>bha</td>
<td>bha</td>
<td>bha</td>
<td>bha</td>
</tr>
<tr>
<td>अफ</td>
<td>U+092E</td>
<td>.ma</td>
<td>ma</td>
<td>ma</td>
<td>ma</td>
<td>ma</td>
<td>ma</td>
<td>ma</td>
</tr>
<tr>
<td>अब</td>
<td>U+092F</td>
<td>ya</td>
<td>ya</td>
<td>ya</td>
<td>ya</td>
<td>ya</td>
<td>ya</td>
<td>ya</td>
</tr>
<tr>
<td>अ़</td>
<td>U+0930</td>
<td>ra</td>
<td>ra</td>
<td>ra</td>
<td>ra</td>
<td>ra</td>
<td>ra</td>
<td>ra</td>
</tr>
<tr>
<td>अऽ</td>
<td>U+0932</td>
<td>la</td>
<td>la</td>
<td>la</td>
<td>la</td>
<td>la</td>
<td>la</td>
<td>la</td>
</tr>
<tr>
<td>अा</td>
<td>U+0935</td>
<td>va</td>
<td>va</td>
<td>va</td>
<td>va</td>
<td>va</td>
<td>va</td>
<td>va</td>
</tr>
<tr>
<td>े</td>
<td>U+0936</td>
<td>.sa</td>
<td>Sa</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
</tr>
<tr>
<td>े</td>
<td>U+0937</td>
<td>.sa</td>
<td>Sa</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
</tr>
<tr>
<td>े</td>
<td>U+0938</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
<td>sa</td>
</tr>
<tr>
<td>े</td>
<td>U+0939</td>
<td>ha</td>
<td>ha</td>
<td>ha</td>
<td>ha</td>
<td>ha</td>
<td>ha</td>
<td>Ha</td>
</tr>
</tbody>
</table>
Appendix B: Proposed Encoding Schema

The characters in our encoding schema are represented in 16 bits as follows: $b_{15}...b_{11}b_{9}b_8-b_7b_6b_5-b_4b_3b_2-b_1b_0$. The bits $b_{15} - b_{11}$ are used to represent the script. The remainder bits are represented as given in the tables 5 and 6 below. Null for the consonant part represents a pure vowel, whereas null for the vowel part represents consonants without a vowel sound. For example, अ is represented as 00000-111-111-000-00. The symbol श्ष which is broken up as (क + ष = क + ष + अ) will be represented as two 16-bit units, 00000-000-000-111-11 representing (क) and 00000-010-001-000-00 representing (ष).

### Table 5
Mapping for 6 consonant bits

<table>
<thead>
<tr>
<th>$b_7b_6b_5$</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>क</td>
<td>ख्</td>
<td>ग्</td>
<td>घ्</td>
<td>ह्</td>
<td>ङ्</td>
<td>च्</td>
<td>छ्</td>
</tr>
<tr>
<td>001</td>
<td>च्</td>
<td>छ्</td>
<td>ज्</td>
<td>झ्</td>
<td>झ्</td>
<td>छ्</td>
<td>च्</td>
<td>छ्</td>
</tr>
<tr>
<td>010</td>
<td>ट्</td>
<td>ठ्</td>
<td>ड्</td>
<td>ढ्</td>
<td>ण्</td>
<td>त्</td>
<td>थ्</td>
<td>द्</td>
</tr>
<tr>
<td>011</td>
<td>न्</td>
<td>घ्</td>
<td>ङ्</td>
<td>झ्</td>
<td>झ्</td>
<td>ङ्</td>
<td>घ्</td>
<td>ङ्</td>
</tr>
<tr>
<td>100</td>
<td>प्</td>
<td>फ्</td>
<td>भ्</td>
<td>फ्</td>
<td>भ्</td>
<td>प्</td>
<td>फ्</td>
<td>भ्</td>
</tr>
<tr>
<td>101</td>
<td>ध्</td>
<td>ध्</td>
<td>ध्</td>
<td>ध्</td>
<td>ध्</td>
<td>ध्</td>
<td>ध्</td>
<td>ध्</td>
</tr>
<tr>
<td>110</td>
<td>उ</td>
<td>ऊ</td>
<td>ऋ</td>
<td>ऍ</td>
<td>ऋ</td>
<td>ऍ</td>
<td>ऊ</td>
<td>उ</td>
</tr>
<tr>
<td>111</td>
<td>ब</td>
<td>प</td>
<td>य</td>
<td>ब</td>
<td>प</td>
<td>य</td>
<td>ब</td>
<td>प</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$b_1b_0$</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>अ</td>
<td>आ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>इ</td>
<td>ई</td>
<td>ए</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>ऋ</td>
<td>ऋ</td>
<td>ए</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>ओ</td>
<td>औ</td>
<td>औ</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>ओ</td>
<td>औ</td>
<td>औ</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 6
Mapping for 5 vowel bits
Modeling the Phonology of Consonant Duplication and Allied Changes in the Recitation of Tamil Taittirīyaka-s

BALASUBRAMANIAN RAMAKRISHNAN

Abstract: The phonetics of the Vedas are described by the prātiśākhya and śikṣā texts. Each Veda has its own prātiśākhya as well as specific śikṣā texts. There is also a Pāṇiniya śikṣā, which talks about general rules applicable to all Veda-s. While there are similarities between the various prātiśākhya and śikṣā texts, there also tend to be important differences, leading to differences in the modes of chanting the Veda-s. Some differences are obvious, but a significant percentage of the differences can be detected only by the trained ear, consonant duplication being in the latter category. While consonant duplication is not always faithfully followed by reciters of classical Sanskrit verses, by even trained pañḍita-s, duplication is faithfully preserved, largely adhering to the Taittirīya-Yajuh Prātiśākhym (TYP) rules, especially by the Tamil Taittiriyaka-s (TT). Additional rules are to be found in various śikṣā texts and some rules are known only from traditional practice. It is important to study the printed texts of the TTs which use the Grantha script since they offer a very concise representation of some unique duplication rules. Finally, analysis of actual recitation by experts, or field-study, is also required to completely understand the phonetic rules.

The aim of this paper can be summarized as:

1. describe duplication rules among TTs, point out where they deviate from the TYP, compare and contrast with other prātiśākhya-s and Pāṇini,
2. develop an algebraic formulation of TT duplication rules,
3. develop a Non-Deterministic Finite State Transducer (ND-FST) model from the algebraic formulation, and finally
4. a Perl implementation of the model as a tool to study TT duplication rules.

181
1 Introduction

Consonant duplication is one of the truly arcane topics in Sanskrit. This paper is about consonant duplication among TTs, a phonological process, when conjunct consonants (saṃyutākṣara) occur. While it should be clear, it is worth mentioning that this is different from the duplication of syllables in a verbal root form, e.g., the third class of verbs reduplicated aorist, etc., which are morphological processes. Consonant duplication occurs when specific groupings of vowels and consonants (to be specified later) occur. The duplicated syllable is always a full consonant and never a vowel and this is made explicit by using the phrase “consonant duplication”. A vowel can be duplicated in the long (dirgha) svarita accent, but the topic of this paper is not accentuation in the Veda. The list of vowels and consonants are mostly the same in classical and Vedic Sanskrit. However, there are a few differences between the different lists, even between the prātiśākhya-s of different Vedas. This paper will not discuss these differences in detail and a good resource for this is Whitney (1871). This paper will also not examine the phonetics of the standard Sanskrit vowels and consonants, which is a well-researched topic. A good source for phonetic analysis, including many Vedic forms, is Allen (1953), and many more references can be found in Scharfe (1973). However, we will discuss phonetics of a few specific vowels and consonant forms which are important in and peculiar to TT recitation, especially during duplication, and which are different from the classical Sanskrit, in the relevant sections.

One of the first people to study duplication was Whitney, in his pathbreaking studies of the prātiśākhya-s of the Atharva and the Taittirīya veda-s (Whitney 1863, 1871). Whitney had specific reservations about the value of the detailed discussions on duplication in the prātiśākhya-s, which he expressed rather forcefully, first in his translation of the Atharva prātiśākhya¹ (Whitney 1863)

“The subject of the duplicated pronunciation of consonants, or of the varṇakrama, as it is sometimes called, is one of the most peculiar in the whole phonetical science of the Hindus. It is also the one, to my apprehension, which exhibits most strikingly their characteristic tendency to arbitrary and artificial theorizing; I have not succeeded in discovering the foundation of fact

¹This can be found under his explanation to 3.28, page 470 (Whitney 1863)
upon which their superstructure of rules is based, or explain-
ing to myself what actual phonetic phenomena, liable to occur
in a natural, or even a strained, mode of utterance, they sup-
posed themselves to have noted, and endeavored thus to reduce
to systematic form. The varṇakrama, however, forms a not in-
conspicuous part of the phonetic system of all the Prātiśākhyas,
and is even presented by Pāṇini (viii. 4. 46-52), although the
latter mercifully allows us our option as to whether we will or
will not observe its rules.”.²

There are two questions which Whitney rightly raises: 1) what the phonetic
basis of duplication is and 2) whether it really makes a difference in Vedic
chanting. In this paper, I’ll concentrate mostly on the latter question. The
concise answer is that duplication does matter quite a bit and is reflected
in the recitation of pañḍita-s trained in the orthodox manner. The first
question is not the focus of this paper. However, in a few places, I will point
out some of the possible phonetic reasons for duplication.

Opinion of the later śikṣā-s

First, we can look at texts which were written after the TYP, i.e., the śikṣā
texts, and see that it is an important topic in these texts as well. The
sarvasammata śikṣā (Finke 1886) actually begins with the invocatory verse

kṛpāluṁ varadaṁ devaṁ praṇipatya gajānanam |
dvitvādimaṁ pravakṣyāmi lakṣaṇaṁ sarvasaṃmatam ||
Having prostrated to the compassionate and wish-granting
elephant-faced God,
I will expound the phonetics of duplication, etc., (which are)
agreeable to all.

Since the sarvasammata śikṣā singles out duplication and is actually largely
about this topic, it can be seen that the later textual tradition also held
that consonant duplication was an important topic for reciters of the Veda.

²He also states in page 313 of his translation of the TYP (Whitney 1871), “Thus is
brought to an end the tedious subject of duplication, the physical foundation of which is of
the obscurest, although the pains with which the Hindu śakhinaḥ have elaborated it, and
the earnestness with which they assert their discordant views respecting it, prove that it
had for them a real, or what seemed like a real, value.”, clearly expressing his reservations
once again on the practical utility of the discussions on duplication.
Furthermore, the śikṣā is somewhat ambitiously titled sarvasammata, i.e., agreeable to all, which implies that a variety of opinions, especially on duplication, existed at the same point in time. This is very clear from the TYP itself, where several contradictory opinions are stated and sometimes with attribution to a specific authority. Whitney had also pointed out very early on that while duplication was a topic treated by all prātiśākhya-s, the TYP is unique in that it presents the contradictory opinion of many different authorities (Whitney 1871). The Vyāsa (Ācārya Śripaṭṭābhirāmaśāstri 1976) and especially the sarvasammata śikṣā clearly try to present a unified theory of duplication by leaving out contradictory opinions and sometimes adding rules not found in the TYP. The sarvasammata śikṣā finishes with the cautionary verse

śikṣā ca prātiśākhyaṃ ca virudhyete parasparam |
śikṣāiva durbaletyāhūḥ śiṃhasyaiva mṛgī yathā ||

(If there is any) mutual contradiction between the śikṣā and prātiśākhya,
They declare that the śikṣā is indeed the weaker, like the deer (in the presence) of a lion.

Clearly the sarvasammata śikṣā’s author was aware that at least some of the śikṣā-s have opinions contradictory to the TYP. However, it should be noted that the same text adds some rules not found in the TYP, and also extends the scope of some rules found in the TYP. Again it was noted by Whitney in his treatment of the svarabhakti in the TYP, where he remarks that the commentator seemed to rely more on his śikṣā text than the TYP itself (Whitney 1871). In this regard, the comment of Scharfe regarding the attitude of Pāṇiniya-s is very perceptive, is applicable in this case as well, and deserves to be quoted in full (Scharfe 1973)

“The aṣṭādhyāyī can be compared to a code of law which is subject to legal interpretation when cases that were not or could not be foreseen by the lawmaker. The courts need a consistent and workable application even to such cases. Lawyers are used to obtaining this application by extrapolating principles embodied in the code which is presumed to be comprehensive and consistent to the minute technical details; seemingly redundant features

---

3It is interesting that the text uses the word śikṣā instead of śikṣā , similar to the Taittirīya Upaniṣad, thus placing the author squarely within the Kṛṣṇa-yajur-veda tradition.
must have their significance. If these extrapolations lead to opposing conclusions this contradiction must be resolved. As a last recourse, the law may be amended. The Pāṇinīya-s are like such lawyers and we miss the point when we castigate them for reading later theories into the original texts.”

Methodology of the Orthodox Practitioners

The methodology so aptly described by Scharfe can be described in general by following a list of Boolean logic rules as given below, and is also applicable to the phonology of duplication:

- If condition $A_1$, then execute Rule 1.
- If conditions $A_1$ AND $A_2$, then execute Rule 2.
- ...
- If conditions $A_1$ AND $A_2$ AND $\cdots$ AND $A_k$, then execute Rule $k$.

A series of rules like the above can be used to give the general and fundamental rule (vidhi) (denoted by Rule 1) and modifying the rule by an exception, followed by an exception to the exception and so on. Note that the TYP itself follows such a formulation by first giving a general rule and adding elements to the list to either discard Rule 1 or modify its performance in certain cases. For example, Rule 2 could be “discard the implementation of Rule 1”, if conditions $A_1$ and $A_2$ occur together, thus providing an exception. The śikṣā-s follow the lead of the TYP and add lawyerly emendations to either modify or discard the rules within in TYP. In this manner, they can claim to be completely consistent with the TYP and yet offer modifications/emendations. Note that the question of why the TYP does not list all the rules followed by the orthodox tradition, or whether some of them were later innovations, does not make sense within this formulation of the orthodox tradition. This is a sensible procedure for interpreting the TYP and śikṣā texts as a unified whole. It is also important for the orthodox tradition to interpret these texts as a unified whole because while in some cases it is clear what the TYP considers as an accepted doctrine, in several other cases what the TYP considers to be an accepted doctrine is known
only from the commentary, as noted by Whitney (Whitney 1871). Thus if the orthodox tradition accepts the TYP as well as the śikṣā texts, to begin with, then the only logical route is the lawyerly one.

Regarding the orthodox method of learning the Vedic recitation, it was pointed out in a recent study of the Vedic tradition in Maharashtra by Larios (Larios 2017) that the “vedamūrti-s” or the orthodox Vedic chanters rarely have even seen a śikṣā text and it has been observed by me in Tamil tradition as well. However the phonological peculiarities of the TTs are very efficiently encoded in their texts published in the Grantha script (Nārāyaṇa Śāstrī 1930, 1931, 1935, Undated[a],[b]; Vaidyanāthasāstri 1905). This should not be confused with an efficient and one-to-one mapping of the actual syllables chanted, e.g., in a manner as described in Scharf and Hyman (2009) from the point of view of computer-based processing or representa-

4It can of course be hypothesized that the TYP represents the “original” rules and the modifications and additions found in the śikṣā-s are later accretions. But this is not correct because the TYP does record a multitude of opinions, especially about duplication, and clearly, the TYP is only one strand among the different schools of thought. It is clear that there were multiple schools of thought regarding certain phonetic and phonological processes, and that the TYP is the first attempt to unify the different strands. It also seems unlikely that there was a single original way of recitation, unless it is assumed that the Taittirīya school was limited originally to a very small geographical area, which is not likely. It’s also unlikely that a historical dating of different doctrines can be obtained by mere textual analysis.

5In page 5 (Larios 2017), “Notably, none of the brāhmaṇas I came in contact with had memorized a śikṣā or prātiṣākhya text at the time of our meeting. Many of those I interviewed during my fieldwork did not possess a copy of such a text, and many others had never seen a śikṣā or prātiṣākhya text in their lives. Yet, as will be shown below, the rules concerning the Vedic recitation are mainly learned through the system of oral transmission, and this includes the pronunciation rules stipulated in the śikṣā or prātiṣākhya texts of each Vedic branch.”

6In Staal (1961) TT recitation is referred to as Tamil Iyer recitation. I prefer the reference as TTs since Iyer, Iyeṅkars and Mādhvas settled in Tamil Nadu all follow the same schools and the phonetics/phonology is the same between these different subsects. The Iyeṅkars usually, but not always, learn the Drāvida pāṭha. But this affects only the text of a few praśna-s of the Taittirīya Āraṇyaka and not the phonetics/phonology. I have observed that the paṇḍīta-s from the Vedic school conducted by one of the four Advaita āmnāya maṭha-s, Dakṣiṇāmnāya Śrīngeri Śrāradā Pīṭham, are very similar to TTs in their recitations, but I do not know if it’s the same case with all reciters from the Karnataka region. I also haven’t had the opportunity of interacting with paṇḍīta-s from the Andhra region, but the few recordings I have heard show at least some differences from the TTs. The Nambudiri recitation is of course quite different from the TTs. Comparing the duplication among the different Southern schools, including the Nambudiris, would be an interesting field study.
tion of general Sanskrit texts, but rather a concise encoding of only the most important śīkṣā rules, which will make complete sense only to people trained within the oral tradition. It should be noted that if the texts were printed exactly as recited, with all duplicated consonants explicitly specified, there would be a prolixity of consonants. This would have had a manifold effect: it would have significantly complicated the task of the scribe, increase the probability of error propagation, as well as hinder text comprehension. The printed texts serve a dual purpose, aid recitation, and comprehension, and this fact was likely not lost to the TTs who started the practice of using the Grantha script. On the other hand, unless some of the more obscure rules are actually represented in the printed text, it might result in the wrong recitation. The Grantha texts offer an admirable combination of precisely encoding arcane duplication rules, yet avoiding prolixity in printing out consonants for people trained within the tradition. The Grantha texts thus make the TYP and śīkṣā texts largely unnecessary for an orthodox reciter. An ability to read Grantha is thus a sine-qua-non for understanding consonant duplication among TTs.

The method by which the Grantha texts achieve this is by writing any conjunct consonant from top to bottom, even if they consist of more than two consonants, or inventing new characters for frequently used complex conjunct consonants. Standalone consonants mean something special, but the meaning is context-dependent and thus the encoding is not one-to-one. For example, a standalone ‘n’ phoneme within a sentence can stand for different ways of pronouncing it, depending on the vowel-consonant cluster surrounding it (as will be seen in a subsequent section). Note again that some of these rules and exceptions can be found neither in the TYP nor in the śīkṣā texts, and are known only from the orthodox tradition. However, these are faithfully reflected in the Grantha texts. In contrast, Devānāgarī typesets typically do not have such complicated conjunct consonants, and the complex phonological rules are largely ignored and not encoded. Another issue is that some conjunct consonants which occur in the TT recitation due to phonological changes, e.g., śna which is transformed into śņa except in the Ḫṇṭhaka borrowings, do not occur at all in classical Sanskrit and are thus not available in Devānāgarī typesets as a standalone conjunct consonant.7 An important effort to rectify these defects, and reflect TYP

---

7The Vaidikavardhini press in Kumbhakonam published these Grantha texts and many excellent prayoga texts but unfortunately went out of business many decades ago. These books are hard to obtain, but facsimile copies of their Taittiriya corpus are available
rules in the Devanāgarī script has been made, but still has some short-
falls compared to the original Grantha texts (Rā Kṛṣṇamūrti śāstri and Rā Gaṇeśvaradrāviḍaḥ 2003a,b). However, this series has a very learned intro-
duction regarding some of the key, though not all, phonological aspects of
TT recitation. Furthermore, this series has very few printing errors, uses a
very pleasing font, and is thus well worth using as a reference text.

Just an ability to read the Grantha texts is not enough and actually
would hinder the proper evaluation of duplication among TTs. I have per-
sonally interacted with a number of kramaṇa-svādhyāyināḥ and have also
learned to recite a good portion of the Taittirīya śākhā, which has been
an invaluable help in understanding duplication rules. Field study would
also clear any questions about whether duplication makes a difference in
Vedic recitation. In my experience with laypeople and experts, I have seen
that most (but not all) laypeople who learn to recite the Vedas, especially
as adults and using texts printed in the Devanagari script, confuse textual
familiarity and chanting with seeming fluency with correct Vedic recitation.
Relying on texts and not listening keenly to expert recitation makes laypeo-
ple, even those who listen to or practice Vedic chanting regularly, unaware
of the phonological sophistication of Vedic chanting, which cannot be com-
pletely captured in textual form. When the correct syllables are not dupli-
cated or wrongly duplicated, the expert recognizes this very quickly, which
answers Whitney’s question on whether duplication makes a difference in
Vedic recitation. The answer is that it does to the expert, but laypeople
may not understand the subtleties. Finally, while actual field study is im-
portant and irreplaceable, a good source for authentic TT recitation is the
audio recordings released by an organization called Vediclinks (Sarma et al.
among the paṇḍita circles in Chennai. One of my teachers Śrī Śrikanṭha Ācāryaḥ showed
me his Kannada text and it seemed to follow the same principle as the Grantha texts with
regard to conjunct consonants and standalone consonants. However, I am not familiar
with the Kannada script and did not perform a careful analysis. A reviewer also pointed
out that the Telugu and Kannada texts follow the same principle as the Grantha texts. It
would be interesting to compare texts in Kannada and Telugu scripts with the Grantha
texts popular in Tamil Nadu. It is also not implied that Devanāgarī typesets do not
have sophisticated conjunct consonant representations, but they are not as clear as the
Grantha scripts which follow a binary rule of conjunct consonants always being represented
as a single top-to-bottom cluster and standalone consonants representing exceptions to the
duplication rules. This can certainly be corrected by introducing new conjunct consonants
in existing Devanāgarī fonts.
2004), which unfortunately seems to be defunct now.\(^8\)

**Fundamental Rule and Notation**

Consonant duplication is specified in fundamentally the same way in different texts, and the basic rule (Condition \(A_1\) in previous section) is common across all prātiśākhya-s as well as Pāṇini:

- Taittirīya Prātiśākhya 14.1 (Whitney 1871) - svarapūrvaṃ vyañjanaṃ dvivarṇaṃ vyañjana-param.
- Śukla-yajuḥ Prātiśākhya 4.99 (Rastogi 1967) - svarāt saṃyogādirdvīrucyate sarvatra.
- R̥k Prātiśākhya 6.1 (Sastri 1931) - svarānusväropahito dvīrucyate saṃyogādiḥ sa krame/vikrame san.
- Atharva Prātiśākhya 3.28 (Whitney 1863) - saṃyogādi svarāt.
- Pāṇini 8.4.47 (Vasu 1898) - anaci ca.

In this paper we will use the terminology from the TYP to refer to vowels, consonants, or groups of consonants. The following abbreviations will be used for a concise description of the rules:

1. **V** - all the consonants (vyañjana-s).

2. **\(C^{i,j}\)** - the mutes (sparśaḥ), the indices \(i\) and \(j\) stand for the row and column in \(5 \times 5\) matrix with the ka, ca, ta, ta and pa series (varga) as the rows. As examples, \(C^{1,4}\) would be gha and \(C^{5,5}\) would be ma.

3. **A** - the semivowels (antasthāḥ) ya, ra, la, and va.

4. **U** - the sibilants (ūṣman) śa, sa, ha, jihvāmūliya, and upadhmānīya.

5. **\(U_a\)** - the four sibilants, śa, sa, sa, and ha.

6. **\(S\)** - vowels and the svarabhakti-s (svara).\(^9\)

---

\(^8\)This is available at [https://archive.org/details/VedicLinks-SriKrishnaYajurVedam-TaitiriyaSamhita](https://archive.org/details/VedicLinks-SriKrishnaYajurVedam-TaitiriyaSamhita) and I would like to thank Mr. N. E. Venkateswaran for bringing this to my attention.

\(^9\)The svarabhakti is the sound the phonemes ‘r’ and ‘l’ attain in certain situations and will be explained in detail later.
7. $S_d$ and $S_h$ stand for long (dirgha) and short (hrasva) vowels respectively.

Specific consonants or vowels will also be used if the rules require specificity.

The basic duplication rule from the TYP 14.1, svara-pūrvam vyañjanam
dvi-varṇaṃ vyañjana-param, actually requires a sequence of consonants after
a vowel, where the sequence length is greater than one. Since two or more
consonants cannot occur after a vowel without a following vowel, except at
the end of a sentence, which is exempted from duplication by TYP 14.15,
the rule can be concisely expressed by the algebraic expression $S_1V_1 \cdots V_kS_2$
where $k \geq 2$. The subscripts stand for the numbering of the vowels and
consonants in the required sequence. The basic transformational rule is thus
explained by the following equation

$$S_1V_1 \cdots V_kS_2 \rightarrow S_1V_1V_1 \cdots V_kS_2 \quad (1)$$

The rule is applicable whether the cluster occurs within a word or across two
words within a sentence, i.e., vākya. In printed texts, there is white space
between two words, when they do not coalesce due to sandhi. However, the
spacing in printed texts is merely for the convenience of comprehension and
does not affect pronunciation. However, there are specific rules for certain
exceptions to duplication, when certain consonants occur at the end of a
word (pada). These word-end rules are to be found in the śīkṣā texts and
in the orthodox tradition and are not specified by the TYP. It is convenient
to treat the word-end exceptions separately. Unless specified, the presence
of white space in the text is ignored in all these rules.

The next section will give a detailed description of all the duplication
rules and exceptions in various situations. Appropriate examples will be
given to illustrate the rules. I use the abbreviations TS, TB, and TA for
the Taittirīya Saṃhitā, Brāhmaṇa and Āraṇyaka respectively. It should be
noted that in some examples, there may be multiple duplicated consonants
and only the consonant which illustrates that particular rule will be shown
explicitly. Another important fact is that the duplication rules are largely
independent of the stress accents udātta, anudātta and svarīta, except in
one case. So the stress markings are not given in the illustrative examples
unless the duplication rule demands it. The rules are very dependent on $V_1$,
and the duplication rules are enumerated by the class to which $V_1$ belongs.
2 Duplication Rules

$V_1$ is a mute

This is the bulk of duplications, since there are 25 mutes, which forms the majority of all consonants.

- $V_2$ is a mute: The mute is duplicated, and if the mute is aspirated, then the corresponding unaspirated mute is used instead as per the rule

\[
\text{dvitiya-caturthayostu vyañjana-uttarayoh pūrvah} | \\
\text{Of the 2nd and 4th after which a consonant (occurs), the} \\
\text{previous (consonant in the series, i.e., the 1st or 3rd is} \\
\text{duplicated).}
\]

There are important exceptions. If $V_1$ and $V_2$ belong to the same series, then there is no duplication, except if the conditions that $V_2$ is the fifth consonant in the series and $V_1$ is not, are met simultaneously. For example, if $V_1$ and $V_2$ are the phonemes ‘t’ and ‘th’ respectively, then there is no duplication. However if $V_1$ and $V_2$ are ‘dh’ and ‘n’, then there is duplication. This is stated by the TYP 14.13 which is an exception to the general rule and 14.14 which is an exception to 14.13 as

\[
\text{savarna-savargiya-parah} | \\
\text{The (consonant which is) of the same quality and series (as} \\
\text{the) later (consonant is not duplicated).} \\
\text{nānuttama uttama paraḥ} | \\
\text{(But) not in the case of a consonant (which is) not the last} \\
\text{(in the series which has) the last (in the series following} \\
\text{it).}
\]

Some examples of duplication in this case are

---

\(^{10}\)The Vyāsa śīksā 363 also gives the two rules concisely as vargiyānanuttarordhve hal savarṇottara eva ca. The same two rules are stated in the Śukla-yajuvṛ Prātiśākhyaṃ (SYP) 4.115 as well, by the exception sva-vargīye cānuttame
**V₂ is a semivowel:** The fundamental rule applies, i.e., $V₁$ is duplicated.

- $\text{āpyāyamān} → \text{āppyāyamān}$ (TS 2.3.5.3)
- $\text{āpākrāmat} → \text{apākkrāmat}$ (TS 2.3.7.1)
- $\text{śuklē} → \text{śukkle}$ (TS 5.3.1.4)
- $\text{āgnīdhram} → \text{āgnīddhram}$ (TS 4.7.8.1)

**V₂ is a sibilant:** It should be noted that only the first mute in each series occurs before a sibilant. As per TYP 1.12,

\[
p \text{ṛksāme} → \text{ṛkhśāme} → \text{ṛkkhsāme}\quad (TS\ 6.1.3.1)
\]
\[
p \text{ḥṛṣtu} → \text{ḥṛthṣu} → \text{ḥṛṭthṣu}\quad (TS\ 1.2.8.1)
\]
\[
p \text{dipsanta} → \text{diphsanta} → \text{dipphsanta}\quad (TS\ 1.2.14.5)
\]
\[
p \text{tat śoḍaśī} → \text{tathśoḍaśī} → \text{tatthśoḍaśī}\quad (TS\ 6.6.11.1)
\]

After this, the normal duplication rule applies. Some examples will clarify this situation:

We now briefly examine the phonetic basis of duplication about which (Whitney 1871) raised questions, which was quoted in the previous section. TYP 2.32 and 2.33 are key śūtra-s in this analysis:

\[
y\text{adupasaṃharati tatkāraṇam}\quad |\quad 2.32
\]
\[
\text{anyeśāṁ tu yatra sparśanaṁ tatsthānam}\quad |\quad 2.33
\]

What comes close (the tip of the tongue), that (is the) cause (of the vowels).\(^{11}\)

The place of the production of others is where contact (with the tongue occurs).

\(^{11}\)The fact that the vowels are referred to in this śūtra, is inferred from the previous śūtra-s.
The key issue is clear articulation and even flow, especially in Vedic recitation. Take the simple case of a word like rudra contrasted with the allied verb roditi. The former has a conjunct consonant, while the latter does not. When a consonant is followed by a vowel, the tongue is first “close” to the organ of production, and when the vowel sound is generated, it is not in contact. The movement of tongue will both be fluid and the sounds can be articulated clearly and with an even flow, as long as only consonant-vowel phoneme pairs are repeated, e.g., the word roditi, which has 3 syllables ro, di and ti. Now take the word rudra consisting of two syllables, ru and dra. The first consonant-vowel phoneme pair is ru, which lends to clear articulation and even flow. The conjunct consonant ‘dra’ requires the tip of the tongue to be first placed at the back of the teeth, and the second phoneme ‘r’ requires the tongue to be moved a little back towards the throat and the ‘a’ phoneme requires no contact. This can happen without a pause and in a fluid manner easily if the phoneme ‘d’ is duplicated, i.e., when the tongue makes contact with the back of the teeth.

It should be noted here that there is a strand of thought that when $V_1$ and $V_2$ are both mutes there should be no duplication. This is recorded in the TYP 14.27 sparśa-sparśa-paṇaḥ, which is one of the exceptions to duplication. According to the commentary, this doctrine is not approved by the TYP (Whitney 1871). It is also clear from the commentary on the Vyāsa śikṣā, and looking at the examples of duplication cited in the commentary, that this is not an approved rule by this text (Ācārya Śrīpaṭṭabhīrāmaśāstrī 1976) as well. However, this is indeed a sound doctrine, except in the case when $V_2$ is the fifth mute in a series. Listening to actual recitations, the duplication of d in rudra is extremely clear, whereas the duplication of the mute g in vāgdevi, supposed to be vāggdevi, is not as clear. We may even say that some duplicated syllables are more duplicated than others! The phonetic basis of duplication certainly deserves a full treatment by itself.

