SANDHI SPLITTER AND ANALYZER FOR SANSKRIT

(With Special Reference to *aC Sandhi*)

Dissertation submitted to Jawaharlal Nehru University

In partial fulfillment of the requirements

for award of the

degree of

MASTER OF PHILOSOPHY

SACHIN KUMAR

SPECIAL CENTRE FOR SANSKRIT STUDIES

JAWAHARLAL NEHRU UNIVERSITY

NEW DELHI-110067

INDIA

2007

ÌuÉÍvÉ· xÉ|xMüiÉ AkrÉrÉlÉ MüISì
जवाहरलाल नेहरू विश्वविद्यालय
lÉD ÌSssÉÏ - 110067

SPECIAL CENTRE FOR SANSKRIT STUDIES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067

July 30, 2007

D E C L A R A T I O N

I declare that the dissertation entitled "**Sandhi Splitter and Analyzer for Sanskrit (with special reference to *aC sandhi*)**" submitted by me for the award of the degree of **Doctor of Philosophy** is an original research work and has not been previously submitted for any other degree or diploma in any other institution/university.

(Sachin Kumar)

ÌuÉÍvÉ· xÉ|xMüiÉ AkrÉrÉlÉ MåüIS
जवाहरलाल नेहरू विश्वविद्यालय
lÉD ÌSssÉÏ - 110067

SPECIAL CENTRE FOR SANSKRIT STUDIES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067

July 30, 2007

C E R T I F I C A T E

This dissertation entitled “Sandhi Splitter and Analyzer for Sanskrit (with special reference to *aC sandhi*)**” submitted by** Sachin Kumar **to** Special Centre for Sanskrit Studies, Jawaharlal Nehru University, New Delhi-110067**, for the award of the degree of Master of Philosophy, is an original work and has not been submitted so far, in part or full, for any other degree or diploma of any University. This may be placed before the examiners for evaluation.**

Dr. C. Upender Rao
(Chairperson)

Dr. Girish Nath Jha
(Supervisor)

To

MY

LATE GRANDFATHER

SH. HEMRAJ MEHANDIRATTA

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to all those without whom I would not have been able to complete this work. The first and foremost, I am extremely thankful to my supervisor Dr. Girish Nath Jha, who provided me his generous support, encouragement and inspiration all through this work. It is his impeccable and persuasive guidance, constructive suggestions which have molded my work in the present shape. I express my deepest sense of gratitude to him.

I also express my sincere gratitude to faculty members of my centre, Shashiprabha Ma'am, Upender Sir, Hari Ram Sir, Rajnish Sir, Ram Nath Sir and Santosh Sir for their teaching and assistance. I also acknowledge all the staff members for their cooperation and extending facilities to complete my work.

I duly acknowledge University Grants Commission (UGC) for providing me the financial assistance. I also extend my special thanks to the library staffs of my University, especially to Malik Sir who generously helped in finding the relevant material.

I am deeply indebted to my Dadi, Papa, Mummy, Chacha, Chachi, Sisters and Brothers for their prayers, affection and immense support. I am grateful to my Shyam Chacha whose constant inspiration and never-ending love is a pillar of strength to me.

My special thanks to Sudhir ji, Chandra ji, Subhash ji, Ainura ji, Narayan ji, Muktanand, Manji, Diwakar, Diwakar Mishra, Surjit, Vijendra and Mayank for helping me to systematize the ideas for my R & D. I extend my special appreciation to Alok, Mukesh and Bala for helping me in data entry.

I am obliged to acknowledge my friends and my seniors like Subhash Sir, Ajay Sir, Piyush Sir, Vijay Bhaiya, Devendra Sir, Vimal Sir, Yogesh Ji, Ved, Nandi, Ramanuj, Chander and many others who gave immense moral courage in the course of my writing. They were always eager to know about the progress of my work till I completed.

Sachin Kumar

Contents

List of Abbreviations

Aṣṭ.	*Aṣṭādhyāyī*
ASR	Academy of Sanskrit Research
JNU	Jawaharlal Nehru University
JSP	Java Server Pages
Kāś. vṛt	*Kāśikāvṛtti*
LTRC	Language Technologies Research Centre
MAT	Machine Aided Translation
MT	Machine Translation
MTS	Machine Translation System
MWSDD	Monier Williams Sanskrit Digital Dictionary
NL	Natural Language
NLP	Natural Language Processing
OCR	Optical Character Recognition
POS	Part of Speech
R&D	Research and Development
RCILTS	Resource Centre for Indian Language Technology Solutions
RSV	Rashtriya Sanskrit Vidyapeetha
SAS	Sandhi Analyzer for Sanskrit
SCSS	Special Centre for Sanskrit Studies
Sid. Kau.	*Siddhāntakaumudī*
TDIL	Technology Development for Indian Languages

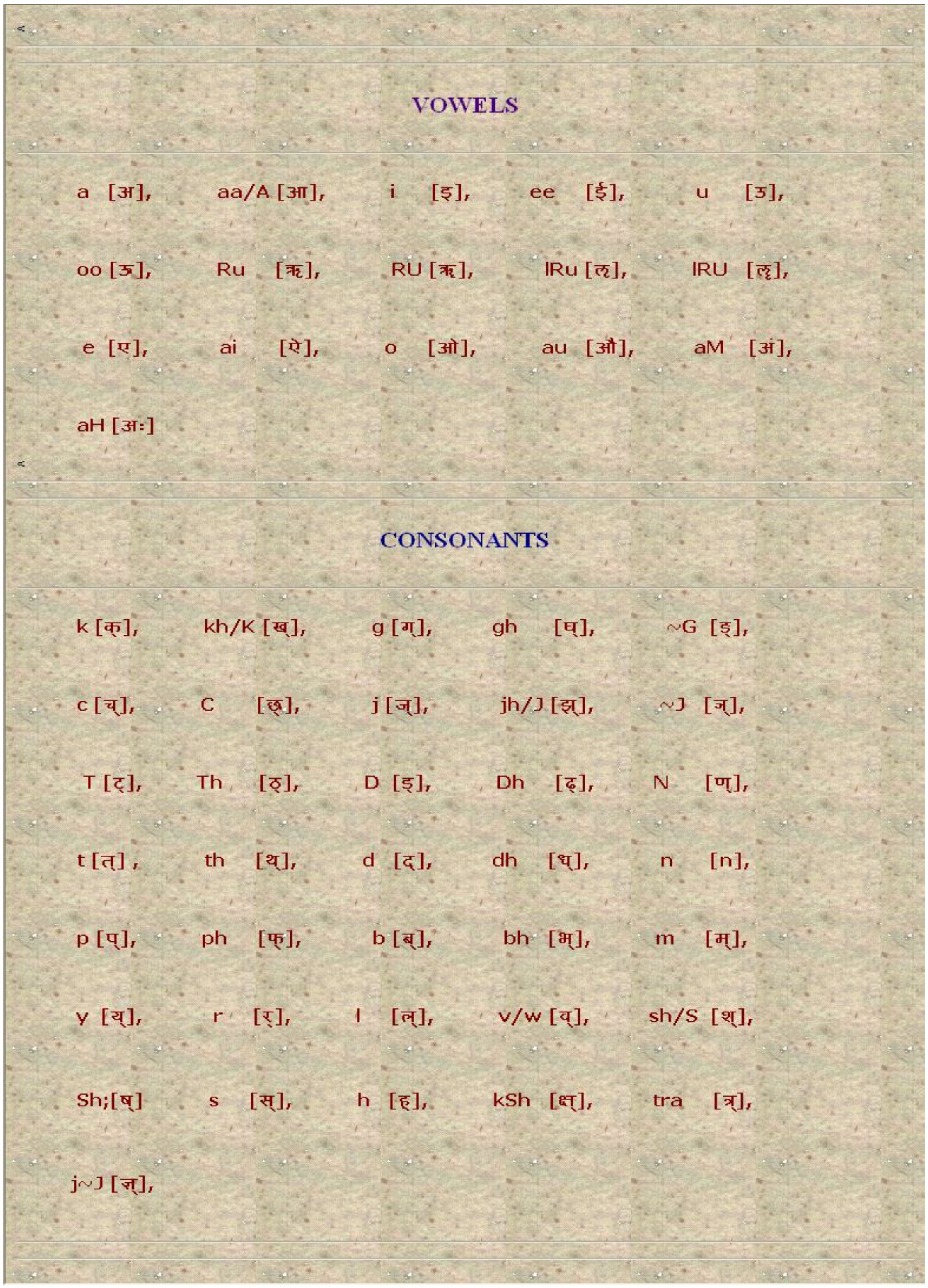
List of Tables

Transliteration key used in the dissertation

A	=	a
AÉ	=	ā
C	=	i
D	=	ī
E	=	u
F	=	ū
G	=	ṛ
H	=	ṝ
I	=	ḷ
L	=	e
Lè	=	ai
AÉå	=	o
AÉè	=	au
Mü	=	k
ZÉ	=	kh
aÉ	=	g
bÉ	=	gh
Xü	=	ṅ
cÉ	=	c
Nü	=	ch
eÉ	=	j
fÉ	=	jh
gÉ	=	ñ
Oü	=	ṭ
Pü	=	ṭh
Qü	=	ḍ
Rü	=	ḍh
hÉ	=	ṇ
iÉ	=	t
jÉ	=	th
S	=	d
kÉ	=	dh
lÉ	=	n
mÉ	=	p
Tü	=	ph
oÉ	=	b

pÉ	=	bh
qÉ	=	m
rÉ	=	y
U	=	r
sÉ	=	l
uÉ	=	v
vÉ	=	ś
wÉ	=	ṣ
xÉ	=	s
W	=	h
¤É	=	kṣ
§É	=	tr
¥É	=	jñ
Å	=	‘
(*Anusvāra*)	=	ṁ
È (*visarga*)	=	ḥ

Devanagri Input Mechanism according to Baraha software (http://www.baraha.com)



VOWELS

a [अ],	aa/A [आ],	i [इ],	ee [ई],	u [उ],
oo [ऊ],	Ru [ऋ],	RU [ॠ],	lRu [ऌ],	lRU [ॡ],
e [ए],	ai [ऐ],	o [ओ],	au [औ],	aM [अं],
aH [अः]				

CONSONANTS

k [क्],	kh/K [ख्],	g [ग्],	gh [घ्],	~G [ङ्],
c [च्],	C [छ्],	j [ज्],	jh/J [झ्],	~J [ञ्],
T [ट्],	Th [ठ्],	D [ड्],	Dh [ढ्],	N [ण्],
t [त्] ,	th [थ्],	d [द्],	dh [ध्],	n [n],
p [प्],	ph [फ्],	b [ब्],	bh [भ्],	m [म्],
y [य्],	r [र्],	l [ल्],	v/w [व्],	sh/S [श्],
Sh;[ष्]	s [स्],	h [ह्],	kSh [क्ष्],	tra [त्र्],
j~J [ज्ञ्],				

Introduction

The scope of the present research is to develop a vowel *sandhi* (*ac sandhi*) analyzer for Sanskrit based on Pāṇinian formulations. While some attempts have been made (Gerard Huet) to develop string segmentation engines based on ad-hoc processing, there is no *sandhi* analyzer which comprehensively analyses a Sanskrit text according to Pāṇinian approach. The present work and the associated algorithm will be useful in solving this long overdue problem in Sanskrit Natural Language Processing (NLP). Though the study is only related to the vowel *sandhi* analysis, but the similar approach can be applicable to build a complete *sandhi* analyzer for Sanskrit.

Sandhi-viccheda is a critical module for any Natural Language (NL) system for Sanskrit. It is because of the synthetic nature of Sanskrit in which words can be combined together to form a larger string of words. So, before processing Sanskrit input text and extracting morphological and syntactical information from it, these conjugated words need to be segmented into their constituents. An automated *sandhi* analysis is a pre-requisite for complete analysis of Sanskrit input text as it will simplify the Sanskrit text and this simplified text can be basis for doing Part of Speech (POS) analysis and doing further grammatical analysis of the text. This complete analysis of Sanskrit text can be used in various NLP applications like Sanskrit- Indian Language Machine Translation System (MTS), tagging of large text corpora, spell checker for Sanskrit, building a Sanskrit text search engine etc. This work, besides being an essential component in NL system of Sanskrit, will also be useful for self-reading and understanding of Sanskrit text.

Sandhi-viccheda has been one of the most challenging aspects of Sanskrit processing. Pāṇini, in *Aṣṭādhyāyī*, describes rules of *sandhi* formation. To apprehend these rules, the understanding of the structure of Aṣṭ., its technical terms, the application of rules and also of context is required. In generation of *sandhi* words, these rules can be understood easily. But reverse *sandhi* analysis is a complex subject. Even to manually segment these *sandhi*-derived words, a mastery over applying *sandhi* rules and knowledge of Sanskrit lexicon is needed. In automated segmentation, evolving reverse rule base for these rules, building an exhaustive lexicon of Sanskrit words to validate the segmentation, applying context and resolving ambiguities are some major issues.

The objectives of this study are:

- to build a reverse vowel *sandhi* rule base and example base of Pāṇinian *sandhi* rules for identification and analysis of vowel *sandhi*,
- to adapt Monier Williams Sanskrit Digital Dictionary (MWSDD) of Louis Bontes for analysis purpose,
- to adapt available e-corpora and customize them for *sandhi* analysis purpose,
- to build a servlet based online Java engine which will consult the rule base, example base and the linguistics resources to analyze vowel *sandhi* in a Sanskrit text, and will be used in any other Sanskrit processing application

For this Research and Development (R&D), the methodology of computational Sanskrit and software engineering has been used. This R&D is based on a hybrid approach of rule base and example base. The study consists of a descriptive, analytical as well as application work. The study is based on the primary and secondary resources available on the topic. The primary sources include the Pāṇinian Aṣṭ., *Siddhāntakaumudī (Sid. Kau.) of Bhaṭṭojidīkṣita*, *Kāśikāvṛtti (Kāś. vṛt) of Vāmana and Jayāditya*, adapted MWSDD by Louis Bontes and adapted and customized e-corpora. Secondary materials include several books of grammar, published articles and information on the internet. To develop a vowel *sandhi* analyzer, the rules and *vārttikas* of vowel *sandhi* from *ac sandhiprakaraṇa* of *Sid. Kau* have been formalized in reverse format. This format has also been manually tried on the corpus of simple Sanskrit stories of *pañcatantra* and necessary modifications have been carried out to stabilize the reverse rule base. As part of the research, various linguistic resources were developed and adapted according to the need of the system. To build a corpus of Sanskrit words, a lexicon of place name, nouns was developed and MWSDD was adapted to *Devanāgarī* UTF-8. A verb database is also adapted to exclude the *tiṅanta* (verb form) of processing. An example base of 1000 *sandhi*-derived words from *pañcatantra* is collected with their split form. All these lexicons are to validate the segmented words and will be on up rise. For online processing of Sanskrit text, a Java based web-application has also been developed.