$V_1$ is the phoneme ‘y’

$V_1$ being ‘y’ phoneme occurs usually only in a conjunct consonant with itself, e.g., rayyai in the TS. In the TB, there is a single occurrence tānchaṇyvanta (TB 1.5.9.3). Since this is after an anusvāra, it does not undergo duplication. In the TA, there are two instances of the word vāyvaśvā (TA 1.1.2 and TA 1.21.1), which will undergo duplication as per the fundamental rule

---

\(^{12}\)The Vyāsa śikṣā itself omits this doctrine.
vāyvaśvā ↔ vāyyvaśvā

$V_1$ is the phoneme ‘l’

lavakārāpurvasparśaśca pauśkarasādēḥ 14.2
sparśā evaikeśāmācāryāṇām 14.3

(That) the mute preceded by the ‘l’ or ‘v’ phonemes (is duplicated is the opinion) of Pauśkarasādi 14.2
(That) the mute alone (is duplicated is the opinion) of some teachers 14.3

The possibilities for $V_2$ when $V_1 = l$ are as follows:

- $V_2$ is a mute: Only the mute is duplicated, and if the mute is aspirated, then the corresponding unaspirated mute is duplicated.

  yajñena kalpatām ↔ yajñena kalppatām (TS 4.5.7.2)
  cāpagalbhāya ca ↔ cāpagalbbhāya ca (TS 4.5.6.1)

This should be contrasted with the situation in the recitation by the Nambudiris. It was first pointed out by Kunhan Raja that the phoneme ‘t’ is pronounced as the phoneme ‘l’ in certain situations, both in Vedic as well as regular Sanskrit by the Nambudiris (Raja 1937). In the case of vedic recitation, the Nambudiris duplicate the ‘l’ phoneme when it has been substituted for a ‘t’ phoneme, but follow the TYP rules otherwise. For example, vatsa which is pronounced valsa would actually be said vallsa in the case of vedic recitation, while kalpayati would be pronounced as kalppayati.

- $V_2$ is the phoneme ‘y’: The fundamental rule applies, i.e., the phoneme ‘l’ is duplicated, e.g.,

  yā kalyāṇī bahurūpā ↔ yā kallyāṇī bahurūpā (TS 7.1.5.7)

- $V_2$ is the phoneme ‘v’: In sarvasammata śikṣā-44

    lakāraśca vakāraśca saṁyoge svarito yadi |
    saṁyuktau tu tadā jñeyāvaśaṁyuktau tadanyathā ||
    Where (there is) a svarita in the combination (of the) ‘l’ and ‘v’ phonemes,
    There (is) a combination (of the two), in other cases it is known to be separate.
A non-conjunct ‘l’ phoneme is reflected in only four instances of this combination, all with the word bailvah or bilvah, and not elsewhere where this combination occurs, by the TTs. However this distinction is not specified by the sarvasammata śikṣā:

- \texttt{bailvo yūpō} \rightarrow \texttt{bailvo yūpō} (TS 2.1.8.1)
- \texttt{bīlvā udātiṣṭhat} \rightarrow \texttt{bīlvā udātiṣṭhat} (TS 2.1.8.2)
- \texttt{bailvo vā} \rightarrow \texttt{bailvo vā} (TB 3.8.19.1)
- \texttt{ṣaḍbailvā} \rightarrow \texttt{ṣaḍbailvā} (TB 3.8.20.1)
- \texttt{khalvāhuḥ} \rightarrow \texttt{khalvāhuḥ} (TS 2.5.1.6)
- \texttt{khalvāṇḍaramityeva} \rightarrow \texttt{khalvāṇḍaramityeva} (TS 2.5.3.7)

Note that in the last example (TS 2.5.3.7), the non-occurrence of a svarita does not matter and the ‘l’ phoneme does not stand separately. This is correctly reflected in the printed Grantha texts. It is not clear if the author of the savrvasammata śikṣā was from a tradition where in the example TS 2.5.3.7 above, the ‘l’ phoneme would be non-conjunct. In practice, there is also a very slight pause after the non-conjunct ‘l’ phoneme, which is pronounced as the phoneme ‘l’ followed by approximately the last quarter of the r phoneme. In Rā Kṛṣṇamūrti śāstri and Rā Gaṇeśvaradrāvidāḥ (2003a), the authors have introduced a notation to make the non-conjunct nature clear to readers. They print a \( \div \) sign after the phoneme ‘l’ to clarify that it is not just the usual phoneme ‘l’. In their previous book (Rā Kṛṣṇamūrti śāstri and Rā Gaṇeśvaradrāvidāḥ 2003b), they followed the principle of the Grantha texts by printing a free standing consonant, in this case an ‘l’ phoneme, which they note was confusing to people used to Devanāgarī typesetting. Note again that a free standing consonant in the Grantha typeset texts have a context based meaning, and the fact that the phoneme ‘l’ in this case is non-conjunct is clear to the practitioners. I have adopted the clarifying notation introduced in Rā Kṛṣṇamūrti śāstri and Rā Gaṇeśvaradrāvidāḥ (2003a) in this paper.

- \( V_2 \) is a sibilant: The Vyāsa śikṣā 381 says:

\[ \text{svarordhvoṣmaṇī repasya lasyāpi svarabhaktitā} \]

\(^{13}\)It should be note that the TTs pronounce all non-conjunct or final consonants, except the phoneme ‘m’ in this manner in their chantings, be it Vedic or otherwise, unlike native Hindi speakers.
The ‘r’ phoneme or ‘l’ phoneme attain svarabhakti when (occurring) after a sibilant before which a vowel is present

The ‘l’ phoneme before a ‘h’ phoneme is called karvinī, and before the ‘ś’, ‘ṣ’ or ‘s’ phonemes is called the hāritā, as per the Vyāsa śikṣā 385-386. However, in practical chanting the two svarabhakti-s are pronounced in a similar fashion. It may be noted here that these svarabhakti-s are pronounced by the TTs very similar to a ‘l’ phoneme followed by the kūriyalukaram sound (John Lazarus 1878) in the Tamil language. In the Grantha texts these svarabhakti-s are indicated by a freestanding ‘l’ phoneme, and the fact that it is a svarabhakti and not non-conjunct is clear only from the context. Some examples are:

\[
\begin{align*}
\text{sahasravalśāḥ} & \rightarrow \text{sahasravalśāḥ} \quad (\text{TS 1.1.2.1}) \\
\text{malhām} & \rightarrow \text{malhām} \quad (\text{TS 1.8.19.1})
\end{align*}
\]

\(V_1\) is the ‘v’ phoneme

For the ‘v’ phoneme as the first consonant in the sequence, there are two possibilities for \(V_2\):

- \(V_2\) is a mute: In this case \(V_2\) can only be an ‘n’ or \(ṅ\) phoneme.

\[
\begin{align*}
\text{manīśāmośīśthadāvne} & \rightarrow \text{manīśāmośīśthadāvṅne} \quad (\text{TS 1.6.12.3}) \\
\text{dadhikrāvṅo akāriṣam} & \rightarrow \text{dadhikrāvṅo akāriṣam} \quad (1.5.11.4)
\end{align*}
\]

- \(V_2\) is the phoneme ‘y’, ‘r’ or ‘l’: The fundamental rule applies, i.e., the phoneme ‘v’ is duplicated, e.g.,

\[
\begin{align*}
\text{divyā āpo} & \rightarrow \text{divyā āpo} \quad (\text{TS 6.1.2.3}) \\
\text{tīvro raso} & \rightarrow \text{tīvro raso} \quad (\text{TS 5.6.1.3}) \\
\text{prayajuravlināt} & \rightarrow \text{prayajuravlināt} \quad (\text{TS 6.1.2.4})
\end{align*}
\]

\(^{14}\)There is a detailed and good study of the phonetics of svarabhakti-s as described in the śikṣā texts in Mohanty (2015). My main concern is not the phonetics of the svarabhakti-s, rather where it occurs and how it is treated in TT recitation. The svarabhakti of the ‘l’ phoneme is slightly different from the non-conjunct ‘l’ phoneme, but even trained paṇḍita-s sometimes blur the distinction.
$V_1$ is the sibilant ‘ś’, ‘ṣ’ or ‘s’

- $V_2$ is a mute: As per the TYP 14.9, after an unvoiced spirant, the first mute of the series is inserted as abhinidhāna:

\[
\text{aghosādūṣmāṇah paraḥ pratham/o bhinidhāna-sparśaparāttasya sasthānah} | \\
\text{After an unvoiced spirant, a first consonant (of the same series as the) mute following it, (is inserted as) abhini-}
\]

Some illustrative examples are

\[
\text{suṣmeṇod} \mapsto \text{suṣpmeṇod (TS 1.2.8.1)} \\
\text{prasnātīḥ} \mapsto \text{prastnātīḥ (TS 2.6.11.2)} \\
\text{viṣṇo} \mapsto \text{viṣṭṇo (TS 1.1.3.1)} \\
\]

- $V_2$ is a semi-vowel: The fundamental rule applies, i.e., the sibilant is duplicated. Some examples are

\[
\text{ava sya vara} \mapsto \text{avassyā vara (TS 1.2.3.3)} \\
\text{śiśriye} \mapsto \text{śiśśriye (TS 1.5.3.1)} \\
\text{uttamaśloko} \mapsto \text{uttamaśśloko (TS 5.7.4.3)} \\
\text{aśvamedhaḥ} \mapsto \text{aśśvamedhaḥ (TS 5.7.5.3)} \\
\]

$V_1$ is the phoneme ‘r’

The ‘r’ phoneme is not duplicated as per TYP 14.15, avasāne ra-visarjanīya-jihvāmūliyopadhmaṇīyāḥ. The rule of Pauṣkarasādi (TYP 14.2) and some unnamed teachers (TYP 14.3), applicable to the ‘l’ and ‘v’ phonemes are applicable to the ‘r’ phoneme as well, as per TYP 14.4 repḥāt paraṃ ca. However, note that all consonants after the ‘r’ phoneme are duplicated as per TYP 14.4, whereas only the mutes are duplicated after the ‘l’ and ‘v’ phonemes.

- $V_2$ is a mute: Only the mute is duplicated, and if the mute is aspirated, then the corresponding unaspirated mute is duplicated. In the Grantha texts, the duplicated ‘t’ before a ‘th’ phoneme and a ‘d’ before a ‘dh’ phoneme are explicitly printed, as mentioned previously.
sāśirkena ↔ sāśirkkeṇa (TS 1.6.10.4)
ūrdhvā yasyāmatiḥ ↔ ārddhvā yasyāmatiḥ (TS 1.2.6.1)

- $V_2$ is the phoneme ‘y’, ‘l’ or ‘v’: This is similar to the previous case of $V_2$ being a mute. The clarity of the duplicated syllable even by trained pañḍita-s however varies quite a bit. It is quite probable that there was a tradition of svarabhakti of the ‘r’ phoneme which occur before the semi-vowels, which is reflected in this variation among the pañḍita-s. This can be seen from the kāṭhaka section of the Taittirīya brāhmaṇa, where the word sūrya is said to have three syllables in it, and would make sense if the ‘r’ phoneme was pronounced as a svarabhakti, which is considered to be a vowel as per the TYP.\(^{15}\)

\[
\begin{align*}
\text{paryāgata} & \leftrightarrow \text{paryyāgata} \quad (\text{TS 1.6.10.3}) \\
\text{tairloka} & \leftrightarrow \text{tairlloka} \quad (\text{TS 5.2.1.7}) \\
\text{evā no dūre} & \leftrightarrow \text{evā no dūrve} \quad (\text{TS 5.2.8.3})
\end{align*}
\]

- $V_2$ is the phoneme ‘ṣ’, ‘ś’ or ‘s’: The phoneme ‘r’ attains svarabhakti if the series has only two consonants, i.e, the series is of the form $S_1 r [ṣ|ś|s ] S_2$, and this type of svarabhakti is called the harinī. This is from the rule in TYP 14.16 listing exceptions to duplication

\[
\text{ūṣmā svaraparāḥ}
\]

Clearly, if there is a third consonant following the ‘ṣ’, ‘ś’ or ‘s’ phonemes, there is no svarabhakti of the ‘r’ phoneme and the other usual duplication rules take over. Some examples are:

\(^{15}\text{See the Kāṭhaka 1.9 (Rā Kṛṣṇamūrti śāstri and Rā Gaṇeśvaradrāvīdaḥ 2003a) where the eight-syllabled mantra of sūrya is described as }\text{ghṛṇīritṛi }\text{dve }\text{akṣare}^\text{til} \text{ }\text{sūrya }\text{iti }\text{trīmi }\text{ }\text{āditya }\text{iti }\text{trīmi }\text{|. The mantra is thus }\text{ghṛṇīḥ sūrya ādityaḥ. This should be only 7 syllables as per the regular mode of counting syllables. The later Sūryopaniṣad (A. Mahadeva Sastri 1921) describes the eight syllabled mantra by adding the salutation Om before this mantra, i.e., }\text{om }\text{ghṛṇīḥ sūrya ādityaḥ, and thus getting the correct number of syllables. The verses from the Sūryopaniṣad 7 are }\text{omityekākṣaraṃ brahma }\text{ | ghṛṇīriti }\text{dve }\text{akṣare }\text{ | sūrya ityakṣaradvayam }\text{ | āditya iti }\text{triṇyakṣarāṇi }\text{| etasyaiva sūryasyāṣṭāksaro manuḥ }\text{| |. If the }\text{kāṭhaka śākin-s actually pronounced the }'r'\text{ phoneme before a }'v'\text{ phoneme as a svarabhakti, then this feature is not preserved in the recitation of this praśna by the TTs, while some other phonological peculiarities of the kāṭhaka praśna-s are still preserved by the TTs. It seems quite likely that the }'r'\text{ phoneme before the semi-vowels received differing treatment from various Vedic groups, and at least some of them continue to this day, e.g., the Tamil R̥gvedin-s actually duplicate the }'r'\text{ phoneme before the semi-vowels.}
V₂ is the h phoneme: The ‘r’ phoneme again attains svarabhakti un-
der conditions similar to the ‘ś’, ‘ṣ’ or ‘s’ phonemes and this type is
called the kareṇū. Just like the ‘l’ phoneme svarabhakti-s, these two
svarabhakti-s are pronounced in an identical fashion, although they
are classified under two different names.
barhiṣā ↔ bar.za hiṣā (TS 1.6.7.2)

However, there is an important exception to the kareṇū svarabhakti.
If V₁ has the svarita accent and V₂ has the anudatta accent, then the
‘r’ phoneme does not attain svarabhakti, and there is duplication of
the h phoneme. This rule is not found in the TYP or the two main
śikṣā-s, but is followed in the Grantha texts and TT recitation.

etadbārhiṁṛyaśaḥ ↔ etadbārhhṛyaśaḥ (TA 4.5.5)

• Special case of the form S₁ṛṛ: This form is also not explicitly described
in the TYP or the main śikṣā texts. However, the ‘r’ phoneme stays
non-conjunct, and is pronounced very similar to the svarabhakti.¹⁶

V₁ is the phoneme ‘h’
The ‘h’ phoneme is not excluded from duplication in the TYP and TT
recitations, although Pāṇini and the SYP exempt it.¹⁷

• V₂ is a mute: In this case V₂ has to be the ‘n’, ‘ṅ’, or the ‘m’ phoneme.
However, there is a peculiarity in the TYP when the ‘n’, ‘ṅ’, or the ‘m’
phonemes occur after the ‘h’ phoneme. A so called nāsikya is inserted
after the ‘h’ phoneme

¹⁶The pronunciation of the ‘r’ phoneme in this situation should be contrasted with that
of the Tamil Rg-vedin-s, who do not pronounce it with a svarabhakti like sound.
¹⁷For example, the SYP 4.100, enjoins duplication of any consonant following the ‘r’
and ‘h’ phonemes, prəṁ tu rephahakārabhyām.
hākārānnananamanamaṁparāṁnāsīkyaṁ 21.14
From a ‘ḥ’ phoneme (with the phonemes) ‘ṅ’, ‘ṅ’ or ‘m’ following (it), a nasal-sound (occurs)

Whitney (Whitney 1871) translates it as “After ḍh, when followed by ṇ, ṇ or ṁ, is inserted a nāsīkya”, which is quite reasonable. However, as he points out the commentator actually interprets this statement as ‘ḥ’ phoneme itself taking up a nasal sound. Whitney points out that this is not a straightforward interpretation of this sūtra and his contention seems correct, when the previous sūtra-s and the commentators explanation of those are examined. The pronunciation of the nasals after the ‘ḥ’ phoneme seems to have a variety of opinions, but the TT pronounce these conjunct consonants with the nasal before the ‘ḥ’ phoneme and transitioning to the ‘ḥ’ phoneme towards the tail-end of the nasal. This is what the commentator of the TYP seems to have in mind as well.

- $V_2$ is a semi-vowel: The fundamental rule applies, i.e., the phoneme ‘ḥ’ is duplicated.

\[
\begin{align*}
grḥyate & \rightarrow \text{grḥhyate (TS 6.5.10.1)} \\
prrahriyate & \rightarrow \text{prahhriyate (TS 7.5.15.1)} \\
śītikāvati hlāduke & \rightarrow \text{śītikāvatihhlāduke (TA 6.4.1)} \\
bahvībhīḥ & \rightarrow \text{bahhvībhīḥ (TS 6.5.9.2)}
\end{align*}
\]

**Anusvāra**

The conversion of the anusvāra at the end of a word is generally the same as in Pāṇini, where the anusvāra is usually transformed into a nasal of the same group (savaṁa-anunāsika). The difference in TT recitation is in how it is treated before the sibilants, the ‘r’ phoneme and some notable exemptions. The Grantha texts explicitly specify the transformation of the anusvāra unlike the Devanāgarī texts which simply use the anusvāra symbol (a dot above the line) before the mutes and leave it to the reader to make the conversion, which is not trivial during recitation. In fact, one of the sure ways of identifying a low quality recitation is to see if the reciter substitutes the ‘m’ phoneme for the anusvāra. Since this is a phonologically important

\[18\text{The commentary says: tasmāṁ naṇama-paraṁ hakāraṁ āruhya nāsikyaṁ bhavati, Thus, ḍh when followed by ṇ, ṇ or ṁ becomes an inserted nāsikya.}\]
distinction, I point out the rules governing the anusvāra, although it is exempt from duplication, which is the main topic of this paper.

- $V_1$ is the ‘ṛ’, ‘ś’, ‘ṣ’, ‘s’ or ‘h’ phoneme: The anusvāra remains and does not become a nasal as per TYP 5.29 na repa paraḥ, the ‘ṛ’ phoneme being specified explicitly since the anusvāra is transformed in the case of the other semi-vowels. In these cases where the anusvāra does not become a savarṇa-anunāsika, the sarvasammata śiksā-43 specifies:

\[
\text{adhyāye taittirīyāṇāmanuvāro yadā bhavet} | \\
tadādyardho gakāraḥ syādāparastvanunāsikāḥ ||
\]

In the recitation of Taittirīyaka-s, when the anusvāra is present,
Then, a half gakāra sound followed by the anunāsika is (chanted).

This should be contrasted with the recitation of the Tamil Ṛg-vedins.

- $V_1$ is followed by a vowel, then the sound is like “g-m”, where the “g” has a svarabhakti like quality to it. Note that this will always be the case with the ‘ṛ’ phoneme. The following two cases apply only to the sibilants.

- If $V_1$ is followed by a consonant and $S_1$ was long, the the sound is like “g”, an ardha-gakāra.

\[\text{The Ṛk-prātiśākhya actually requires doubling of the sibilants after the anusvāra.} \]
– If $V_1$ is followed by a consonant and $S_1$ was short, the the sound is like “g-g”, i.e., a pure phoneme ‘g’ sound followed by an ardha-gakāra.

In the Grantha texts, the first two cases are denoted by the vedic anusvāra symbol, whereas the last case is the duplicated vedic anusvāra. Some examples are:

- ādityānāṁ sadasi $\rightarrow$ ādityānāṁsadasi (TS 1.1.11.2)
- iṣāṁrayīṁām $\rightarrow$ iṣāṁrayīṁām (TS 1.1.14.1)
- aśravaṁhi $\rightarrow$ aśravaṁhi (TA 1.1.14.1)
- vayaṁsyāma $\rightarrow$ vayaṁsyāma (TB 2.11.4.28)
- indraṁsthaviram $\rightarrow$ indraṁsthaviram (TB 2.4.2.20)

- $V_1$ is a mute or the phonemes ‘y’, ‘l’ or ‘v’: The TYP rules governing the behavior are similar to Pāṇini and as follows

  nakāro anunāsikam TYP 5.26
  makāra sparsāparastasya sasthānamanunāsikam TYP 5.27
  antasthāparaśca savarṇamanunāsikam TYP 5.28

Examples are not provided in this case since these rules are well known even in the classical Sanskrit.

- Special cases: In the Vyāsa śikṣā

  jñāghnottarō makāraścedanusvāro ‘tra kevalah  166
  dvimātra iti vijñeyo hyanyadharmavivarjitaḥ  167

  The ‘m’ phoneme before a jña or ghna (exists) as just an anusvāra,
  This is known to be two mātra-s (in length), and indeed obtains a different quality.

This duration (mātra) lengthening is reflected in actual TT practice and a distinct pause occurs after the anusvāra. The duration of the pause is somewhat variable. Distinct pauses can be noted in the chanting of such anusvāra-s in (Sarma et al. 2004) and the paṇḍita-s from the Sringeri vedapāṭhaśāla, but in my field study I have observed some paṇḍita-s do not pause very clearly, and the pause is clear only to the trained ear. The sarvasammata śiksā-32 elaborates further:
nakīṣṭam ghnanti saṃjñānam priyaṁ jñātim tathaiva ca |
dhūṃkṣṇā daṃkṣṇava ityatānuśvāro’pi vidharmakaḥ ||

In the places with nakīṣṭam ghnanti, saṃjñānam, priyaṁ jñātim,
dhūṃkṣṇā and daṃkṣṇava, the anusvāra (is pronounced with a) different quality.

Note that this vidharma quality is observed whenever the combination mīṇa occurs, as specified by the vyāsa śikṣā, e.g., saṃjñāpayantyaindraḥ in TS 6.3.11.2, which is not covered by the verse quoted above. However the other cases, dhūṃkṣṇā and daṃkṣṇava, quoted above are pronounced the same way, i.e., with the lengthening of the duration. Interestingly, it is not an anusvāra, but rather the consonant ŋ, which has the extended mātra. This phonological flourish can be clearly heard in authentic TT recitations. Thus the actual TT practice is a combination of the specification of the two śikṣā-s.

Phonological peculiarities with $V_1$ being the ‘n’ phoneme

Some phonological peculiarities of the ‘n’ phoneme in TT recitation are known only from tradition and are reflected in the Grantha texts.

- The first is when the ‘n’ phoneme occurs in the conjunct consonant nts, which would become nths as per TYP 14.12. The ‘n’ phoneme obtains vidharmam, just like the cases described previously. In practice, a pause occurs after the ‘n’ phoneme.20 The Grantha texts print out a standalone ‘n’ phoneme in this case.

- The second peculiarity is when the conjunct consonant npr occurs. In this case, the Grantha texts print the ‘n’ phoneme as a standalone consonant, but it neither gets vidharmam nor is pronounced like an ‘n’ phoneme at the end of a pada and which is not duplicated (described in the next section). It reflects the fact that the ‘n’ phoneme is just not duplicated, although the basic TYP rule would enjoin it.

---

20 This pause is also reflected in the ‘n’ phoneme obtaining the svarita accent if the previous vowel had the svarita accent.
Thus a standalone ‘n’ phoneme is pronounced in different ways, depending on the context where it occurs. These two rules are not reflected in Rā Kṛṣṇamūrti śāstri and Rā Gaṇeśvaradrāviḍāḥ (2003a).

**V₁ is nakāra or ŋakāra at the end of a pada**

This is a distinct feature of the TT recitation, where in some cases the ‘n’ phoneme or the ŋakāra are not duplicated when occurring at the end of a word and followed by a consonant in the next word. Note that the TYP does not have any explicit rules for this and some rules can be found in the śikṣā-s. This case has been explained very well along with examples in the introduction to Rā Kṛṣṇamūrti śāstri and Rā Gaṇeśvaradrāviḍāḥ (2003a). Thus, the examples will not be repeated in this paper. However, I will summarize the account from the śikṣā-s and what happens in actual TT recitations. The sarvasammata śikṣā-45 says:

```
padāntasya nakārasya yavaheṣu paresu vai |
nakārayavahāsatra tvasmaṃyuktāḥ prakīrtitāḥ ||
The ‘n’ phoneme at the end of a word, followed (by) ya, va or ha (in the next word),
There, the ‘n’ phoneme and the following ya, va or ha are well known as not conjoined.
```

The Vyāsa śikṣā on the other hand says:

```
yavahe parastheṣu nakārascāntagastviti | 364
nasyāntagasya dirghāttu yavahe he ca halpare | 365
parairebhirhi tasyaiva na syāt saṃyuktatā tathā | 366
The ‘n’ phoneme at the end (of a word) which comes prior to a yakāra, vakāra or hakāra (in the next word is not duplicated).
The ‘n’ phoneme at the end of (a word) which occurs after a long (vowel) and followed by a yakāra, vakāra or hakāra, or a ‘h’ phoneme followed by a consonant (The ‘n’ phoneme) followed by these exists without (becoming a) conjunct (consonant).
```

Clearly there is a difference in the account between the two śikṣā-s, since the Vyāsa śikṣā restricts the non-duplication to a word-end ‘n’ phoneme which is preceded by a long vowel. However, neither reflect the true
state of affairs in actual TT recitation, which is much more complicated and includes the īnakāra as well. It is actually summarized in the sarvalaksanañamañjarisaṅgrahah quoted under the above sūtra-s of the Vyāsa śikṣā (Ācārya Śrīpaṭṭābhirāmaśāstri 1976),

prāpnutoḥtau īnanau dvitvaṁ vya-ṝ-hṛ-va-ra-parau ca re hrasvānno vaparo dvitvaṁ sarvatra yottarastuṁ naḥ ||

The īa and na at the end of a word are duplicated when followed by vya, ṛ, hṛ, vra or the ‘r’ phoneme (in the next word)
The ‘n’ phoneme preceeded by a short vowel and followed by a ‘v’ phoneme is duplicated, and the īnakāra is always duplicated when followed by a ‘y’ phoneme.

The non-duplicated consonant is pronounced almost like a svarabhakti, but not quite the same, and the TTs pause very slightly after the ‘n’ or the ‘ṅ’ phonemes, although this may not be discernible to the untrained ear. In the Grantha texts, this non-conjunction is again denoted by having the phonemes ‘n’ phoneme or ‘ṅ’ as a standalone consonant. (Rā Kṛṣṇamūrti śāstri and Rā Gaṇeśvaradrāvidāḥ 2003a) again use the ñ symbol to delineate this behavior.

Rules for the visarga

The visarga changes into the corresponding sibilants in general, except before the conjunct consonant
c21 kṣa. This means that before the ka and pa series, the visarga will change into the jihvāmūliya and the upadhmānīya respectively. The key rule in duplication is TYP 14.9, which was quoted previously. After the unvoiced sibilants, namely the jihvāmūliya and the upadhmānīya, the first mute of the series, i.e., k and p, are inserted as abhinidhāna. Note that the jihvāmūliya and the upadhmānīya themselves are exempted from duplication as per TYP 14.15. A point to note here about the Grantha texts is that they convert the visarga to the corresponding sibilant if followed by śa, ṣa or sa. Otherwise, the use the visarga symbol, including before the conjunct consonant kṣa. It is up to the reader to map it into the jihvāmūliya, upadhmānīya, or an actual visarga. Some examples are:

21See TYP na kṣaparaḥ. Note that the kṣa itself would be converted to khṣa as per TYP 14.12.
2.1 A note on insertion of consonants

There are several cases where consonants are inserted, even in the absence of conjunct consonants, e.g., as described in TYP 14.8. Since these happen in the absence of conjunct consonants, most texts including those in the Devanāgarī script, specify this explicitly. Thus, these cases are not treated in this paper. Another instance of insertion is the so-called yama or twin, described in the TYP 21.12-13, where a corresponding twin is inserted when a non-nasal mute is followed by a nasal mute. Strictly speaking, this describes the phonetic phenomenon when a non-nasal sound transition to a nasal sound, and can be excluded from the category of consonant duplication.

3 Algebraic Formulation

It is assumed for the sake of textual processing that white space exists between two consonants only in the case of a word-end ‘ṅ’ or ‘n’ phoneme. The reason is that only these cases have an effect on pronunciation in certain cases. In practice, useless white space can be erased easily in software developed in high-level languages such as Perl very easily. Note that there can be a white space between a vowel and a consonant, namely splitting across different words when there is no conjunction, to make the text more readable. The following notation is used to write concise equations:

- $[\ ]$ denotes a white space
- $[x_1 | x_2 | \cdots | x_k]$ represents $k$ possible choices. If there are $k$ possible choices in the LHS of the equation, there will be $k$ possible choices in the RHS of the equation and the corresponding choices will be at the same position (index).
- The $\div$ sign will be used to represent non-conjunctivity.
- The $\{r\}$ or $\{l\}$ notation are used to represent the svarabhakti-s.
• A standalone ‘n’ phoneme stands for the special case of it before the conjunct consonants beginning with pr.

• The * symbol is used after the vyañjana to indicate the vidharmatva.

• Wikners transliteration for the anudāṭṭa and svarita are adopted.\textsuperscript{22}

A mathematical representation of the rules, which affect a nakāra or nakāra before a white space, i.e., happening at the end of a pada, is as follows:

\[
S_{n[\ ]} = \sim l \quad (2)
\]
\[
S_{[n\ddot{n}[\ ]}y^{va\ddot{v}r\ddot{h}r\ddot{vra|r}] = S_{[n\ddot{n}[\ ]y^{va\ddot{v}r\ddot{h}r\ddot{vra|r}] \quad (3)
\]
\[
S_{h\ddot{n}[\ ]v = S_{h\ddot{n}v} \quad (4)
\]
\[
S_{d\ddot{n}[\ ]v = S_{d\ddot{n}v} \quad (5)
\]
\[
S_{n[\ ]} v = S_{n\ddot{v}} \quad (6)
\]
\[
S_{\ddot{n}[\ ]y = S_{\ddot{n}\ddot{y}} \quad (7)
\]
\[
S_{\ddot{n}[\ ]v = S_{\ddot{n}\ddot{v}} \quad (8)
\]
\[
S_{h\ddot{n}[\ ]h = S_{h\ddot{n}\ddot{h}} \quad (9)
\]
\[
S_{d\ddot{n}[\ ]h = S_{d\ddot{n}\ddot{h}} \quad (10)
\]

\textsuperscript{22}See https://ctan.org/tex-archive/language/sanskrit.
The rest of the duplication is as follows:

\[ S_1 C_1^{i_1,j_1} V_0 \cdots V_l S_2 = S_1 C_1^{i_1,j_1} V_0 \cdots V_l S_2, \]

if \( V_0 = C_1^{i_1,j_2} \) and \( j_2 \neq 5 \) or \( V_0 = V_1 = C_1^{i_1,5} \)

\[ = S_1 C_1^{i_1,j_1-1+j_1 \mod 2} C_1^{i_1,j_1} V_0 \cdots V_l S_2, \]

otherwise \( \text{(11)} \)

\[ S_1 [r|l|v] C_1^{i_1,j_1} V_0 \cdots V_l S_2 = S_1 [r|l|v] C_1^{i_1,j_1} V_0 \cdots V_l S_2, \]

if \( V_0 = C_1^{i_1,j_2} \) and \( j_2 \neq 5 \) or \( V_0 = V_1 = C_1^{i_1,5} \)

\[ = S_1 [r|l|v] C_1^{i_1,j_1-1+j_1 \mod 2} C_1^{i_1,j_1} V_0 \cdots V_l S_2, \]

otherwise \( \text{(12)} \)

\[ S_1 [r|l] U_a S_2 = S_1 [\{r\} \mid \{l\}] U_a S_2 \text{ (13)} \]

\[ S_1 r^r = S_1 r \div r \text{ (14)} \]

\[ S_1 ! r h - S_2 = S_1 ! r h h - S_2 \text{ (15)} \]

\[ S_1 r A V_0 V_1 \cdots V_k S_2 = S_1 r A V_0 V_1 \cdots V_k S_2, \text{ if } V_0 = A \]

\[ = S_1 r A A V_0 V_1 \cdots V_k S_2, \text{ otherwise } \text{ (16)} \]

\[ S_1 r U C_1^{i_1,j_1} V_0 \cdots V_l S_2 = S_1 r U C_1^{i_1,j_1} V_0 \cdots V_l S_2 \text{ (17)} \]

\[ S_1 r U A V_0 \cdots V_l S_2 = S_1 r U U A V_0 \cdots V_l S_2 \text{ (18)} \]

\[ \text{bailv}[a|o] = \text{bailv} \div v[a|o] \text{ (19)} \]

\[ S_1 m [j\bar{n}|\bar{gh}na] = S_1 m \ast [j\bar{n}|\bar{gh}na] \text{ (20)} \]

\[ [d\bar{a}|d\bar{h}u] \mkshna = [d\bar{a}|d\bar{h}u] \ddot{n} \ast \shn \text{ (21)} \]
Figure 1

ND-FST for processing $V_1 = \text{mute}$
4 Non-deterministic Finite State Transducer Representation

A Non-Deterministic Finite State Transducer (FST) is a 7-tuple (Sipser 2006) \((Q, \Sigma, \Gamma, \delta, \omega, q_0, F)\)

1. \(Q\) is a finite set called the states
2. \(\Sigma\) is a finite set called the alphabet
3. \(\Gamma\) is a finite set called the output alphabet
4. \(\delta\) is the transition function
5. \(\omega\) is the output function
6. \(q_0\) is the start state
7. \(F\) is the set of accept states

The processing is done in units of one sentence. The start state is the beginning of new sentence. The accept state is when the virāma symbol (|), i.e., the end of a sentence, is detected. The symbol \(\phi\) stands for a white-space between two words. Each sentence can be processed multiple times to apply all the rules. Of course, in actual software, this could be parallelized for faster processing. Two ND-FST processing engines are illustrated. Figure 1 shows the processing cycle when \(V_1\) is a mute. The whitespace between words are denoted by the symbol \(\phi\). Note that some “super-states” are used (denoted by gray filling) in order to simplify the diagram. Each super-state will have to be broken into 5 different normal states for the 5 different series in the mutes. Figure 2 shows the processing for a word-end ‘ṅ’ or ‘ṅ’ phoneme. Clearly, ND-FSTs can be developed for all the other rule-groups as well.
Figure 2
ND-FST for processing the ‘n’ phoneme at the end of a pada
5 Software for Studying Duplication

A software using the Perl programming language has been developed to study duplication and will be placed in the public domain. This software accepts an input file which contains Yajurveda sentences in the Wikner transliteration format. Rules such as svarabhakti, etc., are implemented and an output file with text in Wikner’s transliteration format, using principles similar to the Grantha texts, is produced. As an option, all duplicated consonants can also be explicitly specified. This can be included in an skt file, which serves as the input to Wikner’s pre-processor (which was written in the C programming language). Note that the Wikner’s C code was modified slightly to accommodate some TT accentuations and syllables, which are not available from the regular pre-processor. Finally, xelatex can be used to generate pdfs from the tex output of the modified Wikner pre-processor.