The process flow of the system is as follows:

input Sanskrit text

↓

viccheda eligibility tests
(pre-processing)

↓

subanta processing

↓

search of *sandhi* marker and *sandhi* patterns
(*sandhi* rule base)

↓

generate possible solutions
(result generator)

↓

search the dictionary

↓

search the results in the corpora (if not found in the dictionary)

↓

output (segmented text)

The research has been divided into four chapters. The chapter 1 discusses forward and reverse computation of *sandhi*, computational morpho-phonemics, morpho-phonemics in Sanskrit, need for the *sandhi* analyzer and a survey of work related to *sandhi* processing and NLP of Sanskrit and other Indian Languages. The chapter 2 talks about the main characteristics of Sanskrit euphonic rules and a detailed discussion of vowel *sandhi* rules and its exceptions. The chapter 3 is a detailed study of the rule base and example base of *sandhi* and other lexical resources required for the reverse vowel *sandhi* analysis. The chapter 4 discusses the implementation aspects- the front end, Java objects, linguistics resources and how they work. The limitations of the system and its implications for future research have been summarized in concluding part of the dissertation. The appendices contain the sample data of linguistic resources used to develop the *sandhi* analyzer system, the screen shot of the interface and the debugging process of the system. A portable CD has also been enclosed with the dissertation which comprises the sample data of each linguistic resources and a screen-shot of the interface. The system is likely to be stable within a month period and it can be accessed at http://sanskrit.jnu.ac.in

<u>Chapter – I</u>

Sanskrit sandhi and its computation

1.1 Introduction

The computational aspect of *sandhi* has two dimensions: forward computation and reverse computation. These two can be described as follows:

- **Forward Computation of *sandhi***

  Forward computation of *sandhi* means computing Pāṇinian rules for *sandhi* formation leading to resultant sounds from a *samhitā* situation. The rules, according to which either one or more sounds are modified in such cases, are called the rules of *sandhi*. *Sandhi* can be within a word or between two or more words. To do this computationally, a computational representation of *sandhi* rules and an algorithm to generate *sandhi* is needed. For example; to combine the word *rāmasya+ācāryaḥ*; the four rules of simple vowel combination are required:

  (1) /ā/+/ā/→/ā/,
  (2) /ā/+/a/→/ā/,
  (3) /a/+/ā/→/ā/,
  (4) /a/+/a/→/ā/

  In other words, this can be given as
  [simple vowel] → [+long simple vowel] / - [vowel of same type] (*akaḥ savarṇe dīrghaḥ* in Pāṇinian terms)

- **Reverse computation of *sandhi***

  In the reverse *sandhi* case, the above procedure is reversed. Reverse computation of *sandhi* means applying Pāṇinian rules in reverse form to split the *sandhi*-derived words into their constituent morphemes. This parsed or simplified Sanskrit text will be useful in various NLP applications for Sanskrit. This process needs computational representation of *sandhi* rules in reverse format, an algorithm to parse Sanskrit words and linguistic resources to validate the split words. For example, to split the word *rāmasyācāryaḥ* into *rāmasya+ācāryaḥ*; the four rules of simple vowel combination are required in reverse format:

(1) ā/→/ā/+/ā/

(2) /ā/→/ā/+/a/

(3) /ā/→/a/+/ā/

(4) /ā/→/a/+/a/

In other words, this can be given as
[+ simple long vowel] →[two same types of vowel] (*akaḥ savarṇe dīrghaḥ* in reverse Pāṇinian terms)

1.2 Computational Morpho-phonemics

1.2.1 Computational Phonology

Phonology is a subfield of linguistics which studies the sound system of a language. It deals with the analysis, classification and organization of the phonemes of a language. It differs from phonetics in the sense that phonetics is the study of the production, transmission, and perception of speech sounds whereas phonology studies how they are combined, organized, and convey meaning in a particular language. An important part of phonology is to study which sounds are distinctive units within a language. In Sanskrit, for example, /k/ and /n/ are distinctive sounds (i.e., they are phonemes). This can be seen from minimal pairs such as '*karaḥ*' and '*naraḥ*', which mean different things, but differ only in one sound in identical position (word initial position in this case). Similarly, /ṭ/ and /r/ respectively in *kukkuṭaḥ / kukkuraḥ* (word non-final position) and /v/ and /m in gacchāvaḥ / gacchāmaḥ (word non-final position)

Computational Phonology is the field which deals with the computational techniques of the representation and processing of phonological rules and behaviour.[1] This can be useful in NLP applications such as speech recognition, text-to-speech etc. Computational

[1] Bird, Steven. 2003, 'Phonology', in '*The Oxford Handbook of Computational Linguistics'*, Edited by Ruslan Mitkov, New York: Oxford University Press, p. 6.

phonology can be generative as well as analytical. For example formulation of a phonological rule for voicing alternation can be as follows:

+ cons -> [+ voice] /- [+ voice]
(a consonant becomes voiced if a voiced sound follows)

The same rule can be analyzed in the following way:

[+voiced cons] → [-voiced cons] /- [+voiced sound]

1.2.2 Issues in Computational Phonology

There are various issues related to the representation, procedures and implementation of Computational Phonology.[2] These issues can be described as follows:

- Representations:
  - what are the representation formalisms for phonological knowledge, computational and cognitive reasoning, data structures for phonemes, strings of phonemes, syllable structures, feature matrices

- Procedures:
  - what procedures are required for mapping one phonological representation to another and implementing phonological rules

- Implementations:
  - how does one set about designing and making an implementation

1.2.3 Computational Morphology

Morphology is a branch of linguistics which deals with the formation of word. It studies the patterns and rules of grouping sounds into words, their grammatical paradigms and grammatical properties. The basic building blocks of words are morphemes. A morpheme

[2] Jha, Girish Nath. 2007, "*Introduction to Computational Phonology*", Lecture delivered on 5 January 2007 at CDAC, Noida.

is the smallest meaningful linguistic unit. Morphemes are of two types: free morphemes and bound morphemes. Free morphemes are those which can occur as a word by themselves, for example *rāma*. Bound morphemes are the morphemes which occur only in combination with other forms. All affixes are bound morphemes, for example */su>ḥ/* in *rāmaḥ*.

Computational morphology is analysis and generation of word-forms through computational techniques.[3] This morphological information is very useful in analyzing a language because syntactic analysis requires morphological analysis. This morphological information can be used in various NL applications such as parsing, lemmatization, text-to-speech, Machine Translation (MT), spell checker, spell corrector, automatic word separator, text generation and word paradigm builder.

Morphological analysis is a complex task. It has various dimensions which can be described as follows[4] –

- **Complexity of word formation**
  Words are built up by joining morphemes according to the permissible patterns in a language. Typologically, languages are of Agglutinative, Isolating, Inflectional and Polysynthetic types based on how morphemes combine to form words productively

- **Morphological processes:**
  There are essentially three types of morphological processes which determine the function of morphemes. These three processes are inflectional, derivational and compounding.

- **Morpheme combination**

[3] Jha, Girish Nath. 2007, "*Introduction to Computational Morphology*", Lecture delivered on 5 January 2007 at CDAC, Noida.
[4] Oflazer,Kemal. http://folli.loria.fr/cds/2006/courses/Oflazer.ComputationalMorphology.pdf

Morphemes can be combined in a variety of ways to build the words such as concatenation, infixation, circumfixation, templatic combination and reduplication.

1.2.4 Issues in Computational Morphology

- what kind of data needs to be compiled
- what are the morphological rules and how to represent them for computational purposes
- what are possible implementation strategies
- what are potential ambiguities and how to resolve them

1.2.5 Morphophonemics or Morphophonology

Words are composed by concatenating morphemes. Morphotactics governs the rules for this combination of the morphemes. Sometimes in this concatenation process, there occur some phonological changes at morpheme boundary. These modifications and their underlying reasons are studied under morphophonemics or morphophonology.[5] For example, assimilation in Sanskrit where two segments influence each other at word boundary i.e. *tat+ca=tacca, tat+ṭīkā=taṭṭīkā.* Here /t/ (dental) changes to /c/ (palatal) and /*ṭ/ (retroflex)* respectively.

1.2.6 Issues in Morphophonemics

- what are the morphophonemic rules which explain these changes
- how to represent theoretical rules for computational purposes
- how to restrict the generation of ungrammatical words
- how to handle ambiguities

1.2.7 Morphophonemics in Sanskrit

Typologically, Sanskrit belongs to the inflectional category. Words get their forms when bound morphemes combine with the bases and get fused with them. For example, *rāma* +

[5] Troast, Harald. 2003,'*Morphology', in 'The Oxford Handbook of Computational Linguistics*', Edited by Ruslan Mitkov, New York: Oxford University press, p. 36.

ṭā (ina) → *rāmeṇa*. *Sandhi* governs these morphophonemic changes at morpheme or word boundary in terms of alteration to the sounds due to the neighboring sounds or due to the morphological behaviour of adjacent words. *Sandhi* can take place between vowel and vowel, vowel and semivowel, semivowel and semivowel, consonants and consonants and between visarga and other sounds. *Sandhi* is useful in internal structuring of constituents like verbs, and padas (internal sandhi), as well as for the combination of two words (external sandhi).[6] This sandhi is compulsory within the internal structure of a word, in concatenation of *dhātu* (root) and *upasarga* (prefix), and in *samāsa* (compounds), but in a sentence i.e. in the case of the finals and initials of the different words in a sentence, it depends on the will of the writer.[7]

To analyze this, semantic consideration is also required because *sandhi* overlaps with *samāsa*. *Samāsa*, or compounding in Sanskrit, may consist of two or more words. In *samāsa*, only the last word takes case marker and the remaining words are used as *prātipadika* (crude form). In joining these words as well, the *sandhi* rules apply. The final consonant or vowel of preceding word, according to the *sandhi* rules, combines with the initial letters of the following words. For this purpose, a separate *samāsa* analyzer is needed, but *sandhi* analyzer will also be partially helpful by segmenting *samsata pada* (compound word) into *sandhi*-free constituents.

1.3 Need for the sandhi analyzer

Sandhi analyzer will be a very important component in any NL system that attempts to analyze and understand Sanskrit for computational purposes. In the architecture of a computational Sanskrit platform, various linguistic resources such as lexicon, POS Tagger, *kāraka* analyzer, *subanta* analyzer, *tiṅanta* analyzer, *liṅga* analyzer, *sandhi* analyzer, *samāsa* analyzer etc. will be needed. All these resources will be interlinked but *sandhi* analyzer will be a pre-requisite for analyzing a Sanskrit text because words in Sanskrit language are generally written with no explicit boundaries.

[6] Jha, Girish Nath.1993, '*Morphology of Sanskrit Case Affixes: A Computational Analysis*', M.Phil., submitted to JNU, New Delhi. Pp. 16-17

[7] *saṃhitaikapade nityā nityā dātūpasargayoḥ|*
nityā samāse vākye tu sā vivakṣāmapekṣate||

This *sandhi* analyzer module will be useful in many ways. Sanskrit has a vast knowledge reserve of diverse disciplines. To make this knowledge available to the users of other languages, an automatic MTS from Sanskrit to other Indian languages will have to be developed. *Sandhi-viccheda* will be an essential initial step for this work. The other applications of this segmented form of Sanskrit text may be in building a search algorithm and spell checker for Sanskrit corpora. A '*sandhi-aware*' system thus will not only be essential for any larger Sanskrit NL system, but will also be helpful for self-reading and understanding of Sanskrit texts by those readers who do not know or want to go through the rigors of *sandhi viccheda*. It will also be helpful for interpretation and simplification of Sanskrit text. Any NL or NL like Sanskrit compiler will have *sandhi viccheda* as a necessary initial component.

1.4 Survey of R&D and available literature in this area

1.4.1 Work related to *sandhi* processing

French scholar Gerard Huet[8] has done some significant work in this area. He has built an online program named '**The Sanskrit Reader Companion**' for segmenting and tagging simple Sanskrit phrases. While segmenting, it does *sandhi-viccheda* also but the limitation with this program is that it does simple string segmentation applying string de-concatenation techniques and gives multiple results in many cases. For example, for the sentence '*maarjaarodugdha.mpibati*', it gives 7 solutions i.e. 1. *mārjāraḥ (aḥ+d=od), **dugdham (m+p=ṃp), pibati*** *2. mā (ā+a=ā), arja (a+a=ā), aro, dugdham (m+p=ṃp), pibati 3. mā (ā+a=ā), arja (a+a=ā), araḥ* ***(aḥ+d=od), dugdham (m+p=ṃp), pibati*** *4. mārja (a+a=ā), aro, dugdham (m+p=ṃp), **pibati*** *5. mārja (a+a=ā), araḥ* ***(aḥ+d=od), dugdham (m+p=ṃp), pibati*** *6. mā (ā+a=ā), arja (a+a=ā), aro, dugdham (m+p=ṃp), pibati 7. mā (ā+a=ā), arja (a+a=ā), araḥ* ***(aḥ+d=od), dugdham (m+p=ṃp), pibati.***

Here the drawback is that the splitting of words seems to be based on ad-hoc processing and not on Pāṇinian rules. Besides, it does not seem to be using dictionary to verify the results.

[8] The Sanskrit Heritage Site, Huet, Gerard. http://sanskrit.inria.fr/ (accessed on 10.02.2007).

The Technology Development for Indian Languages (TDIL) program of the Ministry of Information Technology (MIT), Govt. of India, in one of its project named '**Computer Assisted Sanskrit Teaching & Learning Environment**' (CASTLE)[9] funded for Jawaharlal Nehru University, New Delhi claims to have developed a *sandhi- viccheda* system which takes a word as input and returns the constituent words in the DOS environment. But this work is also not available for download anywhere on the TDIL website.

Amba Kulkarni, in her **Anusaaraka** project at Rashtriya Sanskrit Vidyapeetha, Tirupati is also developing a *sandhi* analyzer system. Its methodology is that using the *sandhi* rules, the programme splits the given word into two words and then checks whether the two words are recognized by a morphological analyzer. If any of the words is not recognized, the sandhi split function is called recursively.

There are also some softwares on generative *sandhi*. Huet has built an online program named **The Sandhi Engine** for *sandhi* generation. It generates all the three types of sandhi and in the most cases it gives good result. The engine takes input in roman transliteration and returns output in roman as well as Unicode script. It has separate section for external and internal *sandhi*.[10] Academy of Sanskrit Research (ASR), Melkote claims to have developed a system named '**panini'[11]** which handles the generative *sandhi* and shows *sandhi*'s name and *sūtra*, but the site does not have the system, nor there is any publication detailing the system. **Ganakashtadhyayi**,[12] a Sanskrit software developed by Dr. Shivamurthy Swamiji of Sri Taralabalu Jagadguru Brihanmath, Sirigere (Karnataka), does only some preliminary vowel *sandhi* generation.

1.4.2 Work related to NLP of Sanskrit and other Indian Languages

1.4.2.1 ASR, Melkote

[9] TDIL, MIT, GOI website, http://tdil.mit.gov.in/nlptools/ach-nlptools.htm (accessed on 10.02.2007).
[10] The Sanskrit Heritage Site, Huet, Gerard. http://sanskrit.inria.fr/DICO/sandhier.html (accessed on 10.02.2007).
[11] Academy of Sanskrit Research, Melkote, http://www.sanskritacademy.org/About.htm (accessed on 10.02.2007).
[12] Ganakashtadhyayi, www.taralabalu.org/panini (accessed on 10.02.2007).

ASR, Melkote[13] claims to have been working on development of software tools for NLP in Sanskrit and other Indian languages using insights and logic available in ancient texts. The academy website says it has been working with 20 software tools like **Samskrita Vinodah** and **Adhyapika** (interactive multimedia Sanskrit teaching package), **Janani** (synonym retriever for Sanskrit words), **Kriya** (Sanskrit verb generator), **Semusi** (Subanta generator/analyzer), **Prajna** (*tiṅanta* generator/analyzer), **Chetana** (*kṛdanta* generator/analyser), Bodha (Sentence disambiguation system according to *śābdabodha* of *navya nyāya* system), **Ekadanta vidhya** (Sanskrit speech synthesis), **Pratibha** (MT from Sanskrit to Kannada) and lexicon generators for different domains. The academy claims to have developed a software named **Shabdhabodha**[14] which is said to be an interactive analyzer of semantic and syntactic structure of Sanskrit sentence. This software works on DOS 6.0 or higher with GIST (Graphic based Intelligence Script Technology) shell on Windows 95 platform, which are outdated and not compatible to present versions. The software has two sections: user input and input file. The first section takes a sentence as input and shows the corresponding syntactically compatible sentence and otherwise shows all morphological details while the latter section does the same for an input file.

1.4.2.2 The Sanskrit Heritage Site

Dr. Gerard Huet**,** Director, INRIA[15] has developed various computational tools for Sanskrit, which are available online. The **Declension Engine** takes a nominal base with its gender information as input and gives all the nominal inflectional forms as output. The **Conjugation Engine** is for verb generation. It takes root as input and gives all the possible forms of the verb root in its *ātmane* and/or *parasmai* terminations, in *kartṛ* and *karmaṇi/bhāve* voices in eight *lakāra*-s. **Lemmatiser** and **Sanskrit Readers** are the analyzers. While the Lemmatizer tags a given simple inflected noun or a verb (without *upasarga*-s), the Sanskrit Reader Companion does analysis of a given phrase or a simple sentence, segments it into individual words and tags each word.

[13] Academy of Sanskrit Research, Melkote, http://www.sanskritacademy.org/About.htm (accessed on 20.02.2007).
[14] Language Processing Tools: TDIL website, http://tdil.mit.gov.in/download/Shabdbodha.htm (accessed on 20.02.2007).
[15] The Sanskrit Heritage Site, Huet, Gerard. http://sanskrit.inria.fr/ (accessed: 10.02.2007).

1.4.2.3 CDAC, Banglore

DESIKA,[16] a software developed by Indian Heritage Group of the Centre for Development of Advanced Computing (C-DAC), Bangalore, claims to have developed generation and analysis modules for plain and accented written Sanskrit texts. It has an exhaustive database based on *Amarakośa*, the most popular Sanskrit lexicon, rule base using the grammar rules of Pāṇini's *Aṣṭādhyāyi* and heuristics based on *Nyāya* and *Mimāṃasā śāstras* for semantic and contexual processing. This software is available on the TDIL site but does s*ubanta* generation only.

1.4.2.4 IIT, Kanpur

Anglabharti:[17] It is a multilingual Machine aided Translation (MAT) methodology for translation from English to Indian languages. English is a SVO language while Indian languages are relatively of free word-order. Anglabharti uses a pseudo-interlingua approach. It analyzes English (source language) and generates a pseudo lingua for Indian languages (PLIL) applicable to a group of Indian languages (target languages). A language specific text-generator converts the 'pseudo-target' code into target language text. The translation system has also been interfaced with text-to-speech module and Optical Character Recognition (OCR) input.

AnuBharti:[18] it is an example based MT approach which is designed for translation from Hindi to English and other languages. Here the pre-stored examples form the basis for translation. The translation is obtained by matching the input sentence with the minimum 'distance' example sentence. Hindi like all other Indian languages is a relatively free word-group order language. Here the methodology is that first the input Hindi sentence is analyzed into a standardized form of word-order and this standardized Hindi sentence is matched with standardized example-base of target language. Besides these, IIT, Kanpur is also engaged in development of translation system for bi-lingual text in Hinglish (Hindi mixed with English) and system for speech to speech translation.

---

[16] Desika, http://tdil.mit.gov.in/download/Desika.htm, (accessed on 10.020.2007).
[17] Anglabharti, IIt,Kanpur, http://www.cse.iitk.ac.in/users/langtech (accessed on 20.02.2007).
[18] Anubharti, IIt,Kanpur, http://www.cse.iitk.ac.in/users/langtech (accessed on 20.02.2007).

1.4.2.5 IIIT, Hyderabad

Language Technologies Research Centre (LTRC) at IIIT, Hyderabad is a prominent NLP research centre. LTRC with the collaboration of Govt. of India, Carnegie Mellon University's Language Technology Institute, University of Pennsylvania, HP Labs, Google, TCS and other academic institutions aims at developing technologies related to MT among English and Indian languages, speech processing for Indian languages, search engines, information extraction and retrieval for English and Indian languages. LTRC has developed '**Shakti**'[19] system for MT from English to Indian languages. It combines rule-based approach with statistical approach and currently claims to work for three target languages: Hindi, Telgu and Marathi. Besides this, LTRC is also developing several machine readable bilingual dictionaries, tense aspect modality dictionary, multi-word expressions dictionary for language pairs of English-Hindi, English-Marathi, and English-Bengali. In addition to the above, LTRC is also working on various projects such TTS for Telugu and Hindi, Telgu to Hindi Machine Translation, morphological analyzers for Indian languages, POS tagger for Hindi and Bengali. Its Search and Information Extraction Lab (SIEL) focuses on solving problems in the areas of Information Retrieval and Extraction using NLP techniques. SIEL is currently focusing on applications areas like '**Ask Buddha**' (Web based question answering system in News), General Search Engines, Indian Language Search Engines, Document Categorization, Document Summarization, Information Extraction and Ontologies.

1.4.2.6 IIT, Bombay

Resource Centre for Indian Language Technology Solutions (RCILTS), IIT-Bombay[20], led by Dr. Pushpak Bhattacharya is a happening place for NLP in India. The institution aims to offer information technology through Indian languages, and to develop resource information in Indian languages and Sanskrit in a way relevant to the present day needs. The institution has developed an online Hindi Wordnet which is a lexical database for nearly 60000 Hindi words. It takes input in Unicode *Devanāgarī* fonts and gives different synonyms of the word with their example in Hindi sentence. It also uses an inbuilt

[19] Shakti, LTRC, IIIT, Hyderabad, http://www.iiit.net/ltrc/index.html (accessed on 20.02.2007).
[20] IIT, Bombay, http://www.cse.iitb.ac.in (accessed on 1.04.2007).

keyboard to enter the input.[21] The institution is working on POS taggers for Hindi and Marathi and MT systems among multiple languages with a semantic net like representation called the Universal Networking Language (UNL) as interlingua. This interlingua is based on the concepts of language independent words, relations and attributes which are captured in lexical resources like the wordnet.

1.4.2.7 Rashtriya Sanskrit Vidyapeetha (RSV), Tirupati

RSV, Tirupati has been working on developing linguistic resources for NLP in Sanskrit. Prof. K.V. Ramakrishnamacharyulu and Dr. Srinivasa Varkhedi along with Prof. Vineet Chaitanya and Amba P. Kulkarni have initiated many projects and have developed many tools like *pada-ccheda*, which segregates Sanskrit compound words into its components, which works on Sanskrit ISCII text in Linux environment. Apart from this it is also concentrating on *kṛdanta* and *tiṅanta* analyzers and also generators for *subanta, tiṅanta* and *samāsa*.[22] RSV Tirupati[23] along with C-DAC Bangalore, Ahobila Mutt Sanskrit College Madhurantakam Tamil Nadu, PoornaPrajna Samshodhana Mandiram Bangalore, Chinmaya International Foundation Veliyanad Kerala, ASR Melkote Karnataka, IIIT-H, Dept. of Sanskrit H.S.Gour University, Saugar Madhya Pradesh have combined initiative to develop a large Sanskrit Corpus. RSV, Tirupati also worked on a project of Veda and śāstrārtha recording, funded by the Ford Foundation of USA.

1.4.2.8 RCILTS – Utkal University

RCILTS – Oriya Centre at the Department of Computer Science and Application, Utkal University[24] has been working on the various areas of NLP. The institution has developed an Oriya OCR '**DIVYADRUSTI**' and text-to-speech for Oriya, Hindi and Bengali. It is also working on building Oriya Machine Translation (OMT), Ori-Net (Word-Net for Oriya), parsers, morphological analyzers and spell checkers for Oriya language. Besides these Oriya NLP tools, the centre also claims to have developed Sanskrit Word-Net (San-Net) using Navya-Nyāya philosophy and Pāṇinian Grammar. The system has 300

---

[21] http://www.cfilt.iitb.ac.in/wordnet/webhwn/wn.php
[22] Chandrashekhar, R. 2006, '*Part-of-Speech Tagging for Sanskrit*', submitted for Ph.D degree at SCSS, JNU. P. 12-13
[23] RSV, Tirupati, http://rsvidyapeetha.ac.in, (accessed on 20.02.2007).
[24] RCILTS, Utkal University, http://www.ilts-utkal.org, (accessed on 20.02.2007).

Sanskrit words (250 Nominal words and 50 Verbal words) and it explains synonymy, antonym, hyponymy, hypernymy, holonymy and meronymy relationship of words with their analogy, etymology, and definitions.

1.4.2.9 AU-KBC Research Centre

NLP Group at Anna University KB Chandrashekar (AU-KBC) Research Centre, Madras Institute of Technology, Chennai is mainly working on Tamil NLP. The centre has developed Tamil-Hindi Machine Aided Translation (MAT) system which is based on the model of Anusaaraka and has an accuracy of 75%. It has also developed Tamil morphological analyzer which can handle nearly 3.5 million word forms with more than 95% accuracy. The center has also developed Tamil search engine. All these systems have a demo and online service on their website.[25] The centre is also working on developing MT systems between Tamil and other languages particularly English and Hindi, a Tamil Word-net in collaboration with Dr. S Rajendran of Tamil University, Thajavur and a POS tagger for Tamil.

1.4.2.10 The Sanskrit Library

The Sanskrit Library Project, under the guidance of Dr. Peter M. Scharf, Classics Dept., Brown University, is engaged in philological research in Vedic and Classical Sanskrit Language and literature. It aims to digitalize the oral and written literature of Sanskrit. At present the research is going on computational phonology and morphology, developing OCR for Indic scripts and *Vedic* accentuation.[26]

1.4.2.11 Sanskrit Studies Links and Information

The site is developed to consolidate various links related to Sanskrit. It lists various links related to Sanskrit software, Sanskrit tutorials, *Devanāgrī* fonts and transliteration schemes, collection of essays related to Indian tradition, Sanskrit journals, Sanskrit daily

[25] AU-KBC Research Centre, http://www.au-kbc.org/frameresearch.html (accessed: 15.10.2006).
[26] The Sanskrit Library, http://sanskritlibrary.org/ (accessed: 20.05.07)

audio news sites, Sanskrit dictionaries, oral recording of *Veda* and *Vedic* math, Sanskrit studies in Indian and foreign universities, awards related to Sanskrit etc.[27]

1.4.2.12 Jawaharlal Nehru University (JNU)

The RCILTS – Sanskrit, Japanese, Chinese unit of JNU,[28] under the leadership of Prof. G.V.Singh claims to have designed various modules for web based Sanskrit Language Learning System. The various software modules and language resources that the centre has developed include learning materials on Sanskrit lessons and exercises and lexicons on Sanskrit-English and English-Sanskrit and a lexicon on *Nyāya* terms. The centre has also developed a computational module of *Aṣṭādhyāyī* of Pāṇini and verb analyzer and generator. Girish Nath Jha,[29] as part of his M.Phil. dissertation, has developed a Nominal Inflection Generator for Sanskrit using Prolog. The input for the program is nominal base, its class and gender and it generates all the paradigms of *subanta*.

1.4.2.13 Special Center for Sanskrit Studies (SCSS), JNU

This center at JNU has been doing various R&D for computational Sanskrit under the guidance of Dr. Girish Nath Jha since 2002. The tools developed can be used live at (http://sanskrit.jnu.ac.in).

A project on *Amarakosha* (http://sanskrit.jnu.ac.in/amara/index,jsp), under the guidance of Dr. Girish Nath Jha, has been built up in SCSS, JNU. It is a Multilingual Online project, funded by UGC under UPOE program. The Unicode based software supports seven languages- Sanskrit, Hindi, Kannada, Punjabi, Bangla, Oriya and English and allows the user to search the synonym from one language to another. The output displays the grammatical and semantic category of the word, its base word, reference and ontological information. The software also provides the facility to enter and edit the data by language experts. The software will be extended as a multilingual interface, search engine and text processing tool.

[27] Sanskrit Studies Links and Information, http://www.sanskritlinks.blogspot.com/ (accessed on 05.07.2007)
[28] RCILTS, JNU, http://tdil.mit.gov.in/SanskritJapaneseChinese-JNUJuly03.pdf (accessed on 20.02.2007).
[29] Jha, Girish Nath. 1993, '*Morphology of Sanskrit Case Affixes: A Computational Analysis*', M.Phil. submitted to JNU, New Delhi.

R.Chandrashekhar[30], as part of his Ph.D. thesis, has developed a POS tagger for *sandhi*-free classical Sanskrit prose text which is an online system run on Apache Tomcat platform using Java Servlet. The system will be the basic requirement for the further R&D on the Sanskrit-Indian Languages MT Systems.

Subash Chandra,[31]as part of his M.Phil. dissertation, has developed a Sanskrit *subanta* Recognizer and Analyser System which is an online system on Apache Tomcat platform using Java Servlet. The system uses a hybrid approach of Pāṇinian formalism and example-based techniques and gives a comprehensive computational analysis of *subanta-padas* in a (*sandhi-rahita*) Sanskrit text of *Devanāgari* script and does basic tagging of verbs and *avyayas* too. The system can be used for larger processing of Sanskrit, text simplification and MT. The system claims to give an average accuracy of 91.65% accuracy, tested on some selected simple Sanskrit prose texts.

Sudhir Kumar Mishra[32], a Ph.D. scholar, is working on a *Kāraka* Analyzer for *Laukika* Sanskrit prose text based on Pāṇini and Kātyāyana *Kārak*a formulations. This work will be an important component in syntactico-semantic analysis of Sanskrit and thus will be useful in various NLP applications for Sanskrit.

In addition to the above, research works are also going on in the areas like learning Sanskrit language using e-learning approach[33], computational identification and analysis of Sanskrit verb-forms using reverse Pāṇinian techniques as well as example base[34],

---

[30] Chandrashekhar, R. 2006, '*Part-of-Speech Tagging for Sanskrit*', submitted for Ph.D degree at SCSS, JNU.
[31] Chandra, Subash. 2006. '*Machine Recognition and Morphological Analysis of Subanta-padas*', submitted for M.Phil degree at SCSS, JNU.
[32] Mishra, Sudhir Kumar & Girish Nath Jha. 2004, '*Sanskrit Karaka Analyser for Machine Translation*', In the proceedings of ISTRANS-2004, New Delhi, pp. 224-225.
[33] Bhowmik, Preeti & Jha, Girish Nath. 2006, '*Sanskrit Language Pedagogy: an e-learning approach*', In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, p. 150.
[34] Agrawal, Muktanand. 2006, '*Computational Identification and Analysis of Sanskrit Verb-forms*', In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, pp. 126-127.

online indexing of *Ādiparva* of Mahābhārata,[35] computational analysis of gender in Sanskrit noun phrases for MT and analysis of derived nouns in Sanskrit.[36]

Besides the above mentioned centres, the following institutions/organzations/companies are actively engaged in NLP R&D for Indian languages- Thapar Institute of Engineering and Technology, Patiala, Banasthali Vidyapeeth, Rajasthan, Malaviya Centre for Information Technology Localization, BHU, Varanasi, Indian Statistical Institute, Kolkatta, Microsoft India, IBM, HP Lab, HCL, Webdunia etc.

[35] Mani, Diwakar, & Jha, Girish Nath. 2006, '*Online indexing of Ādiparva of Mahābhārata'*, In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, p. 125.
[36] Singh, Surjit Kumar & Jha, Girish Nath. 2006, '*Strategies for Identifying and Processing Derived Nouns in Sanskrit*', In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, p. 131.

Chapter – II

Sandhi formalism of Pāṇini

2.1 System of Pāṇini

Pāṇini's grammar *Aṣṭ.* (approximately 7th BCE) is important for linguistic computation for two reasons. One, it provides a comprehensive and rule based account of a natural language in about 4000 rules - the only complete grammatical account of any language so far. Two, the model of a 'grammar-in-motion' that it provides seems to closely mimic a fully functional Natural Language Processing (NLP) system -

SOUND CLASSES (phonetic module)
|
RULE-BASE (parser/grammar module)
|
LEXICONS (lexical interface modules)

The possibility that a Natural Language (NL) parser based on Pāṇini can help analyze Indian languages has gained momentum in recent years.[37]

The core of Pāṇinian grammar is a set of statement called *sūtra (*rule*)*. A *sūtra* is a statement in a formula form which is brief but unambiguous, concise but comprehensive, impersonal and objective.[38] These *sūtras* are of six types[39]: *samjña (*definitional rule), p*aribhāṣā* (metarule), v*idhi* (operational rule), *niyama (*restriction rule*) atideśa* (extension rule) and *adhikāra* (heading rule). *Aṣṭ.* contains around 4000 *sūtras* which are described in eight chapters (*adhyāya*) of four sub-chapters (*pāda*) each. Rama Nath Sharma[40] summarizes the topics discussed in different sections of the *Aṣṭ.* as follows:

Book I

a. major definitional and interpretational rules
b. rules dealing with extension *(atideśa)*
c. rules dealing with ā*tmanepada-parasmaipada*
d. rules dealing with the *kārakas*

Book II

a. rules dealing with compounds

[37] Jha, Girish Nath. '*The System of Panini*' http://www.languageinindia.com/feb2004/panini.html
[38] alpākṣaramasandhigdhaṃ sārvadviśvatomukham
astobhamanavadyaṃ ca sūtraṃ sūtravido viduḥ
[39] saṃjñā va paribhāṣā ca vidhirniyama eva ca
atideśo'dhikāraśca ṣadvidha sūtralakṣaṇam
[40] Sharma, Rama Nath.2002, '*The Aṣṭādhyāyī of Pāṇini*', New Delhi: Munshiram Manoharlal Publishers Pvt. Ltd., pp.75-76

b. rules dealing with nominal inflection
c. rules dealing with number and gender of compounds
d. rules dealing with replacements relative to roots
e. rules dealing with deletion by *luk*

Book III

a. rules dealing with derivational of roots ending in affixes *san* etc.
b. rules dealing with the derivational of ending in a *kṛt*
c. rules dealing with the derivational of ending in a *tiṅ*

Book IV

a. rules dealing with derivation of a *pada* ending in a *sup*
b. rules dealing with feminine affixes
c. rules dealing with the derivational of nominal stems ending in an affix termed *taddhita*

Books V, VI & VII

a. rules dealing with doubling
b. rules dealing with *samprasāraṇa*
c. rules dealing with the *saṁhitā*
d. rules dealing with the augment (*āgama*) *suṭ*
e. rules dealing with accents
f. rules dealing with phonological operations relative to a presuffixal base (*aṅga*)
g. rules dealing with operations relative to affixes augments etc.

BookVIII

a. rules dealing with doubling (*dvitva*) relative to a *pada*
b. rules dealing with accent relative to a *pada*
c. rules dealing with other phonological operations relative to a *pada*
d. rules dealing with miscellaneous operations relative to a non-*pada*

2.1.1 śiva sūtras or pratyāhāra sūtra

śiva sūtras or *pratyāhāra sūtra* is a set of 14 *sūtras.* Pāṇini uses these *sūtras* to generate *pratyāhāras* (abbreviatory terms). The use of these *pratyāhāras* is to build phoneme-cluster which he uses to economically specify in the domain of application of various rules. These 14 *sūtras* are: 1. *a i u Ṇ 2. ṛ ḷ K 3. e o Ṅ 4. ai au C 5. h y v r Ṭ 6. la Ṇ 7. ñ m ṅ ṇ n Ṃ 8. jh bh Ñ 9. gh ḍh dh ṣ 10. j b g ḍ d ś 11. kh ph ch ṭh th c ṭ t V 12. k p Y 13. ś ṣ s*

R 14. h L. These *sūtras* consist of 42 letters (*varṇas*) - nine vowels (*svara*) and thirty three consonants (*vyañjanas*). The detailed analysis of this alphabet is as follows:

- Vowels

  The Sanskrit alphabet has nine primary vowels which consist five simple vowels (a, i, u, ṛ, ḷ) and four dipthongs or *sandhyakṣara* (e, ai, o, au). Again these vowels, according to length, are classified into short (*hrsva*), long (*dīrgha*) and prolated (*pluta*). Vowels are further classified into acute (*udātta*)[41], grave (*anudātta*)[42] and circumflex (*svarita*).[43] The acute vowel is produced from the upper part of the organ, the grave vowel from the lower and the circumflex means the combination of two. This accentuation is found in the Vedic literature only and has been lost in classical Sanskrit. Each of these vowel may again be divided into two kinds- nasalized or *anunāsika*[44] (which is pronounced through both the mouth and nose) and non-nasal or *ananunāsika* (pronounce only through mouth). On the above divisions, each of /a/, /i/, /u/, /ṛ/ has 18 modifications and /ḷ/ and the dipthongs (e, ai, o, au) have 12 modifications of each.

- Consonants

  The consonants are divided into three categories- stops (*sparśa*), semivowel (*antaḥstha*) and sibilants (*ūṣman*).

  The term stops refer to the sound which is produced by a complete closure in the vocal tract or complete contact of the tongue with the organ of pronunciation. The stops sound, according to the organ of pronunciation, are divided into five groups- *kavarga* (k, kh, g, gh, ṅ), *cavarga* (c, ch, j, jh, ñ), *ṭavarga* (ṭ, ṭh, ḍ, ḍh, ṇ), *tavarga* (t, dh, d, dh, n), *pavarga* (p, ph, b, bh, m).