6 Some Examples of Duplication

Examples of typesetting are given below with regular Devanāgarī typesetting first, Grantha-mode typesetting next, and finally the Grantha-mode with all duplicated syllables. This will clearly illustrate the differences between the Grantha mode and traditional Devanāgarī typesetting, as well as how consonants are duplicated. It should be noted that these do not cover all the TYP duplication rules.

1. Example 1, TA 3.12.33.

   (a) sa̯hasraśi̯raḥ puruṣaḥ | sa̯hasraśkaḥ sa̯hasrapaṭ | sa bhūmiṁ viśvato vṛtvā | atyātiṣṭhaddaśāṅgulam | puruṣa evedaṁ sarvām | yadbhūtaṁ yacca bhavyāṁ | utāṁśatattvasyeśāṁ | yadannēna tirohāti | etāvāṇasya mahīmā | atoṣṭi-yāṇiśca pūrīṣaḥ |

   (b) sa̯hasraśi̯raḥ-sā puruṣaḥ | sa̯hasraśkaḥ-sa̯hasraḥpaṭ | sa bhūmiṁviśvato vṛtvā | atyātiṣṭhaddaśāṅgulam | puruṣa evedaṁ sarvām | yadbhūtaḥyacco bhavyāṁ | utāṁśatattvasyeśāṁ | yadannēna tirohāti | etāvāṇasya mahīmā | atoṣṭi-yāṇiśca pūrīṣaḥ |

   (c) sa̯һasraśi̯rḥ-sā puruṣaḥ | sa̯hasraśkhaḥsa̯hasraḥraḥ | sa bhūmiṁviśvato vṛtvā | atyātiṣṭhaddaśāṅgulam | puruṣa
2. Example 2, TB 1.1.1.

(a) brahma saṃdhāttāṃ tanmē jinvatam | kṣatraṇi saṃdhāttāṃ tanmē jinvatam | iṣāṇi saṃdhāttāṃ tāṃ mē jinvatam | urjaṇi saṃdhāttāṃ tāṃ mē jinvatam | rayiṇi saṃdhāttāṃ tāṃ mē jinvatam | puṣṭiṇi saṃdhāttāṃ tāṃ mē jinvatam | prajāṇi saṃdhāttāṃ tāṃ mē jinvatam | paśūntsaṃdhāttāṃ tāṃmē jinvatam | stūtośi janādhāḥ | devāstvā sukrāpāḥ pranāyantu ||

(b) brahma saṃdhāttantanmē jinvatam | kṣatraṇi sandhāttantanmē jinvatam | iṣāṇi sandhāttantām mē jinvatam | urjaṇi sandhāttantām mē jinvatam | rayiṇi sandhāttantām mē jinvatam | puṣṭiṇi sandhāttantām mē jinvatam | prajāṇi sandhāttantām mē jinvatam | paśūntthsandhāttantānṃē jinvatam | stūtośi janādhāḥ | devāstvā sukrāpāḥ pranāyantu ||

(c) brahma sandhāttantanmē jinvatam | kṣatraṇi sandhāttantaṇmē jinvatam | iṣāṇi sandhāttantaṇām mē jinvatam | urjaṇi sandhāttantaṇām mē jinvatam | rayiṇi sandhāttantaṇām mē jinvatam | puṣṭiṇi sandhāttantaṇām mē jinvatam | prajāṇi sandhāttantaṇām mē jinvatam | paśūntthsandhāttantaṇānnmē jinvatam | stūtośi janādhāḥ | devāsttva sukkrāpāḥppranāyantu ||

3. Example 3, TA 1.22.86.1.

(a) puṣkaraṇāhī puṣkaradaṇḍāhī puṣkaraścā sāṁsthīrya |

(b) puṣkaraṇāhī puṣkaradaṇḍāhī puṣkaraścā sāṁsthīrya |

(c) puṣkaraṇāhī puṣkaraṇāhī puṣkaraścā sāṁsthīrya |

7 Conclusion

Duplication of consonants is uniquely complex and creates many interesting phonological flourishes in TT recitation. The duplication largely follows the TYP and the two main śīkṣā -s, though not completely. The instances where
the śikṣā -s add additional rules and where the two main śikṣā -s disagree with each other have been pointed out. All aspects of duplication and the exceptions can be appreciated only by a trained ear. Duplication can be expressed by algebraic equations as well as by an ND-FST. A Perl script has been written to output the duplicated consonants in an exact manner. The phonetic basis of duplication and comparisons between different Taittirīya traditions would be interesting future projects.

Acknowledgments

I would like to first thank my father for introducing me to the fascinating subject of vedic chanting at a very early age. I would like to thank all my teachers, and in particular, the following teachers have offered invaluable insights on chanting: Śrī Nārāyaṇa Śāstriṇāḥ of Mylapore, Chennai, Śrī Śrīkaṇṭha Ācāryāḥ of Los Angeles, California, Śrī Yajñēśvara Śāstriṇāḥ of Chicago, Illinois, and Śrī Satyanārāyaṇa Bhaṭṭāḥ of Andover, Massachusetts. I would like to thank the reviewers whose comments greatly helped improve the clarity of the presentation.
References


— 1931. Taitirīyayajurbrāhmaṇam dvitiyabhāgaṁ, sasvaram. Śāradavilāsamudrākṣaraśālā, Kumbha -ghoṇam.


— Undated(a). Facsimile copy of Kṛṣṇa yajurvediāya taitirīya-saṃhitā dvitiyabhāgaṁ, sasvaram. Śāradavilāsamudrākṣaraśālā, Kumbha -ghoṇam.

— Undated(b). Facsimile copy of Kṛṣṇa yajurvediāya taitirīya-saṃhitā prathamabhāgaṁ, sasvaram. Śāradavilāsamudrākṣaraśālā, Kumbha -ghoṇam.


Vaidyanāthasāstri. 1905. Taittirriya Āranyakam, kāṭhakabhāgasahitaṁ drāviḍapāṭhakramayutañca. Śāradavilāsamuraṅṣaraśāla, Kumbha - ghoṇam.

Vasu, Srisa Candra. 1898. The Aṣṭādhyāyī of Pāṇini interpreted according to The Kāśikāvrtti of Jayāditya and Vāmana and translated into English, Book VIII. Sindhu Charan Bose.


Word complementation in Classical Sanskrit

Brendan S. Gillon

Abstract: Classical Sanskrit has very flexible word order. This presents a challenge to the application to Classical Sanskrit of categorial grammars and their type logical extensions, which generally assume a fixed total order on the words of the language. The paper outlines the facts pertaining to complementation in Classical Sanskrit and proposes a form of the cancellation rule which accommodates Classical Sanskrit’s free word order of words and their complements.

1 Introduction

Syntacticians generally distinguish between complements and modifiers. Generative syntacticians, wittingly or unwittingly, use some form of a categorial grammar to handle complementation. This approach works well enough for those fragments of a language where complement order is rigid, but it does not handle in a satisfactory way derogations from rigid word order. Moreover, for languages such as Classical Sanskrit, where complement word order appears to be completely free, off the shelf categorial grammars are utterly unsatisfactory. However, it is possible to alter the standard version of a categorial grammar to accommodate in a deft fashion the free ordering of a word’s complements, as found, for example, in Classical Sanskrit. The basic idea is to take advantage of the mathematically well-known equivalence between sequences of length $n$ on a set and functions from the set of positive integers up to and including $n$ into the set.

Like other Indo-European languages, Classical Sanskrit distinguishes between nouns, verbs, adjectives and prepositions. And like other languages, its words from each of these categories have complements, some obligatory, some optional and some even excluded. In what follows, I shall assume that the reader knows these distinctions and how they apply. I shall also assume that words are assigned lexical categories which are ordered pairs.
The first coordinate is its part of speech and its second is a complement list. For example, the English verb to greet is assigned the category \( \langle V, \langle \text{NP} \rangle \rangle \), where \( V \) indicates that the word is a verb and where \( \langle \text{NP} \rangle \) indicates that it requires a noun phrase complement. The English verb to introduce is assigned the category \( \langle V, \langle \text{NP}, \text{PP} \rangle \rangle \), where the complement list shows that the verb takes two complements, a noun phrase complement, followed by a prepositional phrase complement. Intransitive verbs are assigned the category \( \langle V, \langle \rangle \rangle \). In other words, intransitive verbs have empty complement lists. To enhance readability, I shall write the labels for these categories with the part of speech label to the left of a colon and the complement list to the right. Here are the categories of the three verbs just mentioned in this modified notation: \( V: \langle \text{NP} \rangle \), \( V: \langle \text{NP}, \text{PP} \rangle \) and \( V: \langle \rangle \).

The remainder of the paper proceeds in two steps. The first step is to set out the data pertaining to complementation in Classical Sanskrit. The Classical Sanskrit data are drawn from Apte’s *A Practical Sanskrit-English Dictionary* (Apte 1957), Monier-Williams’s *A Sanskrit English Dictionary* (Monier-Williams 1899) and Apte’s *Student guide to Sanskrit composition* (Apte 1885). To help fix ideas, I shall provide examples from English as well. These data are taken from Gillon (2018), which in turn draws on Quirk et al. (1985) and Huddleston (2002). The second step is to set out the proposal. I shall conclude the paper with an overview both of what has been covered and of what has not been.

## 2 Survey of the data

We begin with prepositions since their complements are the simplest to describe. In English, *in* and *into* are both prepositions. They both admit a single complement. The preposition *into* requires a complement, the preposition *in* does not. The latter permits its complement to be omitted. When the complement of a preposition is omitted, the argument corresponding to the omitted complement has its value determined contextually, either through its cotext or its setting, in ways familiar from the ways in which third-person pronouns have their values fixed contextually.

(1) Dan stood in front of the house. When the phone rang,*
     he suddenly ran into.
     he suddenly ran in.
(where the asterisk is the usual sign meaning that the expression to which it is prefixed is judged as odd.)

English, as it happens, also has words, traditionally classified as adverbs, which are often, but not always, compounded from prepositions and which exclude a complement. For example, the English adverb *afterwards*, though it expresses a binary relation, excludes any complement.\(^1\)

(2) Alice lived in Montreal until 2010.
   Afterwards, she moved to Vancouver.
   After that, she moved to Vancouver.

Many Classical Sanskrit prepositions take a complement for which, as is well known, its case has to be specified. For example, the complement of the preposition *adhas* takes a noun phrase whose head is the sixth case, for example, *tarūṇām adhas* (beneath the trees) (Apte 1885, §112). As with English, some Classical Sanskrit prepositions, for example *upari*, have optional complements:

(3.1) *muhūrtād upari* (Monier-Williams (1899) sv *upari*)
      *after a minute*

(3.2) *upari payāḥ pibet* (Monier-Williams (1899) sv *upari*)
      *afterwards he should drink milk*

Next come adjectives. Though not common, some English adjectives require a complement.

(4.1) *Max is averse.*
(4.2) Max is averse to games.
      (Quirk et al. 1985, ch. 16.69)

Others admit optional complements. When omitted, their construal is either contextually fixed, as illustrated by the examples in (5), or is reciprocal, as illustrated by the examples in (6).

(5.1) Bill lives faraway.
     Bill lives faraway from here.

(5.2) Although Bill lives faraway, he visits his parents regularly.
     Although Bill lives faraway from *them*, he visits *his parents* regularly.

\(^1\)Similar examples are found in French and are detailed in Grevisse (1964, §901).
Though cases where an adjective expresses a binary relation but excludes a complement are rare, they do exist. The English adjective *alike*, which excludes any complement, is particularly instructive in this regard, since it has two synonyms, the adjective *similar* whose complement is optional, and the preposition *like*, whose complement is obligatory.

(6.1) Bill and Carol are alike.
(6.2) Bill and Carol are similar (to each other).
(6.3) Bill and Carol are like each other.

Classical Sanskrit adjectives too take complements and the case of their complements has to be specified. For example, the adjective *sukha*, in the sense of good for, takes a fourth case complement. Other adjectives include *dūra* (*distant*), which takes a sixth case complement, *kovida* (*proficient*) which takes a seventh case complement, *sama* (*same*) which takes a third place complement, *prabhu* (*being a match for*) which takes a fourth case complement and *kalpa* (*fit*), *nikaṭa* (*near*) and *samīpa* (*near*), which all take sixth case complements.

English relational nouns are nouns with complements. A few require a complement, such as the word *lack* (Herbst 1988, p. 5) or *sake* (Chris Barker pc). A few exclude complements, such as *stranger* and *foreigner*. Most, however, admit optional complements. This is true, for example, of kinship nouns. When their complements are omitted, the construal of the missing complement is indefinite, or existential, as is illustrated by the equivalence of the sentences in (7).

(7.1) Bill is a father.
(7.2) Bill is a father of someone.

Finally, many nouns resulting from the nominalization of a verb also admit optional complements, even if the complements of the verbs from which they derive are obligatory.

While it is unclear whether or not Classical Sanskrit has relational nouns whose complements are required or excluded, it is obvious that it has relational nouns with optional complements. This is the case for kinship nouns. It is also true for nouns such as *pārśva* (*side*), *paryanta* (*edge*), *vīṣaya* (*object*), *sthāna* (*object*) (Apte 1885, §11b: U4). The default case assignment is the sixth case, but other case assignments are also possible, for example, the seventh case for *kāraṇa* (*cause*) or *spṛha* (*desire*).

We now come to verbs and their complements. To fix ideas, let me summarize the situation with verbs in English. Like other well-studied lan-
guages, English verbs may take none, one, two, or sometimes even three complements. An English verb which admits no complement (e.g., *to bloom*) is known as an intransitive verb. English linguistics has no term for verbs which take exactly one complement. However, traditional English grammar does have a term for verbs which takes an adjective phrase as its sole complement. They are called copular, or linking, verbs. They include verbs such as *to appear*, *to become* and *to sound*. It also has a term for verbs which take a noun phrase as its sole complement. As the reader knows, they are called transitive verbs. But English also has verbs whose sole complements are prepositional phrases such as *to approve of*, *to rely on* and *to wallow in*, among many, many others. In addition, English has verbs whose sole complements are clauses, where clauses include finite interrogative and declarative clauses, interrogative and declarative infinitival phrases, and gerundial phrases. Here are examples of each of these kinds of verbs: *to note*, *to wonder*, *to decide*, *to recall* and *to enjoy*.

English verbs may also take two complements. The best known of such verbs are verbs such as *to give*, one complement of which is known as the direct object and the second as the indirect object. However, other pairs of complements are also common, for example, where one complement is a noun phrase and the other an adjective phrase (e.g., *to consider*) or where one is a noun-phrase or prepositional phrase and the other is a clause (e.g., *to convince* and *to say* respectively). Finally, there are a few verbs which take three complements such as *to bet*, *to trade* and *to transfer*.

Let us now turn to Classical Sanskrit verbs. There has been no study of word complementation in Classical Sanskrit, not even of just verb complementation. Still, enough for my purposes here can be gleaned from the sources mentioned earlier. Classical Sanskrit, like other Indo-European languages has verbs with no complements, one complement, two complements, and three complements. Verbs with just one complement include verbs whose sole complement is a clause, an adjective phrase, and a noun phrase. I do not know whether or not Classical Sanskrit has any verbs whose sole complement is either an adverbial phrase or a prepositional phrase. Verbs with two complements include verbs where one complement is a noun phrase and the other either another noun phrase, an adjective phrase, or a clause, where a clause may be an interrogative or declarative finite clause, a participial phrase or an infinitival phrase. There are also a few verbs with three complements.

I shall illustrate the principal features of verb complements with verbs
taking just one complement. The noun phrase complements of single complement Classical Sanskrit verbs, though typically taking the second case, may, depending on the verb, take any of the seven cases. The verb *ghrā* (*to smell*) takes a second case nominal complement, while *tuṣ* (*to be pleased with*) takes a third case nominal complement, *ruc* (*to please*) takes a fourth case nominal complement, *vīyuj* (*to separate from*) a fifth case, *samjñā* (*to remember with regret*) a sixth case nominal complement and *vyavahṛ* (*to deal with*) a seventh case complement. In addition, copular verbs such as *vṛt* (*to be*) take a first case complement, which may be either a noun or an adjective.

Copular verbs exhibit an important property of many words which take complements: the single complement they admit may be from any of several categories. I shall refer to words which admit alternate complements *polyvalent* complements. The copular verb is the best-known example. In English, the copular verb admits one complement which may be either an adjective phrase, a noun phrase, a prepositional phrase or an adverbial phrase; in Classical Sanskrit, the complement may be a noun phrase in any of the seven cases, an adjective phrase or a prepositional phrase.

Another polyvalent Classical Sanskrit verb, well known to students of Classical Sanskrit grammar, is the verb *div* (*to play*), whose complement is a noun phrase in either the third (instrumental) or second (accusative) case. (Apte 1885, §59 obs.)

(8.1) *akṣair dīvyati.*
He plays dice.

(8.2) *akṣān dīvyati.*
He plays dice.

English also has many, many such verbs. A particularly compelling example is the English verb *to appoint*, which admits alternately a prepositional phrase complement and a noun phrase complement, for this verb is synonymous with another English verb *to choose*, which admits only a prepositional phrase complement.

(9.1) Dan appointed Alice as chief minister.

(9.2) Dan appointed Alice chief minister.

(10.3) Dan chose Alice as chief minister.

(10.4) *Dan chose Alice chief minister.*

Though polyvalency is common with verbs, it is not unique to verbs.
Another feature of verb complements is their optionality. I call words with optional complements polyadic. It is a feature of all word classes in English and it seems to be common in other languages, including both Chinese and Classical Sanskrit. (Details pertaining to polyadic Classical Sanskrit words are given in Gillon (2015).)

3 Proposal

Let me now turn to the proposal. To understand the proposal for Classical Sanskrit complementation, let me sketch out the proposal for complementation in English. The reader familiar with categorial grammar will immediately recognize the similarity between the phrase formation rule given below for English and the cancellation rule of categorial grammar.

Careful study of English complementation (Quirk et al. (1985); Huddleston (2002)) shows that English verbs alone have nearly four dozen kinds of complements. (The details are found in (Gillon 2018, ch. 10). The general schema, to a first order approximation, is this:

(11) **ENGLISH PHRASE FORMATION RULE SCHEMA**

If \( e|X:⟨C_1, \ldots, C_n⟩ \) and \( f_i|C_i \), for each \( i \in \mathbb{Z}_n^+ \), then \( ef_1 \ldots f_n|X:⟨⟩ \)

(where \( 1 \leq i \leq n \)).

In the schema, \( e, f_1, \ldots, f_n \) are expressions. Each expression which is a simple word is assigned a category of the form \( X:⟨C_1, \ldots, C_n⟩ \), where \( X \) is either \( A \) (adjective), \( N \) (noun), \( P \) (preposition) or \( V \) (verb) and \( C_i \) is a complement, which itself may be a phrase, such as a noun phrase, an adjective phrase, a verb phrase, a prepositional phrase, or a clause of some kind. In effect, the rule says that, when an expression is assigned a lexical category which admits \( n \) complements is followed by \( n \) complements of corresponding categories, the resulting expression forms a phrase, that is, it forms a constituent requiring no complements. Obviously, this is an enriched cancellation rule, as familiar from categorial grammar. (See Gillon (2012) for details, as well as Gillon (2018, ch. 10).)

An application of an instance of the cancellation rule is given in the derivation below, where the expression *greeting Bill* is assigned the category VP, which is an abbreviation for \( V:⟨⟩ \).
In the derivation above, the total, or linear, order of the expressions matters, as one expects for English. Also, since we are talking about complementation here and subject noun phrases are not complements, the last step results from an additional rule for clause formation.

While every instance of phrase formation in English is an instance of this schema, not all instances of this schema are instances of phrase formation in English. For example, it is thought that no English word has more than three complements. It, therefore, follows all instances of the schema where \( n \) is greater than or equal to 4 do not form English phrases.

Associated with the syntactic rule in (9) is the following semantic rule.

(12) **ENGLISH PHRASE VALUATION RULE SCHEMA**

Let \( \langle U, i \rangle \) be a structure for an English lexicon.

If \( e|X:⟨C_1, \ldots, C_n⟩ \) and \( f_j|C_j \), for each \( j \in \mathbb{Z}_n^+ \), then

\[
v_i(e f_1 \ldots f_n|X:⟨⟩) = \{x: ⟨x, y_1, \ldots, y_n⟩ \in v_i(e|X:⟨C_1, \ldots, C_n⟩) \text{ and } y_j \in v_i(f_j|C_j), \text{ for each } j \in \mathbb{Z}_n^+\}.
\]

Before seeing how these schemata can be generalized to handle words with alternate complements and optional complements, let us see how these simpler schemata can be applied to Classical Sanskrit. The basic idea is to change the correspondence between a complement list and the complements by having two \( n \)-tuples of the same length to the requirement that there be a bijective function from the complements onto the complement list which preserves various syntactic specifications of the complement expressions. In particular, the bijection must preserve phrasal and clausal categories and, in the case of phrases, preserve the case of a phrase’s head noun.

(13) **SANSKRIT PHRASE FORMATION RULE SCHEMA**

Let \( e|X:⟨C_1, \ldots, C_n⟩ \), let \( f_j|A_j \), for each \( j \in \mathbb{Z}_n^+ \), and let \( g \) be a bijection from \( \{A_i: i \in \mathbb{Z}_n^+\} \) into \( \{C_j: j \in \mathbb{Z}_n^+\} \) such that the phrasal category and case, if the category has case, of \( A \) is identical with that of \( g(A) \). Then, \( e f_1 \ldots f_n|X:⟨⟩ \).

Consider the verb *trai* (*to rescue*), which takes two complements:
(14) bhīmād duḥśāsanaṃ trātum.

to save Duḥśāsana from Bhīma.

(Apte 1885, §78: Ve 3)

Its lexical entry is trai|⟨NP₂, NP₅⟩, where the subscripts indicate the case requirement on the noun phrase complement. Now consider the expressions bhīmād, duḥśāsanaṃ and trātum, under any ordering, form a constituent and there is a bijection from the expressions bhīmād and duḥśāsanaṃ into the complement list ⟨NP₂, NP₅⟩ which preserves the phrasal category and the case associated with the phrasal category.

\[
\begin{array}{ccc}
\text{trātum} & \text{duḥśāsanaṃ} & \text{bhīmād} \\
V:⟨\text{NP₂, NP₅}⟩ & \text{NP₂} & \text{NP₅} \\
\text{VP}
\end{array}
\]

In the derivation tree above, the expressions are not totally ordered with respect to one another. In other words, any permutation of the lexical items results in the same derivational step. This, of course, contrasts with English.

Let us now consider a conservative extension of the categories and the cancellation rule which permits a simple treatment of word polyvalence. The basic idea is to replace each specification of a complement in the complement list with a set of complement categories. In the case where a word has just one complement for a position in its complement list, then its position is filled with a singleton set whose sole member is the relevant category; if the position has more than one complement which can be associated with the position, then it is assigned the set of all the categories associated with it. Cancellation occurs when a complement to the word occurs in the corresponding set in the complement list.

(15) PHRASE FORMATION RULE SCHEMA (first extension)

For each \(j \in \mathbb{Z}_n^+\), let \(C_j\) be a complement category, let \(C_j\) be a non-empty subset of the complement categories and let \(C_j\) be a member of \(C_j\).

If \(e|X:⟨C_1, \ldots, C_n⟩\) and \(f_j|C_j\), for each \(j \in \mathbb{Z}_n^+\), then \(ef_1\ldots f_n|X:⟨⟩\).

For example, the verb to choose takes one complement and its complement is a noun phrase, whereas the verb to appoint takes one complement but its complement may be either a noun phrase or a prepositional phrase (headed by the preposition as). The contrasting complement lists are illustrated below in the contrasting lexical entries for the two verbs.
Here are examples of Classical Sanskrit polyvalent verbs:

(17.1) \textit{div}|V: \{\text{NP}_2, \text{NP}_3\}
(17.2) \textit{namaskṛ}|V: \{\text{NP}_2, \text{NP}_4\}
(17.3) \textit{praṇam}|V: \{\text{NP}_2, \text{NP}_4, \text{NP}_6\}

We must now adjust the semantic rule schema paired with the extended-phrase formation rule schema so that the semantic rule schema accommodates the revisions in the phrase formation rule schema.

(16) PHRASE VALUATION RULE SCHEMA (first extension)

Let \(\langle U, i \rangle\) be a structure for an English lexicon.

For each \(j \in \mathbb{Z}_n^+\), let \(C_j\) be a complement category, let \(C_j\) be a non-empty subset of the complement categories and let \(C_j\) be a member of \(C_j\).

If \(e|X: \langle \mathcal{C}_1, \ldots, \mathcal{C}_n \rangle\) and if \(f_j|C_j\) (for each \(j \in \mathbb{Z}_n^+\)), then
\[
v_i(ef_1 \ldots f_n|X: \langle \rangle) = \{x: \langle x, y_1, \ldots, y_n \rangle \in v_i(e|X: \langle \mathcal{C}_1, \ldots, \mathcal{C}_n \rangle) \text{ and } y_j \in v_i(f_j|C_j), \text{ for each } j \in \mathbb{Z}_n^+\}.
\]

The final complication, occasioned by polyadic words, or words with optional complements, has been addressed for English and it can be addressed for Classical Sanskrit, but it is not possible to set out the details in the brief time allotted for the paper.

4 Conclusion

As should be well known, the treatments in generative linguistics of the formation of a phrase from its head and the head’s complements amounts to some form of the cancellation rule of Categorial Grammar. While this rule works for phrases where the word order of the head word and its complements is total, or linear, it does not apply to languages where the order is not. Classical Sanskrit is such a language. In this paper, I have shown how a simple enrichment of lexical categories and a modest change in the cancellation rule permits languages where the head word and its complements are freely ordered to be brought within the purview of a cancellation rule. In addition, I showed how the enrichment can be conservatively extended to handle complement polyvalence, that is, where expressions of different syn-
tactic categories may appear in the same complement position. Finally, I stated, but did not show, that the conservative extension can be still further extended conservatively to apply to complement polyadicity, that is, where complements to a word are optional.

Three important points were not addressed: how is a clause formed from a subject noun phrase and a verb phrase; how does one distinguish in Classical Sanskrit complements from modifiers; and how does the cancellation rule introduced here affect computational complexity. Let me bring this brief article to a close by saying a few words about each point.

Since the focus on this paper is complementation, the question of the correct rule for clause formation does not arise, aside from providing an example derivation of an entire clause. Of course, in Categorial Grammar and its various type logical enrichments, the usual way to handle the formation of a clause from a subject noun phrase and a verb-phrase is through the cancellation rule, in effect, treating the subject noun phrase as a kind of complement to the verb. But this is not empirically satisfactory, as it collapses the distinction between complements and subjects, something which may in fact be warranted in the case of Classical Sanskrit.

The second point pertains to how to distinguish between modifiers and complements in Classical Sanskrit. There is some debate among linguists as to how to make this distinction. Distinguishing between modifiers and complements is an empirical question and one whose answer will vary from language to language. It is not a question which has been addressed either by scholars using contemporary linguistic ideas or by scholars of the Indian grammatical tradition. The latter fact is not surprising, since complementation is not a notion used in the Indian grammatical tradition. Nonetheless, languages do share properties, and in the absence of empirical work on the subject, I have followed the lead of what are generally regarded as complements in the study of various Indo-European languages, of which Classical Sanskrit is an example.

Finally, I have not addressed the question of the computational tractability of the enriched cancellation rules proposed here. This important question, raised by a reviewer, is not one I am currently in a position to address.²

²In this regard, let me bring to the reader’s attention work by Alexander Dikovsky and collaborators seeks to address the problem which relatively free word order poses for a cancellation rule of the kind found in categorial grammar: Dekhtyar and Dikovsky (2008) and Dekhtyar, Dikovsky, and Karlov (2015). This work was kindly brought to my attention by a reviewer of this paper.
References


TEITagger: Raising the standard for digital texts to facilitate interchange with linguistic software

Peter M. Scharf

Abstract: For several years, members of the International Sanskrit Computational Linguistics Consortium working to facilitate interchange between digital repositories of Sanskrit texts, and digital parsers and syntactic analyzers have recognized the need to standardize reference to particular passages in digital texts. XML has emerged as the most important standard format for document structure and data interchange, and TEI as the most important standard for the XML markup of textual documents. TEI provides methods to precisely describe divisions in texts from major sections to individual morphemes, and to associate various versions with each other. Responsible text archives, such as TITUS and SARIT, have adopted the TEI standard for their texts. After a workshop to train doctoral candidates at the Rashtriya Sanskrit Sansthan to mark-up texts in accordance with TEI in May 2017, the Sanskrit Library developed software to semi-automate the process with extensive use of regular expressions and meter-identification software, and is currently marking-up all of its texts using the TEITagger. The result will be a large repository of digital Sanskrit texts that can furnish text to the Sanskrit Heritage parser and the University of Hyderabad’s parser and syntax analyzer to allow passages parsed and analyzed for dependency structure to be interlinked with their originals.