[41] uccairudāttaḥ, 1.2.29
[42] nīcairanudāttaḥ, 1.2.30
[43] samāhāraḥsvaritaḥ, 1.2.31
[44] mukhanāsikāvacano'nunāsikaḥ, 1.1.8

The term *antaḥstha* means occupying an intermediate position between a consonant and vowel. /y/, /v/, /r/, /l/ are *antaḥstha* sounds. Phonologically, these sounds are consonants because their role in syllables is the same as of consonants. But phonetically, they are vowel-like in character, because they lack the friction or closure normally associated with consonants.

The sibilants are four are in number. The difference of sibilants with stops is that the sibilants are less fricative because they are produced through an open position of mouth. /ś/, /ṣ/, /s/, /ḥ/ are sibilants sounds. Except of these sounds, there are other sounds also which are not listed in *śivasūtras*, but they are used in Sanskrit language. These sounds are: *visarga, jihvāmūlīya, upadhmānīya*, *anusvāra* and four *yamas.* These sounds are collectively called *ayogāvaha.*

2.1.2 The Place and Manner (*uccāraṇa sthāna and prayatna*)

Pāṇini, in order to define *savarṇa* (homophonic letters), describes the place and manner of articulation. *Savarṇa* means letter belonging to the same category of letters which have the same place and manner of articulation.[45] For example, the eighteen varieties of /a/, due to its short, long and prolated nature and also due to its accents and nasalization, are *savarṇa* to one another, but these vowels are not *savarṇa* to the consonants even if they may have the same place of articulation.[46]

The place of articulation means the point of contact in the vocal tract where obstruction between active and passive articulators occurs and they give a shape to the air stream into a sound. The places of articulation for Sanskrit sounds are: Velar (*kaṇṭha*), Palate (*tālu*), Retroflex (*mūrdhā*), Dental (*danta*), Labial (*oṣṭha*), Velar-palatal (*kaṇṭha- tālu)*, Velar-labial (*kaṇṭha- oṣṭha)*, Labio-dental (*danta-oṣṭha)*, Tounge root (*jihvāmūla*) and Nasal (*nāsikā*).

[45] tulyāsyaprayatnaṃ savarṇam, 1.1.9
[46] nā’’jjhalau, 1.1.10

The manner of articulation means the arrangement of different speech organs in producing a sound. It includes the type of closure of trachea, the degree of obstruction of the air stream by the articulators and the flow of the air. This manner for Sanskrit sounds is of two types: buccal (*ābhyantara prayatna*) and extra-buccal (*bāhya prayatna*). Buccal manner means the degree of obstruction of the air stream by the articulators while extra-buccal means the type of closure of trachea and the flow of the air. Buccal manner is of 5 types: stops (*spṛṣṭa*), slight closed (*īṣat spṛṣṭa*), slight open (*īṣata vivṛta*), closed (*saṃvṛta*) and open (*vivṛta*) while extra-buccal manner is of 11 types: lax (*vivāra*), tense (*saṃvāra*), breath (*śvāsa*), resonance (*nāda*), voiced (ghoṣa), unvoiced (*aghoṣa)*, unaspirated (*alpaprāṇa*), aspirated (*mahāprāṇa*), acute (*udātta)*, grave (*anudātta*)and circumflex (*svarita)*.

Manner → / Place ↓	open	lax, breath, unvoiced, unaspirated stops	lax, breath, unvoiced, aspirated, stops	tense, resonant, voiced, unaspirated stops	tense, resonant, voiced, aspirated, stops	nasal, stops	slight closed, unaspirated	slight open, aspirated	closed	ayogāvaha
Velar	a (18)	K	kh	g	gh	ṅ		h	a	ḥ
Palatal	i (18)	C	ch	j	jh	ñ	y	ś		
Retroflex	ṛ (18)	ṭ	ṭh	ḍ	ḍh	ṇ	r	ṣ		
Dental	ḷ (12)	T	th	d	dh	n	l	s		
Labial	u (18)	P	ph	b	bh	m				upadhmānīya
Velar-palatal	e,ai (12)									

Velar-labial	o, au (12)									
Labio-dental							v			
Tounge root										jihvāmūlīya
Nasal										ṃ, yama

Table 2.1: chart of place and manner of articulation

2.2 Sandhi

The word *sandhi* refers to a wide variety of phonological changes at morpheme or word boundary in which two letters combine and they have certain changes. Pāṇini, in *Aṣṭ.*, has not used the word '*sandhi*', instead he uses the word '*saṃhitā*' which is defined as '*paraḥ sannikarṣaḥ saṃhitā*'[47]. *Saṃhitā* means the close proximity of two letters either within a word or between two words which results into the natural phonetic combination of these letters. In the words of Vidhata Mishra, "*when the vedic hymns or the running prose passage of the yajurveda were split up into their different constituent parts namely the words of padas by the padakāras, the word saṃhitā or saṃhitāpāṭha came into use as contrasted with the the padapāṭha. Saṃhitā in the sense of 'putting together' (sam 'together' and 'dhā 'to put') occurs in Ṛgveda Prātiśākhya (2.2.2). In its technical sense of 'the closest connection of words or their parts' it is first found in Nirukta (1.1.17).*[48] While distinguishing *sandhi* and *saṃhitā*, Śripatidatta, in *vṛtti* of *kātantra pariśiṣṭa* (1.95)[49] says that the coming together of the letters without any intervening vowel and consonant and the pronouncing immediately after the other without an interval of even half a mora (*mātrā*) is *saṃhitā*. And when letters come in this way, *sandhi* takes place.[50]

[47] aṣṭādhyāyī, 1.4.108
[48] Mishra, Vidhata. 1972, '*A Critical Study of Sanskrit Phonetics*', Varanasi : The Chowkhamba Sanskrit Series Office, p. 96
[49] varṇāntarāvyavahitayodvayovarṇayoḥ susannikarṣo bhavati | sa hi nitratiśayamānantaryam | ardhm ā tr ākālenavyavāyaḥ saṃhitocyate | pāṇī kuṇḍe nadhau vadhvau plavate gāyati | saṃitāyāmeva sandhyaḥ syuḥ|
[50] Ibid. no. 12, p. 96

2.2.1 Sandhi: morphophonological or morpholexical alternation

W.S.Allen[51] says that there are two main types of alternation: morphophonological and morpholexical. In the first, the variation is determined by the phonetic environment while in the second type, the alternation depends not upon any phonetic environment, but upon the selection of neighbouring morphemes without regard to their phonetic form. He explains these two types of alternation from Sanskrit by illustrating the different past participal forms *matta* and *panna*. He says that it is simply the selection of the roots *mad*- and *pad*- respectively that determines whether the suffix shall be /ta/ or /na/, i.e. the /ta/ or /na/ alternation is morholexical. On the other hand, the fact that the root takes the form *mat*-in *matta* and *pan*-in *panna,* is a matter of morphophonological alternation, being determined by the nature of the suffixial initial, viz. /t/ in the one case and /n/ in the other.[52] One more example from the infixation process in the verb also fits here. In the verb forms *bhavati* and *karoti,* it is the selection of the root *bhū* (*bhvādigaṇa*) and *kṛ* (*tanādigaṇa*) that determines respectively whether the infix will be /a/ (śap) or /u/. This is morpholexical alternation. On the other hand, the process in which the /ū/ of *bhū* changes to /o/$\rightarrow$/av/ and /ṛ/ of *kṛ* changes to /ar/ in *karoti,* is a matter of morphophonological alternation because it is being determined by internal process of euphonic combination.

Sandhi is related with the morphophonological alternation. The main use of *sandhi* is to make ease of pronunciation in speech. For example, final voiceless stop, in Sanskrit, is followed by a voiceless initial and a voiced is followed by a voiced initial which means that there is a less complexity in the pronunciation from the final to the initial letter than there would be a voiceless final were followed by a voiceless initial or vice versa.[53]

2.2.2 External and Internal sandhi

External and internal are the two processes of *sandhi*. Internal process governs the combination of suffix with root or stem in declension, conjugation and derivation. In its external process, the rules of *sandhi* determine the changes of final and initial letters of

[51] Allen, W.Sidney. 1972, '*Sandhi : The Theoretical, Phonetic and Historical Bases of Word-Junction in Sanskrit*', Mouton, The Hauge, pp 13-15
[52] Ibid. no. 15, pp. 14-15
[53] Ibid. no. 15, pp. 15

words in a sentence and also the final and initial letters of the components of compounds (*samāsa*). So it is difficult to understand Sanskrit sentence without knowing these rules. This gives a very practical importance to the subject of *sandhi*.

Generally the rules of internal *sandhi* agree with the rules of external *sandhi*, but on some occasions they have exceptions too. For example, final /i/or /ī/, /u/ or /ū/, /ṛ/ or /ṝ/ and /ḷ/, if followed by vowel or dipthongs, are generally changed to /y/,/v/,/ar/,/al/ respectively, but in declension or conjugation, /i/or /ī/, /u/ or /ū/, /ṛ/ or /ṝ/ are changed to /iy/, /uv/, /ir/ respectively. For example: *bhū+i= bhuvi, gṝ+ati=girati*.[54]

2.2.3 Types of sandhi

Bhaṭṭoji Dikṣita, in *Sid. Kau.*, describes *sandhi* in five *prakaraṇa*: *ac sandhi prakaraṇa, prakṛtibhāva prakaraṇa, hal sandhi prakaraṇa, visarga sandhi prakaraṇa,* and *svādī sandhi prakaraṇa*. *Ac sandhi* deals with changes which occur at the combination of two vowels. For example, *rāmasya+ācāryaḥ= rāmasyācārya*. *Prakṛtibhāva sandhi* means that there is no euphonic modification in a vowel even if it is immediately followed by a vowel. These vowels are called *pragṛhya*. For example: *harī+etau = harī etau. Hal sandhi* means the changes which occur when a consonant or vowel combine with a consonant. For example, *rāmas+cinoti=rāmaścinoti.* When the vowel or consonant sounds join with the *visarga*, then the *visarga* undergoes some changes. This is called the *visarga sandhi*. For example: *hariḥ+avadat=hariravadat. Svadī sandhi* includes the changes which occur when nominal bases are joined with case terminations. For example, *śiva su+arcyaḥ= śiva u (su-ru-u) arcyaḥ= śivo arcyaḥ.*

The main characteristics of Sanskrit euphonic rules[55] and their classification are as follows:

- Assimilation

[54] Max Muller, F. 1977, '*A Sanskrit Grammar*', New Delhi: Asian Educational Series,, p.53
[55] शर्मा, देवीदत्त. 1974, '*संस्कृत का एतिहासिक एवं संरचनात्मक परिचय*', चण्डीगढ: हरियाणा साहित्य अकादमी, पृष्ठ 80-86

Assimilation refers to the influence exercised by one sound segment upon the articulation of another, so that the sound becomes more alike or identical. For example: palatization and retroflexation. If dental sounds are followed by palatal or retroflex sounds, the dental sound is changed to palatal or retroflex respectively. For example: *tat+ca=tacca, tat+ṭīkā=taṭṭīkā.* Also the different type of assimilation of vowels like *tav+indraḥ=tavendraḥ, tau+iti=tāviti.* Similarly the sound /m/ in the end of a word (*padānta*) is changed to /ṃ/ (*anusvāra*) and this *anusvāra* (followed except by semivowels and sibilants) is changed to the nasal sound of the next stop sound. For example:*tvam+karoṣi=tavaṅkaroṣi,aham+karomi=ahaṅkaromi,śīgram+calati=śī grañcalati*.

- Voicing

  If an unvoiced sound at word boundary is followed by a voiced sound, then the unvoiced sound is changed to the voiced sound. For example: *jagat+īśaḥ=jagadīśaḥ, vāk+jālam=vāgjālam, samrāṭ+gacchati=samrāḍgacchati*

- Devoicing

  If a voiced sound at the end of a word is followed by an initial unvoiced sound of the next word, voiced sound is changed to unvoiced sound. For example: *vipad+su=vipatsu, ud+sthānam=utthānam*

- Nasalization

  If the unvoiced sound is followed by the nasal sound, the unvoiced sound is changed to the nasal sound of its own group. This is optional with voicing but if the following nasal sound is /m/ and that is a part of a suffix, then this nasalization is compulsory. For example: *etat+manuṣya=etanmanuṣya, tat+mayam=tanmayam, vāk+mātram=vāṅmātram*

- Deaspiration

If the aspirated sound in the end of a word is followed by the unaspirated sound of a next word, the aspirated sound is changed and this aspiration is transformed. For example: Ex. *labh+taḥ=labdhaḥ, rundh+thaḥ=runddha*ḥ

- Visarga

  In Sanskrit, *visarga* in the end of a word, followed by different sound, is changed to different forms. For example: *namaḥ+karoti=namaskaroti, puruṣaḥ+asti=puruṣo'sti, rāmaḥ+āyāti=rāma āyāti, devaḥ+icchati=deva icchati, aśvāḥ+ime=aśvā ime*

- Doubling

  An aspirated sound between vowel sounds is doubled and this doubled sound is unaspirated sound of that sound class. For example: *pari+bhuja=paribbhuja, a+khidate=akkhidate, san+atra=snnatra, pratyaṅ+ātmā=pratyaṅṅātmā*

- Vowel lengthening

  When the two similar simple vowels (a, i, u, ṛ, ṝ) come together, their long form replaces both of them. For example: *namasi+īśvaram = namasīśvaram, muninā+atāmi.*

2.3 Vowel Sandhi

Vowel *sandhi* (*ac sandhi*) occurs at the combination of two vowels. The main characteristic of this *sandhi* is absence of hiatus because two contiguous vowels, in the formation of a word, in compound (*sāmāsa*) and in sentence, influence each other mutually and lead to different kinds of resultant changes.

2.3.1 Types of vowel sandhi

- *yaṇ sandhi*

when /i/, /u/, /ṛ/, /ḷ/ are followed by /a/, /i/, /u/, /ṛ/, /ḷ/, /e/, /o/, /ai/, /au/, they are replaced by /y/, /v/, /r/, /l/ respectively.[56] The following chart shows the different combinations and examples of *yaṇ sandhi*:

Table 2.2: outline of forward *yaṇ sandhi*

Word Initial Letter → / **Word Final Letter ↓**	a / ā	i / ī	u / ū	ṛ / ṝ	ḷ	e / ai	o / au
i / ī	-ya- / -yā- (*dadhi+atra = dadhyatra*)	--	-yu- / -yū-	-yṛ- / -yṝ-	-yḷ-	-ye- / -yai-	-yo-/ -yau-
u / ū	-va- / -vā- (*madhu+ariḥ= madhvariḥ)*	-vi- / -vī- (*madhu+iva= madhviva*)	--	-vṛ- / -vṝ-	-vḷ-	-ve- / -vai-	-vo- / -vau-
ṛ / ṝ	-ra- / -rā- (*dhātṛ+aṃśaḥ= dhātraṃśaḥ*)	-ri- / -rī-	-ru- / -rū- (*kartṛ+uta = kartruta*)	--	--	-re- / -rai-	-ro- / -rau-
ḷ	-la- / -lā- *(ḷ+ākṛti=lākṛti)*	-li- / -lī-	-lu- / -lū	--	--	-le- / lai-	-lo- / -lau

- *ayādi sandhi*

  when /e/, /o/, /ai/, /au/ are followed by a vowel, they are replaced by /ay/, /av/, /āy/, /āv/ respectively.[57]

Table 2.3: outline of forward *ayādi sandhi*

Word Initial Letter →	Any vowel

[56] iko yaṇaci, 6.1.74, sthāne'ntaratamaḥ, 1.1.49

[57] eco'yavāyāvaḥ, 6.1.75

Word Final Letter ↓	
e	-ay + following vowel- (*sakhe+iha=sakhayiha*)
o	-av + following vowel- (*prabho+ehi=prabhavehi*)
ai	-āy + following vowel- (*śriyai+arthaḥ= śriyāyarthaḥ*)
au	-āv + following vowel- (*tau+iti=tāviti*)

Besides the above general rule, there are some supplementary rules and *vārttika* also which explain the words which can not be explained through the above general rule.

If /o/, /au/ are followed by suffix beginning with /y/, they are also changed to /av/, /āv/ respectively.[58] For example: *go+yam = gavyam, nau+yam = nāvyam*

If the vowel /o/ of the word 'go' is followed by suffix 'yūtiḥ', /o/ changes to /av/.[59] For example: *go+ yūtiḥ= gavyūtiḥ*

If the root *kṣi* and *ji* is followed by the suffix *yat* in the sense of (to be capable of), both *kṣi (kṣe) and ji (je)* will have /ay/ *ādeśa*.[60] For example: *kṣi+yam = kṣayyam, ji+yam = jayyam*

The root *krī* (in the sense of 'to be bought'), has /ay/ *ādeśa.*[61] For example: *krī+ yam = krayyam*

[58] vānto yi pratyaye , 6.1.76
[59] adhvaparimāṇe ca, vārttika (henceforth vā.)
[60] kṣayyajayyau śakyārthe, 6.1.78
[61] krayyastadarthe, 6.1.79

- *guṇa sandhi*

  If /a/ or /ā/ is followed by a vowel (i, u, ṛ, ḷ), both initial and following vowels are changed to *guṇa* vowel (a, i, u)[62]

Table 2.4: outline of forward *guṇa sandhi*

Word Initial Letter → / Word Final Letter ↓	i / ī	u / ū	ṛ / ṝ	ḷ
a / ā	-e- (*tava+indraḥ= tavendraḥ)*	-o- (*sā+uktā= soktā*)	-ar- (*sā+ṛdhiḥ= sardhiḥ)*	-al- (*tava+ḷkāraḥ = tavalkāraḥ)*

The above rule has an exception too. If a root beginning with /ṛ/ is preceded by a preposition ending in /a/ or /ā/, the two vowels are merged to /ār/ instead of /ar/.[63] This root may be *nāmdhātu* also.[64] For example: *upa+ṛcchati = upārcchati, pra+ṛṣabhīyati = prārṣabhīyati.*

- *vṛddhi sandhi*

  If /a/ or /ā/ are followed by /e/, /o/, /ai/, /au/, then both initial and following vowels are changed to *vṛddhi* vowel (ā, ai, au).[65]

Table 2.5: outline of forward *vṛddhi sandhi*

Word Initial Letter → / Word Final Letter ↓	e / ai	o / au

[62] ekaḥ pūrvaparayoḥ, 6.1.81, ād guṇaḥ, 6.1.84
[63] upasargādṛti dhātau, 6.1.88
[64] vā supyāpiśaleḥ, 6.1.89
[65] ekaḥ pūrvaparayoḥ, 6.1.81, vṛddhireci, 6.1.85

a / ā	-ai- (*tava+eva=tavaiva*)	-au- (*tava+oṣṭhaḥ=tavauṣṭhaḥ)*

- *dīrgha sandhi*

  If /a/, /i/, /u/, /ṛ/, /ḷ/ are followed by *savarṇa* vowel, both are replaced with their long vowel.[66]

Table 2.6: outline of forward *dīrgha sandhi*

Word Initial Letter →	a / ā	i / ī	u / ū	ṛ / ṝ	ḷ
Word Final Letter ↓					
a / ā	-ā- (*uktvā+apagac chati= uktvāpagacchat i*)				
i / ī		-ī- (*nadī+īdṛśī= nadīdṛśī)*			
u / ū			-ū- (*kiṃtu+udeti = kiṃtūdeti)*		
ṛ / ṝ				-ṝ- (*kartṛ+ṛju=kartṝju*)	
ḷ					ṝ

There are two *vārttikas* available on the above rule.

If /a/, /i/, /u/, /ṛ/, /ḷ/ are followed by *savarṇa* /ṛ/ or /ḷ/, both the initial and following vowels are optionally replaced by /rṛ/[67] and /lḷ/.[68] For example: *hotṛ+ṛkāraḥ= hotrṛkāraḥ (or hotṝ kāraḥ), hotṛ+ḷkāraḥ = hotlḷkāraḥ (hotṝkāraḥ)*

- *pūrvarūpa sandhi*

[66] ekaḥ pūrvaparayoḥ, 6.1.81, akaḥ savarṇe dīrghaḥ, 6.1.97

[67] ṛti savarṇe ṛ (rṛ) vā, (vā.)

[68] ḷti savarṇe ḷ (lḷ) vā, (vā.)

If /e/ or /o/ in the end of a word is followed by a short /a/, then /e/ or /o/ replaces the both and the absence of /a/ is shown by ’ (*avagraha*).[69]

Table 2.7: outline of forward *pūrvarūpa sandhi*

Word Initial Letter → / **Word Final Letter ↓**	a
e	-e’- (*hare+ava=hare’va*)
o	-o’- (*viṣṇo+ava= viṣṇo’va*)

- *pararūpa sandhi*

  If a preposition ending in /a/, /ā/ is followed by a verb beginning with vowel /e/ or /o/, then /e/ or /o/ respectively replaces both the initial and following vowels.[70]

Table 2.8: outline of forward *pararūpa sandhi*

Word Initial Letter → / **Word Final Letter ↓**	e (initial letter of verb)	o (initial letter of verb)
a / ā	-e- (*pra+ejate=prejate*)	-o- (*upa+oṣati=upoṣati*)

The rule has some supplementary rules and *vārttika* too.

If the initial vowel /o/ of the words *otuḥ* and *oṣthaḥ* in a compound is preceded by /a/ or /ā/, the two vowels may coalesce into /au/ (*vṛddhi*) or /o/ (*pararūpa*) optionally.[71] For

[69] ekaḥ pūrvaparayoḥ, 6.1.81, eṅaḥ pad āntādati, 6.1.105
[70] ekaḥ pūrvaparayoḥ, 6.1.81,eṅi pararūpam , 6.1.91

example: *adhara+ oṣthaḥ=adharoṣthaḥ/ adharauṣthaḥ, sthūla+otuḥ = sthūlautuḥ/ sthūlotuḥ. This vārttika* is only applicable in case of compound; otherwise it will follow *vṛddhi sandhi*. For example: *tava+oṣthaḥ=tavauṣthaḥ*

If /a/ is followed by the word *om* and prefix ā (āṅ), /a/ will be merged to the following vowel.[72] For example: *śivāya+om namaḥ = śivāyom namaḥ, śiva+ehi(ā+ihi) = śivehi*

The *pararūpa sandhi* has an exception too. If /a/ is followed by the verb *eti, edhati* and *ūhaḥ(ūṭh),* the *vṛddhi sandhi,* instead of *pararūpa or guṇa,* takes place.[73] For example: *upa+eti = upaiti, upa+edhate = upaidhate, praṣṭha+ūhaḥ= praṣṭhauhaḥ*

2.3.2 Exceptions of vowel *sandhi*

There are certain cases in which the final vowel is not liable to any *sandhi* rules. These vowels are called *pragṛhya*. They remain unchanged even if followed by a vowel.[74] This non- changeability is called *prakṛtibhāva sandhi*. In the following rules *prakṛtibhāva sandhi* (fully or optionally applicable) is described.

If /i/, /u/, /ṛ/, /ḷ/, in the end of a word (except in compound)[75] are followed by a vowel (except of /i/, /u/, /ṛ/, /ḷ/), the *sandhi (yaṇ sandhi*) can be omitted there but in that case the long final vowel will be shortened.[76] For example: *cakrī+atra = cakri atra/ cakryatra*

Similarly if /a/, /i/, /u/, /ṛ/, /ḷ/, in the end of a word and in compound, is followed by /ṛ/, the *guṇa sandhi* can be omitted there and the long vowel will be shortened.[77] For example: *brahmā+ṛṣiḥ = brahma ṛṣiḥ/ brahmarṣiḥ, sapta+ṛṣīṇām = sapta ṛṣīṇām/ saptarṣīṇām*

---

[71] otvoṣṭhayoḥ samāse vā (vā.)
[72] omāṅośca, 6.1.92
[73] etyedhatyūṭhsu, 6.1.86
[74] plutapragṛhyā aci nityam , 6.1.121
[75] na samāse (vā.)
[76] iko'savarṇe śākalyasya hrasvaśca, 6.1.123
[77] ṛtyakaḥ, 6.1.124

If the word '*go*' in the end of a word is followed by a short /a/, the *pūrvarūpa sandhi* can be dropped there and there may be *prakṛtibhāva sandhi* optionally.[78] For example: *go+agram = go agram/ go'gram*. There may also be *ava (avaṅ)* ādeśa to '*go'*.[79] For example: *go+agram = gavāgram*. This *ava (avaṅ)* ādeśa will be compulsory if the word '*go*' is followed by the word '*indra'*.[80] For example: *go+indraḥ=gavendraḥ.*

Final /ī/, /ū/, /e/ of a word in dual numbers are *pragṛhya* and so remains unchanged when followed by a vowel.[81] This is an exception to *yaṇ sandhi*. For example: *harī+etau = harī etau, viṣṇū+imau = viṣṇū imau, pacete+imau = pacete imau*

The terminations of the word *amī* and *amū*, respectively (nominal, plural, masculine) and (nominal, masculine) of the pronoun *adas* is *pragṛhya* and so it is not liable to any euphonic change.[82] For example: *amī+ī*śāḥ = *amī ī*śāḥ, *am*ū+*ī*śāḥ = *am*ū *ī*śāḥ

Indeclianbles consisting of a single vowel (except of /ā/) are *pragṛhya* and so not liable to any *sandhi* rule.[83] For example: *i+indraḥ = i indraḥ, u+umeśaḥ = u umeśaḥ*

The final /o/ of indeclinable is *pragṛhya.[84]* There are six indeclinables ending in /o/- *o, āho, utāho, ho, aho, atho*. For example: *aho+apehi = aho apehi*

The termination /ī/, /ū/ of the word (ending in 7th case marker) are *pragṛhya.[85] For* example: *somo gaurī (ī)+adhiśritaḥ = somo gaurī adhiśritaḥ*

Indeclinables ending in single vowel /u/ (uñ), followed by *iti* is optionally *pragṛhya*. This /u/ may also be changed to nasalized /u/ also.[86] For example: *u+iti = u iti /viti/* ũ*iti*

---

[78] sarvatra vibhāṣā goḥ, 6.1.118
[79] avaṇ sphoṭāyanasya, 6.1.119
[80] indre ca, 6.1.120
[81] īdūded dvivacanaṃ pragṛhyam, 1.1.11
[82] adaso māt, 1.1.12
[83] nipāta ekājanāṅ, 1.1.14
[84] ot, 1.1.15
[85] īdūtau va saptamyarthe, 1.1.18
[86] uña ũ, 1.1.17

If the vowel /u/ (uñ) (preceded by *pratyāhāra may*) is followed by any vowel, /u/ changes to /v/ or remains *pragṛhya optionally*.[87] For example: *kimu+uktam = kimu uktam/ kimvuktam.*

[87] maya uño vo vā, 8.3.33

<u>Chapter – III</u>

Lexical Resources for Reverse Sandhi Analysis

3.1 Introduction

The present chapter describes the lexical resources needed to develop a vowel *sandhi* analyzer which will analyze a Sanskrit text according to Pāṇinian formalism. This chapter also describes a reverse algorithm to automatically segment words which are combined together according to vowel *sandhi* rules.

3.2 Viccheda patterns

Viccheda patterns are the formalization of the *sandhi* rules in the reverse format. The primary basis of this reverse rule base is *ac sandhiprakaraṇa* of *Sid. Kau.*. But each and every rule of *ac sandhiprakaraṇa* is not included in the rule base. The rules which are directly related to the processing of vowel *sandhi* in reverse format are included and modified according to the need of automatic reverse computation of *sandhi*. The rule base has two objects: marker and its corresponding pattern with *sandhi* name. Here 'marker' means the resultant *sandhi* sound where *sandhi* is possible and at which point the programme will segment the word for possible splitting and 'pattern' is the corresponding sound of the marker which will replace the marked sound. For example: ा=ा+आ:दीर्घसन्धि अकः सवर्णे दीर्घः (*ā=ā+ā:dīrga sandhi akaḥ savarṇe dīrgaḥ*) is a reverse rule of *dīrgha sandhi*. Here ा (*ā*) is a marker and ा+आ (*ā+ā)* is the corresponding pattern.

The markers and patterns in the rulebase are based on Pāṇinian formalism of generative *sandhi*, but they are not exactly reverse to the forward *sandhi* formalism. For example, in forward *yaṇ sandhi, /*इ/ or /ई/(i or ī) are changed to /य्/ (*y)*, but in reverse rule base, /य/ (ya) has been stored as a possible *sandhi* sound which will replace /इ/ or /ई/ (i or ī). This is because of encoding scheme in *Devanāgarī* Unicode (UTF-8) in which the consonants are represented as syllabic and therefore to separate vowel from them, *halanta* is added after them. So while storing these consonants as possible *sandhi* sounds, they have been stored with the vowel.

One more point about the markers is that there are more than one markers of same type in the rule base. For example: the marker /े/ (*e*) covers the rule for *pararūpa sandhi, guṇa sandhi* and the rule *omāṅośca*. But each marker has different patterns according to the rules for which it represent. So each marker, if found in the input string, will be validated for possible valid *viccheda* by lexicon check.

The reverse rule patterns are of three kinds -

- the RHS has two letters (i.e. ि+अ),
- the RHS has one letter followed by a blank space (ु+ :)
- the RHS has a blank space followed by a vowel (+इ).

In the first pattern type, the first letter (i.e. before +) will replace the marked sound and /अ/ will be the initial letter of the second word in *viccheda*. The reason to list /अ/ (/a/) separately as initial letter of the second word is that the marker will merge following /अ/ (/a/) within it. For example: in the word 'दध्यत्र' (*dadhyatra*) /य/ (*ya*) will be replaced with ि+अ (*i+a*) and the output will be दधि+अत्र (*dadhi+atra*). In the second pattern type, the first letter of the pattern (i.e. before +) will replace the marked sound and the second word will start with the following vowel (i.e. *mātrā* attached to the marked sound). For example: in the word 'मध्विव' (*madhviva*) the programme will look for the marker (the first /व/ (*va*)) from the left and replace it with ु+ *:* and the output will be मधु+इव (*madhu+iva*). In the third pattern type, blank space means that first part of segmented word will be /अ/ (/a/) ending and the second part of segmentation will start by the given vowel. For example: in the word 'तवेन्द्रः' (*tavendraḥ*) /े/ (e) will be marked as (+इ) and the output will be तव+इन्द्रः (*tava+indraḥ*).

The rule base has been built up in the following format:

LHS (search marker)=RHS(replace the search string with this and split the input into two parts from here+prefix this to the remaining part)

```
v i c c h e d a   p a t t e r n
|                                 |
LHS                               R       H       S
                                  |               |
                                  replace         prefix
                                  |                |
                                  the LHS         the remaining
```

ऽ= +अ:(पूर्वरूपसन्धि, एङःपदान्तादति);ाय=ै+ :(अयादिसन्धि एचोऽयवायावः);ाय=ै+अ:(अयादिसन्धि एचोऽयवायावः);य=े+ :(अयादिसन्धि एचोऽयवायावः);य=े+अ:(अयादिसन्धि एचोऽयवायावः);य=ी+ :(यण् सन्धि इकोयणचि);य=ी+अ:(यण् सन्धि इकोयणचि);य=ि+ :(यण् सन्धि इको यणचि);य=ि+अ:(यण् सन्धि इकोयणचि);ाव=ौ+ :(अयादिसन्धि एचोऽयवायावः);ाव=ौ+अ:(अयादिसन्धि एचोऽयवायावः);व=ो+ :(अयादिसन्धि एचोऽयवायावः);व=ो+अ:(अयादिसन्धि एचोऽयवायावः);व=ू+ :(यण् सन्धि इको यणचि);व=ू+अ:(यण् सन्धि इको यणचि);व=ु+ :(यण् सन्धि इको यणचि);व=ु+अ:(यण् सन्धि इको यणचि);व्=ो+ :(वान्तो यि प्रत्यये/अध्व परिमाणे च);व्=ौ+ :(वान्तो यि प्रत्यये);े= +ई:(गुणसन्धि आद् गुणः);े= +इ:(गुणसन्धि आद् गुणः);े=ा+ई:(गुणसन्धि आद् गुणः);े=ा+इ:(गुणसन्धि आद् गुणः);े= +ए:(पररूपसन्धि एङि पररूपम्/ओमाङोश्च);े=ा+ए:(पररूपसन्धि एङि पररूपम्/ओमाङोश्च);ो= +ऊ:(गुणसन्धि आद् गुणः);ो= +उ:(गुणसन्धि आद् गुणः);ो=ा+ऊ:(गुणसन्धि आद् गुणः);ो=ा+उ:(गुणसन्धि आद् गुणः);ो= +ओ:(पररूपसन्धि एङि पररूपम्/ओमाङोश्च);ो=ा+ओ:(पररूपसन्धि एङि पररूपम्/ओमाङोश्च);ल=ऌ+ :(यण् सन्धि इको यणचि);ल=ऌ+अ:(यण् सन्धि इको यणचि);ल्= +ऌ:(गुणसन्धि आद् गुणः);ल्=ा+ऌ:(गुणसन्धि आद् गुणः);ार्= +ॠ:(वृद्धिसन्धि उपसर्गादृति धातौ/वा सुप्यापिशलेः);ार्=ा+ॠ:(वृद्धिसन्धि उपसर्गादृति धातौ/वा सुप्यापिशलेः);र्=

+ऋ:(गुणसन्धि आद् गुणः);र्= +ॠ:(गुणसन्धि आद् गुणः);र्=ा+ऋ:(गुणसन्धि आद् गुणः);र्=ा+ॠ:(गुणसन्धि आद् गुणः);र=ृ+ :(यण् सन्धि इको यणचि);र=ृ+अ:(यण् सन्धि इको यणचि);र=ॄ+ :(यण् सन्धि इको यणचि);र=ॄ+अ:(यण् सन्धि इको यणचि);ै= +ऐ:(वृद्धिसन्धि वृद्धिरेचि);ै=ा+ऐ:(वृद्धिसन्धि वृद्धिरेचि);ै= +ए:(वृद्धिसन्धि वृद्धिरेचि);ै=ा+ए:(वृद्धिसन्धि वृद्धिरेचि);ौ= +औ:(वृद्धिसन्धि वृद्धिरेचि);ौ= +ओ:(वृद्धिसन्धि वृद्धिरेचि);ौ=ा+औ:(वृद्धिसन्धि वृद्धिरेचि);ौ=ा+ओ:(वृद्धिसन्धि वृद्धिरेचि);ा=ा+आ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ा=ा+अ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ा= +आ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ा= +अ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ी=ी+ई:(दीर्घसन्धि अकः सवर्णे दीर्घः);ी=ी+इ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ी=ि+ई:(दीर्घसन्धि अकः सवर्णे दीर्घः);ी=ि+इ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ू=ू+ऊ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ू=ू+उ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ू=ु+उ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ू=ु+ऊ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ॄ=ृ+ऋ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ॄ=ॄ+ॠ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ॄ=ृ+ॠ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ॄ=ॄ+ऋ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ॄ=्ऌ+ऋ:(दीर्घसन्धि अकः सवर्णे दीर्घः);्ॠ=ृ+ऋ:(वा.ऋति सवर्णे ऋ ॠ वा);्ल्ऌ=ृ+ऌ:(वा. ऌति सवर्णे ऌ वा);

The detailed description of this rule base and how it works is described as follows:

3.2.1 Rule-base for *yaṇ* sandhi

In reverse format of *yaṇ sandhi*, /य/(*ya*),/व/(*va*),/र/(*ra*),/ल/(*la*) will be replaced by /इ//ई/(*i*,ī),/उ//ऊ/(u,ū),/ऋ//ॠ/(ṛ,ṝ),/ऌ/(ḷ) respectively. The rule-base table for *yaṇ sandhi* is as follows:

Table 3.1: outline of reverse *yaṇ* sandhi

Marker[88]	Pattern
य	ी+ :
य	ी+अ
य	ि+ :

[88] Here these markers are stored comprising /अ/(a).