1 XML and TEI

In the age in which oral productions and hand-written documents were the predominant mode of expressing knowledge and exchanging information, each individual articulation or manuscript had its own format determined by the author and heard or read by other individuals. In the age of the print medium, presses produced multiple copies of individual productions which
could be widely distributed to numerous other individuals. At the outset of the digital age, as Scharf and Hyman (2011, p. 2) and Scharf (2014, p. 16) noted, presentation of individual productions imitated the print medium. Document creators and software engineers created works to present knowledge to human readers. As Goldfarb (1990) noted, unfortunately the tendency persists as “their worst habits” as if their production were meant only for human eyes, and had no need to coordinate with software developed by others. In 1969, however, Goldfarb, Mosher, and Lorie at International Business Machines Corporation (IBM) developed the Generalized Markup Language (GML), so called based on their initials (Goldfarb 1990, p. xiv), to mark up documents in terms of the inherent character of their constituents, such as prose, header, list, table, etc., to enable software to format the documents variously for various devices, such as printers and display screens, by specifying a display profile without changing the document itself (Wikipedia contributors 2017). Over the next decade, Goldfarb and others developed the international Standard Generalized Markup Language (SGML), International Standards Organization (ISO) document 8897, to describe documents according to their structural and other semantic elements without reference to how such elements should be displayed. Thus in contrast to the Hyper-Text Markup Language (HTML) which was designed to specify the display format of a text, SGML separates the inherent structure of a document from how it is presented to human readers and “allows coded text to be reused in ways not anticipated by the coder” (Goldfarb 1990, p. xiii).

The eXtensible Markup Language (XML) is an open-source meta-language consisting of a stripped-down version of SGML formally adopted as a standard by the World Wide Web Consortium (W3C) in February 1998. In the couple of decades since, XML has become the single most important standard format for document structure and data interchange. Wüstner, Buxmann, and Braun (1998) noted, “XML has quickly emerged as an essential building block for new technologies, offering a flexible way to create and share information formats and content across the Internet, the World Wide Web, and other networks.” Benko (2000, p. 5) noted, “XML is expected to become the dominant format for electronic data interchange (EDI).” A few years ago, Zazueta (2014) noted, “XML emerged as a front runner to represent data exchanged via APIs early on;” whereas “Javascript Object Notation (JSON), emerged as a standard for easily exchanging Javascript object data between systems.” He continues,
API designers these days tend to land on one of two formats for exchanging data between their servers and client developers - XML or JSON. Though a number of different formats for data have been designed and promoted over the years, XML’s built in validation properties and JSON’s agility have helped both formats emerge as leaders in the API space.”

Benko (2000, p. 2) also noted that two of the seven benefits the W3C defines for establishing XML include the following:

- Allow industries to define platform-independent protocols for the exchange of data.
- Deliver information to user agents in a form that allows automatic processing after receipt.

As a simple metalanguage consisting of just seven characters (<, >, /, =, " , '), XML allows users to develop markup languages of an unlimited variety. In order to facilitate interchange of textual documents, the Text Encoding Initiative (TEI) developed a community-based standard for the representation and encoding of texts in digital form. The TEI Guidelines for Electronic Text Encoding and Interchange define and document a markup language for representing the structural, renditional, and conceptual features of texts. They focus (though not exclusively) on the encoding of documents in the humanities and social sciences, and in particular on the representation of primary source materials for research and analysis. The Text Encoding Initiative also makes the Guidelines and XML schema that validate them available under an open-source license. TEI has become the most important standard for the XML markup of textual documents. Hence to facilitate the interchange, cross-reference, and unanticipated use of digital Sanskrit text, it is imperative that digital archives of Sanskrit texts make their texts available encoded in XML in accordance with the TEI Guidelines.

2 Sanskrit digital archives and the use of TEI

A number of organizations and individuals, such as GoogleBooks, The Million Books Project, Archive.org, the Digital Library of India, and the Vedic Reserve at Maharishi International University, have made images and PDF documents of Sanskrit printed texts available, and a number of libraries,
such as the University of Pennsylvania in Philadelphia and the Raghunath Temple Sanskrit Manuscript Library in Jammu, have made images of their Sanskrit manuscripts available. Such productions have greatly facilitated access to primary source materials; yet that access is limited exclusively to being read by a human being. Although Jim Funderburk developed software to search headwords in a list and highlight that headword in digital images of dictionary pages, and Scharf and Bunker developed software to approximate the location of passages in digital images of Sanskrit manuscripts, the results of such software are also merely displays for a human reader. PDFs do not facilitate automatic processing after receipt.

Numerous groups and individuals of various backgrounds have created digital editions of Sanskrit texts and made them available on portable digital storage media and the Web. As opposed to image data, these documents consist of machine-readable character data. Most of these are structured in simple data structures, such as lines of text numbered with a composite chapter-section-line number, in text files or directly in HTML files. These documents are intended to permit access by a human to passages by searching as well as for sequential reading. While the various providers of digital text are too numerous to mention, one site has emerged as a central registry. The Göttingen Register of Electronic Texts in Indian Languages (GRETIL) lists about eight hundred such Sanskrit texts. These texts are openly available for download so that others may subject them to various sorts of linguistic processing such as metrical, morphological, and syntactic analysis. As great a service as making these texts available in digital form is, GRETIL exerted minimal discipline on its early contributors so that there is great variability in the specification of metadata. In many cases, the source edition of the text is unknown. In addition, each contributor was free to structure the document as he wished, so there is great variability in the manner of formatting verse and enumerating lines.

Although GRETIL offers the texts in a few common standard encodings including UTF8 Unicode Romanization, there is variability in how the contributors employed capitalization, encoded diphthongs versus contiguous vowel sequences, punctuation, etc. Texts available from other sources use Devanāgarī Unicode, different ASCII meta-encodings, or legacy pre-Unicode fonts. Scharf and Hyman (2011) and Scharf (2014) have already dealt with the issues regarding character encoding. Here I address higher-lever text and document structure encoding.

Even by 2006, at the start of the International digital Sanskrit library
integration project, the Thesaurus Indogermanischer Text- und Sprachmateri
talien (TITUS), which contributed its texts for integration with diction-
naries produced by the Cologne Digital Sanskrit Dictionaries project via morphological analysis software produced by Scharf and Hyman at Brown, had begun partially using TEI tags to mark up the structure of its texts and metadata. Over the past four years, the Search and Retrieval of Indic Texts project (SARIT) marked up all of the texts which had previously been made available in various ad hoc formats at the Indology website, and some twenty additional texts, in a consistent encoding in accordance with the TEI standard. The site (http://sarit.indology.info) currently houses fifty-nine Sanskrit TEI documents made available under a Creative Commons license and provides clear instructions for how to mark up Sanskrit texts in accordance with TEI.

3 TEI training

At the bequest of the SARIT project, in an initial attempt to spur large-
scale encoding of Sanskrit texts in accordance with the TEI standard, I conducted a one-week e-text tutorial at the Rashtriya Sanskrit Sansthan’s Jaipur campus in February 2010. While several participants produced TEI versions of small portions of texts, the workshop failed to instigate the collaboration of technical expertise and abundant Sanskrit-knowing labor that SARIT had hoped. In May 2017, however, I was invited by the Rashtriya Sanskrit Sansthan to conduct a two-week TEI workshop at its Ganga Nath Jha campus in Allahabad. There I trained twenty Sanskrit doctoral candidates in how to encode texts and catalogue manuscripts in accordance with TEI Guidelines. In an additional week I worked with these students to encode twenty Sanskrit works in accordance with TEI, ten of which were delivered complete in the next month.

During the workshop, I trained students to analyze the structure of a plain text data-file with Sanskrit text in numbered lines or verses and to construct regular expressions to recognize strings of text with fixed numbers of syllables. We constructed regular expressions to recognize a few common verse patterns and had the students submit the verses found to the Sanskrit Library’s meter analyzer produced and described by Melnad, Goyal, and Scharf (2015a,b). Once we knew that verses with a certain number of syllables were typically in a certain metrical pattern, we constructed
replacement expressions to transform the recognized pattern to well-formed TEI line group elements (lg) with subordinate line (l) and segment elements (seg) for each verse quarter (pāda) and to insert type, analysis, and metrical pattern attributes (type, ana, met) in the (lg) tag. The replacement expressions inserted the enumeration provided by the source document in (n) and (xml:id) attributes in the (lg) tag, and typed and lettered the verse quarters as well. Where complex numbers compiled the numbers of text divisions, subdivisions, and passages within subdivisions, the regular expression placed just the last in a separate group, and the replacement expression inserted that number in the value of the n attribute while putting the whole number in the value of the xml:id attribute. For example, the regular expression and replacement expression shown in Figure 1 was primarily responsible for transforming the following verse of the Bhagavadgītā (in Sanskrit Library ASCII encoding) to the well-structured TEI (lg) element with its subsidiaries shown in Figure 2:

06024070a ApUryamARam acalapratizWaM; samudram ApaH praviSanti yadvat
06024070c tadvat kAmA yaM praviSanti sarve; sa SAntim Apnoti na kAmakAmI

I say, “primarily responsible,” because in fact the leading zeroes on the number of the verse were captured by this regular expression so that ‘070’ was inserted in the value of the n attribute; an additional regular expression removed them.

Now one will notice that the original text document conveniently indicated the break between the two verse quarters in each line of a Triṣṭubh verse by a semicolon and space. This indication allowed the regular expression to group just the text of each verse quarter without leading or trailing spaces. However, no such indication was given for the break between verse quarters in an Anuṣṭubh verse because there is frequently no word-break at the pāda boundary of the ubiquitous śloka. One would want to preserve the information whether or not there is a word break there, yet would not want a pāda to begin with a space. Hence after a regular expression inserted each verse quarter in a seg element, subsequent regular expressions moved leading spaces, where found, from the beginning of the second seg to the end of the first and set the second verse quarter on a separate line. Thus the first verse of the Bhagavadgītā,
Figure 1

Regular expression and replacement expression to transform a plain text verse in Triṣṭubh meter to TEI
I also trained students in the workshop to compose regular expressions to capture the speaker lines such as *Dhrtrāṣṭra uvāca* that introduce speeches and to compose replacement expressions to put these in speaker elements. Similarly, I taught them to mark up prose sentences and paragraphs in s and p elements, to put speeches in sp elements, to insert head and trailer elements, to locate and capture enumeration of divisions, to insert div elements, to insert the whole in body and text elements, to insert page and line break elements, and to mark up bibliography. I then had them insert these elements in a teiHeader template in the TEI element, and to validate the complete TEI document. Figure 4 shows the first short speech of the Bhagavādgcita with the speaker element in the context of parent sp, div, body, and text opening tags. Let me remark that guidelines for how to mark up Sanskrit text in accordance with TEI are conveniently available on the SARIT website.1

---

1http://sarit.indology.info/exist/apps/sarit-pm/docs/encoding-guidelines-simple.html
Figure 3
*Bhagavadgītā 1.1 in Anuṣṭubh meter*

```xml
<lg type='anuzwuB' n='1' xml:id='m06023001'>
  <l>
    <seg n='a'>Darmakzetre kurukzetre </seg>
    <seg n='b'>samavetA yuyutsavaH .</seg>
  </l>
  <l>
    <seg n='c'>mAmakAH pARqavAS cEva </seg>
    <seg n='d'>kim akurvata saMjaya ..1..</seg>
  </l>
</lg>
```

Figure 4
*TEI markup of a speech in the context of division, body, and text elements*

```xml
<text xml:lang='sa-Latn-x-SLP1'>
  <body>
    <div type='aDyAya' n='1'>
      <sp>
        <speaker n='1s' xml:id='m06023001s'>DftarAzwra uvAca</speaker>
        <lg type='anuzwuB' n='1' xml:id='m06023001'>
          <l>
            <seg n='a'>Darmakzetre kurukzetre </seg>
            <seg n='b'>samavetA yuyutsavaH .</seg>
          </l>
          <l>
            <seg n='c'>mAmakAH pARqavAS cEva </seg>
            <seg n='d'>kim akurvata saMjaya ..1..</seg>
          </l>
        </lg>
      </sp>
    </div>
  </body>
</text>
```
4 TEITagger software

After the experience of teaching Sanskrit students with minimal technical literacy to transform a plain text document to well-structured XML in accordance with TEI in a series of well-ordered steps, it occurred to me that I could also teach a machine to do the same. Ralph Bunker, the technical director of the Sanskrit Library, had previously developed software called Linguistic Mapper at my request so that I could compile a driver file that contained a sequence of regular and replacement expressions that implemented historical sound change rules between a proto-language and a descendant language. We created TEITagger by modifying Linguistic Mapper to process a series of such sets of regular and replacement expressions that matched specified numbers of syllables in certain arrangements that approximated metrical patterns. By creating a regular expression that counted the correct number of syllables per pāda we could convert every such verse to proper TEI markup in \textit{lg} elements, with each line in an \textit{l} element, and each pāda in a \textit{seg} element. At the same time we could number the verse in an \texttt{n} attribute, insert an \texttt{xml:id}, and insert the presumed meter name and metrical pattern in a \texttt{type} attribute. The meter name and metrical pattern in the first version of TEITagger was presumed on the basis of the syllable count, not automatically checked against a pattern of light and heavy syllables.

We then revised TEITagger to include the feature of submitting a segment of text that matched a certain regular expression to our meter identification software that would identify the meter of a whole verse by checking the passage against specified patterns of light and heavy syllables as defined by classical metrical texts. If a match is found TEITagger version 2 automatically inserts the meter name, general type, and metrical pattern in \texttt{type}, \texttt{ana}, and \texttt{met} attributes of the \textit{lg} element. To simplify the regular expression formulation in the command driver file for this program, we composed macros to represent vowels, consonants, syllables, syllable codas, and the typical terms used in the lines that introduce speeches. These macros are shown in Figure 5.

To further simplify testing segments of text for any meter type with any number of syllables, we introduced an iterative loop command and iteration variable in version 3. Thus, for example, with a command that consists of the single regular expression and replacement expression shown in Figure 6, TEITagger can evaluate every segment of text in a file with four verse
quarters each consisting of $n$ syllables per verse quarter, where the variable $n$ is tested in order from 28–1 thereby testing for all of the verses with the same number of syllables per verse quarter. Metrical patterns with the same number of syllables per verse quarter include all 468 of the samavṛttta and upajāti types as well as some of the ardhasamavṛttta and viṣamavṛttta type. Similar expressions can be composed to match verses with unequal numbers of syllables per verse quarter. Such metrical patterns include those of the ardhasamavṛttta type and mātrāvṛttta type as well as irregular variations of more regular patterns. The current version (17) also passes verse lines and individual pādas to the meter analyzer to detect their patterns in irregular verses.

**Figure 5**

*TEITagger macros*
**Figure 6**

TEITagger iterative command to match verses with four pādas with \( n \) syllables per pāda, where an arbitrary range can be specified for \( n \).
The TEITagger driver file also accepts commands to insert header and footer files so that one can add the opening XML file tags, open and close body and text tags, open and close TEI tags, and a teiHeader. Finally, TEITagger will pretty print the file if it is a valid XML file.

5 Philological use of the TEITagger software

Metrical analysis of Vedic, epic, and classical Sanskrit texts is not new. For instance, metrical analysis of the Mahābhārata has produced interesting results that bear on the critical composition of the text and its history. Edgerton (1939) distinguished regular versus irregular varieties of Triṣṭubh and Jagatī meters that were significantly divided between the Virāṭaparvan and Sabhāparvan respectively and thereby demonstrated separate composition and probably subsequent insertion of the Virāṭaparvan in the text of the Mahābhārata. He also described several regular patterns in the hypermetric and hypometric irregular varieties based upon the location of the caesura.

Fitzgerald (2006) reported the results of analyzing a database of the Triṣṭubh and Jagatī verses he assembled over the past couple of decades. He analyzed these metrical patterns into five segments: initial and final syllables, and three sets of three syllables each: the opening, break, and cadence. He identified three standard varieties of Triṣṭubh: (1) a regular Upajāti consisting of the alternating pādas of Indravajrā and Upendravajrā, (2) Śālinī, and (3) Vātormī; and a standard variety of Jagatī: an Upajāti consisting of alternating pādas of Vamśasthā and Indravamśā. Fitzgerald (2009) isolated two measurable variables: (1) the degree of uniformity among the pādas of the Triṣṭubh stanzas, and (2) the set of major Triṣṭubh features that were eliminated in the creation of the classical standard triṣṭubh. He isolated passages on the basis of runs of Triṣṭubh and Jagatī verses and measured the uniformity within verses in these passages to attempt to locate discontinuities that might signal different periods of composition of the passages. Fitzgerald (2004) argued, “if we are able to make reasonable arguments about historical fissures in the text, we thereby enrich our understanding of the text’s possible meanings ...by distinguishing multiple voices, dialogical tension, and innovation within the otherwise synchronic, unitary, received text.” In his careful unpublished study of the episode of the dice match, he was able to counter the conclusions of Söhnen-Thieme (1999), and to con-
clude that “this whole episode, the Upajāti passage of chapter 60 in which Duḥśāsana drags Draupadī into the sabhā by the hair, is likely later than most or all of the rest of this episode.”

Work of the sort that Edgerton and Fitzgerald have done with careful evaluation of statistics gathered with great effort over a long time could be vastly simplified and assisted by the automation provided by TEITagger. After testing TEITagger version 2 on the Bhagavadgītā, within a week, I tagged the entire critical edition of the Mahābhārata, including those with irregular patterns such as those with hypermetric or hypometric pādas. A driver file of nearly a thousand lines individually matched every possible combination of the syllable counts per pāda, triple-line and single line verses as well as the normal double-line verses. For example, a separate set of a regular expression and its replacement expression targets triple-line Triṣṭubh verses with a hypermetric first pāda, another targets such verses with a hypermetric second pāda, etc. The driver file assumed that such deviant metrical patterns ought to be classified under a certain type despite the failure of the meter analyzer to find a regular type. The task preceded and inspired the development of our iteration command and commands to send verse lines and pādas to the meter analyzer described in the previous section. The driver file I developed to tag the Bhāgavatapurāṇa with these features added consists of only 318 lines.

TEITagger version 2 tagged 73,436 verses and 1,057 prose sentences in 386 paragraphs. The verses include 68,860 Anuṣṭubhs, 2,970 Triṣṭubhs, 431 Jagatī, 322 Indravajrā, 0 Upendravajrā, 496 of the standard Upajāti variety alternating the two preceding, 88 Śālā, 78 Vāṇī (other Upajātis), 31 Aparavaktra (an ardhasamavr̥ttta meter), 22 Praharṣinī, 16 Rucirā, 9 Mālinī, 4 Vasantatilakā, 4 Puṣpitāgrā, 1 Šārdulavikr̥dita, 1 Halamukhī, 1 Āryāgīti (a type of Āryā), 1 mixture of half Kāmakṛḍā and half Kāmukī, and a hundred unidentified. The unidentified metrical patterns include for instance, 1 mixture of half Kāmukī and half unidentified, 1 mixture of a deviant pāda with subsequent Anuṣṭubh, jagatī, and Triṣṭubh pādas, as well as 98 other uninvestigated unidentified patterns.

The results of TEITagger version 2 are presented in Table 1 in comparison with some of the results Fitzgerald (2009) reported. One can see that there is a minor discrepancy of one passage in the enumeration of the prose passages. The cause of this discrepancy needs to be investigated. Yet otherwise there is astonishing consistency in the enumeration of the prose and verse passages. There is a discrepancy of just two verses of the Anuṣṭubh
meter. The discrepancy of 41 Triṣṭubh/Jagatī verses and 52 fancy meters is probably largely due to TEITagger’s incorrect assumption that a number of irregular meters with 11–12 syllables per pāda were of this type rather than fancy metrical patterns. For if the meter analyzer failed to identify a verse, TEITagger relied on syllable count alone to classify it.

Using TEITagger version 17 with the more refined feature of sending verse lines and quarters to the meter analyzer, and with some revision of the meter analyzer itself, I reevaluated the metrical patterns of the *Mahābhārata*. In this version, I made no assumptions about the conformity of deviant patterns to regular types; instead, where the meter analyzer failed to find a match for a verse, I permitted it to seek a match of each line of the meter, and failing to find a match for a line, to seek a match for each pāda in the line. Where lines or pādas within a verse were identified as the same, the metrical information was combined so that along with a single type classification for the verse only the deviant lines or pādas are classified separately. Labels consisting of the meter names in SLP1 for each different meter found within a verse are separated by a forward slash in the value of the type-attribute of the lg-element that contains the verse in the TEI file. These labels are preceded by letters indicating the pādas so labeled.

Table 2 shows the numbers of verses with one to six metrical identifications for the verse as a whole or parts of the verse individually. Table 3 shows the meters recognized. Column three of Table 3 shows the number of the meter indicated in column one that was recognized as a verse. Column four shows the number of additional sets of double lines recognized within triple-line meters. Column five shows the number of lines recognized in verses not recognized as verses or sets of double lines. Column six shows the number of pādas recognized in lines not recognized as lines. The first line of each section divided by double horizontal lines tallies the numbers of that general metrical type. Rows beginning with *Upajāti* in bold in the Triṣṭubh and Jagatī sections tally the numbers for the Upajāti type patterns listed in subsequent rows within the same section. The Upajāti numbers are included in the tally for the section as a whole as well. At the bottom of the table, the row labeled *Identified* in bold summarizes the total number of verses, additional pairs of lines, additional lines, and additional verse quarters recognized. The row labeled *No type* shows the number of verses not recognized before querying the meter analyzer regarding lines and pādas, and the total number of pādas that remain unidentified. The pādas that remain unidentified are provided with the label no_type within the value
Table 1
Metrical and non-metrical passages in the Mahābhārata identified by TEITagger v. 2 compared with those identified by Fitzgerald

<table>
<thead>
<tr>
<th>passage type</th>
<th>syllables/pāda</th>
<th>TEITagger 2009</th>
<th>Fitzgerald 2009</th>
</tr>
</thead>
<tbody>
<tr>
<td>passages</td>
<td>73,822</td>
<td>73,821</td>
<td></td>
</tr>
<tr>
<td><strong>prose</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>paragraphs</td>
<td>386</td>
<td>385</td>
<td></td>
</tr>
<tr>
<td>sentences</td>
<td>1,057</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>verse</strong></td>
<td>73,436</td>
<td>73,436</td>
<td></td>
</tr>
<tr>
<td>Anuṣṭubh</td>
<td>8</td>
<td>68,860</td>
<td>68,858</td>
</tr>
<tr>
<td><strong>Triṣṭubh/Jagatī</strong></td>
<td>11–12</td>
<td>4,385</td>
<td>4,426</td>
</tr>
<tr>
<td>Triṣṭubh</td>
<td>11</td>
<td>2,970</td>
<td></td>
</tr>
<tr>
<td>Indravajrā</td>
<td>11</td>
<td>322</td>
<td></td>
</tr>
<tr>
<td>Upendravajrā</td>
<td>11</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Upajāti</strong></td>
<td>11</td>
<td>662</td>
<td></td>
</tr>
<tr>
<td>Indravajrā/Upendravajrā</td>
<td>11</td>
<td>496</td>
<td></td>
</tr>
<tr>
<td>Śāḷā</td>
<td>11</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>Vaṃī</td>
<td>11</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>Jagatī</td>
<td>12</td>
<td>431</td>
<td></td>
</tr>
<tr>
<td><strong>Fancy meters</strong></td>
<td></td>
<td>100</td>
<td>152</td>
</tr>
<tr>
<td>Halamukhī</td>
<td>9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Aparavakra</td>
<td>13/12</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Puṣpitāgrā</td>
<td>12/13</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Praharṣiṇī</td>
<td>13</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Rucirā</td>
<td>13</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Vasantatilakā</td>
<td>14</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Mālinī</td>
<td>15</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Kāmakrīḍā/Kāmukī</td>
<td>15/16</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Śārdulavikrīḍita</td>
<td>19</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Āryāgīti</td>
<td>7 caturmātrās + 2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>unidentified</td>
<td></td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
of the type-attribute in the TEI file. No lines or line pairs are so labeled because if they are unidentified their pādas are sent to the meter analyzer individually for analysis. The row labeled Total in bold shows the total number of verses in the Mahābhārata in column three but in column six just the total number of pādas analyzed individually.

Table 2
Mixed metrical patterns in the Mahābhārata identified by TEITagger v. 17

<table>
<thead>
<tr>
<th>type</th>
<th>identified</th>
<th>not fully</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>single</td>
<td>70,242</td>
<td>3,194</td>
<td>73,436</td>
</tr>
<tr>
<td>mixed</td>
<td>689</td>
<td>2,505</td>
<td>3,194</td>
</tr>
<tr>
<td>double</td>
<td>85</td>
<td>4</td>
<td>89</td>
</tr>
<tr>
<td>triple</td>
<td>468</td>
<td>994</td>
<td>1,462</td>
</tr>
<tr>
<td>quadruple</td>
<td>129</td>
<td>1,451</td>
<td>1,580</td>
</tr>
<tr>
<td>quintuple</td>
<td>5</td>
<td>23</td>
<td>28</td>
</tr>
<tr>
<td>sextuple</td>
<td>2</td>
<td>33</td>
<td>35</td>
</tr>
</tbody>
</table>

TEITagger version 17 found matches for each of the fourteen varieties of Triṣṭubh Upajāti patterns and the several Jagatī Upajāti patterns named separately. It also found several additional samavrta metrical patterns for lines and verse quarters not found by analyzing whole verses. Rows headed by these meter names show blanks in the columns for verses and lines where no verses or lines of that type were found. These initial results of applying TEITagger to analyze the metrical patterns in the Mahābhārata demonstrate its capacity to reveal detailed information about a massive work and to mark up the results in a way that permits computational compilation so that these results may be presented to scholars in ways that may inspire further insight.

Table 3
Metrical patterns in the Mahābhārata identified by TEITagger v. 17

<table>
<thead>
<tr>
<th>meter type</th>
<th>syllables/ pāda</th>
<th>verse</th>
<th>2/3 lines</th>
<th>line</th>
<th>quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anuṣṭubh</td>
<td>8</td>
<td>68,360</td>
<td>10</td>
<td>521</td>
<td>633</td>
</tr>
<tr>
<td>Anuṣṭubh3</td>
<td>8</td>
<td>68,322</td>
<td>10</td>
<td>518</td>
<td>610</td>
</tr>
<tr>
<td>Pramāṇikā</td>
<td>8</td>
<td>38</td>
<td>0</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>Vidyunmālā</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>meter type</td>
<td>syllables pāda</td>
<td>verse</td>
<td>2/3 lines</td>
<td>line</td>
<td>quarter</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>-------</td>
<td>-----------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>Vibhā</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hanisaruta</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triṣṭubh</td>
<td>11</td>
<td>1,355</td>
<td>62</td>
<td>970</td>
<td>3,252</td>
</tr>
<tr>
<td>Indravajrā</td>
<td>11</td>
<td>171</td>
<td>3</td>
<td>271</td>
<td>941</td>
</tr>
<tr>
<td>Upendravajrā</td>
<td>11</td>
<td>94</td>
<td>0</td>
<td>174</td>
<td>805</td>
</tr>
<tr>
<td>Vātormī</td>
<td>11</td>
<td>1</td>
<td>30</td>
<td>0</td>
<td>597</td>
</tr>
<tr>
<td>Rathoddhatā</td>
<td>11</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Śālīnī</td>
<td>11</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>909</td>
</tr>
<tr>
<td>Upajāti</td>
<td>11</td>
<td>1,046</td>
<td>29</td>
<td>525</td>
<td>0</td>
</tr>
<tr>
<td>Bhadrā</td>
<td>11</td>
<td>68</td>
<td>2</td>
<td>167</td>
<td>0</td>
</tr>
<tr>
<td>Hanisī</td>
<td>11</td>
<td>90</td>
<td>0</td>
<td>188</td>
<td>0</td>
</tr>
<tr>
<td>Kīrti</td>
<td>11</td>
<td>114</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Vānī</td>
<td>11</td>
<td>98</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mālā</td>
<td>11</td>
<td>73</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Śālā</td>
<td>11</td>
<td>82</td>
<td>0</td>
<td>170</td>
<td>0</td>
</tr>
<tr>
<td>Māyā</td>
<td>11</td>
<td>50</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Jáyā</td>
<td>11</td>
<td>50</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bālā</td>
<td>11</td>
<td>82</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Árdrā</td>
<td>11</td>
<td>68</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rāmā</td>
<td>11</td>
<td>62</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rddhi</td>
<td>11</td>
<td>85</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Buddhī</td>
<td>11</td>
<td>67</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Siddhi</td>
<td>11</td>
<td>57</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Jagatī</td>
<td>12</td>
<td>411</td>
<td>4</td>
<td>94</td>
<td>343</td>
</tr>
<tr>
<td>Vanīṣasthā</td>
<td>12</td>
<td>359</td>
<td>3</td>
<td>73</td>
<td>181</td>
</tr>
<tr>
<td>Indravanīśā</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>95</td>
</tr>
<tr>
<td>Bhujāṅgaprayāta</td>
<td>12</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Kāmadattā</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Vaiśvadevī</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>Śruti</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Upajāti</td>
<td>12</td>
<td>48</td>
<td>0</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Śaṅkhanidhi</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Padmanidhi</td>
<td>12</td>
<td>2</td>
<td>0</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>Vanīśamālā</td>
<td>12</td>
<td>45</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fancy</td>
<td></td>
<td>116</td>
<td>0</td>
<td>37</td>
<td>32</td>
</tr>
<tr>
<td>meter type</td>
<td>syllables pāda</td>
<td>verse</td>
<td>2/3 lines</td>
<td>line</td>
<td>quarter</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>-------</td>
<td>-----------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>Halamukhī</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Śuddhavirāj</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Aparavaktra</td>
<td>13/12</td>
<td>27</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Puṣpitāgrā</td>
<td>12/13</td>
<td>33</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Praharṣini</td>
<td>13</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Rucirā</td>
<td>13</td>
<td>28</td>
<td>0</td>
<td>11</td>
<td>28</td>
</tr>
<tr>
<td>Prabhavatī</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Vasantatilaka</td>
<td>14</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Praharaṇakalikā</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Mālinī</td>
<td>15</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Śārdūlavikriṣṭita</td>
<td>19</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Upagīti</td>
<td>5cm+l+1cm+g</td>
<td>6</td>
<td>0</td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>Āryagīti</td>
<td>7cm+gg</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Identified</td>
<td>70,242</td>
<td>76</td>
<td>1,622</td>
<td>4,267</td>
<td></td>
</tr>
<tr>
<td>No type</td>
<td>3,194</td>
<td></td>
<td></td>
<td></td>
<td>4,297</td>
</tr>
<tr>
<td>Total</td>
<td>73,436</td>
<td></td>
<td></td>
<td></td>
<td>8,564</td>
</tr>
</tbody>
</table>

### 6 Communication between TEI files and linguistic software

As mentioned in section 1, one of the principal benefits of encoding Sanskrit texts using TEI XML is to fulfill the need to coordinate directly, without human intervention, with software developed by others, possibly in ways not anticipated. In particular, by encoding Sanskrit texts in TEI we anticipate coordinating a large repository of digital Sanskrit texts with parsers and syntax analyzers, such as the Sanskrit Heritage parser and the University of Hyderabad’s संसाधनी. TEI provides robust standardized methods to coordinate various versions of texts and to refer to particular divisions and segments within a text so that parsed and syntactically analyzed passages may be interlinked with their originals. Naturally, the highest levels of coordination between versions would require standardized identification of the repository that houses the original file from which a passage was taken and submitted to a linguistic analysis tool on another site. An attribute value pair such as simply `repository='sl'`, or more officially `repository='US-RiPrS1'` using the International Standard Identifier for
Libraries and Related Organizations (ISIL), ISO 15511, might identify the Sanskrit Library as the repository. Obviously standardized identification of the file within the repository is required, either by collection and item identifiers or by filename. These identifiers should be interpretable programmatically as a URL, or be a URL directly provided with a submission. For example, if I submit the first verse of the unanalyzed text of the Mahābhārata to the Sanskrit Heritage parser I might provide the URL http://sanskritlibrary.org/texts/tei/mbh1.xml with my submission.