य	ि+अ
व	ू+ :
व	ू+अ
व	ु+ :
व	ु+अ
र	ृ+ :
र	ृ+अ
र	ॄ+ :
र	ॄ+अ
ल	ऌ+ :
ल	ऌ+अ

To illustrate one example, in the input word धावत्यश्वः *(dhāvatyaśvaḥ)*, the programme will start looking for the marker from the left side and will find /ा/(ā), /व/(va) /य/(ya). It will then search for corresponding pattern to each marker one by one. At each stage of pattern replacement, the segmented words will be validated by lexical check. If both the words are found in the corpus, the system will return them as outputs. If both the words are not found, the programme will look for the next marker in a word till the segmentation of the word is validated through lexical check . In case no segmentation is validated, the input will be returned as it is. In the above example the marker /य/(ya) will be replaced by ि+अ(i+a) and the result will be *धावत्यश्वः* = धावति+अश्वः (*dh*āvatyaśvaḥ= *dh*āvati + *aśvaḥ*)

3.2.2 Rule base for *ayādi sandhi*

In the reverse format of *ayādi sandhi,* /य/(ya)[89], /व/(va)[90], /आय/(āya), /आव/(āva) will be changed to /ए/(e), /ओ/(o), /ऐ/(ai), /औ/(au) respectively. The rule base of *ayādi sandhi* is as follows:

Table 3.2: outline of reverse *ayādi sandhi*

Marker[91]	Pattern
ाय[92]	ै+ :
ाय	ै+अ
य	े+ :
य	े+अ
ाव	ौ+ :
ाव	ौ+अ
व	ो+ :
व	ो+अ

[89] Here /य/(ya) is stored instead of /अय्/(ay) because the /अ/(a) of /अय्/(ay) will be merged with the preceding consonant in Devanāgarī Unicode script, so it will be impossible to search /अय्/(ay) in a word.

[90] Here /व/ is stored instead of /अव्/(av) because the /अ/(a) of /अव्/(av) will be merged with the preceding consonant in Devanāgarī Unicode script, so it will be impossible to search /अव्/(*av*) in a word.

[91] Here also these markers are stored comprising /अ/(a).

[92] The marker /ाय/(*āya*) has been put prior to the marker /य/(ya) in rule base so that /य/(ya) may not supervene upon /ाय/(*āya*) while searching for marker. This is the same with /ाव/(*āva*) also.

For example: in the word 'तस्मायिषुम्'*(tasm*āyiṣum) /ाय/(āya) will be labeled as the marker and will be replaced by ै+ :(ai+ :) and it will be तस्मै+इषुम् (tasmai+iṣum). Both these words will be found in the corpus and they will be returned as output.

The *ayādi sandhi* has a supplementary rule *vānto yi pratyaye* also, but its rule base will be different because here /व्/ (v) of /अव्/ (av) and /आव्/ (āv) are followed by a consonant /य्/ (y) and not vowel, so in its rule base, /व्/ (v), instead of /व/ (va), is stored. Its rule base is as follows:

Table 3.3: outline of extension of reverse *ayādi sandhi*

Marker	Pattern
व्	ो+ :
व्	ौ+ :

3.2.3 Rule base for *guṇa* sandhi

In reverse *guṇa sandhi*, /ए/(e),/ओ/(o),/र्/(r),/ल्/(l) will be marked as a possible *sandhi* sound and will be replaced as follows:

Table 3.4: outline of reverse guṇa *sandhi*

Marker	Pattern
े	+ई
े	+इ
े	ा+ई
े	ा+इ
ो	+ऊ

ो	+उ
ो	ा+ऊ
ो	ा+उ
ल्	+ऌ
ल्	ा+ऌ
र्	+ॠ
र्	+ॠ
र्	ा+ऋ
र्	ा+ॠ

For example: in the word '*तवल्कारः*' (*tavalkāraḥ*), the marker /ल्/(l) will find two formula and the output will be *तव+ऌकारः* (*tava+ḷkāraḥ*)

The rule *upsargādhṛti dhātau* is an exception to *guṇa sandhi*. Its rule base can be as follows[93]:

Table 3.5: outline of reverse guṇa sandhi - exception

Marker	Pattern
ार्	+ऋ
ार्	ा+ऋ

3.2.4 Rule base for *vṛddhi sandhi*

[93] This rule base will cover *vā supyāpiśaleḥ* also.

In reverse *vṛddhi sandhi* /ऐ/(ai) and /औ/(au) will be labeled as a marker and will be replaced as follows:

Table 3.6: outline of reverse *vṛddhi sandhi*

Marker	Pattern
ै	+ऐ
ै	ा+ऐ
ै[94]	+ए
ै	ा+ए
ौ	+औ
ौ[95]	+ओ
ौ	ा+औ
ौ[96]	ा+ओ
ौ[97]	+ऊ

For example: in the word *तवैव* (*tavaiva*), the marker /ै/(*ai*) will have two formulae and the output will be *तव*+ *एव* (*tava+eva*)

3.2.5 Rule base for *dīrgha sandhi*

In *dīrgha sandhi* /आ/(ā),/ई/(ī),/ऊ/(ū),/ॠ/(ṝ) are marked as the marker and are replaced as follows:

Table 3.7: outline of reverse *dīrgha sandhi*

[94] This marker includes the rule base for *etyedhatyūṭhsu* also.
[95] This marker is also applicable to the *vārtika otvośṭhayoḥ samāse vā*
[96] This marker is also applicable to the *vārtika otvośṭhayoḥ samāse vā*
[97] This marker is added to cover the rule *etyedhatyūṭhsu*

Marker	Pattern
ा	ा+आ
ा	ा+अ
ा	+आ
ा	+अ
ी	ी+ई
ी	ी+इ
ी	ि+ई
ी	ि+इ
ू	ू+ऊ
ू	ू+उ
ू	ु+उ
ू	ु+ऊ
ॄ	ृ+ऋ
ॄ	ॄ+ॠ
ॄ	ृ+ॠ
ॄ	ॄ+ऋ
ॄ	्ऌ+ऋ

For example: in the word ‘*उक्त्वापगच्छति*’ (*uktvāpagacchati*), the marker /ा/ (ā) has four replacements and the output will be *उक्त्वा+अपगच्छति* (*uktvā+apagacchati*)

There are two *vārtikas* available on the above rule. These *vārtikas* are *ṛti savarṇe ṛ rrṛ vā* and *ḷti savarṇe ḷ vā.* Their rule base can be as follows:

Table 3.8: outline of reverse *dīrgha sandhi*- exception

Marker	Pattern
्ॠ	ृ+ऋ
्ॡ	ृ+ऌ

3.2.6 Rule base for *pūrvarūpa sandhi*

In reverse *pūrvarūpa sandhi* /ऽ/ (‘) will be labeled as marker. Their rule-base will be as follows:

Table 3.9: outline of reverse *pūrvarūpa sandhi*

Marker	Pattern
ऽ	+अ

For example: in the word ‘*हरेऽव*’ (hare’va), the marker /ऽ/(‘) will be replaced as *हरे+अव* (hare+ava).

3.2.7 Rule base for *pararūpa sandhi*

In reverse *pararūpa sandhi* /ए/(e) and /ओ/(o) will be labeled as marker. Their rule base[98] will be as follows:

Table 3.10: outline of reverse *pararūpa sandhi*

Marker	Pattern

[98] This rule base of *pararūpa sandhi* also incorporates rule base for rule *omāṅośca* because marker and pattern will be same for both.

◌े	+ए
◌े	◌ा+ए
◌ो[99]	+ओ
◌ो[100]	◌ा+ओ

For example: in the word '*उपोषति*' (*upoṣati*), the marker will be /◌ो/ (o)and the output will be *उप*+*ओषति* (upa+oṣati*)*

3.3 Sandhi Lexicon

The need of the *sandhi* lexicon is to confirm the segmented words as valid Sanskrit words. For this purpose, it uses the following linguistic resources

- Adapted data from MWSDD
- Custom Sanskrit corpora

For a valid *viccheda*, both the segments must be present in either of the linguistic resources. If the word has more than one sounds marked for *sandhi,* then only the first word must be present in either of the linguistic resources. The remaining string in this case will continue with the process of rule pattern matching, splitting and search in the linguistic resources. The sample data of these linguistics resources is described below.

अयथपुरम्;अयथबलम्;अयथामात्रम्;अयथायथम्;अयथावत्;अयथोक्तम्;अया;अयि;अयुगपद्;अयुतशस्;अये;अरण्यवत्;अरम्;अररे;अरे;अरेरे;अर्च्य;अर्जुनतस्;अर्थकारणात्;अर्थतस्;अर्थिसात्;अर्धशस्;अर्प्य;अर्वाक्;अलकम्;अलग्लम्;अलम्;अलंकृत्वा;अलंतराम्;अलले;अल्पशस्;अव;अवतरम्;अवक्षायम्;अवगृह्य;अवघ्रायम्;अवचक्षणम्;अवज्योत्य;अवधूय;अवप्लुत्य;अवमत्य;अवमन्य;अवमर्शम्;अवयवशस्;अवरतस्;अवरस्तात्;अवरार्धतस्;अवरोप्य;अवष्टभ्य;अवसलवि;अवसवि;अवसल्ल;अवसाय्य;अवाक्;अवाग्नाभि;अवाप्य;अविघ्नतस्;अविचार्य;अविजित्य;अविद्यमानवत्;अवि

[99] This marker is also applicable to the *vārtika otvośṭhayoḥ samāse vā*
[100] This marker is also applicable to the *vārtika otvośṭhayoḥ samāse vā*

धा;अविधानतस्;अविधिपूर्वकम्;अविपर्यासम्;अविभज्य;अविरामम्;अविलग्नम्;अविवेचम्;अविवे
नम्;अविशेषतस्;अविश्रामम्;अविस्वरम्;अवृथा;अव्यतिषङ्गम्;अव्यवानम्;अव्याप्य;अशम्;अशे
षतस्;अश्रुतवत्;अष्टकृत्वस्;अष्टधा;असंव्यवहितम्;असंश्रवणे;असंश्रावम्;असंस्वादम्;असकृत्;अस
त्कृत्य;असद्यस्;असमक्षम्;असमवहितम्;असमिध्य;असमीक्ष्य;असंप्रति;असंप्राप्य;असंभव्यम्;अ
संभ्रान्तम्;असम्यक्;असर्वशस्;असाक्षात्;अस्यसि;अस्ति;अस्तिनास्ति;अस्था;अस्मत्रा;अस्मद्वत्;
अह;अहर्जरम्;अहर्व्यत्यासम्;अहश्शस्;अहह;अहहारे;अहहा;अहावस्;अहे;अहो;आ;आकण्ठम्;आक
र्णम्;आकर्णमूलम्;आकल्पम्;आकल्पान्तम्;आकालम्;आकालिकातीरम्;आकीम्;आक्ली;आगण्ड;
आगत्य;आगमनतस्;आगम्य;आगूर्य;आग्रन्थम्;आचतुरम्;आचन्द्रतारकम्;आचन्द्रम्;आचम्य;
आच्छिद्य;आच्य;आजन्म;आजरसम्;आजानु;आजीवम्;आजीवितान्तम्;आज्ञाय;आट्;आढ्यपदि;
आत्;आताली;आतिष्ठद्गु;आत्मसात्;आत्मार्थम्;आत्मार्थे;आदाय;आदितस्;आदित्यवत्;आदिश्य;आ
दीप्य;आदृत्य;आदृष्टिगोचरम्;आद्रृष्टिप्रसरम्;आदीर्य;आद्वादशम्;आद्वारम्;आधाय;आधीकृत्य;आधू
य;आधार्य;आनत्य;आनर्दम्;आनिशम्;आनाय्य;आनुकूल्यतस्;आनुषक्;आन्ऋशंस्यतस्;आन्तम्;आ
पाततस्;आपीय;आपाटलिपुत्रम्;आपेषम्;आपोथ्य;आपृक्;आपूर्य;आपृच्छय;आप्रदिवम्;आप्रपदम्;
आप्रावृषम्;आप्लाव्य;आप्लुत्य;आबालम्;आबाल्यम्;आब्दम्;आब्रह्म;आब्रह्मसभम्;आभाष्य;आभूत
संप्लवम्;आम्;आमन्थ्य;आमध्याह्नम्;आमन्त्र्य;आमरणम्;आमूर्धान्तम्;आमूलम्;अटति;ईश्धर;रा
म;अवतार;रमा;ईश;प्रति;एक;विद्या;आलय;गिरि;ईश;हरि;मही;इन्द्र;शची;शेष;प्रति;एक;मधु;आचा
र्य;वधू;उदय;आहार;महा;उत्सव;रमा;एक;आकृति;आकार;औषधि;आदि;अर्थ;पुरुष;सत्य;उदधि;आ
पन;पूर्व;अंचल;वन;मत्स्य;वराह;कृष्ण;विष्णु;महावीर;बुद्ध;कल्कि;ब्रह्म;महेश;

3.4 Search corpus

Various linguistic resources have been developed or adapted to increase the efficiency of *sandhi* analysis. The primary purpose of this corpus is to exclude the words for which *sandhi* processing is not required. This will save processing time for these words. Besides, this corpus will also be used while validating the segmented words For example in validating the segmentation of the word *astyuttarasyām* as *asti* + *uttarasyām*

3.4.1 Verb database

A verb database of around 90,000 verb forms has been adapted. The sandhi analyzer will exclude the *tiṅanta* (verb forms) individually found in the input text for processing because a separate *tiṅanta* analyzer is being developed. But the verb forms attached to the prefix (upasarga) or *subanta* form need segmentation. The sample data of verb database is as follows:

आसन्;उपकरोति;अनिच्छातः;इच्छन्ति;जागर्ति;विशदीकरोतु;सन्तु;परित्यजति;उपकरोति;रक्ष
न्तु;अरोचत;प्रचलति;पश्यतु;याति;आसीत;प्रासारयत्;उपाविशत्;आदिशत्;उत्पतत्;उदपतन्;अ
धावत्;आगच्छन्;न्यवर्तत्;कर्तिष्यति;अकृन्तत्;प्रत्यागच्छत्;आलोकयन्;लिम्पन्ति;भूषयन्ति;
प्रेषयन्ति;प्रज्वालयन्ति;अपनयति;धारयन्ति;भक्षयन्ति;समायोजयन्ति;प्रभवति;कथयन्ति;म
न्यन्ते;पूजयन्ति;उपयुज्यते;उत्पादयति;सञ्चालयति;प्राविशत्;अपतत्;अव्रजत्;अभणत्;त्रस्यत
;अस्मि;परिपालय;अकथयन्;आज्ञापयति;आकर्णयत्;अकुर्वन्;अलभत्;असि;संजायते;अजायत;
प्रारभत;प्राशंसत्;अलङ्करोति;प्रयतेमहि;भ्रमति;उद्भवन्ति;हरति;धारयन्ति;भ्रमन्ति;गुञ्जन्ति;प्र
सीदन्ति;समायोज्यते;धारयन्ति;खादन्ति;प्रसीदन्ति;अनुभवन्ति;रञ्जयति;रोचते;नास्ति;वर्जये
त;जयते;प्रविचलन्ति;प्रविशति;नमस्करोमि;इच्छसि;ददामि;याचे;प्रदास्ये;ददामि;पिबामि;इच्छा
मि;आरोहामि;नेच्छामि;श्रूयताम्;आस्ताम्;आगच्छताम्;कुरु;चालय;स्वीकरोतु;अक्रन्दत्;अवदत्;
जीवतु;अर्हति;ग्रहीष्यति;अक्षाम्यत्;असहत;जीवति;अतिष्ठन्;अहरत्;मरिष्यन्ति;आगमिष्यति;अ
नुभवन्तु;तिष्ठन्तु;गच्छत्सु;चिन्तयामास;विनङ्क्ष्यति;विषीदत;हन्ति;संवादयामि;आनुष्ठिते;उवा
च;समायातः;उच्यताम्;श्रुणु;करिष्यामि;गच्छ;क्षम्यताम्;आसते;विद्यते;मनोरञ्जयामः;प्रभवामः
;अलभत;गृह्णाति;प्रदर्श्यन्ते;प्रतीयन्ते;ददाति;शक्नुमः;वर्धयन्ति;अपश्यन्;आस्ते;अचिन्तयत्;म्रि
यते;नीयते;क्रियते;प्राप्स्यामि;अन्वभवत्;अवसत्;प्राप्नोत्;न्यवसत्;अलभत;कुर्याम;कारयति;वि
तरति;अभिधीयते;शक्यन्ते.अन्तर्भवन्ति;अभिलषति;प्रवर्तयति;अतिक्रामति;तरति;कथ्यते;नि
र्मीयन्ते;वर्धते;आपद्यते;उड्डाययति;

3.4.2 avyaya database

The *avyaya* (indeclinables) list has around 500 enteries and this is to exclude the *avyaya* words from *sandhi* processing. The sample data of *avyaya* file is as follows:

अ;कश्चित्;सदैव;अकस्मात्;अकाण्डे;अग्निसात्;अग्नी;अघोः;अङ्ग;अजस्रम्;अञ्ज्सा;अतः;अति;
अतीव;अत्र;अथ;अथकिम्;अथवा;अथो;अद्धा;अद्य;अद्यापि;अधरात्;अधरेद्युः;अधरेण;अधः;अध
स्तात्;अधि;अधिहरि;अधुना;अधोऽधः;अध्ययनतः;अनिशम्;अनु;अनेकधा;अनेकशः;अन्तः;अ
न्तरा;अन्तरेण;अन्यतः;अन्यत्;अन्यत्र;अन्यथा;अन्यदा;अन्येद्युः;अन्वक्;अप;अपरेद्युः;अपलुप
म्;अपि;अपिवा;अभि;अभितः;अभीक्षनम्;अमा;अमुत्र;अम्;अयि;अये;अरम्;अरे;अरेरे;अर्जुनतः;
अलम्;अल्पशः;अवगाहे;अवचक्षे;अवदत्तम्;अवरतः;अवः;अवश्यम्;अव्यथिष्यै;अष्टधा;असकृत्;अ
साम्प्रतम्;अस्तु;अह;अहह;अहो;अह्नाय;अअ;आण;आतः;आतृदः;आदह;आदितः;आम्;आरात्;आ

र्यहलम्;आविः;आः;आहु वध्यै;आहो;आहोस्वित्;इ;इतरेद्यु ः;इतः;इति;इत्;इत्थम्;इदानीम्;इद्धा;इव ;इह;ई;ईषत्;उ;उच्चैः;उत;उताहो;उत्तरतः;उत्त्रत्र;उत्त्रात्;उत्त्राहि;उत्त्रेण;उत्तरेद्यु ः;उदकसात्;उदाकि ;उदेतु ः;उन्मनी;उप;उपजोष;म्;उपधा;उपरि;उपरिषतात्;उपर्यु परि;उपांशु;उभयतः;उभयत्र;उभय था;उभयद्यु ः;उभयेद्यु ः;उषा;ऊ;ऋते;ऋधक्;ॠषिवत्;ए;एकत्र;एकदा;एकधा;एकपदे;एकशः;एतहि;ए व;एवम्;एषे;ऐ;ऐकध्वम्;ऐषमः;ओपरि;ओम्;औ;कच्चित्;कथञ्चन्;कथञ्चित्;कथमपि;कथमपि ;कथंकथमपि;कथा;कदा;कदाचन्;कदाचित्;कदापि;कर्तवे;कर्हि;कर्हि चित्;कामम्;कार्षापणशः;कि ञ्च;किन्तमाम्;किमङ्ग;किमपि;किमिति;किमिव;किमु;किमुत्;किंपुनः;किल;कु;कुतः;कुत्र;कुवि त्;कुह;कूपत्;कृतम्;क्व;क्वचित;क्वापि;क्षत्रियवत्;क्षमा;खालु;गु रुवत्;च;चतुर्धा;चतु ः;

3.4.3 Subanta corpus

The *sandhi* analyzer, before *sandhi* analysis, will do the *subanta* analysis of the given input. This program has already been developed as an M.Phil dissertation and is giving a satisfactory result and work is going on to make it more comprehensive.[101] As the Sanskrit words in the lexicon are stored in *prātipadika* form, the *subanta* analysis will tag the input as base words and case terminations. *Sandhi* analyzer will process the base words only. The *subanta* analyzer will make use of two files: *Subanta*_ExampleBase and *Subanta*_Rulebase. *Subanta*_ExampleBase has the nominal words with their base form and case terminations while the *Subanta*_Rulebase has reverse rules for subanta analysis. Subanta analyzer will first check for solutions in the example base data file. If found, it does not check the rule base. If not found, it will analyze the *subanta pada* according to the rule base.

3.4.4 Place Name database

The place name list is stored in *prātipadika* form. After subanta analysis, these words will be searched in the lexicon, so that sandhi processing can be omitted for these words.

अयोध्या;अनु राधापु र;अजन्ता;अमरावती;अङ्ग;अनू प;अवन्ति;अवन्तिपु र;आम्रकूट;अरावलि;अर्बु दाचल;अरुणाचल;अयोमु ख;आभीर;अरिणिका;अर्जु नायन;आनर्त्त;आरिय;इन्द्रप्रस्थ;उच्छ;उदयपु र; उज्जयिनी;उडुपि;उत्कल;उच्छल;उदयगिरि;उशीनर;एरिकेण;एलोरा;ओंकारेश्वर;कुरुक्षेत्र;कपिलव स्तु;कौशाम्बी;कुरु;कुशीनगर;कांची;कामरूप;कोणार्क;कल्याणी;कन्धार;काम्पिल्य;कंटकशिला;का

[101] Chandra, Subash. 2006. ‘*Machine Recognition and Morphological Analysis of Subanta-padas*’, submitted for M.Phil degree at SCSS, JNU.

शी;कुशावती;किरात;कुशस्थली;कर्णावती;कोशल;कनखल;क्रौञ्च;कैलाश;किष्किन्धा;कर्णूल;का
ञ्ची;कामगिरि;कलिङ्ग;कपिशा;कम्बोज;कुशीनारा;कान्यकुब्ज;काश्मीर;कर्णसुवर्ण;कुलुट;कुशद्वी
प;कैकेय;कदम्ब;कुण्डिनपुर;खण्डगिरि;गान्धार;गौड;गुरुग्राम;गन्धवती;गया;गिरिव्रज;गोमेदद्वीप;
गिरिनगर;घण्टशाला;चन्द्रगिरि;चित्रकूट;चम्पा;चेदि;चीनभुक्ति;जनकपुर;जम्बूद्वीप;तक्षशिला;तोश
लि;ताम्रलिप्ति;तुरुष्क;तुर्वश;ताम्रपर्णी;थानेश्वर;द्वारिका;दशपुर;देवगिरि;दशार्ण;दण्डकारण्य;दक्षिणा
पथ;धनुष्कोटि;धार;नृसिंहपुर;नैमिषारण्य;नीचैर्गिरि;नवद्वीप;नन्दिवर्धन;नासिक;पृथूदक;पञ्चवटी
;पुरुषपुर;पाञ्चाल;पाटलिपुत्र;प्रयाग;प्रभास;पावा;पद्मावती;पारियात्र;प्रतिष्ठान;पिष्टपुर;प्राग्ज्योतिष्पु
र;ब्रह्मदेश;ब्रह्मावर्त्त;भृगुकच्छ;भोज;मानस;माल;महेन्द्र;मलय;मथुरा;माया;मिथिला;महिषी;मगध;
मत्स्य;माहिष्मती;मल्ल;मध्यमिका;मतिपुर;मद्र;मूर्क;महाबलिपुरम;मस्था;मान्यखेत;मुखलिङ्ग
म;मनोरवासर्रण;यापु;यौय;रामगिरि;रैवतक;

3.4.5 Noun database

The noun list is also stored in *prātipadika* form. This list also includes the basic word list of Amarakosha. After *subanta* analysis, these words will be searched in the lexicon, so that these words can also be left out from sandhi processing.

अरिहन्त;अरिजीत;अरिन्दम;अरिवलगन;अरिवलि;अरिवरसु;अरिवोलि;अरिवुचेल्वन;अरिवुमधि;
अरिवुमणि;अरिवुनम्बी;अर्जित;अर्जुन;अर्क;अर्णव;अर्णेश;आरोग्य;अरशद;अरुल;अरुलचेल्वन;अरु
लसेल्वन;अरुमुगन;अरुण;अरूण;आरुणि;अरविन्द;आर्य;आर्यमान;आर्यन;आसव;असीम;असगर;
आशीष;अशोक;अशरफ़;आशु;आशुतोष;आश्विन;अश्वत्थामा;असिज;असीम;अशीत;असलम;असुम
न;आशुमान;अश्लेष;अतनु;अतल;अटल;अतुल;अतुल्य;अतिय;अथर्वन;आत्मा;आत्मज;आत्मजा;
आत्मज्योति;आत्मानन्द;आत्रलरसु;आत्रे;आत्रेय;अवधेश;अवनीन्द्र;अवनीश;अविक्षित;अविनाश;
अवकाश;अवतार;अय्यप्पा;अय्यप्पन;आयोग;आयुष;अज़ीज़;आज़म;अज़हर;आफ़रीन;आभा;अच
ल;आदर्श;अधीर;आदिश्री;अदिति;आद्रिक;अग्रत;अग्रिम;अहिल्या;ऐशानी;ऐश्वर्य;ऐश्वर्या;अजन्ता;अ
खिल;अक्षय;अक्षित;अलकनन्दा;अलका;अल्का;अल्मस;अल्पना;अमल;अम्ल;अम्बिका;अम्बुजा;
अमिता;अमित;आमोदिनी;आम्रपाली;अमृता;अमृतकला;अम्रुश;अनघ;अनहित;अनला;अनामिका;
आनन्दमयी;आनन्दी;आनन्दिनी;अनन्या;अनन्य;अनसुया;अनसूया;आँचल;अङ्गना;अङ्गारिका;
अनीश;अनिता;अञ्जलि;अञ्जना;अंजु;अंजुश्री;अनिन्दिता;अनिका;अङ्किता;अन्नपुर्णा;अनुष्का;
अंशुला;अंतरा;अनुह्य;अनुमति;अनुपमा;अनुराधा;अनुवा;अन्वेष;अपाला;अपराजिता;अपर्णा;अप्स
रा;आराधना;आरती;अर्चा;अर्चना;अर्पणा;अर्पिता;अरशिआ;अरुन्धती;अरुणा;अरुणी;अरुणिमा;अ
सावरी;आश्चर्य;असगरी;आशा;आशना;आश्विनी;असिता;अस्मिता;आत्मजा;आत्रेयी;अवनी;

3.5 Example database

Besides the above rule base for *sandhi* analysis, the system makes use of the *vārttika list* as well as *sandhi* example base. The main purpose of these resources is to find the segmented form of commonly occurring or exceptional examples of Sanskrit *sandhi* words. If words from these resources occur in the input, the lexical lookup method will be applied to get the segmented form for these words.

3.5.1 *Vārttika* list

The *vārttika* list has the *sandhi* derived words with their split forms. The criterion of the *vārttika* list is the application of the rule which is restricted to few specific words or the words which can not be analysed through rule base. That is why some *vārttika*s which can be interpreted through rule base have been put in the rule base while some Pāṇinian rules, with limited or specific applications, are stored in the *vārttika* list. The examples in the *vārttika* list are only collected from SK. The sample data of *vārttika* list is as follows:

गवेन्द्रः=गो+इन्द्रः:इन्द्रेच;क्षय्यम्=क्षि+यम्:क्षय्यजय्यौ शक्यार्थे;जय्यम्=जि+यम्:क्षय्यजय्यौ शक्यार्थे;क्रय्यम्=क्री+यम्:क्रय्यस्तदर्थे;अक्षौहिणी=अक्ष+ऊहिनी:अक्षादूहिन्यामुपसंख्यानम् (वा.);स्वैर=स्व+ईर:स्वादीरेरिणोः(वा.);स्वैरिणी=स्व+ईरिन्:स्वादीरेरिणोः(वा.);मद्ध्वरिः=मधु+ अरिः:इको यणचि;मध्वरिः=मधु+अरिः:इको यणचि;धात्रंशः=धातृ+अंशः:इको यणचि;धात्त्रंशः=धातृ+अंशः:इको यणचि;हर्य्यनुभवः=हरि+अनुभवः:अचो रहाभ्यां द्वे;नह्य्यस्ति=नहि+अस्ति:अचो रहाभ्यां द्वे;इन्न्द्रः=इन्द्रः:त्रिप्रभृतिषु शाकटायनस्य;राष्ट्रम्=राष्ट्रम्:त्रिप्रभृतिषु शाकटायनस्य;चक्रि अत्र=चक्री+अत्र:इकोऽसवर्णे शाकल्यस्य ह्रस्वश्च;

3.5.2 Example List

This example list will have around 1000 commonly occurring Sanskrit *sandhi* words (according to the prose literature like *Pañcatantra*, *Daśakumāracaritam*, *Bṛhatkathā* etc). These words will be stored with their *viccheda* strings.

कोऽपि=कः+अपि;नमोऽस्तु=नमः+अस्तु;तद्यथानुश्रूयते=तद्+यथा+अनुश्रूयते;तेऽपि=ते+अपि;ता
न्यधीत्य=तानि+अधीत्य;स्वजनोऽपि=स्वजनः+अपि;यदपूज्योऽपि=यद्+अपूज्यः+अपि;यदगम्यो
ऽपि=यद्+अगम्यः+अपि;इत्येवं=इति+एवम्;केनापि=केन+अपि;तत्रैकस्य=तत्र+एकस्य;यथेच्छया
=यथा+इच्छया;अतोऽहं=अतः+अहम्;अपृष्टोऽत्राप्रधानः=अपृष्टः+अत्र+अप्रधान;ममाकुलं=मम+आ
कुलम्;कुलीनोऽपि=कुलीनः+अपि;व्यवहारोऽयं=व्यवहारः+अयम्;योऽनाहूतः=यः+अनाहूतः;ततोऽहं
=ततः+अहम्;सोऽत्र=सः+अत्र;तेनादिष्टः=तेन+आदिष्टः;नास्त्येकमपि=न+अस्ति+एकम्+अपि;तेना
पि=तेन+अपि;नागच्छामि=न+आगच्छामि;अपृष्टोऽपि=अपृष्टः+अपि;सोऽस्ति=सः+अस्ति;दुष्टोऽ
स्ति=दुष्टः+अस्ति;महाकायोऽयं=महाकायः+अयम्;यतोऽयं=यतः+अयम्;प्रीतोऽस्मि=प्रीतः+अस्मि
;धर्मोऽयं=धर्मः+अयम्;स्वभावोऽत्र=स्वभावः+अत्र;ह्यविज्ञातं=हि+अविज्ञातम्;ज्ञायतेऽस्य=ज्ञायते+
अस्य;ततोऽहमत्रागतः=ततः+अहम्+अत्र+आगतः;कोलाहलोऽश्रावि=कोलाहलः+अश्रावि;ममोपरि=
मम+उपरि;प्रत्ययोऽत्र=प्रत्ययः+अत्र;प्रणम्योपविष्टः=प्रणम्य+उपविष्टः;दृष्टोऽसि=दृष्टः+असि;जीव
न्तोऽपि=जीवन्तः+अपि;सेवकोऽपि=सेवकः+अपि;चाल्पानि=च+अल्पानि;मोदकेनापि=मोदकेन+
अपि;मृदुलापि=मृदुला+अपि;इत्याह=इति+आह;नाधिकारेण=न+अधिकारेण;

Chapter – IV

Online Sandhi Analyzer System(SAS)

4.1 Introduction

This chapter describes the development part of the vowel S*andhi* Analyzer for Sanskrit (SAS) which is a partial implementation of the research done for M.Phil dissertation. The present system uses Java in the web format to analyze vowel *sandhi* in a given Sanskrit word or sentence or a text. The input and output representation script for the system is *Devanāgarī* UTF-8. There is an optional run-in-debug mode also which displays the internal states of SAS.

The analysis procedure of the system uses lexical lookup method as well as rule base method. Before *sandhi* analysis process, pre-processing, lexical search of *sandhi* strings in *sandhi* example base and *subanta*-analysis takes place respectively. The pre-processing will mark the punctuation in the input. After that, the program checks the *sandhi* example base. This example base contains words of *sandhi*-exceptions (*vārttika* list) and commonly-occurring *sandhi* strings (example list) with their split forms. These words are checked first to get their split forms without parsing each word for processing. Although the extent and criteria of storing words in example database will always be a limitation, but still this will be useful as it will save processing time for the stored words and step-up the accuracy of the result. After lexical search, *subanta* analyzer gets the case terminations (*vibhakti*) separated from the base word (*prātipadika*). *Subanta analyzer* also has a function to look into lexicon for verb and *avyaya* words to exclude them from *subanta* and *sandhi* processing. The *subanta* analysis will be helpful in the validation of the split words generated through reverse *sandhi* analysis as the Sanskrit words in lexicon are stored in *prātipadika* form. The reason to accumulate the words in *prātipadika* form is that *sandhi*-derived words in input Sanskrit text may have any of the case terminations. After *subanta*-normalization of input text, the system will look for fixed word list of place name, nouns and MWSDD. The words found in these resources will be let off form processing. The *sandhi* recognition and analysis will be according to the process outlined in the chapter III.