A second level of standardized identification is required to identify the type of analysis. When the Sanskrit Library analyzed the TITUS archive’s texts for inclusion in 2006, it discovered a surprising variety in the degree and type of analysis of sandhi. Some of these encoding practices can be specified in the encoding description of a document. However, standard designation of various degrees of analysis is needed to coordinate versions. At the least, one might consider standard designation for the types of analysis of Sanskrit texts described in Table 4. For clarity, it is strongly recommended that these different degrees of analysis be located in separate files, not combined in a single file. TEI provides simple means of coordinating such versions by synchronizing element identifiers (xml:id).

Once a file containing the version of a text with a specific degree of analysis is identified, standardized reference to particular sections and passages is required. TEI provides machine-readable methods for declaring the element used and the structure of references within two elements of the teiHeader:

- tagsDecl
- refsDecl

The tagging declaration may be used to document the usage of specific tags in the text and their rendition. Figure 7 shows the tagsDecl element used for the Sanskrit Library’s TEI edition of the critical edition of the Mahābhārata. Because the value of the partial attribute is specified as false, the tags listed as values of the gi attribute of the tagUsage elements are all the elements and the only elements that occur under the text element. The lg, l, and seg elements are used to mark up verses as shown in figures 2, 3, and 4, in the last of which are shown also the use of the body, div, sp, and speaker elements. The p and s elements are used to mark up paragraphs

---

Table 4

Degrees of analysis of Sanskrit texts

1. continuous text (saṁhitā-pāṭha)
   a. with breaks only where permitted in Devanāgarī script, i.e. only after word-final vowels, visarga or anusvāra
   b. with breaks where permitted in Roman script, i.e. after consonants as well
   c. with breaks where permitted in Roman script with designation immediately following characters representing sounds that result from single replacement sandhi at word boundaries

2. sandhi-analyzed text (pada-pāṭha)
   a. with word final visarga throughout, without designation of compound constituents
   b. distinguishing visarga originating in final s from visarga from final r
   c. with designation (but not analysis) of compound constituents as permitted in Devanāgarī script, i.e. after constituent-final vowels, visarga or anusvāra
   d. with designation (but not analysis) of compound constituents as permitted in Roman script, i.e. after constituent-final consonants as well
   e. with designation (but not analysis) of compound constituents as permitted in Roman script, with designation immediately following characters representing sounds that result from single replacement sandhi at constituent boundaries
   f. with analysis of sandhi between compound constituents as well

3. morphologically analyzed text
4. lexically and morphologically analyzed text
5. syntactically analyzed text
   a. dependency structure
   b. phrase structure
and sentences in prose. The numbers listed as values of the `occurs` attribute in the `tagUsage` elements indicate the number of occurrences of the element named in the value of the `gi` attribute. The numbers shown are those for the *Svargārohaṇaparvan*. Those mentioned as values of the `selector` attribute of the `rendition` element with `xml:id='skt'` are all the elements and the only elements that render Sanskrit text in SLP1 to be transcoded to Unicode Roman, Devanagari, or another Indic Unicode encoding for redisplay. These elements provide all that is necessary to extract Sanskrit text from the encoding for display in HTML, and for submission as a unit to metrical, morphological and syntactic analysis software. The attribute values of the elements listed in the `rendition` element with `xml:id='sktat'` lists all the attributes and the only attributes whose values are Sanskrit text in SLP1 to be transcoded. These attribute values are Sanskrit terms that might be used to display menus in an HTML display to select divisions such as parvan, and adhyāya.

The reference declaration describes the reference system used in the text.\(^3\) TEI offers the possibility of describing the pattern of canonical references formally in a manner amenable to machine processing. A regular expression describing the pattern of the canonical reference is paired with a replacement expression that describes the path to the attributes that contain the referenced numbers (n attributes of `div` and `lg` elements in verse in the *Mahābhārata*, and of `p`, and `s` in prose). Figure 8 shows the `refsDecl` element of the Sanskrit Library’s TEI edition of the *Svargārohaṇaparvan*. The pattern shown in the `matchPattern` attribute of the first `cRefPattern` element describes a canonical reference to any verse quarter in the *Mahābhārata*. The three sets of digits separated by periods refer to the parvan, adhyāya, and verse; the letter refers to the pāda, for example, 6.24.70a refers to the first pāda of the seventieth verse of the twenty-fourth adhyāya of the sixth parvan shown in Figure 2. (The 24th adhyāya of that parvan is the second in the *Bhagavadgītā*.) The first of the two `cRefPattern` elements gives a replacement expression that matches a path that has verses directly as children of a `div` element; the second, one that has verses as children of an intervening `sp` element within an adhyāya. Subsequent `cRefPattern` elements describe shorter references to whole verses, adhyāyas, and parvans. These elements and attributes directly provide an unambiguous method to

---

Figure 7

The `tagsDecl` element in the Sanskrit Library’s TEI edition of the Svargārohaṇaparvan of the Mahābhārata
resolve canonical references to particular passages. Yet, processed in the opposite direction, from the replacement path to the match expression, the references provide a means to compose canonical references from n attributes.

Once a standard system of exact references to specific passages in unanalyzed continuous text has been adopted, reference to various versions of analyzed passages are easily constructed by specifying in addition the degree of analysis described in Table 4. One method of doing this in a TEI document would be to specify the degree of analysis as a value of the ana attribute of the text element. Another would be for archives to add a standard addition to the filename.

Linguistic software that produces TEI output would add elements subordinate to those containing text in the TEI document that contains the continuous text. A document that contains analyzed sandhi but no further analysis would insert each word (pada), including compounds (samasta-pada), in a w element. A document that contains compound analysis would insert the lexical constituents of compounds in a w element subordinate to the compound’s w element. Although the types of analysis described in Table 4 do not envision tagging non-lexical morphemes such as the infix a and suffix ti in the verb gacchati, such morphemes would be inserted in an m element. TEI provides attributes that may be used for lexical and morphological analysis of each word in a w element. The stem of the word is made the value of the lemma attribute. We have chosen to make the lexical identifier a value of the type attribute and the morphological identifier a value of the subtype attribute. Figure 9 shows our TEI mark up of the sandhi analysis of the first verse of the Bhagavadgītā, MBh. 6.23.1, and Figure 10 shows our TEI mark up of the lexical and morphological analysis of the same verse. Where authors deliberately compose passages that are amenable to more than one analysis (śleṣa), alternative analyses — whether of verses, lines, verse quarters, prose passages, or individual words — may be analyzed in separate files where, in order to permit coordination, they may be supplied with the identical division numbers and xml:ids as their unanalyzed passages and the preferred analysis.

As a result of standardized coordination of markup and reference between Sanskrit text archives and Sanskrit computational software, HTML displays showing the unanalyzed version of a verse might be able to include a set of links to various analyzed versions for the convenience of students and scholars of Sanskrit. Conversely, displays of the results of analysis of a passage might also provide links to the unanalyzed source.
Figure 8

The *refsDecl* element in the Sanskrit Library’s TEI edition of the Svargārohaṇaparvan of the Mahābhārata
Figure 9

TEI mark up of the sandhi analysis of MBh. 6.23.1, the first verse of the Bhagavadgītā
Figure 10

TEI mark up of the lexical and morphological analysis of MBh. 6.23.1, the first verse of the Bhagavadgītā
References


Preliminary Design of a Sanskrit Corpus Manager

GÉRARD HUET and IDIR LANKRI

Abstract: We propose a methodology for the collaborative annotation of digitalized Sanskrit corpus tagged with grammatical information. The main features of the proposal are a fine grain view of the corpus at the sentence level, allowing expression of inter-textuality, sparse representation allowing non-necessarily sequential acquisition, and distributed collaborative development using Git technology. A prototype Sanskrit Corpus Manager has been implemented as a proof of concept, in the framework of the Sanskrit Heritage Platform. Possible extensions and potential problems are discussed.

1 Introduction

Several digital libraries for Sanskrit corpus have been developed so far. We may mention the GREtil site of Göttingen’s University,\(^1\) with a fair coverage, under various formats. The Sarit site,\(^2\) developed by Dominik Wujastyk, Patrick McAllister and other Indology colleagues, contains a smaller corpus, but it follows a uniform format compliant with the Text Encoding Initiative (TEI) standard, and has a nice interface. Furthermore it benefits from a collaborative acquisition framework using Git technology. The Sanskrit Library\(^3\) developed by Peter Scharf and colleagues, also follows the TEI, and benefits from the tagging services of the Sanskrit Heritage Platform, since individual sentences link to its segmentation cum tagging service. DCS\(^4\) developed at Heidelberg University by Oliver Hellwig, is the most advanced from the point of view of linguistic analysis, since it is fully

\(^1\)Göttingen Register of Electronic Texts in Indian Languages http://gretil.sub.uni-goettingen.de/gretil.htm
\(^2\)Search And Retrieval of Indic Texts http://sarit.indology.info
\(^3\)Sanskrit Library http://www.sanskritlibrary.org
\(^4\)Digital Corpus of Sanskrit http://kjc-sv013.kjc.uni-heidelberg.de/dcs/
annotated with morphological tags indexing a lexicon of stems. Its development involved several iterations of deep learning algorithms (Hellwig 2009, 2015, 2016). Covering at present 560,000 sentences, it is today the closest analogue for Sanskrit of the Perseus Digital Library for Greek and Latin corpus.  

Several other efforts are currently under development, although unfortunately with little standardization effort. Not all digital libraries are publicly available. For instance, the TITUS Thesaurus of Indo-European text is accessible only to scholars participating in the acquisition effort.

There exist now several computational linguistics tools that process Sanskrit text in order to parse it under a grammatical representation that can be considered an approximation to a formal paraphrase of its meaning. Typically, a sentence will yield a stream of morphological tags. The DCS analyser of Oliver Hellwig, based on statistical alignment on a data base of lemmas trained from a seed of human-annotated tags, has the advantage of being fully automatic. The Sanskrit Heritage Platform under development at Inria Paris offers a service of segmentation with tagging (at two levels, inflection and morphology of stems), linking into a choice of two dictionaries. It also has a surface parser using kāraka analysis that can be used for learners on simple sentences, but is not sufficient for corpus processing (Huet 2007). It also links with Amba Kulkarni’s Saṃsādhanī analyser, that helps produce a dependency graph (Kulkarni 2013). This structure captures the semantic role (kāraka) analysis of a sentence, provided it is not too dislocated. Furthermore, an auxiliary tool helps the annotator to transform a dislocated sentence into its prose order by proper permutation of its segments.

Thus it seems that the time is ripe to consider establishing a common repository that would store digital Sanskrit libraries in annotated form, either automatically, or with the help of competent annotators using interactive tools. We present here a preliminary proposal for the design of a Sanskrit corpus manager concept, that could serve as a seed repository for the collaborative editing of texts, and that could support navigation and search through appropriate further tools. We have developed a simplified implementation of the concept, using technology available off-the-shelf as free software. We shall conclude by listing problems in the managing of a joint corpus repository.

5Perseus [http://www.perseus.tufts.edu/hopper/](http://www.perseus.tufts.edu/hopper/)
6TITUS [http://titus.uni-frankfurt.de/index.htm](http://titus.uni-frankfurt.de/index.htm)
7Saṃsādhanī [http://scl.samsaadhani.in](http://scl.samsaadhani.in)
2 Specificities of Sanskrit corpus

Processing Sanskrit by computer is in some sense easier than processing other natural languages, at least if we restrict ourselves to the classical language. It benefits of the grammatical tradition [ vyākaraṇa] dating back from hoary times, since the grammar of Sanskrit was fixed by Pāṇini 25 centuries ago in his Aṣṭadyāyī, which was initially descriptive, but later became prescriptive. Classical Sanskrit was not the vernacular local prakrit, which is used only in the theater. It was the language of the educated [śiṣṭa]. And thus, it was assumed grammatically correct, which means that we may align our segmentations to a precise recursive definition.

Granted, there are many non-Paninian forms in epics literature, and there are many corrupted texts. But we may record exceptions, and corrupted texts may perhaps be amended. Of course philologists will shudder at the thought of amending texts, but they must excuse my irreverence, considering that in my professional trade, programs with bugs must be corrected, and only secondarily treated as historical artifacts in the version-maintaining repository. The main merit of mistakes is to trace possible filiations of versions, since scribes often copied without amending their sources, and thus errors would be transmitted. But this assumption is not always met, and thus the classical phylogenetic tradition is challenged (Hanneder 2017). In any case, I am making the assumption that the corpus recorded in the global repository has been edited to the point of being grammatically correct. Possibly as a result of the interactive use of grammatical tools, in as much as they may be used as editing assistants.

Actually, the Sanskrit language is not that regular. Even seemingly regular processes such as compounding pose problems in the proper analysis of written texts, since compounding is not associative, and accent is not marked in writing. Furthermore, there are many different styles, not just prose and poetry. The grammatical sūtra style is very concise, closer to algebraic rules, with phonemes used both for linguistic and meta-linguistic notation. The śāstra style of scholastic Sanskrit (Tubb and Boose 2007) is also highly artificial. The Indian tradition of formal debate ( vāda) (Tripathi 2016) produced texts that are layers upon layers of commentaries, with counter-arguments ( pūrvapakṣa) alternating with upheld theses ( uttarapakṣa, siddhānta). Poets indulged in obscure constructions, rare lexemes, very long compounds, and dislocated sentences. Furthermore, the inherent ambiguity of phonetic enunciations where word boundaries are blurred by sandhi gave rise to a
whole new genre of śleṣa – double entendre – where ambiguous segmentation yields possibly opposite meanings (Bronner 2010). For instance, consider nakṣatrapathavartinā rājñā from Daṇḍin’s Kāvyādārśa. It may mean a glorious king “following the path of the stars” (nakṣatra-patha-vartinā rājñā), or a despicable king, “not following a noble path” (na kṣatra-pathavartinā rājñā), playing on the oronyms nakṣatra and na kṣatra. Here specific philological apparatus is needed in order to display the two readings, it is not just a matter of choice between segmentations, since both readings are intended. But if linear text is given up in benefit of graphical display, we may visualise the mixed two readings as shown in our Reader tool, see Figure 1.

![Figure 1](image)

Daṇḍin’s prototype śleṣa

Other difficulties in interpreting Sanskrit text are the absence of distinctive sign for proper names (like capitals in Roman script), making e.g. kṛṣṇa ambiguous between the divine hero and the black color, and the ambiguity of prādi compounds such as nirvācyā, that may mean “what should not be talked about” (with nis preposition acting as negation) as well as “what should be explained” (now compositional future participle of verb nirvac). Another problem, at the discourse level this time, is indirect speech, whose ending is marked with particle iti, but whose beginning must be guessed from the context. All these reasons show that editing a text in order to express several possible meanings with distinct morphological annotations, explained through distinct grammatical analyses, is a much more difficult task than simply listing raw sentences in sandhied form.
Finally, Sanskrit literature abounds in inter-textuality features. Mantras are quoted, stories are retold in a manifold manner, bards adapt orally transmitted tales, learned commentaries pile up on each other, numerous anthologies of poems and maxims (subhāṣita, nyāya) share a lot of material, mahākāvyaśas expand episodes of epics, etc.

Considering all these difficulties, we propose a set-up for progressive computer-aided tagging of selected portions of corpus, with documented intertextuality, as an alternative to TEI-style full digitalization of corpus in raw form. Thus one of the important requirements is that the (partial) corpus be represented at a low level of granularity, typically a śloka for poetry, or a sentence for prose.

3 Available technology

The main paradigm of the proposed annotation scheme is that it should be a distributed service, not just available from a server for consultation of readers, but itself the locus of collaborative annotation activity. This is in line with the recommendation of Peter Robinson (Robinson 2009): “The most radical impact of the digital revolution is to transform scholarly editing from the work of single scholars, working on their own on single editions, to a collaborative, dynamic and fluid enterprise spanning many scholars and many materials”.

In the software development domain, now Git technology (Chacon and Straub 2014) is the de facto standard for such collaborative development. Originally designed to serve as versioning cum distribution for the Linux effort, it quickly replaced all previous software management systems. It has several implementations, one managing the GitHub site, popular for open-source development. The GitLab software offers additional functionalities, notably in terms of security.

A Git project consists of branches evolving with time, each branch carrying a hierarchy of files. The hierarchy corresponds directly to the structure of the file system of the host operating system. The files typically contain source code of software, and of its documentation. But they may be of whatever format. Collaborators of the project have a local copy of the relevant branch on their own computer station. So they may not only compile and install locally the software, but they may modify it and add to it. After local testing, the developer may request the supervisor of the branch to update
the global site with his modifications. On approval, the software merges the modifications with the current version, a possibly complex operation.

Git is a major change of paradigm in the collaborative development of massive efforts. It is now used for the dynamic management of documents of various nature. This is a mature technology, with the finest available algorithms in distributed computing, alignment, compaction, cryptography. It looks like the ideal collaborative tool for developers of a digital library.

The other obvious technological decision is to use Web technology for the user interface. HTML and XML support Unicode for presenting all writing systems. Web services are now the absolute standard for distributed services.

4 Implementing a prototype as a proof of concept

A 2-months effort in summer 2017 was defined as a student Master project. The second author, in the Master program of University Paris Diderot, and an Ocaml expert, was offered an internship at Inria for its implementation. He familiarized himself rapidly with the sources of the Sanskrit Heritage Platform, put at this occasion on Inria’s GitLab site for distributed development under Git. At the same time, a second Git project was launched as the Sanskrit Heritage Resources, to distribute the lexical resources used by the Platform machinery, as well as the Sanskrit morphology XML databanks that it produces.

The requirement was to implement a corpus manager as a Web service, using the Sanskrit Heritage Platform as interactive tagging service, and producing progressively an annotated corpus as a sub-branch of the Sanskrit Heritage Resources Git project. The hierarchical structure of the corpus is directly mapped on the directory structure of the UNIX file system.

4.1 The Sanskrit Heritage Corpus Manager

Three levels of capabilities have been defined. The Reader capacity is available to any user. As its name indicates, he is only allowed to read the library, but not to modify it. The Annotator capacity allows addition and correction to the corpus files. The Manager capacity allows addition and correction to the directory structure. These three capacities are mapped respectively to permissions of the UNIX file system, and to roles in the Sanskrit corpus project, initially located as a component of the Sanskrit Heritage Resources Git project.
Texts are available as leaves of the directory structure, such as “KAvya/BANa/KAdambarI/”. In Manager mode, one may add to this structure, or edit it. In Reader mode one may navigate through it, through a simple Web interface with scrolling menus. In Annotator mode you may add to the text, or give corrections. For instance, let us assume that, in Annotator mode, we input a sentence in the initially empty directory “KAvya/BANa/KAdambarI/”. We are forwarded to the Sanskrit Heritage Reader page (Goyal and Huet 2016), where we input in the text window the following string:

rajojuṣe janmani sattvavṛttaye sthitau praṣānām pralaye tamaḥspṛṣe |
ajāya sargasthitināśahetave trayīmayāya triguṇātmane namāḥ ||

The segmenter returns with 155,520 solutions represented on a single HTML page as a graphical display where the annotator, if familiar with the tool, very quickly converges to the desired solution (in 13 clicks). When this is done, a Save button prompts the annotator, who may save this segmentation in the corpus, or abort. On saving he is returned to the corpus interface, which prompts him for the next sentence. The screen he sees at this point is represented in Figure 2. It indicates that now branch “KAvya/BANa/KAdambarI/” supports a text where sentence 1 is now listed (in IAST notation according to local settings, could be Devanāgarī or both).

This sentence 1 is itself a link, to the display of the Heritage reader, as shown in Figure 3. Note that this is what we saved, showing the unique so-
Huet and Lankri

olution selected by the annotator. Note also that what we see is just the usual display of the Reader: you may click on segment rectangles in order to get their morphology. The morphology is itself linked to the dictionary, which can be set either to the Sanskrit-French Heritage dictionary maintained at the site, or to the electronic version of the Sanskrit-English Monier-Williams dictionary. It is not just an HTML static page, it is a dynamic page where all services of the Platform are available. Including backtracking on the annotator choices, via the Undo service! You may also click on the Unique Solution tick and continue the analysis with gender agreement. Or by clicking on the UoH Analysis Mode tick, go further into kāraka analysis with Amba Kulkarni’s dependency analyser. Thus, it would be easy to extend the service with other displays under various analyses, provided with proper meta-data.

Figure 3
Displaying sentence 1

Now let us return to Figure 2. Please note the “> 1” button. It invites the annotator to continue his tagging task, with another sentence. If you click on it, a scrolling menu prompts you with the sentence number, and an Add button. You may at this point choose not to continue in sequence, for instance choose sentence 4. On pressing Add we are back in the loop with the Reader.

After entering sentence 4, we get the image of this piece of corpus as Figure 4. There are now two mouse-sensitive buttons: one marked “2 – 3” for filling the hole between 1 and 4, the other one, marked “> 4”, for entering sentences after the 4th one. This illustrates the partial nature of the annotated corpus. Scholars may concentrate on often quoted parts, or verses they have a special interest in, without having to tag continuously from the beginning of the work. This is important, in view of the non-fully-automatic nature of the annotating task. It also allows work splitting
between annotators of the same text. Merging their contributions will be managed by the Git mechanisms.

When a Reader browses the corpus, he will see exactly the same display, except that the adding buttons will not appear.

When an annotator has completed some tagging, he may call a routine that will store his annotations in the local repertory of the Corpus Git project. He may then use the commit Git command to commit his annotations, with proper documentation. Once in a while he may distribute his contributions to the community by using the push Git command in order to merge his work with those of the other annotators, under control of the project Managers.

4.2 Application to citations analysis in the Heritage dictionary

This prototype of a corpus manager has been implemented as an auxiliary service of the Heritage platform segmenter. It is currently being put to use to manage citations in the Heritage hypertext dictionary. Its current 800 citations are being progressively tagged, and entered in a standalone branch “Heritage_citations” of the corpus. The corpus structure is implemented as a sub-project of the Heritage_Resources Git project, and as such is incorporated in the Heritage_platform server data, at installation time. Thus the facility is available for testing to whoever installs the two software packages through Inria’s GitLab server, as projects https://gitlab.inria.fr/huet/Heritage_Resources.git and https://gitlab.inria.fr/huet/Heritage_Platform.git respectively. The public server
site http://sanskrit.inria.fr has been updated with the corpus manager, available as a “Corpus” service from the site directory at the bottom of its pages. Of course only Reader mode is available at the public site. But the distribution version, available through Git, will allow annotators to develop their own tagged corpus, and possibly merge them in the common Git repository when registered as an official Annotator.

An example of such analysed citation may be viewed in our dictionary at entry kunda. Please visit URL http://sanskrit.inria.fr/DICO/21.html#kunda. This entry is illustrated by a quotation from śloka 6.25 of Kālidāsa’s Ritusāṃhāra, underlined as mouse-sensitive. Clicking on it brings you to the corresponding corpus page, where the sentence is displayed as a list of colored segments, as shown in Figure 5. Clicking on a segment brings its lemma, with lexicon access to the root items. Although it has the same look-and-feel as the segmentation tool, it is actually displayed by the corpus manager, navigating in Reader mode in its “Heritage_citations” branch. This can be verified by clicking on the “Continue reading” button, which brings you to this branch directory, where the śloka appears as item 10. This shows the smooth integration of this tool within other services.

Figure 5
Annotated quotation

5 Extending the prototype to other tools

The extreme simplicity of this design makes it easily extensible to other grammatical tools implemented as Web services. All that is needed to incorporate them is to include a save button in the pages that return the result of their analysis, with the functionality of saving their HTML source in the corpus hierarchy. Or, even, in the style of our own implementation,
to store the sentence analysis data as parameters for the invocation of a
dynamic corpus crawler. Conversely the Add facility of the corpus manager
will have to be made aware of the variety of such services, and its display ac-
commodated to show all analyses of the given śloka by the various services.
This assumes of course that these services are reachable from the putative
annotators, either installed on their station’s own Web server, or available
at publicly available Internet servers. The Heritage set of services may be
used both ways, since it is itself distributed as an open-source system from
its Git project repository. Should the concept prove itself useful, it would
be easy to separate the Corpus Manager from the Heritage distribution, and
make it a stand-alone facility.

It is to be remarked that having several grammatical tools available
for displaying corpus in analysed form does not induce any commitment
on a standard display, each tool may keep its look-and-feel, and links to
its specific functionalities. We are not demanding either to synchronize or
align taggings effected by various tools. Annotators using one tool may
tag sentences irrespective of whether they have been already processed with
some other tool. All we have to agree on is the directory structure and its
metadata format (under control by the Git users with Manager capability),
and in the designation scheme of individual files representing the analyses.

6 Design of inter-textuality functionalities

This simple prototype provides for the moment a strictly hierarchical view
of the corpus. This is too restrictive, since it allows no sharing. For instance,
in the skeleton corpus of “Heritage_citations”, we would like to link item
10 to its original in Ṛtusamḥāra. Of course we could enter its duplicate
in its proper branch, say “KAvya/KAlidAsa/Ritusamhara/6/25”. But we
would like to document this by recording its “absolute” link in the “Her-
itage_citations” branch at item 10. This would be an easy extension of the
current mechanism. But this is only one simple example of inter-textuality.
Some of the citations are not to a full śloka, but perhaps to a portion, or
a simplification, or a reordering of some original quotation. Thus we would
need to design a notation to document such partial sharing between different
branches of the corpus.
6.1 Collating recensions and manuscript segments

We also want to be able to use the tool for recording, and comparing, various manuscripts traditions of the same text. Actually, the idea of this low-granularity corpus representation arose from a presentation by Pr Brockington at a seminar in Paris in December 2016 (Brockington 2016). He showed there two representations of various manuscripts of Sanskrit epics.

The first one, extracted from traditional phylogenetic methods (Phillips-Rodriguez, Howe, and Windram 2009) represents a tree of manuscripts of Mahābhārata, expressing the growth of the material over time. It has been obtained through phylogenetic analysis performed on sargas 43-47, 51, 59-60 and 64-65 of the Dyūtaparvan by the Supernetwork method in the SplitsTree package. The sigla used are those of the Critical Edition, with J substituted for Ñ and Z for Ś. It is reproduced in Figure 6 below (courtesy Wendy Phillips).

The second one is a Venn diagram of Rāmāyaṇa’s manuscript relationships, reproduced in Figure 7 (taken from (Brockington 2000), courtesy John Brockington). This Venn diagram representation (possibly completed by the suitable ordering of the verse portions) is a more informative view of relationships between manuscript groups, since it represents the (multi-)set of all ślokas of all manuscripts, each one represented as a subset, possibly intersecting in complex ways with other manuscripts. In other words, the Rāmāyaṇa is there considered as a Boolean expression in terms of its manuscripts segments, a more detailed concept than the phylogenic tree, although not currently producible automatically from recensions in an obvious manner.

This suggests that our śloka-level corpus ought to accommodate notation amenable to express complex sharing relationships between the manuscripts, such as:

\[ A = B [1 - 250] ; C [5 - 23] ; B [251 - 300] \]

expressing that manuscript \( A \) is the same as \( B \) with interpolation of a portion of \( C \). Such sharing relationships ought to turn into extra annotations on the corpus data representations, so that navigation through the various versions would be available.
6.2 Paths management on shared corpus

It should be obvious at this point that an extra level of abstraction is needed in order to be able to name contiguous portions of corpus recensions that are shared across manuscript versions, such as $C[5-23]$ in the notation above. This path in our corpus tree is shared between recensions $A$ and $C$. If we want to express this sharing in our corpus structure, and thus avoid the duplication of śloka annotations between $A$ and $C$, we shall need to introduce the notion of path through a dag\(^8\) of branches, of which our corpus structure is only a specific spanning tree. This induces a need to express the concept of the successor of a śloka node along a given path, since in our example node $B.250$ has successor $B.251$ along path $B$, but $C.5$ instead along path $A$. Thus we need to record this information in node $B.250$, so that we may later navigate along path $A$ by following the path $B$ until its 250th node, and then continue from node $C.5$, until node $C.23$, which will be followed by node $B.251$ along the $A$ path.

This of course assumes that the numbering of ślokas is now a function of its path, so that e.g. śloka $B.251$ appears at index 269 along path $A$, since śloka $B.251$ is shared with $A.269$ The same mechanism could allow for instance to assign to index Bhagavadgītā.1.1 the same śloka as Mahābhārata.6.63.23.

\(^8\)directed acyclic graph
Huet and Lankri

The determination of the portions of text that are amenable to sharing is decided by the human corpus managers/annotators, not a fully automatic process, since we do not want to share maximally by identifying all identical ślokas across all texts. For instance, we shall not identify all the evocations of a ritual mantra across all texts, with absurd cluttering of a unique node with all possible successors in all the texts. Furthermore, we do not want that two occurrences of the same verse in one text lead to looping paths.

6.3 Cohabitation of readings

Representing the padapātha form of a Sanskrit utterance is the first level of its interpretation. Assigning morphology to its segments is a further level of interpretation. Assigning kāraka semantic roles consistent with nominal cases and verbal voices is still a deeper interpretation; linking anaphoric references to their antecedent and cataphoric ones to their postcedent, together with entity-name recognition, brings analysis at the discourse level. Accommodating these various levels of analysis of a text will need adaptations to our corpus representation structure. The basic idea is that a piece of corpus represents more than the raw text as a stream of phonemes, and that paths through the fine-grain structure represent not just a list of phonetic productions, but a specific reading of this text.

Thus we must admit paths that represent different glosses of a given text, possibly contradictory. For instance, we would need different path assignments for Bhagavadgītā according to Śaṅkara and to Madhva respectively, so that e.g. BhG{24.2.17} appears as nāsatovidyatebhāvonābhāvovidyatesataḥ on the first path, and nāsatovidyate’bhāvonābhāvovidyatesataḥ on the second.\(^9\) Note that in this example, the use of avagraha does disambiguate the two readings, but as stream of phonemes they are the same. This is a case showing that we need two different nodes in our corpus representation for common phonetic material, since their meanings are not compatible. Note that the two readings are oronyms, but this is not a case of śleṣa, where the two meanings are intended. We could talk of XOR-onyms, contrasted with AND-onyms (the genuine śleṣas), for which we want to represent the two readings together in the same structure. The XOR/AND terminology stems from Boolean algebras in Stone form, such as Venn diagrams.

Genuine śleṣas are often used for expressing simili figures of style, as Bāṇa demonstrated ad nauseum in Kādambarī. Their translation in lan-

\(^9\) communicated by Pr. Madhav Deshpande
languages such as French or English necessitates heavy paraphrases weaving the description and its metaphor as coordinated phrases.\textsuperscript{10} Giving a notation to represent the two readings without duplication is an interesting challenge: we want to represent minimally the two segmentations while sharing their common segments. Note that the graphical interface of the Heritage Reader gives a possible solution to this problem, since we may keep the two sets of segments, without any duplication, by trimming all segments that appear in neither. See Figure 1.

The design of a proper notation for annotated corpus is beyond the scope of the present paper, and is the affair of professional philologists, but our prototype could provide a test bed for such experiments.

Actually, we could also include in the corpus directories information concerning studies of a particular śloka or portion of text, mentioning bibliographic references to relevant literature. It could also refer to discussions concerning specific grammatical points, respective validity of the various annotations, etc. Each Sanskrit śloka could have its own blog page, and the global corpus structure could evolve into social networking for Sanskrit text!

7 Remaining problems

Our toy corpus manager raises serious issues which will have to be well assessed before scaling up to a durable infrastructure.

First of all, we are suggesting that a common repository of analysed Sanskrit text be agreed upon by both developers of computational linguistics tools and scholars managing digital libraries. This raises issues of a legal and sociological nature. Certain institutions will want to control what they deem is their intellectual property. Certain scholars will refuse to compromise with their freedom of doing things their own way. Even if a critical mass of individuals agree on sharing their work on a common source-free repository, we know from experience that committee work is not always optimum to design a technical artifact. Apparently simple issues such as the naming of branches may reveal complex problems, the solutions of which may not be easy to agree on.

\textsuperscript{10}In French, śleṣa is limited to curiosities like the holorime “Gal, amant de la Reine, alla, tour magnanime, galamment de l’arène à la tour Magne à Nîmes” and jokes like “mon frère est ma sœur” playing on the oronyms ma sœur/masseur
Another important issue is durability. Our proposal assumes that the analyzing tools will be perennial, in as much as their proper availability is necessary for the display of their analyses. This is implicit from the fact that we are not restricting ourselves to displaying static XML or HTML pages, but allow the execution of Web services (cgi-bin executables in the Web jargon) which demand availability of programming languages and their compilers over the life span of the digital library. Thus robustness and maintainability of the satellite tools are critical. Versioning is another issue, since our analysis tools are not finished products, but experimental software that keeps evolving, and that may depend on lexical resources that also evolve themselves. Thus non-regression analysis tools will have to be developed, in order to correct taggings that are no longer identified or are no longer unique after a change of version. However, please note that improvements in precision that do not compromise recall often do not require revisiting the analysed corpus, which should be robust to such upward-compatible improvements.

Finally, let us emphasize that our proposal concerns just the foundations of a collaborative framework for the grammatical annotation of Sanskrit text, and has no pretense at providing philological tools such as collating software. Such tools will have to be re-thought over this low-level representation of the corpus.

8 Conclusion

We have presented general ideas concerning a Sanskrit corpus manager, and implemented a prototype with the Sanskrit Heritage Platform to test the main concepts. The main design imperative is that corpus managing ought to be a collaborative effort, allowing text annotation on a variety of grammatical analysis services. The prototype implementation, in a restricted setting, shows that the infrastructure development is actually rather simple, if one uses off-the-shelf technology such as Web services and Git repositories. It is hoped that this proposal will spur interest from philologists and computational linguists, and hopefully contribute to their increased collaboration.
References


Hanneder, Jürgen. 2017. To edit or not to edit. Pune Indological Series I, Aditya Prakashan, Pune.


Enriching the digital edition of the Kāśikāvṛtti by adding variants from the Nyāsa and Padamañjarī

Tanuja P. Ajotikar, Anuja P. Ajotikar, and Peter M. Scharf

1 Introduction

1.1 Importance of the present work

As is well-known, the Kaśikāvṛtti (KV.), written by Jayāditya and Vāmana in the seventh century CE, is the oldest extant complete running commentary on Pāṇini’s Aṣṭādhyāyī (A.). While several complete editions of the text have been published, and a critical edition by Sharma, Deshpande, and Padhye (1969–1970), it is known that the KV. has textual problems. Sharma, Deshpande, and Padhye’s critical edition is based on only nine manuscripts as well as four previous editions while in the New Catalogus Catalogorum Raghavan and Kunjunni Raja (1968, 116b–188a) have noticed more than two hundred manuscripts. Efforts to produce a truly critical edition begun nearly thirty years ago led to the publication of an edition of the pratyāhāra section by Haag and Vergiani (2009). Now with new funding under the direction of Malhar Kulkarni at the Indian Institute of Technology Bombay, a project promises to produce an edition of the first pāda of the first adhyāya.

Manuscripts in India last no more than about five hundred years. The oldest readable manuscript of the KV. dates only to the early fifteenth century. Yet several centuries earlier, the KV. was commented upon in the Kāśikāvivaraṇapañjikā by Jinendrabuddhi in the eighth or ninth century and then in the Padamañjarī by Haradatta in the thirteenth century. These commentators provide information about the constitution of the text of KV. in several ways: by direct citation and incipits, as well as less directly by a discussion on the text. The information provided by commentators
hundreds of years prior to the oldest manuscript is invaluable to reliably establish the text of the KV. It would be extremely helpful for the community of Sanskrit grammarians if an edition supplemented with the readings available in the commentaries of the KV is made available.

The Osmania edition seldom mentions variants reported in the commentaries, and, when it does, occasionally does so erroneously. Kulkarni et al. (2016) include an appendix indicating which readings of the Osmania edition of the KV on the pratyāhāra sūtras are supported by the Nyāsa (NY) and Padamañjarī (PM). In that appendix, they use various signs to indicate which reading is supported by NY, which is supported by PM, and which is supported by both of them. It is a useful appendix, yet it covers a small fraction of the text and it lacks information concerning readings in commentaries that differ from the Osmania edition, whether the PM, which is a later commentary, is aware of the reading given by the NY, how many readings are regarded as wrong by these commentators, etc. Therefore, there is a need to create an edition that presents this information accurately for the whole text to the community of Sanskrit grammarians in particular and Sanskrit scholars in general.

2 Method of data collection

The complex, diffuse, and extensive nature of the data in the commentaries regarding readings in the KV. begs for systematic digital methodology. The digital medium provides a means to collect and organize complex information reliably, and to present that information in multiple uncomplicated views. The Text-Encoding Initiative (TEI) provides a means to indicate supporting and variant readings in a critical apparatus. The digital text of the Osmania edition (1970) of the KV. is available from the Sanskrit Library in a sandhi-analyzed form in the Sanskrit Library’s Phonetic Encoding (SLP1). Hence we choose to undertake the production of a digital edition of the sandhi-analyzed KV. with critical apparatus tagged in accordance with TEI in SLP1. We proceed in this undertaking despite the fact that the source digital text is not yet reliably proofread and is not yet itself marked up according to TEI. We propose to so mark it up during the course of our work in accordance with the method demonstrated by Scharf (2018).
2.1 TEI critical apparatus tags

TEI offers the following elements and attributes to mark up a critical apparatus:

1. The \texttt{app} (apparatus) element is used to group together each lemma and all its variations; it has two child elements: \texttt{lem} and \texttt{rdg}.

2. The \texttt{lem} (lemma) element is an optional child of the \texttt{app} element. In this context, the term \textit{lemma} signifies the accepted reading in the base text.

3. The \texttt{rdg} (reading) element is a required child of the \texttt{app} element used to indicate variations from the base text.

4. The \texttt{loc} (location) attribute of the \texttt{app} element specifies the location of the lemma in the base text.

5. The \texttt{wit} (witness) attribute specifies which commentary supports the reading. This attribute is used in both of the elements \texttt{lem} and \texttt{rdg}.

6. The \texttt{type} attribute is used to specify whether the reading is termed occasional, wrong or desired in the apparatus.

At present the \texttt{loc} attribute specifies only the canonical number of the sūtra under which the lemma occurs. In a sandhi-analyzed TEI text fully marked as described by Scharf (2018), the location will be specified in addition precisely to the paragraph, sentence, and possibly word.

2.2 Sigla

The \texttt{wit} attribute’s values are sigla that indicate which commentary and variants reported in commentaries witness a particular reading. The following sigla are used:

1. \texttt{ny} stands for the reading given by the \textit{Nyāsa}.

2. \texttt{pm} stands for the reading given by the \textit{Padamañjarī}.
2.3 Types of readings

Four different types of readings are found in each of the two commentaries. We indicate these by the following values of the type attribute in SLP1 encoding:

1. **apapÅWa** indicates that the reading is considered wrong by the commentator.

2. **kvacit** indicates that the reading is mentioned as a variant found by the commentator somewhere other than in his principal text.

3. **yukta** indicates that the reading in question is not received by the commentator, but suggested by him as the correct reading.

4. **pratIka** indicates that the reading is an incipit that supports but is not identical to the lemma.

2.4 Samples

Below are shown three samples of TEI tagging in our critical apparatus. The first shows a lemma supported by both commentaries. The second shows a lemma for which each commentator has given a different reading. The readings are assumed to be found by each commentator in his principal manuscript of the KV, since the readings are provided without any comment regarding their source. The third example shows a lemma supported by Jinendrabuddhi’s principal text and partially supported by Haradatta’s, yet for which Jinendrabuddhi remarks that the reading in another manuscript is incorrect.

```xml
<app loc='A1.1.1'>
  <lem wit='ny pm'>vfdDiSabdaH</lem>
</app>

<app loc='A1.1.2'>
  <lem>jayanti</lem>
  <rdg wit='ny'>paWanti paWan</rdg>
  <rdg wit='pm'>pacanti jayanti</rdg>
</app>

<app loc='A1.1.30'>
  <lem wit='ny'>tvayakA kftam</lem>
</app>
```
3 Issues

3.1 Data representation

Gathering comments regarding readings from commentaries differs from the collation of manuscripts. When a critical edition is prepared, the assumption is that each manuscript covers the entire span of text edited unless comments to the contrary are made in the manuscript description in the introduction or a comment regarding omission is made in the critical apparatus. Hence only variants are reported in the critical apparatus and it is assumed that silence regarding a witness reports its support for the adopted text. Readings that are identical to the lemma are not reported. In contrast, commentaries on scientific texts, and on grammatical texts in particular, generally do not mention or comment upon every word of their base text. Even the KV., as a commentary on every sūtra of the Aṣṭādhyāyī, does not mention every word of every sūtra as there found. Subcommentaries as a rule specifically mention only a small proportion of the words in the base text. Since the full text is not always cited, one cannot assume that silence regarding reading in the base text indicates support. Therefore, while collecting readings from the commentaries, it is necessary to note explicit comments regarding support along with variants. The notation of positive readings, as well as variants, has the additional advantage of allowing us to analyze how much of the existing text in the Osmania edition is supported by each of these commentaries.

To compile statistics concerning the percent of text covered, supported, and departed from by the commentaries calls for a consistent unit of enumeration. Traditional accounting of the extent of text in India used the orthographic syllable (अन्तर) as the basic unit. The most accurate modern method would be to use the character. We plan to use characters in the phonetic encoding scheme SLP1. Neither a word nor a reading can accurately serve as such a unit as will become clear shortly; however, tabulating lemmata and calculating the number of characters in each will provide an accurate measure of the extent covered by each commentary.
3.2 Omissions

Omissions are recorded in TEI with an empty rdg tag and optionally supplied with a cause attribute that explains the reason for the deletion of the text. Possible values of the cause attribute suggested by the TEI editors include for example the following:

1. homeoarchy, which indicates the accidental skipping of a line of text because its beginning is similar to preceding or following lines, and
2. haplography, which indicates the inadvertent omission of a repeated written letter or letters.

While such explanations may be relevant for omissions in manuscripts, they are hardly relevant to edited commentaries where presumably editors have corrected such errors. The reason for the absence of a certain word or sentence in any commentary is usually inexplicable. Hence it is not useful to use the cause attribute. To represent the absence of a segment of the base text in the commentary, an empty rdg element is used, for example, as follows:

On A. 1.1.51 उर्गुप्त: while explaining the importance of the first word इ: in the sūtra, the KV. as in the Osmania edition gives two counterexamples, खेयम् and गेयम्य. Both the NY. and PM. witness and explain the first counterexample. The text of the NY. quotes the example and states its derivation as follows: खेयमिति। इ च खन: इति काप्। इकारकान्तादेशः। श्रावणः।।

The PM. says: खेयमिति। इ च खनः।। They then both proceed directly to the derivation of the counterexample to the second word in the sūtra, सौधातकः, skipping any mention or discussion of the word गेयम्य given in the Osmania edition. The fact that both commentaries proceed from the explanation of the first example relevant to the first word in the sūtra directly to discussion relevant to the second word in the sūtra implies that the second counterexample on the first word in the Osmania edition was not in the text of the KV. referred to by the NY. and PM. The omission of the second word by the commentators is represented by an empty rdg element as follows:

```xml
<app loc='A1.1.51'>
<lem>geyam</lem>
<rdg wit='ny pm'/>
```

1 We cite the text of the Osmania editions with sandhi as is but drop quotes and references, and use only the daṇḍa as punctuation.
3.3 Problem in lemmatizing words as examples

It is not always appropriate to select individual words as the unit of a lemma. Frequently sentences are used as examples, particularly where interword phonetic changes are demonstrated or where syntax is relevant. Each such example should be understood as a single unit rather than as a series of individual words. For instance, A. 1.1.12 अद्द्यो मात्/terms प्रगृह्ला a vowel ई or ऋ preceded by a म् in forms of the demonstrative pronoun अद्द्य, thereby preventing by A. 6.1.125 sandhi with a following vowel such as would occur by A. 6.1.77. If A. 1.1.12 did not include the word अद्द्य:, then any vowel ई or ऋ preceded by a म् would be termed प्रगृह्ला and not undergo sandhi. The KV. on this rule shows the importance of the word अद्द्य: by citing two counterexamples: शम्य्यत्र and दाम्य्यत्र. If each of these counterexamples were represented as a sequence of two individual words with sandhi analyzed श-मी ऋत्र, दाम्मी ऋत्र, as currently in the sandhi-analyzed digital edition, the significance of the counterexamples would vanish. Hence, sandhi is restored in these and similar cases, and each such example is treated as a single lemma.

3.4 Problems in lemmatizing altered sequences of examples

The KV. cites two examples on A. 1.1.51: कर्त्ता and हर्ता. The order of the examples is significant in establishing the correct text; hence how that order is attested in both manuscripts and commentaries is pertinent. The NY. quotes these examples in the same order as कर्त्ता हर्ता इति before further explaining each form. If there were a variant that inserted another example between these two examples, then certainly that variant would be post-NY. It would be possible to represent each of these examples in a separate app element and to represent an addition by an app element between them that pairs a rdg element with an empty lem element. Conversely, it would be possible to represent an omission by an app element that pairs a lem element with an empty rdg element. However, such a method is more cumbersome and generally not adopted in critical editing. Hence, where the sequence of examples is an issue showing some variation in the commentaries, the sequence is represented by tagging those examples in a single app element.

\[\text{\texttt{app loc='A1.1.51'}}\]
Similarly, in many cases it is simpler and more comprehensible to annotate variants of a sentence by taking the whole sentence as a single unit rather than its phrases or individual words as units. For example, on A. 1.1.57, the Osmania edition reads तुकि कर्तव्येः न स्थानिविद्ध भवति। and the NY. reads तुकि न स्थानिविद्धवतीति। Since positive readings are reported as well as variants, there are three ways to report this reading. One way is to tag every word and report the absence of the word कर्तव्येः as an omission. The second way would be to tag the phrase तुकि कर्तव्येः, and the third would be to tag the entire sentence. Under either of the first two methods, we still require app elements to represent the support of the manuscripts for the other words or phrase in the sentence. Hence it is simpler to tag the whole sentence as a single unit and to treat the reading available in the NY. as a single variant as follows:

<app loc='A1.1.57'>
<lem>tuki kartavye na sTAnivat Bavati</lem>
<rdg wit='ny'>tuki na sTAnivat Bavati</rdg>
</app>

Moreover, it is often the case that if an edition selects small units such as individual words and represents variants in the form of the omission of those words, the reader requires more effort to understand what the exact reading of the witness is because he has to reconstruct the sentence from fragments. Sanskrit commentators themselves describe such additional effort as prolixity of understanding (प्रतिपत्तिगौरव). Thus, we tag the data on the level of the word, phrase, or sentence according to the demand of the situation. The following are a couple of additional examples of the omission of words handled as variants of phrases or sentences.

Under A. 1.1.47, the Osmania edition reads स्थाने-योग-प्रत्ययपरत्वस्य ऋयम् अपवादः। The PM. omits the word ऋयम् and reads स्थाने-योगप्रत्ययपरत्वस्यापवादः। Instead of representing this omission in three app elements, the first and third taking स्थाने-योगप्रत्ययपरत्वस्य and अपवादः as lemmata with the PM. as witness, and the second with ऋयम् as lemma and an empty rdg element with the PM. as witness, we treat the whole sentence as a single variant and tag it in a lem element under a single app element as follows:

<app loc='A1.1.47'>
<lem>sTAneyogapratyayaparatvasya ayam apavAdaH</lem>
</app>
On A. 1.1.48 the KV. reads: रे ऋतिरि। नौ ऋतिनु। The NY. reads ऋतिरि। ऋतिनु।

According to the Osmania edition, the KV. supplies the examples ऋतिरि and ऋतिनु with the base words रे and नौ of the final constituents of the compounds which undergo replacement of their final vowels with a short vowel by A. 1.2.47. Both the NY. and PM. omit these base words and attest only the examples. This can be represented in three ways: (1) by taking each word individually and representing रे and नौ as omitted, (2) by taking the set of both examples as a single unit and representing the omission of these two base words as one variant, or (3) by the medial course of taking each set of base word plus example as a unit and representing the omission of the base word in each as a variant consisting of just the example. Here we chose the third course and placed each set of base word and example in a lem element under an app element and the reading in a rdg element witnessed by the NY. and PM. as follows:

```xml
<app loc='A1.1.48'>
  <lem>rE atiri</lem>
  <rdg wit='ny pm'>atiri</rdg>
</app>
<app loc='A1.1.48'>
  <lem>nO atinu</lem>
  <rdg wit='ny pm'>atinu</rdg>
</app>
```

### 3.5 Difference in order

On A. 1.1.47 the Osmania edition has three examples: विरूणान्ति, मुञ्जन्ति, and पयांसिः. The NY. has the variant रूणांदि without the preverb वि instead of विरूणान्ति, and places this example last in an order different from that of the Osmania edition: मुञ्जन्ति, पयांसिः, and finally रूणांदि. Two differences are relevant: the change in the order of examples, and a variant for one of them. As above, these differences could be represented as an omission and an addition. However, it is simpler to tag all three words in the KV. in a single lem element under one app element, to treat the reading in the NY. as a single variant, and to record it in a single rdg element.

```xml
<app loc='A1.1.47'>
  <lem>viruRadDi . muYcati . payAMsi</lem>
</app>
```
3.6 Inferring readings from explanations

Jinendrabuddhi and Haradatta often provide explanations that permit one to infer that they had certain readings of the Kāśikār̥tti even though they do not directly cite the reading. For example, the Osmania edition on A. 1.3.63, आम्म्रत्ययवत्कृत्रेण उन्न्योगस्य, cites two examples: ईक्ष्यचक्रे and ईहास्यचक्रे. The NY. comments on these examples as follows:

ईचाश्च के इत्यादि। ईच दर्शने। ईह चेतायाम्। उह वितर्क। लित। इजादे: इत्यादिनां। आम: इति तेलुंक।
ईचाश्च के etc. After the roots ईच् ‘see’, ईह ‘strive’, and उह ‘conjecture’, the affix लित् is introduced (by A. 3.2.115); आम is introduced by A. 3.1.36 इजादेय गुरुमतो उन्न्यः; and the affix लित् is deleted by A. 2.4.81 आम:.

Here the NY. refers to three verbal roots, namely ईच्, ईह, and उह. The Osmania edition gives only two forms which are derived from the roots ईच् and ईह. The citation of the additional verbal root उह in the NY. is relevant to the form उहाश्च which must have been an additional example. Hence the text of the KV. received by the NY. must have had three examples, the third of which the established text in the Osmania edition lacks. We tag such an inferred reading in the same way we tag a direct reading. An addition is tagged conversely to the way an omission is tagged by providing an empty lem-element with an associated reading in a separate app-element (cf. §3.2). Thus the present case is tagged as follows:

```xml
<app loc='A1.3.67'>
  <lem wit='ny pm'>IkzAYcakre</lem>
</app>
<app loc='A1.3.67'>
  <rdg wit='ny'>UhAYcakre</rdg>
</app>
```

Similarly, under A. 1.4.20 अयस्मयादैः छन्दसि, the KV. explains the purpose of the rule in the following words: भएवसम्माधिकारे विधानात् तेन मुखेन साधुल्लमययादैः विधोयते। By means of the inclusion of this rule in the section headed by the terms पद and भ, the fact that the words included in the
list beginning with अयस्मय are correct is provided.’ In this explanation, the Osmania includes the phrase तेन मुखेन. The NY. comments on this sentence as follows: कथं पुनरेषां साधुत्तविधीयतं इत्याह भपदसञ्ज्ञाधिकारे इत्यादि। ह्वारम्। मुखेन। उपाय इत्यनथानैतस्म। ‘In answer to the question, “But how is the validity of these words established?” he says, “By means of the inclusion of this rule in the section headed by the terms पद और भूत etc.” ह्वार ‘door’, मुख ‘mouth’, उपाय ‘means’ — there is no difference in meaning. Because of the fact that the words मुख and उपाय follow the word ह्वार, they may serve to explain the latter. In that case, the word मुख would not be a quotation from the base text. Hence, Jinendrabuddhi’s comment may indicate that the word ह्वार was read instead of the word मुख in the version of the KV. available to him. The sentence in the reading received by Jinendrabuddhi would then have been the following: भपदसञ्ज्ञाधिकारे विधानातेन ह्वारेश साधुत्वमयस्मयाद्विन्यान्यात्विधीयते। The PM. demonstrates that this supposition is correct and that Haradatta received the same reading as Jinendrabuddhi. For Haradatta states यदि सञ्ज्ञा विधीयतं आनन्तर्यात्मसञ्ज्ञाविधानस्य निपातान्य स्थात् ...भपदसञ्ज्ञाधिकारे इत्यादि। ह्वारम्। उपाय। ‘If this rule provided a term, due to the fact that it occurs just after (the provision of the term भूत in A. 1.4.18), mention would be made only of words that occur by the provision of the term भूत. The term पद would not occur, nor would the conjunction of the terms भूत and पद। ...भपदसञ्ज्ञाधिकारे etc. The word ह्वार means उपाय। The PM. explicitly mentions the word ह्वारेश and does not mention the word मुख at all. Instead it explains the word ह्वार by the word उपाय। Hence, the PM. clarifies the statement in the NY. and must be based on the same text that inspired the statement in the NY. Although neither Jinendrabuddhi nor Haradatta refers to the word ह्वारेश directly as a citation by using the word इत्यादि after it, their comments are a direct indication of a variant of the reading in the Osmania edition. We represent this case as follows:

```
<app loc='A1.4.20'>
<lem>muKena</lem>
<rdg wit='ny pm'>dvAreRa</rdg>
</app>
```

The following is another case where Haradatta’s comments imply a variant reading. Under A. 1.4.3 यू स्नास्यू नदै। the KV. explains the word यू in the sūtra as हूँ च ऊ च यू। The PM. quotes this statement in the KV. and further says, क्विचितु विभक्तानमेव पद्यते ‘But in some places the form is read ending in a nominal termination.’ This statement indicates that the nominative dual
form चिकिर्जव was read in some manuscript available to Haradatta. We represent this inferred reading in the apparatus as follows:

```
<app loc='A1.4.3'>
  <lem wit='ny pm'>I ca U ca yU</lem>
  <rdg wit='pm' type='kvacit'>yvO</rdg>
</app>
```

### 3.7 Mistakes in the editions of the commentaries

Unfortunately the editions of the NY. and PM. include mistakes. We have discovered errors of mistaken sandhi analysis, mistaken sentence division, and mistaken quotation in our work so far. The following are three examples.

On A. 1.1.39, there is a set of counterexamples: आध्येव, चिकिर्जवे, and कुम्भकरिण्य:. The Osmania edition of the NY. reads चिकिर्जव: इति। At first glance, it seems that this is a variant for चिकिर्जवे. चिकिर्जवे is the dative singular of the nominal base चिकिर्जु, and चिकिर्जव: is the nominative plural. The description of the form in the NY. is of the dative singular. The NY. explicitly states that the form is a dative singular of the nominal base चिकिर्जु, formed by applying the fourth-triplet nominal termination के and the गुण replacement of the final vowel ओ by A. 7.3.111 चेततिति। Thus the nominative plural form चिकिर्जव: does not fit the description given by the NY., and the correct form is चिकिर्जवे. Hence there is no variant for the word चिकिर्जवे in KV. in the text of the NY.

How did the erroneous word चिकिर्जव: come to be found in the edition of the NY.? The editors of the Osmania editions often analyze sandhi in an attempt to be helpful to readers. Their original manuscripts must all have read चिकिर्जव: इति with regular sandhi. In the Osmania edition of the NY., the editors regularly analyze sandhi of examples and quotations followed by इति and place them in quotation marks. The sandhi of चिकिर्जव: इति can be analyzed in two ways: चिकिर्जवे इति and चिकिर्जव: इति। Thus wrong sandhi dissolution created what appears to be a variant in the NY. when in fact the text has no such variant. On the basis of internal evidence, we infer the correct reading and report it as follows:

```
<app loc='A1.1.39'>
  <lem wit='ny pm'>cikIrzave</lem>
</app>
```

The same sandhi error is made by the editors of the Osmania edition of the NY. on A. 1.1.67. The Osmania edition of the KV. states तस्मात् इति
Enriching the digital edition of the *Kāśikāvatī* 289

Here the Sanskrit library sandhi-analyzed text reads पञ्जुम्यर्थरनदेश in the locative. The Osmania edition of the *NY.* states तस्मात् इति। पञ्जुम्यर्थरनदेश: इति। First of all there should not be a full-stop after इति in the phrase तस्मात् इति. Moreover the sandhi-dissolution पञ्जुम्यर्थरनदेश: इति is wrong. As in the preceding example, the proper dissolution is पञ्जुम्यर्थरनदेश as in the Sanskrit Library’s sandhi-analyzed text. Hence we do not report this case as a variant, but take it as support for the text of the *KV.* as analyzed in the Sanskrit Library edition and report it as follows:

```xml
<app loc='A1.1.67'>
  <lem wit='ny pm'>tasmAt iti paYcamyarTanirdeSe</lem>
</app>
```

On A. 1.1.56 the Osmania edition of the *NY.* includes an erroneous sentence break and erroneous indication of a quotation of the base text. The Osmania edition of the *KV.* reads न ओलिधिरनलिधि: इत्यथः। In the Osmania edition of the *NY.*, Jinendrabuddhis’ explanation of the compound ओलिधिरनलिधि is edited as follows: ष वुनः समासो मयुर्यंकादिवात समासं कृत्वा नञ्जसमास: कृतः। न ओलिधिरनलिधि: इति। The editors of the *NY.* put a daṇḍa after कृतः and put single quotes around न ओलिधिरनलिधि: to indicate that it is a quotation from the *KV.* This is a mistake. Careful reading of the text indicates that the दशड should be removed and the passage ending with इति read as a single sentence as follows: ष वुनः समासो मयुर्यंकादिवात समासं कृत्वा नञ्जसमास: कृतो ओलिधिरनलिधिरिति। “But that compound, formed because it is included in the list beginning with मयुर्यंक, is formed as a negative tatpurusa compound (नञ्जसमास: न ‘not’ ओलिधि ‘a phonetic operation’ = ओलिधि.” The cited phrase न ओलिधिरनलिधि: is not a citation to the *KV.*; it does not refer to the base text. It is a typical compound analysis of a nañtatpurusa compound. Such an analysis may have been made originally by a commentator on the *KV.*, even by Jinendrabuddhi himself, rather than by the authors of the *KV.*. Hence without independent support from manuscripts, it should not be adopted in the text of the *KV.* on the basis of the explanation provided in the *NY.*. However, since the editors of the Osmania edition of the *KV.* have adopted the sentence न ओलिधिरनलिधि: इत्यथः। in their base text, the editors of the Osmania edition of the *NY.* marked न ओलिधि: ओलिधि: as a quotation from the base text. Unfortunately this is misleading. If it were a quotation from the base text it would have included the closing words इत्यथः। We do not accept that the
text of the *NY.* supports the reading न अतिविधिनलिविधि: इत्यर्थः in the *KV.* and hence refrain from including it in our critical apparatus.

### 3.8 Discrepancies in quotations within different sections in the same commentary

There are many occasions where the commentary on the *KV.* on one sūtra cites text from the *KV.* on another sūtra. Both the *NY.* and *PM.* do this. We mark these cases as support or variants of the text they cite just as we do citations to the base text in commentaries on the cited base text under the same sūtra. If the citation does not differ from commentary on the base text on the same sūtra, we make no addition. However, if the citation constitutes a variant that differs from one under the base text on the same sūtra, or support for the reading of the base that received no support from the commentary on the base text on the same sūtra, we add an additional *rdg* element containing the new reading with a *source* attribute indicating the sūtra under which that reading was found.

For example, on *A.* 2.3.19, the Osmania edition of the *KV.* reads पितुत्र क्रियादिसम्बन्धः शब्दनोच्यते।पुत्रस्य तु प्रतीयमान इति तस्याप्राणान्यम। On *A.* 1.1.56, the *NY.* quotes the text exactly as given in the *KV.* on *A.* 2.3.19. However, while commenting on *A.* 2.3.19, instead of पुत्रस्य तु प्रतीयमान इति तस्याप्राणान्यम, the *NY.* quotes पुत्रस्य तु प्रतीयमानतवाप्राणान्यम, adding the affix त्वात् and omitting इति तस्य. Thus the *NY.* gives two different readings for the same base text at two different places. We report both of these readings as follows:

```xml
<app loc='A2.3.19'>
  <lem wit='ny' source='A1.1.56'>pituH atra kriyAdisambanDaH Sabdena ucyate. putrasya tu pratIyamAnaH iti tasya aprADAnyam</lem>
  <rdg wit='ny'>putrasya tu pratIyamAnatvAt aprADAnyam</rdg>
</app>
```

### 4 Sample results

Below we report the results of 578 readings gleaned from the our tagged data of the third quarter of the first chapter of the *Aṣṭādhyāyī* (*A.* 1.3). Indicated is the number of times the commentators agree with or differ from the base text, agree with or differ from each other, report, approve of or disapprove of variants.
1. Only the NY. agrees with the base text: 227
2. Only the PM. agrees with the base text: 131
3. The NY. and the PM. share the same reading which agrees with the base text: 155
4. Only the NY. differs from the base text: 24
5. Only the PM. differs from the base text: 23
6. The NY. and the PM. share the same reading which differs from the base text: 9
7. The NY. and the PM. each mention a reading which differs from the reading of the other: 6
8. The PM. is aware of variants: 9
9. The PM. received a different reading for which it suggests a better option: 1

Ten percent (10%) of the readings gleaned from the commentators in A. 1.3 support a change in the base text of the KV. The project of collecting readings from commentators, therefore, promises to contribute significantly to the establishment of a more correct text of the KV.

5 Conclusion

The issues discussed demonstrate the depth of understanding required to determine what each commentator must have read and the care required to represent that information accurately. The method of preparing a critical apparatus of readings of the KV. attested in the NY. and PM. described above provides a reliable and well-structured database of valuable information about the text of the KV. and its historical transmission that is both human and machine-readable. This database will serve as a valuable resource for producing a critical edition of the KV. The results of this project will also reveal the textual history of the KV. between when Jinendrabuddhi wrote his commentary in the eighth or ninth century, Haradatta wrote his in the thirteenth century and the more recent dates of the extant manuscripts of the text. The database will permit one to determine systematically how
much of the text of the *KV.* was known to each of the commentators. It will reveal how many variations occurred in the transmission of the text and how many readings have been lost to us in the course of time. The methods used in this project are applicable to similar philological work to prepare an edition and determine the textual history of any Sanskrit text with commentaries or indeed of any commented text extant in the form of manuscripts.
References


Sharma, Aryendra, Khanderao Deshpande, and D. G. Padhye, eds. 1969–1970. Kāśikā. a commentary on Pāṇini’s grammar by Vāmana & Ja-
From the Web to the desktop: IIIF-Pack, a document format for manuscripts using Linked Data standards

TIMOTHY BELLEFLEUR

Abstract: This paper describes the implementation of a document file format for the representation of the composite image, text, and additional data, focusing on the use case of manuscripts. The organizational methodology follows emerging standards for Linked Data, as well as some standards already in use by scholars and projects in Sanskrit Digital Humanities. It also presents a model for scholars in need of organizing this relevant data to begin to do so in a manner that facilitates future transition into online spaces.

1 Introduction

Textual scholars face a number of practical challenges when organizing and working with their materials in the digital space. While common, established formats exist to serve the needs of the individual artifacts of the process (principally images and text), few adequately provide for the compilation of these pieces, and fewer still for connections between them. As a result, scholars typically must make do with collections of related files, deficient formats, and even ad hoc strategies for organizing and referencing their data. These issues can be particularly severe for textual projects in Indology, which often deal with large amounts of data. Solutions to these sorts of challenges have been developed, but are targeted towards the Internet, where interconnected networks of disparate data resources are commonplace and the necessity of navigating through them requires both robust and extensible standards. However, as theoretically ideal as working in a networked online space might be, practical concerns of access, convenience, distribution, and
portability dictate that we still often require offline documents, even if those documents are of a particular composite variety.

In this paper, I describe the implementation of a document file format for the representation of the structured, composite image, text, and extensible additional data, focusing on the use-case of manuscripts. The organizational methodology follows emerging standards for Linked Data, employing the International Image Interoperability Framework (Appleby et al. 2017b), JSON-LD (Sporny, Kellogg, and Lanthaler 2014), and Web Annotation Data Model (Sanderson, Ciccarese, and Young 2017) standards as well as their related ontologies for data description and linking. It also incorporates some standards already in use by scholars and projects in Sanskrit Digital Humanities, such as the XML-based Text Encoding Initiative Guidelines (TEI Consortium 2017) for textual representation.

The overall objectives of this project are two-fold. First, it presents an offline-friendly document format that can serve in many cases as a replacement for composite document containers such as Adobe Corporations’ Portable Document Format (PDF) while being immediately more extensible for the inclusion of textual and other related data as well as interconnections between the document components. Second, it presents a model for scholars in need of organizing this relevant data to begin to do so in a manner that facilitates future transition into online spaces with as little friction as possible.

To provide a specific use case for this project: In Dr. Adheesh Sathaye’s ongoing digital critical editing work on the medieval Sanskrit story collection *Vetālapaṅcaviṃśati*, the primary dataset includes scans or photographs of some 90 manuscripts in several different Indic scripts along with electronic text transcriptions using IAST romanization. Thus far, the most convenient solution for organizing and storing these resources has been to compile the manuscript images into PDF files, to keep each transcription as a separate file, and to annotate each folio in the transcription to its corresponding PDF page number. While this method is simple enough to implement and requires no special software besides Adobe Acrobat, it is not ideal for a number of reasons. Links between text and images are defined, but there is no easy way to navigate from one to the other; despite being representations of the same text (facsimile image and electronic text), the files for each representation of a manuscript are essentially isolated from each other. The PDF format—however ubiquitous and relatively efficient it functions as a simple multi-image file format—is difficult to extend, parse, and embed.
into other programs, as well as relying on commercial software for optimal creation and editing. As retrieval of specific folio images, linking textual transcription data more directly with these images and navigating these links becomes necessary, it is likely that the project will need to abandon its current PDF-based scheme. Adopting existing standards for Linked Data serves the project’s goal of describing its data and relationships between that data in a well-defined, extensible way, where it will come to include editorial secondary data in an eventual online space. However, even in their simplest useful implementations, these standards currently do not provide comparable simplicity or convenience to that of a single, editable document file. Furthermore, implementing an online-only method of storage, retrieval, and editing enforces its own complexities and restrictions on interacting with the data, especially since the primary data objects can be conceived of simply as discrete documents of text and image data. By adapting Linked Data models to the extent necessary for use in an offline idiom, we can maintain and even increase ease of use for this project (and other document-centric projects) while simultaneously employing the same methodologies used in online spaces.

2 Background

The term Linked Data comes from the World Wide Web Consortium’s Semantic Web project and has come to encompass a variety of standards for identifying, describing, representing, and exchanging structured data and the relationships between this data using standard Web technologies. Linked Data formats follow the model of the Resource Description Framework (RDF), which represents data in a graph of resources (nodes or vertices) and the relationships (or edges) between them. A basic premise of Linked Data is that every node should possess a unique URI (Uniform Resource Identifier, like a web address), which may be used to retrieve it or information about it or be used as an unambiguous reference to the resource by other objects. Description of the information in these resources is standardized by the use of a variety of different controlled vocabularies (widely referred to as “ontologies”) which each define a set of terms and the interpretation of their values.1 A number of syntax notations exist for serializing

---

1 Among the most widespread of these ontologies is the Dublin Core Metadata Initiative Terms for general metadata (DCMI Usage Board 2012, also used widely in digital libraries
these RDF graphs in existing common formats used in data interchange such as XML (via RDF/XML) and JSON (via JSON-LD). Employing this model, further standards have been developed for the representation of specific complex data structures in a well-defined manner. Among these, the International Image Interoperability Framework and the Web Annotations Data Model are central to the design of this project.

The International Image Interoperability Framework (IIIF) is a comprehensive Linked Data standard providing a core model for representing image collections in the JSON-LD format (the Presentation API), as well as a set of additional APIs for dealing with the querying and presentation of these resources. IIIF resources may be described in terms of their often-multiple structural and logical arrangements and divisions and extensively linked with related resources such as text transcriptions and annotations. These linkages are facilitated through the use of the World Wide Web Consortium’s Web Annotations Data Model, which defines a schema for creating detailed metadata about and relationships between resources, as well as specifying a flexible set of methods for targeting specific portions of a given resource relevant to the annotation or relationships involved.

In the IIIF’s core model (see Figure 1), a document consists of a series of virtual canvases onto which content such as images are associated by means of annotations (using the Web Annotation Data Model). These canvases are organized into one or more sequences that provide a logical ordering. Although each of these component resources has its own unique identifier, a set of required structural resources is defined within a single manifest resource which also contains overall metadata for the document. Additional resource types such as collections (groupings of manifests), ranges (groupings of canvases), annotation lists (collections of annotation data), and layers (groupings of annotation lists) provide logical structures for organizing associated data. In some cases, these additional resources may be defined directly in the manifest. However, most resources representing data beyond the basic structure of a document are defined externally and their identifiers referenced in the manifest so that they may be retrieved as necessary. The most ubiquitous of these external resources are annotation lists, which contain all annotations and resource linkages for canvases besides the basic associated images.

and archival studies), the Simple Knowledge Organization System for classification and taxonomy (Miles and Bechhofer 2009), and the Friend of a Friend ontology for describing persons and agents (Brickley and Miller 2014).
From the Web to the desktop

Figure 1

IIIF resource hierarchy (Appleby et al. 2017b).
The IIIF model satisfies all the basic needs of a structured, image-based document, much like Adobe’s Portable Document Format but in a more easily parseable and extensible way. For composite documents like manuscripts, which may not only have multiple divisions and organizational structures but also associated transcription data, the advantages of the IIIF become more pronounced. However, since it is designed for use online Linked Data services and to be efficient for large quantities of variable annotations and metadata, the IIIF specification only permits a limited number of resources to be fully described within its manifest and does not specify a method of packaging additional associated resources together. This leads to a significant quantity of individual resource files within an IIIF document’s structure. For an online service, where these resources can be stored in a database or as a complex file structure that manages user queries, this poses a minimal challenge to accessibility in most cases. However, if a user wishes to retrieve a document’s entire collection of resources all at once, a solution must be devised. It is a primary aim of this paper to propose such a method that bridges the divide between complex online structure and simple document files while maintaining the benefits of the IIIF model.

3 IIIF-Pack Format Structure

The format proposed here, provisionally named IIIF-Pack, provides a method for packaging an IIIF resource into a single file along with its related parts and optional external resources. Taking inspiration from the Open-Document (Durusau and Brauer 2011) and Microsoft Office Open XML (ISO/IEC 2012) formats, the separate resources are compiled together under a well-defined file and folder structure using the ubiquitous Zip archive format originally developed by PKWARE and implemented widely in open-source libraries such as Zlib (Gailly and Adler 2017). This file archive strategy solves the issue of compiling numerous files as well as providing fast, adequate compression for text-based resources and an index through which individual files may be efficiently extracted, appended, or removed from the archive container. While the internal path structure of the Zip format does not rely on the order of files it contains—this is managed by the central directory section at the end of the file—it is prudent that the IIIF-Pack format prioritizes storing static resources at the beginning of the archive to facilitate performant modification of its contents.
Rather than identifying individual resources with URIs according to Internet-style addresses, identifiers for resources within the IIIF-Pack file are defined as absolute pathnames, with the root of the archive acting as the root of the virtual path. IIIF resources within this structure follow the recommended nomenclature for given resource types (see Figure 2) with the leading `{scheme}://{host}/{prefix}/{identifier}` segments omitted. All resources types except “content” are assumed to be in the JSON-LD graph description format. Whenever one IIIF resource is defined within another, such as in the manifest, these are loaded into the resource graph along with their parent and become directly accessible by their identifier name. Where a resource may be requested by its identifier but not already present within the loaded graph, the local file extension `.json` is assumed by the parser and the file is dereferenced from the archive. Accordingly, the document manifest is stored in the root of the archive as `manifest.json` and assigned the URI `/manifest`.

By directly adopting the IIIF’s organizational model, IIIF-Pack benefits from more than simply a standardized method of description designed for complex document structures. It also facilitates easy translation between both online and offline spaces for projects that may want to transition at some future point. In that case, all that is required is to extract the files and add the appropriate `{scheme}://{host}/{prefix}/{identifier}` to each resource’s identifier. This process also functions in reverse, should a project wish to package an existing IIIF resource into a single document file. Furthermore, it remains closely compatible with software developed to work with the IIIF standards.

For the initial use case of IIIF-Pack as a document format for manuscripts, the two principal types of content resources are images and text transcriptions. In virtually all cases, images make up the largest bulk of the data in a document, and as such, efficient image compression strategies are critical. While the IIIF Presentation API does not specify a restricted set of image formats to be associated with its canvases, the IIIF Image API (Appleby et al. 2017a) identifies seven commonly-supported formats which

---

2Implementing IIIF collections in a single package is certainly possible as well, though not explored here. In this case, the IIIF document “`identifier`” segment of the path would need to be defined and retained.

3The IIIF considers “Content” type resources to be any internal resources that are not IIIF structural resources or annotations. In practice this usually means images, but may include additional associated data.
users may want to extract from a document: JPEG, TIFF, PNG, GIF, JPEG2000, PDF, and WebP. The inclusion of PDF as an image format in this list is notable not only for its wide support but also because the PDF standard supports a variety of highly-efficient algorithms for compressing bi-level (black and white) images, in particular the JBIG2 standard (Ono et al. 2000). JBIG2 provides the highest lossless compression of bi-level images currently available by employing symbol recognition and arithmetic encoding and is thus invaluable in efficient storage of manuscript images, many of which may be acquired in black and white as photocopies or print-outs of microfilm. Individual JBIG2 images may be stored in their own file containers with a minor increase in efficiency over single-image PDFs, however, this method is less well-supported, while all major PDF libraries support decoding the format.

For textual data resources, the prevailing schema in use in Sanskrit Digital Humanities is the Text Encoding Initiative’s TEI Guidelines (TEI Consortium 2017) and related standards such as EpiDoc (Elliott et al. 2017). The SARIT project (Wujastyk et al. 2017) is one of the leading contribu-

---

**Figure 2**

*IIIF recommended URI patterns (Appleby et al. 2017b).*

<table>
<thead>
<tr>
<th>Resource</th>
<th>URI Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td><code>{scheme}://(host)/(prefix)/collection/{name}</code></td>
</tr>
<tr>
<td>Manifest</td>
<td><code>{scheme}://(host)/(prefix)/(identifier)/manifest</code></td>
</tr>
<tr>
<td>Sequence</td>
<td><code>{scheme}://(host)/(prefix)/(identifier)/sequence/{name}</code></td>
</tr>
<tr>
<td>Canvas</td>
<td><code>{scheme}://(host)/(prefix)/(identifier)/canvas/{name}</code></td>
</tr>
<tr>
<td>Annotation</td>
<td><code>{scheme}://(host)/(prefix)/(identifier)/annotation/{name}</code></td>
</tr>
<tr>
<td>AnnotationList</td>
<td><code>{scheme}://(host)/(prefix)/(identifier)/list/{name}</code></td>
</tr>
<tr>
<td>Range</td>
<td><code>{scheme}://(host)/(prefix)/(identifier)/range/{name}</code></td>
</tr>
<tr>
<td>Layer</td>
<td><code>{scheme}://(host)/(prefix)/(identifier)/layer/{name}</code></td>
</tr>
<tr>
<td>Content</td>
<td><code>{scheme}://(host)/(prefix)/(identifier)/res/{name}.{format}</code></td>
</tr>
</tbody>
</table>
tors to the adoption of this format in digital Indology today. These XML-based standards define the representation of tagged, annotated textual content, structure, and emendation in a variety of flexible ways. Associating a given part of an XML document to an IIIF canvas is effectively accomplished using the Web Annotations Data Model’s selector system, which supports identifier-based selection through the use of XPath (Robie, Dyck, and Spiegel 2017), CSS-style selectors (Çelik et al. 2011), and the XPointer Framework (Grosso et al. 2003), for which the TEI standard has contributed several registered schemes.

4 IIIF-Pack Example Document

The figures below describe a simple example IIIF-Pack file structure and its components, using the image and textual data for a single manuscript from the aforementioned *Vetālapaṇcaviṁśati* digital critical edition project. The manuscript data includes 217 folio images extracted from a PDF document into individual JBIG2-compressed files along with a single-file TEI XML transcription of the text. Figure 3 illustrates the file and folder structure inside the archive. Figure 4 illustrates the opening portion of the IIIF manifest file, including the full definition of the first canvas resource in the document. Figure 5 shows the annotation list for a single canvas, containing the association of transcription data with the manuscript folio it represents using XPointer selectors on the associated TEI document. Finally, Figure 6 contains the beginning of the definition file for the transcription layer resource, which groups together all of the individual annotation lists providing granular links to each folio’s transcription. Although it is omitted from Figure 4, the manifest resource is also directly associated with an annotation list identifying the full TEI document as a transcription of the full IIIF resource. The resulting IIIF-Pack file is 20% smaller in size (9.3MB) than the image source PDF (11.8MB), despite including IIIF structural and annotation data as well as a text transcription.
Figure 3
Example IIIF-Pack internal file layout.

Figure 4
Beginning of manifest file for example document.
Figure 5
Annotation list for canvas of folio 1 (verso) in example document.

Figure 6
Beginning of transcription layer definition in example document.
5 Software Support

As with many Linked Data standards, general-use software tools for IIIF resources are fairly sparse. The most mature of these tools is Mirador, an open-source, web-based image and annotation viewer (Project Mirador Contributors 2017) designed for digital libraries and museums. The rerum.io project, based out of the Center for Digital Humanities at St. Louis University, has begun to develop online tools and services for facilitating user onboarding as part of their early adoption of the IIIF model (Cuba, Hegarty, and Haberberger 2017). The most noteworthy of these is a tool for generating IIIF manifests and other Presentation API structures from a list of images. Two actively maintained web server frameworks exist for serving images according to the IIIF Image API specification, providing support for retrieving specific canvases or regions of IIIF resources in multiple formats. Overall, the state of the software suggests that while the IIIF standards have gained traction, there is still significant work to be done to broaden their scope of use. For the IIIF-Pack format, I have developed rudimentary parsing and viewing components in order to test its viability as a proof-of-concept. These include support for a separate JBIG2-format image decoder to eliminate the overhead of using single-page PDF files where JBIG2 compression is desired.

The development of two tools is critical to the success of this project: First, a cross-platform IIIF-Pack file viewer that can display the canvases along with their associated textual data according to the document’s defined sequences and annotations (see mock-up in Figure 7). I am examining employing a subset of Mirador as a basis for this application using the cross-platform Electron framework (GitHub Inc. et al. 2017) employed widely in desktop software built on web technologies. The second tool simplifies the workflow of producing a complete IIIF resource structure from source images and transcriptions and packaging them into an IIIF-Pack file. Following these, a library that facilitates performant, in-place editing of the IIIF-Pack file’s resources will provide the full suite of functionality for the document format. I expect these tools to begin public-facing testing by the first half of 2018 and plan to demonstrate them at the 17th World Sanskrit Conference in July.
Figure 7
Side-by-side view of manuscript folio image and its TEI-XML transcription.
6 Conclusion

While this project is in an early stage, I hope that its utility is already apparent. Textual scholars need not abandon the convenience of offline documents to benefit from advances in structuring and connecting their data. Given the promise of emerging new standards and the unfortunate state of existing document formats in meeting a variety of needs, some way of bringing current advances to bear on the practical needs of scholars is sorely needed. Furthermore, the broad extensibility of these standards leaves many additional possibilities left unexplored here—for example, the inclusion of digital stemmatic and other philological data, or metadata integration with growing online databases such as PANDiT (Bronner et al. 2017). By adopting Linked Data standards designed for the web, we may benefit from their manifest organizational proficiencies and future developments without requiring either that textual work happen natively online or that a significant reconception of the idea of a document takes place. In doing so, we can also prepare our work for the ongoing movement into interconnected online spaces in a less onerous, more effective way.
References


New Vistas to study Bhartṛhari: Cognitive NLP

Jayashree Aanand Gajjam, Diptesh Kanojia and Malhar Kulkarni

Abstract: A sentence is an important notion in the Indian grammatical tradition. The collection of the definitions of a sentence can be found in the text Vākyapadiya written by Bhartṛhari in fifth century C.E. The grammarian-philosopher Bhartṛhari and his authoritative work Vākyapadiya have been a matter of study for modern scholars, at least for more than 50 years, since Ashok Aklujkar submitted his Ph.D. dissertation at Harvard University. The notions of a sentence and a word as a meaningful linguistic unit in the language have been a subject matter for the discussion in many works that followed later on. While some scholars have applied philological techniques to critically establish the text of the works of Bhartṛhari, some others have devoted themselves to exploring philosophical insights from them. Some others have studied his works from the point of view of modern linguistics, and psychology. Few others have tried to justify the views by logical discussions.

In this paper, we present a fresh view to study Bhartṛhari, and his works, especially the Vākyapadiya. This view is from the field of Natural Language Processing (NLP), more specifically, what is called Cognitive NLP. We have studied the definitions of a sentence given by Bhartṛhari at the beginning of the second chapter of Vākyapadiya. We have researched one of these definitions by conducting an experiment and following the methodology of silent-reading of Sanskrit paragraphs. We collect the Gaze-behavior data of participants and analyze it to understand the underlying comprehension procedure in the human mind and present our results. We evaluate the statistical significance of our results using the T-test and discuss the caveats of our work. We also present some general remarks on this experiment and the usefulness of this method for gaining more insights into the work of Bhartṛhari.
1 Introduction

Language is an integral part of the human communication process. It is made up of structures. There are sentences, which are made up of words, which in turn are made up of syllables. There has been a lot of discussion about which among these is a minimal meaningful unit in the language. The notions of a sentence and a word have been described in different fields of knowledge such as grammar, linguistics, philosophy, cognitive science, etc. Some provide a formal definition of a sentence, while others give the semantic definition. The Vyākaraṇa, Mīmāṃsā and Nyāya schools of thought in Sanskrit literature hold some views about the nature of a sentence. The grammarian-philosopher Bhartṛhari enumerated eight definitions of a sentence given by early grammarians and Mīmāṃsakas in the second Kāṇḍa (Canto) of his authoritative work Vākyapadīya.

The question that how does a human being understand a sentence has been dealt with in the field of psycholinguistics for the last 20 years. Various studies conducted in the last decade have addressed this question by using several experimental methods. There are many off-line tasks\(^1\) such as Grammaticality Judgement task, Thematic Role Assignment task, etc. which are helpful in examining how the language-users process the complete sentences. In addition to these off-line techniques, psycho-linguists have investigated a number of sophisticated on-line language comprehension methodologies. Some of them are behavioral methods such as Acceptability Judgement, Speed-Accuracy Trade-off, Eye-Movement Behavior, Self-Paced Reading, etc. Some are neuro-cognitive methods such as electroencephalogram (EEG),\(^2\) Event-Related brain Potentials (ERPs),\(^3\) functional Magnetic Resonance Imaging (fMRI),\(^4\) Positron Emission Tomography (PET)\(^5\) etc.

---

\(^1\)These methodologies are called as ‘off-line’ because they study the comprehension process after the participant performs the task, most of which are the pen-paper methods.

\(^2\)EEGs measure the electrical activities of the brain while performing a task by applying electrode/s to the scalp.

\(^3\)ERPs provide a very high temporal resolution. The spontaneous electrical activity of the brain is measured non-invasively by means of electrodes applied to the scalp (Choudhary 2011).

\(^4\)fMRIs are BOLD (Blood Oxygen Level Dependent) techniques and used while studying both neurologically healthy adults and people with reading disabilities, mostly the brain-damaged patients.

\(^5\)PETs are the neuroimaging techniques which are based on the assumptions that areas of high radioactivity are correlated with the brain activities.
which study the ongoing or real-time cognitive procedure while a participant performs a task.

This paper addresses one of the eight definitions given by Bhartṛhari. The main goal is to study this definition from the cognitive point of view i.e. to study the underlying comprehension procedure in human beings taking this definition as the foundation. It also allows us to find the cases of the linguistic behavior of the readers in which this definition holds true. We use an Eye Tracker device to collect the Gaze (Eye) Movement data of readers during the procedure of silent reading\(^6\) of Sanskrit paragraphs.

**Gaze Tracking: An Introduction**

Gaze tracking is the process of measuring a gaze point or the movement of the participants’ eyes. The device which measures the eye-movements is called as Eye-Tracker. We use an ‘SR-Research Eyelink-1000 Plus’\(^7\) which mainly comprises of two PCs (Host PC and Display PC), a camera and an infrared illuminator. It performs the monocular eye-tracking with a sampling rate of 500Hz (one sample/2 millisecond). The Host PC is used by the supervisor for navigating through the experiment. A supervisor can set up the camera, perform the eye-calibration process, check and correct the drifts, present the paragraphs to the readers, and record the session on the Host PC. Similarly, Display PC is used by the reader for reading the paragraphs and answering the questions. The pupil of the participant is captured by the camera and the eye-movements are captured by the infrared illuminator. These eye-movements are mapped to the data that is presented to the participant on the Display PC with the help of some image processing algorithms.

Eye-Tracker records several eye-movement parameters on the Area of Interest (AOI) such as *Pupil size*, *Fixations* and *Saccades*. An AOI is an area of the display that is *of the concern*, like a word or a sentence or a paragraph, which in our case is a word. A *Fixation* is when the gaze is focused

---

\(^6\)The oral and silent reading represents the same cognitive process. However, readers decrease processing time on difficult words in silent as compared to oral reading. (Juel and Holmes 1981). For the current paper, we focus on the silent-reading methodology of the paragraphs.

\(^7\)More information can be found at the link: [http://www.sr-research.com](http://www.sr-research.com)
on a particular interest area for 100-500 milliseconds. A *Saccade*\(^8\) is the movement of gaze between two fixations which occurs at an interval of 150-175 milliseconds.\(^9\) Specifically, due to its high sampling rate, Eye-Tracker is also able to capture *Saccadic-Regressions* and similarly *Progressions*. A *Regression* a.k.a *Back-tracking* is a backward-moving saccadic movement in which the reader looks back to something that they had read earlier. On the contrary, a *Progression* is a forward-moving saccadic path.

The availability of embedded inexpensive eye-trackers on hand-held devices has come close to reality now. This opens avenues to get eye-tracking data from inexpensive mobile devices from a huge population of online readers non-intrusively, and derive cognitive features. For instance, *Cogisen*: has a patent (ID: EP2833308-A1)\(^{10}\) on eye-tracking using an inexpensive mobile webcam.

Till date, there has been lots of research which have been carried out using eye movement data on various tasks such as reading (texts, poetry, musical notes, numerals), typing, scene perception, face perception, mathematics, physics, analogies, arithmetic problem-solving and various other dynamic situations (driving, basketball foul shooting, golf putting, table tennis, baseball, gymnastics, walking on uneven terrain, mental rotation, interacting with the computer screens, video game playing, etc.) and media communication (Lai et al. 2013) etc. *Reading researchers* have applied eye-tracking for behavioral studies as surveyed by Rayner (1998). Recently, some researchers have even used this technique to explore learning processes in complex learning contexts such as emergent literacy, multimedia learning, and science problem-solving strategies.

In Section 2, we discuss the related work in the fields of Sanskrit grammatical tradition and cognitive NLP. In the next Section 3, we present our approach which focuses on the experimentation details and we present the analysis and results in Section 4. Section 5 gives the evaluation of our work.

---

\(^8\) The word ‘Saccade’ is a French-origin word. It was Luis Émile Javal (French eye specialist and a politician) who named the movement of the eyes as ‘Saccades’ for the first time in 19th C.

\(^9\) As far as human anatomy is concerned, eyes are never still; there are small movements/tremors of the eyes all the time. They are called as ‘Nystagmus’ (Rayner 1998). These eye movements are involuntary and hence not measured by the machine. The movements of the eyes which are deliberate, occur at the interval of 150-175 ms and they are considered as the features for the analysis.

\(^{10}\) [http://www.sencogi.com](http://www.sencogi.com)
which is followed by the Section 6 on discussion. We conclude this paper in Section 7 by suggesting possible future work.

2 Related Work

In this section, we discuss the work that has been done on the notions of sentence and sentence-meaning by Indian and Western scholars in subsection 2.1. The studies that have been carried out in the fields of Cognitive NLP are presented in subsection 2.2. We also present a bird’s eye view of our research area in the figure at the end of this section.

2.1 Sentence Definitions and Comprehension

Sanskrit grammatical tradition is started with Pāṇini’s Ashtadhyayi. Pāṇini in his work doesn’t define a sentence explicitly. However, few modern scholars attribute a sentence as the base of the derivational process in Pāṇini’s grammar (Kiparsky and Staal 1969). This view is criticized by Houben (2008) and SD Joshi and Roodbergen (2008). According to some scholars, the notion of Kāraka (Huet 2006) or the notion of Sāmarthya (Deshpande 1987; Devasthali 1974) are Pāṇini’s contribution to the syntax. The latter view is opposed by Mahavir (1984). After Pāṇini, Kātyāyana who wrote Vārttikas on the rules of Aṣṭādhyāyī gave two definitions of the sentence\(^{11}\) (P.2.1.1 Vt.9) (A sentence is chiefly the action-word, accompanied by the particle, nominal words, and adjectives) and ekatiṅ vākyāṃ (P.2.1.1 Vt.10) (a sentence is that [cluster of words] containing a finite verb [as an element]). for the first time, which are said to be formal in their nature and not referring to the meaning content (Laddu 1980; Matilal 1966; Pillai 1971). Deshpande (1987) argued that Kātyāyana’s claim that each sentence must have a finite verb relates to the deeper derivational level and not to its surface expressions. Hence, a sentence may or may not contain a finite verb on the surface level and there can be a purely nominal sentence (Bronkhorst 1990; H. Coward 1976; Tiwari 1997). Patañjali in his Mahābhāṣya discussed the integrity of a sentence in terms of having only one finite verb. According to him, a sentence must have only one finite verb, and also purely nominal sentences may not be considered as complete. The word asti (is) should be understood in those sentences (Bronkhorst 1990). Modern scholars dis-

\(^{11}\)ākhyātaṃ sāvyayakārakaviśeṣaṇaṃ vākyāṃ
cussed that a sentence having two identical finite verbs\textsuperscript{12} doesn’t militate against the integrity of a sentence (Deshpande 1987; Jha 1980; Laddu 1980; Pillai 1971).

\textbf{Bhartṛhari}, for the first time, deals with the semantic issues in the second \textit{Kāṇḍa} i.e Vākyakāṇḍa of \textit{Vākyapadīya} (VP). We can find a comprehensive treatment on various theories of sentence and their meanings along with their philosophical discussions. He enumerates eight views on the notion of a sentence which are held by earlier theorists in India. The verse is:

\begin{quote}
\begin{verbatim}
Ākhyātaśabdaḥ saṅghāto jātiḥ saṅghātavartinī
eko‘navayah śabdaḥ kramo buddhyanusamḥṛtīḥ |
padamādyam prthaksarvam padam sākāṇkṣamanyapi
Vākyam prati matirbhinnā bahudhā nyāyavādinam || (VP.II.1-2)
\end{verbatim}
\end{quote}

The definitions are as follows: (1) \textit{Ākhyātaśabdaḥ} - The verb, (2) \textit{Saṅghātaḥ} - A combination of words, (3) \textit{Jātiḥ saṅghātavartinī} - The universal in the combination of words, (4) \textit{Eko‘navayah śabdaḥ} - An utterance which is one and devoid of parts, (5) \textit{Kramaḥ} - A sequence of words, (6) \textit{Buddhyanusamḥṛtīḥ} - The single whole meaning principle in the mind, (7) \textit{Padamādyam} - The first word, and (8) \textit{Prthak sarvam padam sākāṇkṣam} - Each word having expectancy for one another. These eight views on the sentence are held by earlier grammarians and Mīmāṃsakas. They look at the sentence from different angles depending upon the mental dispositions formed due to their discipline in different Śāstras.\textsuperscript{13}

The definitions \textit{jātiḥ saṅghātavartinī, eko‘navayah śabdaḥ} and \textit{buddhyanusamḥṛtīḥ} can be categorized under Bhartṛhari’s theory of \textit{Sphoṭa} which believes that a sentence is ‘a single undivided utterance’ and its meaning is ‘an instantaneous flash of insight’. This definition is studied by various modern scholars in their respective works. (H. Coward 1976; Lundo 2015; Pillai 1971; Raja 1968; Sriramamurti 1980; Tiwari 1997). Some modern scholars have studied the theory of \textit{Sphoṭa} in different perspectives. G. H. Coward (1973) showed the logical consistency and psychological experience\textsuperscript{14} of \textit{Sphoṭa} theory, while Houben (1989) compared Bhartṛhari’s

\textsuperscript{12}The definition \textit{ekatīṅ vākyam} is explained by \textit{Patañjali} by giving the illustration of \textit{brūhi brūhi}, which indicates that a verb repeated is to be regarded as the same. \textit{Kaiyyaṭa}, the commentator on the \textit{Mahābhāṣya}, also takes the term \textit{eka} as identical.

\textsuperscript{13}Avikalpe‘pi vākyārthe vikalpā bhāvanāśrayāḥ | (VP II.116)

\textsuperscript{14}Coward argues that, according to traditional Indian \textit{Yoga}, the \textit{Sphoṭa} view of language is practically possible. It is both logically consistent and psychologically realizable.
Śabda to Saussure’s theory of sign (Houben 1989). Later on, Akamatsu (1993) tried to look at this theory in the philosophical and historical context of the linguistic theory in India.

In contrast with the theory of Sphoṭa, Mīmāṃsakas hold the view that a syllable has a reality of its own and the word is a sum-total of the syllables and the sentence is only words added together. The remaining definitions such as ākhyātasabdaḥ, saṅghātaḥ, kramaḥ, padamādyam and prthak sarvam padam sākāṅkṣam are categorized under this view. Various modern Indian scholars (Bhide 1980; Choudhary 2011; Gangopadhyay 1993; Iyer 1969; Jha 1980; Sriramamurti 1980) have discussed the compositionality of a sentence in modern times. This view is also studied by various Western psycholinguists such as Sanford and Sturt (2002), and criticized by Pagin (2009) who asserts that it is not enough to understand the meanings of the words to understand the meaning of the whole sentence. Studies by Foss and Hakes (1978), Davison (1984), Glucksberg and Danks (2013) and Levy et al. (2012) proved that the sequence is the important parameter in understanding the English sentence. Similar studies by McEuen (1946) and Davison (1984) have shown that people usually tend to skip the first word in the sentence unless it is semantically loaded.

We study the very first definition i.e. ākhyātasabdaḥ which states that a single word ākhyāta (‘The Verb’) is the sentence. The explanation of this definition as given by Bhartṛhari himself in VP.II.326 suggests that if a mere verb denotes the definite means of the action (i.e. the agent and accessory) in the sentence then that verb should also be looked upon as a sentence. In the introduction to the Ambākartrī commentary on the VP by Pt. Raghu-natha Sarma, he discusses this view by giving examples such as pidhehi. He mentions that when someone utters the mere verb i.e. *pidhehi* (‘Close’ [imperative]), it also necessarily conveys the *karma* of the action which is *dvāram* (‘the door’), in which case, the mere verb *idhehi* can be considered as a complete sentence (Sarma 1980). This view is emphasized by later modern scholars by saying that if a linguistic string is to be considered as a sentence, it should have the expectancy on the level of the semantics and not.

15Houben suggested that in both the works a purely mental signifier plays an important role.
16ākhyātasade niyataṃ sūdhanaṃ yatra gamyate | tadapyekaṃ samāptārtham vākyamityabhidhiyate ||” (VP.II.326)
17pidhehi ...atra dvārami karmaśeṣāt paripūrṇarthatve ‘dvāram pidhehi’ iti vākyam bhavatyeva /
just on the word-level (Laddu 1980; Pillai 1971). As stated by the commentator Puṇyarāja, this definition believes that the meaning of a sentence is of the nature of an action, which means the meaning of the finite verb becomes the chief qualificand in the cognition that is generated and other words in the sentence confirm that understanding of a particular action (Huet 2006; Pillai 1971). Moreover, as said in the commentary, this definition does not deny the status of the sentence of the linguistic string which contains other words besides the verb. But it emphasizes the fact that, sometimes a single verb can also convey the complete meaning, hence can be looked upon as a sentence. Depending upon these views established by the commentary, we can explain the word ākhyātaśabdaḥ in both ways viz. the compound ākhyātaśabdaḥ is analyzed either as ākhyātaḥ eva śabdaḥ (i.e. Karmadhāraya Samāsa- ‘The verb’ [itself can also be considered as a sentence.]) or as ākhyātaḥ śabdaḥ yasmin tat (i.e. Bahuvrīhi Samāsa- ‘the linguistic string consisting the verb’ [is a sentence.]), both of which are qualified as ‘a sentence’. However, one cannot decide whether this definition leaves out purely nominal sentences when it comes to assign the status of the sentence.

Some earlier work on this view in the field of Psycholinguistics such as McEuen (1946) proves that in the English language, the sentence cognition takes place even if the verb is unavailable. The same view is put forward later by Choudhary (2011). He showed that in verb-final languages such as Hindi, comprehenders do not wait for the verb in case they have not

---

18. riya vaikyartaḥ

19. Kriya kriyantaradbhinnit niyattadharsadhana

Prakranti pratipattrunnam bhadha sambodhatah ||” (VP.II.414)

20. tatrakhyatasabdvo vakyamti vadinaam akhyatasabda eva vakyamiti nabhiriyaḥ ...kintu kvacid akhyatasabdo’pi vakyam, yatra karaasabdrayogaṃ vinā kevlakhyatasabd-draayoge’pi vakyarthvagatiḥ ... (Ambākartri on VP.II.1-2)

21. We, in this paper, have studied the latter view, and presented the sentences having verbs and other words as the stimuli to the participants. For studying the first view, which requires presenting the only-verb sentences, it would have led to the loss of context when it comes to the written language cognition. Hence, in stead of presenting only-verb sentences, we have dropped the agent-denoting word from the sentence, which would help us to find out, whether the verbs express their means of actions and are as comprehensible as the sentences having the complements too.

22. We also tried to present these kind of sentences, to study if the nominal sentences are as much comprehensible as the sentences having verbs, or whether it amounts to the excessive cognitive load in the readers which makes them to consider the verb for the better understanding of it.
been reached to it yet but they process the sentence incrementally. The study by Osterhout, Holcomb, and Swinney (1994) showed that the verb has complement-taking properties. Hence, it is the major element in the procedure of sentence-comprehension.

Considering these studies as the motivation, we test the definition of the verb by using an experimental method i.e. by using readers’ **Eye Movement Behavior** on the data which contains verbs, which contains purely nominal sentences and which lack the agents. We are aware that there might be some shortcomings with this definition. There can be the cases or situations in which this definition doesn’t hold true or holds true partially. The aim of this paper is to find out the cases in which it does. Hence, we carry out an experiment to find out the situation in which this definition is valid and also provide statistical evidence for the same.

### 2.2 Cognitive NLP

It is very clear from the vast number of studies that Eye Movement behavior can be used to infer cognitive processes (Groner 1985; Rayner 1998; Starr and Rayner 2001). *The eye is said to be the window into the brain* as quoted by Majaranta and Bulling (2014). Rayner (1998) has mentioned in his work that the reading experiments have been carried out in different languages such as English, French, Dutch, Hebrew, German (Clematide and Klenne 2013), Finnish, Japanese and Chinese etc. There are few studies on Indian languages such as Hindi (Ambati and Indurkhya 2009; Choudhary 2011; Husain, Vasisht, and Srinivasan 2014; Salil Joshi, Kanojia, and Bhattacharyya 2013) and on Telugu (Ambati and Indurkhya 2009). The writing style is mainly from left to right except for Hebrew (right to left). Khan, Loberg, and Hautala (2017) studied the eye movement behavior on Urdu numerals which is written bidirectionally. The orthography has been both horizontal and vertical (Japanese and Chinese). These works have been taken place at various levels of language such as typographical, orthographical, phonological (Miellet and Sparrow 2004), lexical (Husain, Vasisht, and Srinivasan 2014), syntactic (Fodor, Bever, and Garrett 1974), semantic, discourse, stylistic factors, anaphora and coreference (Rayner 1998). Few studies were conducted on fast readers versus poor readers, children

---

**23**Such as in poetry, some concern is also to be given to the sequence (*kramaḥ*) of the words. While learning a new language, every word including first word (*padamādyāṃ*) seems to play the major role, etc.
versus adults versus elderly adults, multilingual versus monolinguals (De Groot 2011), normal readers versus people with reading disabilities such as dyslexia, aphasia (Levy et al. 2012), brain damages or clinical disability (Rayner 1998), schizophrenia, Parkinson’s disease (Caplan and Futter 1986) or oculomotor diseases. Various methodologies were followed such as eye contingent display change, moving window technique, moving mask technique, boundary paradigm, Naming task, Rapid Serial Visual Presentation (RSVP) versus Self-paced reading, reading silently versus reading aloud, etc.

The experiments that took place on reading have been used mainly to understand the levels underlying the comprehension procedure. Apart from that, a study for word sense disambiguation for the Hindi Language was performed by Salil Joshi, Kanojia, and Bhattacharyya (2013) where they discuss the cognitive load and difficulty in disambiguating verbs amongst other part-of-speech categories. They also present a brief analysis of disambiguating words based on different ontological categories. Martínez-Gómez and Aizawa (2013) use Bayesian learning to quantify reading difficulty using readers’ eye-gaze patterns. Mishra, Bhattacharyya, and Carl (2013) proposes a framework to predict difficulty in translation using a translator’s eye-gaze patterns. Similarly, A. Joshi et al. (2014) introduce a system for measuring the difficulties perceived by humans in understanding the sentiment expressed in texts. From a computational perspective Mishra, Kanojia, and Bhattacharyya (2016) predict the readers’ sarcasm understandability, detects the sarcasm in the text (Mishra, Kanojia, Nagar, et al. 2017a) and analyze the sentiment in a given sentence (Mishra, Kanojia, Nagar, et al. 2016) by using various features obtained from eye-tracking.

Eye-tracking has been used extensively for Natural Language Processing (NLP) applications in the field of Computer Science, apart from the immense amount of studies done in the field of psycholinguistics. Mishra, Kanojia, Nagar, et al. (2017b) model the complexity of a scan path, and propose the quantification of lexical and syntactic complexity. They also perform sentiment and sarcasm classification (Mishra, Dey, and Bhattacharyya 2017) using neural networks using eye-tracking data via the use of a convolutional neural network (CNN) (LeCun et al. 1998). They refer to the confluence of attempting to solve NLP problems via cognitive psycholinguistics as Cognitive NLP.

Our method of analyzing eye-movement patterns in the Sanskrit language is the first of its kind and is inspired by these recent advancements.
The *Bird’s eye view* of our research area is presented in Figure 1. The highlighted and bold text is our research interest in the current paper.

![Diagram](image-url)

**Figure 1**

*A brief analysis of our research area*

## 3 Our Approach

We describe our approach to dataset creation in Subsection 3.1, experiment details which includes participant selection in Subsection 3.2, feature description in Subsection 3.3, followed by the methodology of the experiment in Subsection 3.4.

### 3.1 Dataset Creation

We prepare a dataset of 20 documents consisting of either a prose (Total 13) or a poetry (a *subhāṣita*) (Total 7) in the Sanskrit language. Prose documents mainly contain the stories taken from the texts such as *Pañcatantra*, *Vaṃśavṛkṣaḥ* and *Bālanītikathāmālā*. *Subhāṣitas* are taken from the text *Subhāṣitamañjūṣā*. The stories are comprised of 10-15 lines each, and each *subhāṣita* is 2 - 4 verse long. We create three copies of 20 paragraphs as the experiment demands and manipulate them as follows:
• **Type A:** These are 20 documents that do not contain any changes from the original documents. They are kept as they were.

• **Type B:** In this set of documents, we remove the finite and infinite verbs completely which results in a syntactic violation in the respective sentences. These are purely nominal sentences. In poetry, instead of removing the verbs, we replace the verbs with its synonym verb to maintain the format of the poetry. The motivation behind this kind of modification is to test how much does a verb contributes to the comprehension of a sentence, both syntactically and semantically. There are 20 documents of this kind.

• **Type C:** Here, the verbs are kept constant but we drop the *kartā* in the sentences. *kartā* being semantically loaded in the sentence, we choose to drop it for the demand of the experiment i.e. to investigate whether a mere verb without its agent can denote the meaning of the whole sentence. *Kartās* are not removed from the sentences which did not have finite or infinite verbs in the original document to avoid the possibility of insufficient information. This kind of modification will throw some light on the view that the verb itself can be considered as a sentence. In Type C of poetry, the stimulus is degraded by replacing the original finite verbs by distant-meaning finite verbs by retaining the same grammatical category. Even though these verbs bear the syntactic integrity of the sentence, they tend to be semantically incompatible with the other words in the linguistic string. This incompatibility leads to semantic inhibition while processing it, which in turn allows the reader to reconstruct the meaning of the sentence all over again. There are 20 documents of this kind.

The paragraphs do not contain text which readers might find difficult to comprehend. We normalize the text to avoid issues with vocabulary. We control the orthographical, typographical, and lexical variables that might affect the outcome of the experiment. We maintain a constant orthography throughout the dataset. The passages are shown in Devanāgarī script and the writing style is from left to right. We keep the font size large, customize the line spacing to optimum and adjust the brightness of the screen for the comfort of the participant. We ensure that there is no lexical complexity in the prose. We minimize it by splitting the *sandhis* (total 70), separating the compound words with the hyphens (total 51), and also by adding commas.
in appropriate places for the easier reading. The verses are not subject to this kind of modification. This forms our original document. Sentences in the original dataset vary in their nature with respect to the verbs. There are 7 purely nominal sentences, 33 sentences with no finite verb but the kṛdantas and 70 sentences having at least one finite verb in them. There are no single-sentence paragraphs which eliminate the possibility of insufficient contextual information while reading. In poetry, there are 26 finite verbs in total, each verse having 3 to 4 finite verbs in it. Two linguists validate our dataset with 100% agreement that the documents are not incomprehensible. This forms the ground truth for our experiment.

All these types of documents (i.e. Type A, B, and C) are shuffled in such a way that no reader gets to read both types of the same paragraph. Hence, we tried to maintain the counter-balance to remove the bias of the paragraphs. 20 of such shuffled paragraphs make one final dataset. There are three final datasets: Datasets 1, 2, and 3. Out of the 20 participants, 7 participants are presented with Dataset 1, 6 participants with Dataset 2 and remaining 7 participants with Dataset 3. We formulated two multiple-choice questions in each paragraph. The first question of which is one and the same for all paragraphs which helps us get the reader’s viewpoint about the meaningfulness of the paragraph concerned. The second question is based on the gist of that paragraph which works as a comprehension test for the readers, which also ensures that people have read attentively and eliminates the cases of mindless reading. The answers given by the participants on both questions are used by us to decide the inter-annotator agreement and the accuracy rate.
3.2 Experiment Details

We chose 20 participants with a background in Sanskrit. They have been learning Sanskrit for a minimum of 2 years to a maximum of more than 10 years. The participants are neurologically healthy adults who belong to the age group of 22 to 38. They are well-acquainted with the Sanskrit language, however, they were not aware of the modifications made to the datasets beforehand. All of the participants can understand, read, and speak multiple languages. While most of the participants are native speakers of Marathi; few of them have Kannada, Telugu, and Hindi as their native language.

They are provided with a set of instructions beforehand which mentions the nature of the task, annotation input method, and necessity of head movement minimization during the experiment. We also reward them financially for their efforts. They are given two sample documents before the experiment so that they get to know the working of the experimentation process.

3.3 Feature Description

The eye-tracking device records the activity of the participant’s eye on the screen and records various features through gaze data. We do not use all the feature values provided by the device for our analysis, but only the ones which can provide us with the prominence of a word (interest-area) and in turn, show us the importance of words that belong to the same category. These are features which are calculated based on the gaze behavior of the participant, and we use for our analysis:

---

24 The number of participants is less owing to the restriction that we needed our readers to know Sanskrit. We chose the readers with a normal or corrected vision since the readers who use bi-focal eyeglasses would pose a minor possibility of erroneous eye-movement data. Moreover, some other human-related aspects such as very dark or very light irises, downward-pointing eyelashes, naturally droopy eyelids, the headrest not fitting the person’s head or even the incorrigible head motions amount to the calibration fails and errors while reading. We aim to increase the number of participants in future experiments.

25 We chose to present the Sanskrit data to the participants instead of their native languages because it would be more faithful to study the definition, taking the same language which was the lingua franca at the time when these definitions were enlisted. Nonetheless, we also aim to conduct the same definition on the native speakers and carry out the contrastive study for a better understanding of the definition.
1. **Fixation-based features**

Studies have shown that attentional movements and fixations are obligatorily coupled. More fixations on a word are because of incomplete lexical processes. The more cognitive load will lead to more time spent on the respective word. There are some variables that affect the time spent on the word such as word frequency, word predictability, number of meanings of a word or word familiarity, etc. (Rayner 1998). We consider the Fixation duration, Total fixation, Fixation Count for the analysis. These are motivated by Mishra, Kanojia, and Bhattacharyya (2016)

(a) **Fixation Duration (or First Fixation Duration)**
   First fixations are fixations occurring during the first pass reading. Intuitively, increased first fixation duration is associated with more time spent on the words, which accounts for lexical complexity.

(b) **Total Fixation Duration (or Gaze Duration)**
   This is a sum of all fixation durations on the interest areas. Sometimes, when there is syntactic ambiguity, a reader re-reads the already read part of the text in order to disambiguate the text. Total fixation duration accounts for the sum of all such fixation durations occurring during the overall reading span.

(c) **Fixation Count**
   This is the number of fixations in the interest area. If the reader reads fast, the first fixation duration may not be high even if the lexical complexity is more. But the number of fixations may increase in the text. So, fixation count may help capture lexical complexity in such cases.

2. **Regression-based feature**

Regressions are very common in complicated sentences and many regressions are due to comprehension failures. A short saccade to the left is done to read efficiently. Short within-word saccades show that a reader is processing the currently fixated word. Longer regression (back the line) occur because the reader did not understand the text. Syntactic ambiguity (such as Garden Path sentences etc.), syntactic violation (missing words, replaced words) and syntactic unpredictability lead to shorter saccades and longer regressions. We consider
the feature Regression Count i.e. a total number of gaze regressions around the AOI (Ares of Interest).

3. **Skip Count** -
   Our brain doesn’t read every letter by itself. While reading people keep on jumping to the next word. The predictable target word is more likely to be skipped than an unpredictable one. We take Skip count as a feature to calculate the results. Skip count means whether an interest-area was skipped or not fixated on while reading. This is calculated as the number of words skipped divided by total word count. Intuitively, higher skip count should correspond to lesser semantic processing requirements (assuming that skipping is not done intentionally). Two factors have a big impact on skipping: word length and contextual constraint. Short words are much more likely to be skipped than long words. Second, words that are highly constrained by the prior context are much more likely to be skipped than those that are not predictable. Word frequency also has an effect on word skipping, but the effect is smaller than that of predictability.

4. **Run Count** -
   Run count is the number of times an interest-area was read.

5. **Dwell Time-based feature** -
   Dwell time and Dwell Time percentage i.e. the amount of time spent on an interest-area, and the percentage of time spent on it given the total number of words.

3.4 **Methodology**

As described above in Section 3.1, we modified the documents in order to test the syntactic and semantic prominence of a verb in both prose and poetry. Such instances of modification of the data may cause a syntactic violation, semantic inhibition, and leads to insufficient information to comprehend the document, at the surface level of the language. It enforces the reader to re-analyze the text. The time taken to analyze a document depends on the context (Ivanko and Pexman 2003). While analyzing the text, the human brain would start processing the text in a sequential manner, with the aim of comprehending the literal meaning. When such an **incongruity** is perceived, the brain may initiate a re-analysis to reason out such disparity (Kutas
and Hillyard 1980). *As information during reading is passed to the brain through eyes, incongruity may affect the way eye-gaze moves through the text. Hence, distinctive eye-movement patterns may be observed in the case of the successful finding of a verb, in contrast to an unsuccessful attempt.*

This hypothesis forms the crux of our analysis and we aim to prove this by creating and analyzing an eye-movement database for sentence semantics.

4 Analysis & Results

As stated above, we collect gaze data from 20 participants and use it for our analysis. We try to verify the first sentence definition given by Bhartṛhari. With our work, we find that the verb is the chief contributor to the sentence-semantics and enjoys more attention than other words in the process of sentence comprehension. To study how does a reader uses a verb in constructing the meaning of a linguistic string, we analyze the time one spends on the particular verb (dwell-time percentage), the number of times one backtracks (regression out count) or skips (skip count) the verb, the number of times the verb is read through (run count) and fixated upon (fixation count). We analyze these features on the verbs vs. non-verbs in Datasets 1, 2 and 3 and present the results in the Figures 2 (dwell-time percentage), 3 (regression count) and 4 (skip count) in the form of graphs.

The analysis of dwell-time percentage, regression count and skip count proves our point that verbs are prominent elements while constructing the sentence meaning. It can be clearly seen that *verbs are spent more time on, regressed about more and skipped a lesser number of times than non-verbs.* All the participants except a few correlate with our hypothesis. We observe that in Figure 2, Participant 5 (P5) has spent less time on the verbs but we also observe, as shown in Table 1, that P5 lacks in agreement compared to the other annotators. Participants 11 (P11), 12 (P12), and 18 (P18) do not lack in agreement, still, they do not read verbs as much as the other consistent participants and hence are clearly outliers. Even though these four participants have not fixated on the verb for more time, the number of times they regressed around verbs is significantly higher as shown in Figure 3. Figure 4 shows that verbs are unanimously skipped for a lesser number of times than non-verbs, hence it is proved that a reader cannot afford to skip verbs while constructing the sentence meaning.
A Comparison of Dwell-Time Percentage on Verbs and Non-Verbs for all Datasets, and all participants

A Comparison of Regression Count on Verbs and Non-Verbs for all Datasets, and all participants

A Comparison of Skip Count on Verbs and Non-Verbs for all Datasets, and all participants
We also strengthen this view by analyzing the Type A vs. Type B vs.
Type C documents and also consider the answers provided by the readers
in Section 6.

5 Evaluation

We perform the evaluation of our work and calculate the inter-annotator
agreement (IAA) for each participant with all the others, on the same
dataset. We perform this for both the questions posed to the participants,
separately. We also evaluate the answers provided by the participants to
ensure that none of them were performing an inattentive reading of the
documents. We show our evaluation in Tables 1, 2, and 3 for Dataset 1,
2 and 3 respectively. Overall, the agreement of our participants ranges
between 0.45 (Moderate Agreement) to 0.95 (Almost perfect Agreement)
for Question 1. For Question 2, the agreement ranges from 0.5 (Moderate
Agreement) to 0.95 (Almost perfect Agreement). The Accuracy (Acc),
as shown in the tables, ranges from 0.6 to 1, which means that our
participants were substantially accurate and were attentive during the ex-
periment. The inter-annotator agreement points our the tentative outliers
and helps us analyze the results of our experiment. We find that both the
inter-annotator agreement and accuracy of our experiment are substantial.

We also perform statistical significance tests based on the standard t-test
formulation assuming unequal variances for both variables, for all partici-
pants and display the $p$-values in Tables 4, 5, 6 for Datasets 1, 2, and 3
respectively. For these datasets, we compare Verbs with all the other words
for the features Regression Count (RC) and Skip Count (SC). We find out
that a number of regressions performed by a user around verbs are much
more than around other words. For these features, we also show the dif-
ference between the means of verbs and non-verbs ($M_D$), and the $p$-value
($P$). Our T-Test parameters were variable values, the hypothesized mean
difference was set to zero, and the expected cut-off for the T-Test is 0.05.
Our evaluations show that these values are statistically significant for most
of the participants.
Inter-annotator agreement (IAA) and Accuracy (Acc) Scores

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th></th>
<th>Q1</th>
<th>Q2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IAA</td>
<td>IAA</td>
<td>Acc</td>
<td>IAA</td>
<td>IAA</td>
</tr>
<tr>
<td>P1</td>
<td>0.7</td>
<td>0.5</td>
<td>0.6</td>
<td>P8</td>
<td>0.85</td>
</tr>
<tr>
<td>P2</td>
<td>0.8</td>
<td>0.9</td>
<td>0.95</td>
<td>P9</td>
<td>0.75</td>
</tr>
<tr>
<td>P3</td>
<td>0.8</td>
<td>0.9</td>
<td>0.9</td>
<td>P10</td>
<td>0.75</td>
</tr>
<tr>
<td>P4</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>P11</td>
<td>0.65</td>
</tr>
<tr>
<td>P5</td>
<td>0.45</td>
<td>0.85</td>
<td>0.9</td>
<td>P12</td>
<td>0.7</td>
</tr>
<tr>
<td>P6</td>
<td>0.9</td>
<td>0.55</td>
<td>0.6</td>
<td>P13</td>
<td>0.85</td>
</tr>
<tr>
<td>P7</td>
<td>0.85</td>
<td>0.7</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1
Dataset 1

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th></th>
<th>Q1</th>
<th>Q2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IAA</td>
<td>IAA</td>
<td>Acc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P14</td>
<td>0.8</td>
<td>0.8</td>
<td>0.75</td>
<td>P8</td>
<td>0.141</td>
</tr>
<tr>
<td>P15</td>
<td>0.65</td>
<td>0.65</td>
<td>0.75</td>
<td>P9</td>
<td>0.147</td>
</tr>
<tr>
<td>P16</td>
<td>0.85</td>
<td>0.9</td>
<td>0.95</td>
<td>P10</td>
<td>0.112</td>
</tr>
<tr>
<td>P17</td>
<td>0.9</td>
<td>0.8</td>
<td>0.7</td>
<td>P11</td>
<td>0.194</td>
</tr>
<tr>
<td>P18</td>
<td>0.75</td>
<td>0.85</td>
<td>0.85</td>
<td>P12</td>
<td>0.163</td>
</tr>
<tr>
<td>P19</td>
<td>0.5</td>
<td>0.9</td>
<td>0.9</td>
<td>P13</td>
<td>0.211</td>
</tr>
<tr>
<td>P20</td>
<td>0.8</td>
<td>0.7</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3
Dataset 3

Mean Difference and p-values from T-Test for Regression Count (RC) and Skip Count (SC)

<table>
<thead>
<tr>
<th></th>
<th>RC</th>
<th>SC</th>
<th></th>
<th>RC</th>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0.159</td>
<td>0.000</td>
<td>0.061</td>
<td>0.038</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>0.234</td>
<td>0.000</td>
<td>0.078</td>
<td>0.012</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>0.250</td>
<td>0.000</td>
<td>0.180</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>0.126</td>
<td>0.001</td>
<td>0.112</td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>0.062</td>
<td>0.050</td>
<td>0.029</td>
<td>0.194</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>0.183</td>
<td>0.001</td>
<td>0.064</td>
<td>0.029</td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>0.091</td>
<td>0.029</td>
<td>0.089</td>
<td>0.005</td>
<td></td>
</tr>
</tbody>
</table>

Table 4
Dataset 1

<table>
<thead>
<tr>
<th></th>
<th>ROC</th>
<th>SC</th>
<th></th>
<th>ROC</th>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8</td>
<td>0.141</td>
<td>0.001</td>
<td>0.129</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td>0.147</td>
<td>0.001</td>
<td>0.134</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td>0.112</td>
<td>0.005</td>
<td>0.143</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>P11</td>
<td>0.194</td>
<td>0.000</td>
<td>0.025</td>
<td>0.237</td>
<td></td>
</tr>
<tr>
<td>P12</td>
<td>0.163</td>
<td>0.003</td>
<td>0.012</td>
<td>0.364</td>
<td></td>
</tr>
<tr>
<td>P13</td>
<td>0.211</td>
<td>0.000</td>
<td>0.106</td>
<td>0.001</td>
<td></td>
</tr>
</tbody>
</table>

Table 5
Dataset 2

<table>
<thead>
<tr>
<th></th>
<th>ROC</th>
<th>SC</th>
<th></th>
<th>ROC</th>
<th>SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P14</td>
<td>0.188</td>
<td>0.000</td>
<td>0.058</td>
<td>0.053</td>
<td></td>
</tr>
<tr>
<td>P15</td>
<td>0.072</td>
<td>0.033</td>
<td>0.058</td>
<td>0.053</td>
<td></td>
</tr>
<tr>
<td>P16</td>
<td>0.244</td>
<td>0.001</td>
<td>0.077</td>
<td>0.015</td>
<td></td>
</tr>
<tr>
<td>P17</td>
<td>0.129</td>
<td>0.003</td>
<td>0.055</td>
<td>0.059</td>
<td></td>
</tr>
<tr>
<td>P18</td>
<td>0.120</td>
<td>0.030</td>
<td>-0.030</td>
<td>0.189</td>
<td></td>
</tr>
<tr>
<td>P19</td>
<td>0.021</td>
<td>0.247</td>
<td>0.044</td>
<td>0.106</td>
<td></td>
</tr>
<tr>
<td>P20</td>
<td>0.253</td>
<td>0.002</td>
<td>0.059</td>
<td>0.049</td>
<td></td>
</tr>
</tbody>
</table>

Table 6
Dataset 3
6 Discussion

We discussed the core features of our work *i.e.* Dwell-time Percentage, Regression Count, Skip Count, Run Count, and Fixation Count in Section 4. In this section, we would like to further analyze the result of work by exploring the answers provided by our participants. We break down our documents into the categories of prose and poetry. In Figures 5 and 6, we show the answer counts of our participants, when they find the documents absolutely non-meaningful, or lacking information *i.e.*, somewhat meaningful. For all participants, over document Types A, B, and C, we find that Type A (Original Data) is marked non-meaningful least number of times.

In case of a prose (Figure 5), Type B documents lack verbs. It can clearly be seen that our participants do not understand the documents most of the time, and mark them either as completely non-meaningful or lacking in information. We do not hint them to look for verbs as psycholinguistic principles do not allow an experiment to be biased in the participants’ mind. Non-presence of verbs in Type B documents affects both syntax and the semantics of the documents and it can be seen that purely nominal sentences fail to convey the complete semantics of the sentence. In Type C for prose (Figure 5), we see that our participants are confused by the removal of agent-denoting words, but are still able to grasp the context, and hence their answers do not depict absolute meaninglessness of the documents. Even though verbs are retained in document type C, the removal of agent words leads to insufficient information.

For poetry (Figure 6), Type B documents have the presence of synonymous verbs, and Type C have verbs with very distant meanings and no correlation with the semantics of the original verb present. Hence, Type B documents are marked as lacking in information by our participants many times as compared to Type A documents. They do not mark even one of them as absolutely meaningless as a synonym of a verb is present and they are still able to grasp the context which bears a strong impact on the conclusion we draw. On a similar note, Type C documents that have verbs but with very distant meanings are marked lacking in information most number of times, as a correlation cannot be established between the expected sense of the original verb and the current verb present in the document.

We explore further and manually analyze the saccadic paths of our participants to find out that in document types A, B, and C, the saccadic-regressions vary as per our hypothesis. We present a sample in Figures 8,
9 and 10. For a randomly chosen single participant, who has above average IAA and good accuracy, we find that the amount of regression on document Type c increases in comparison to Type A since the document lacks a agent in some sentences. But, for Type B, we can observe that the regressions increase further when the verb is completely removed from the document.

As stated before, the definition that we have studied might not be valid in all cases. Our aim is to find out the cases in which it does. In the conclusion of this research, we can say that we have found one such case in which Bhartṛhari’s definition Ākhyātaśabdaḥ is valid and that is: when the lexical complexity is minimized in the Sanskrit texts, readers rely on the verbs in order to understand the complete meaning of the sentence, without which the sentence-meaning seems incomplete. Hence, we can conclude that verbs play the most important role in the syntax and semantics of a sentence, nonetheless, in most of the cases, they demand their complements (i.e. means of action) to represent the complete semantics of a sentence. We can also conclude that the purely nominal sentences in Sanskrit are less meaningful than the corresponding original sentences.

Similarly, we would also like to present Figures 12 (Run Count) and 13 (Fixation Count) which further strengthen our discussion. We can see in both the figures that a number of times a verb has been read is always more than the number of times other words have been read.

![Figure 5](image1)  
*Figure 5*  
*For Prose*

![Figure 6](image2)  
*Figure 6*  
*For Poetry*

*Figure 7*  
*Meaninglessness of documents as reported by Participants on different document sets*
Figure 8  
Regressions on Type-A

Figure 9  
Regressions on Type-B

Figure 10  
Regressions on Type-C

Figure 11  
Regression sample from a participant

Figure 12  
A Comparison of Run Count on Verbs and Non-Verbs for all Datasets, and all participants

Figure 13  
A Comparison of Fixation Count on Verbs and Non-Verbs for all Datasets, and all participants
Limitations

The data selected for our experiment does not vary in its nature. We only use stories in prose, and the poetry is also borrowed from the same text. We would like to clearly state that we know this is a limitation of our work. It will be more insightful to conduct similar experiments on different kinds of texts. For the same experiment on ‘verbs’, data can also be modified in many other ways. Moreover, a spoken word, when accompanied by gesture and facial expression and when given a special intonation, can convey much more than the written word. This experiment it limited to the written sentences only and it tests the comprehension only from the reader’s point of view.

7 Conclusion & Future Work

We present a fresh view to study Bhartṛhari’s Vākyapadīya, especially the definitions given by him on the syntactic and the semantic level. We pick sentence definition one viz. Ākhyātaśabdaḥ, that the “verb” can also be considered as a sentence. We discuss his work in brief and perform an experiment to study this definition from a cognitive point of view. We employ the eye-tracking technique and follow the methodology of silent-reading of Sanskrit paragraphs to perform the above-mentioned experiment in order to have a better understanding of the definition. We aim to extend our work under the purview of Cognitive NLP and use it to resolve computational problems. With our work, we open a new vista for studying sentence definitions in the cognitive point of view by following an investigational technique.

Our results show that humans tend to read verbs more than they read other words and they are deemed most important. We assert that verbs play a prominent role in the syntax and semantics of a sentence, nonetheless, in most of the cases, they demand their complements to represent the complete semantics of a sentence. It is proved that a human being, cognitively, searches for a verb in a sentence, without which the unity of a sentence tends to be incomplete. Purely nominal sentences in the Sanskrit language are less meaningful than the original sentences. We show the statistical significance of our results and evaluate them using the standard T-test formulation. We also discuss the manual analysis of saccadic paths and answers given by our participants to verify our results. We are aware that, the method followed
by us is one way of justifying Bhartṛhari and there could be other ways that can strengthen the same results.

In the future, we aim to conduct more experiments on different kinds of texts in the Sanskrit language which have different sentence-construction styles. For the same experiment on ‘verbs’, data can also be modified in other ways such as changing the place of the verb in the sentence, removing the sentence boundary markers, replacing the conjunctions, negatives, discourse markers, etc. We also aim to verify other sentence definitions using eye-tracking. We would like to employ other tools such as EEG and work in multi-lingual settings to further delve deeper into the cognition of a human mind so that we can understand the definition in a better perspective. We would also like to study the comprehension among the native speakers vs. bilingual so that we can study whether the definitions by Bhartṛhari are generic in nature. We hope to gain more insights into the field of Cognitive NLP with the help of our work.

**Acknowledgements**

We thank our senior colleague Dr. Abhijit Mishra who provided insights and expertise that greatly assisted this research. We are grateful to Vasudev Aital for his assistance in the data-checking process and all the participants for being part of this research. We would also like to extend our gratitude to the reviewers for their comments on an earlier version of the manuscript, although any errors are our own.
References