<u>The development part of the dissertation consists of the following components:</u>

1. A JSP front end run on Apache Tomcat 4.0

2. Java objects for

   - pre-processing
   - analyzing *subanta*
   - checking fixed lists
   - analyzing *sandhi*

3. Lexical resources

   - verb database
   - *avyaya* database
   - *vārttika* list of *sandhi*
   - example base of *sandhi*
   - *subanta* files (*subanta* example base, *subanta* Rule base)
   - placeNameList, noun List, MWSDD
   - rule base of *sandhi*
   - lexicon of Sanskrit words for validation
   - test corpus

The design of the *sandhi* analyzer is as follows:

input Sanskrit text

↓

viccheda eligibility tests
(pre-processing)

↓

subanta processing

↓

fixed list checking

↓

search of *sandhi* marker and *sandhi* patterns
(*sandhi* rule base)

↓

generate possible solutions
(result generator)

↓

search the dictionary

↓

search the results in the corpora (if not found in the dictionary)

↓

output (segmented text)

Diagram 4.1: model of vowel *sandhi* analyzer

4.2 The web interface of SAS (*viccheda.jsp*)

The front-end for the system is developed in utf-8 enabled Java Server Pages (JSP) and HTML. The front-end of the software enables the user to interact with the SAS engine with the help of a web-server (Apache Tomcat in this case). The JSP technology helps create web based applications by combining Java code with HTML. The web server runs the Java code and displays the results as HTML. The following code snippet instructs the page to set the language and content encoding for the page -

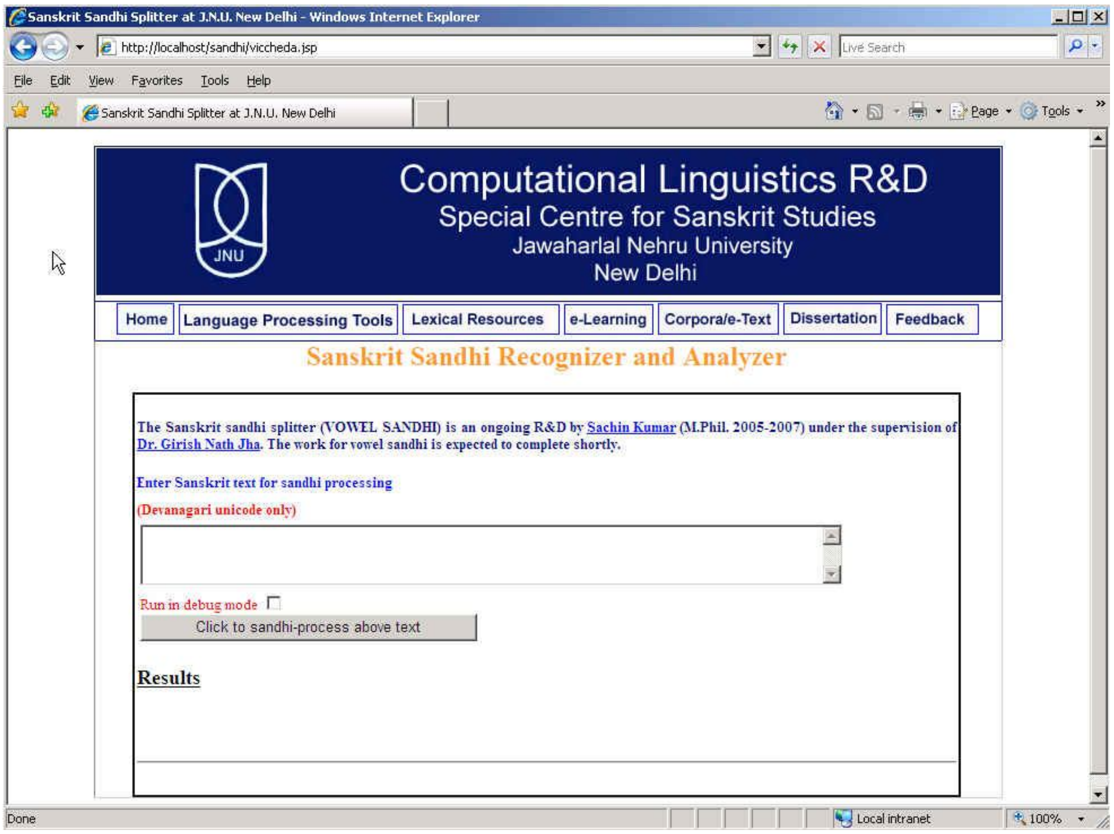
```
<html>
<head>

</head>
<%@ page
        language="java"
        pageEncoding="utf-8"
        contentType="text/html; charset=utf-8"
        import="java.util.*"
%>
```

The Apache –Tomcat web server is a Java compatible server. It is available under open source and is considered a fast web server with a lot of security features. The folder structure under a typical JSP website is as follows:

$$\begin{tikzcd}
\text{Web server installation folder} \arrow[d] & \\
\text{Webapps} \arrow[d] & \\
\text{website name (\textit{sandhi})} \arrow[d, dash] \arrow[dr, dash] & \\
\text{JSP page (viccheda.jsp)} & \text{web-inf} \arrow[d] \\
 & {}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{Classes} \arrow[d] \\
\text{Package (sandhiAPI)} \arrow[d] \\
\text{Java objects}
\end{tikzcd}$$

The screen shot of the interface is as follows:



Sanskrit Sandhi Splitter at J.N.U. New Delhi - Windows Internet Explorer

http://localhost/sandhi/viccheda.jsp

Computational Linguistics R&D
Special Centre for Sanskrit Studies
Jawaharlal Nehru University
New Delhi

Home | Language Processing Tools | Lexical Resources | e-Learning | Corpora/e-Text | Dissertation | Feedback

Sanskrit Sandhi Recognizer and Analyzer

The Sanskrit sandhi splitter (VOWEL SANDHI) is an ongoing R&D by Sachin Kumar (M.Phil. 2005-2007) under the supervision of Dr. Girish Nath Jha. The work for vowel sandhi is expected to complete shortly.

Enter Sanskrit text for sandhi processing

(Devanagari unicode only)

Run in debug mode ☐

Click to sandhi-process above text

Results

Done | Local intranet | 100%

The request made by the user upon clicking ‘Click to sandhi-process above text’, on interface is received by the request object in viccheda.jsp and it sends it to the servlet engine for processing. The code for this function is as follows:

```
<%

        request.setCharacterEncoding("UTF-8");
        String itext = request.getParameter("itext");

        int dbg = 1;

        if  (request.getParameter("debug") == null)
                dbg = 0;

        String ch = "checked";

        if (dbg == 0)
                ch = "";


        if (itext==null)
                itext = "";


        Viccheda v = new Viccheda(dbg);
```

The servlet engine assigns the task to the pertinent Java object for completion and response generation. The response/s thus collected from different Java objects by the servlet engine are sent back to the user. This can be seen as follows:

```
<h2><u>Results</u></h2>

                <% if (itext.length()>0) { %>
                <%=v.splitText(itext) %>
                <% } %>
```

4.3 Viccheda Modules

Viccheda.java is the parent java class of the programme which calls all the other classes and assigns different tasks to them. It first reads all the lexical resources on initialization. The reading is done as UTF-8 stream. The buffered-reader object converts the input stream into String buffered objects. The initialization of the *viccheda* class also creates the preprocessor, SupAnalyzer, FixedListChecker, STokenizer and Segmenter classes. This process can be described in the following way:

4.3.1 Preprocessor

Sandhi pre-processor is a component which will mark and normalize the input text for punctuations etc before the text is processed for *sandhi* segmentation. It will do the following functions sequentially:

- mark the punctuations in the input
- check the length of the word to determine if *sandhi* processing is required. If the length of the word is less than the threshold number, then it will pass the word as output without processing.
- check the *sandhi* example base to see if the *viccheda* strings can be obtained from these resources. This will save the processing time for these words.

The detailed description of each component with programming code can be described as follows:

4.3.1.1 Check Punctuation

The preProcess() call commands the string to checkPunct() function for searching the punctuations in the input and tagging them. In addition to tagging the punctuations, this function also moves out unwanted foreign letters or punctuations from inside of a *Devanāgari string*. It makes use of following data types for identifying different kinds of punctuations –

```
String punctProper = ",.'()[]{}$#@!%^&*-_+=|\\?/<>~`:;\"\u0964 \u0965";

String punctRoman= "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

String punctNum = "0123456789\u0966 \u0967 \u0968 \u0969 \u096A \u096B \u096C \u096D \u096E \u096F";

          String punctBullets = ")].<>-{}([* \u0970";
          String punctBulletsExcept = "\u091A \u0928 \u0939";
```

```
        String sub_punct_left = "'([{*`\"";

        String sub_punct_right = "')]}!*`\";,?.\u0964 \u0965";


        String punct = punctProper + punctRoman + punctNum;

        String ts = "";
        char c = ' ';
        int idx = -1;

        //PUCTUATIONS

        boolean found = false;

        found = false;


        if (debug==1)
            err =  err+"token="+tkn+" tkn length="+tkn.length()+"<br>";

        String tmp = "";

        int charCount = 0;

        tkn = tkn.trim();

        StringBuffer            punctuations            =new
StringBuffer("\u0964=[PUN_VV];\u0965=[PUN_VV];|=[PUN_VV];||=[PUN_SA];,=[P
UN_LV];?=[PUN_PC];!=[PUN_AC];‘=[PUN_UC];’=[PUN_SC];(=[PUN_VU1];)=[PU
N_VS1];[=[PUN_VU2];]=[PUN_VS2];{=[PUN_VU1];}=[PUN_VS2];-
=[PUN_DS];:=[PUN_CL];/=[PUN_BS];+=[PUN_PL];.=[PUN_BIN];*=[PUN_LAG];a=[
PUN_AB];b=[PUN_AB];c=[PUN_AB];d=[PUN_AB];e=[PUN_AB];f=[PUN_AB];g=[P
UN_AB];h=[PUN_AB];i=[PUN_AB];j=[PUN_AB];k=[PUN_AB];l=[PUN_AB];m=[PU
N_AB];n=[PUN_AB];o=[PUN_AB];p=[PUN_AB];q=[PUN_AB];r=[PUN_AB];s=[PUN
_AB];t=[PUN_AB];u=[PUN_AB];v=[PUN_AB];w=[PUN_AB];x=[PUN_AB];y=[PUN_
AB];z=[PUN_AB];A=[PUN_AB];B=[PUN_AB];C=[PUN_AB];D=[PUN_AB];E=[PUN
_AB];F=[PUN_AB];G=[PUN_AB];H=[PUN_AB];I=[PUN_AB];J=[PUN_AB];K=[PUN
_AB];L=[PUN_AB];M=[PUN_AB];N=[PUN_AB];O=[PUN_AB];P=[PUN_AB];Q=[PU
N_AB];R=[PUN_AB];S=[PUN_AB];T=[PUN_AB];U=[PUN_AB];V=[PUN_AB];W=[P
UN_AB];X=[PUN_AB];Y=[PUN_AB];Z=[PUN_AB];0=[AB_SAM];1=[AB_SAM];2=[
AB_SAM];3=[AB_SAM];4=[AB_SAM];5=[AB_SAM];6=[AB_SAM];7=[AB_SAM];8
=[AB_SAM];9=[AB_SAM];\u0966=[SAM];\u0967=[SAM];\u0968=[SAM];\u0969=[SA
M];\u096A=[SAM];\u096B=[SAM];\u096C=[SAM];\u096D=[SAM];\u096E=[SAM];\u0
96F=[SAM];“=[UN_UCd];”=[UN_SCd]");
```

4.3.1.2 Check example base

The *viccheda* class, on initialization, will read the example base and will store it as StringBuffer object as can be seen in the following code:

```
br4 = new BufferedReader( new InputStreamReader(new
FileInputStream("C:/Program Files/Apache Tomcat
4.0/webapps/sandhi/datafiles/example_base.txt"),"utf-8") );
```

StringBuffer examples = new StringBuffer(br4.readline());

Then the checkExampleBase(tkn) function will call this data.

4.3.2 Subanta Analyzer

SupAnalyzer is the java class for *subanta* analysis. The purpose of *subanta* analysis is to remove the case termination of the given input. This will be helpful to increase the accuracy of *sandhi* validation process. This program, before doing *subanta* analysis, marks verb and *avyaya* words and pass them as outputs without processing. The constructor of the RSubanta class will load the verb and *avyaya* lexicons as buffered reader object as follows:

```
public RSubanta(int dbg){
          debug = dbg;

          try{
               br1 = new BufferedReader( new
InputStreamReader(new FileInputStream("C:/Program
Files/Apache Tomcat 4.0/webapps/subanta/WEB-
INF/classes/avyaya.txt"),"utf-8") );
               br2 = new BufferedReader( new
InputStreamReader(new FileInputStream("C:/Program
Files/Apache Tomcat 4.0/webapps/subanta/WEB-
INF/classes/verb.txt"),"utf-8") );
```

After that, the SupAnalyzer class will call this in the following way:

```
 if ( s.indexOf("[") == -1  &&  s.length()>0 )
               //serach verbs and avyayas
               s =checkNS(s, "av");

          // if not a punctuation AND avyaya then only
check verbs
          if ( s.indexOf("[") == -1 &&  s.indexOf("AV") ==
-1 && s.length()>0 )
               //serach verbs and avyayas
               s =checkNS(s, "verb");
```

For *subnata* analysis, the program first checks the example base to get *subanta* analysis from there. The words found in the example base are not processed according to the rule base. After this, the rule base processes the remaining words and splits it into the base and affix. It does it by splitting certain number of last characters from the *subanta pada*. It first splits 8, then 7 and likewise up to the last character. At each step of segmentation, it validates the segmented form by checking the affix in the rule base. If the affix is found in the rule base, it assumes the analysis to be correct and returns it as output. If the affix is not validated in the rule base, it proceeds to create another combination of base + affix. The java code for this function is as follows:

```
if (newTkn.length()>8){
     tknPart8 = newTkn.substring(newTkn.length()-8,
newTkn.length());
     tknPartR8 = newTkn.substring(0,newTkn.length()-8);
}

if (newTkn.length()>7){
     tknPart7 = newTkn.substring(newTkn.length()-7,
newTkn.length());
     tknPartR7 = newTkn.substring(0,newTkn.length()-7);
}

if (newTkn.length()>6){
     tknPart6 = newTkn.substring(newTkn.length()-6,
newTkn.length());
     tknPartR6 = newTkn.substring(0,newTkn.length()-6);
}
```

```
if (newTkn.length()>5){
      tknPart5 = newTkn.substring(newTkn.length()-5,
newTkn.length());
      tknPartR5 = newTkn.substring(0,newTkn.length()-5);
}

if (newTkn.length()>4){
      tknPart4 = newTkn.substring(newTkn.length()-4,
newTkn.length());
      tknPartR4 = newTkn.substring(0,newTkn.length()-4);
}

if (newTkn.length()>3){
      tknPart3 = newTkn.substring(newTkn.length()-3,
newTkn.length());
      tknPartR3 = newTkn.substring(0,newTkn.length()-3);
}

if (newTkn.length()>2){
      tknPart2 = newTkn.substring(newTkn.length()-2,
newTkn.length());
      tknPartR2 = newTkn.substring(0,newTkn.length()-2);
}

if (newTkn.length()>1){
      tknPart1 = newTkn.substring(newTkn.length()-1,
newTkn.length());
      tknPartR1 = newTkn.substring(0,newTkn.length()-1);
}

if (tknPart8.equals(ky)) {
      Base = tknPartR8;
}
else if (tknPart7.equals(ky)) {
      Base = tknPartR7;
}
else if (tknPart6.equals(ky)) {
      Base = tknPartR6;
}
else if (tknPart5.equals(ky)) {
      Base = tknPartR5;
}
else if (tknPart4.equals(ky)) {
      Base = tknPartR4;
}
else if (tknPart3.equals(ky)) {
      Base = tknPartR3;
}
else if (tknPart2.equals(ky)) {
      Base = tknPartR2;
}
```

```
else if (tknPart1.equals(ky)) {
     Base = tknPartR1;
}
else
     Base = tkn;

//take out case_num information from Val
StringTokenizer stz = new StringTokenizer(val,"+");

while(stz.hasMoreTokens()){
     case_number = stz.nextToken();
     if(case_number.indexOf(".")>0){
          break;
     }
}
```

4.3.3 Fixed List checking

After *subanta* analysis, the input text has been segmented into base and affixes. Now only base words will be processed for *sandhi* analysis. But before that these words will be checked into MWSDD, place name list and noun list. The words found in these resources, will be exempted from *sandhi* processing. These different files are merged to one text file which is named as lexicon. The java code for this lexicon checking will be as follows.

```
br3 = new BufferedReader( new InputStreamReader(new FileInputStream("C:/Program Files/Apache Tomcat 4.0/webapps/sandhi/datafiles/lexicon.txt"),"utf-8") );
```
(Here viccheda.java is reading this)

Segmenter.java will call it as follows

```
tkn = checkLexicon(tkn);
```

4.3.4 Sandhi Analysis

After preprocessing, example base checking, *subnata* analysis and fixed list checking, the system will do the *sandhi* analysis. The java class segmenter does this analysis. This process is executed in the following two steps:

4.3.4.1 Sandhi marking and pattern identification

The segmenter class will check the *sandhi* rule-base in the database to mark the resultant *sandhi* sounds (marker) for potential splitting and to identify the *sandhi* patterns for *viccheda*, corresponding to the marked sound.

4.3.4.2 Result generator

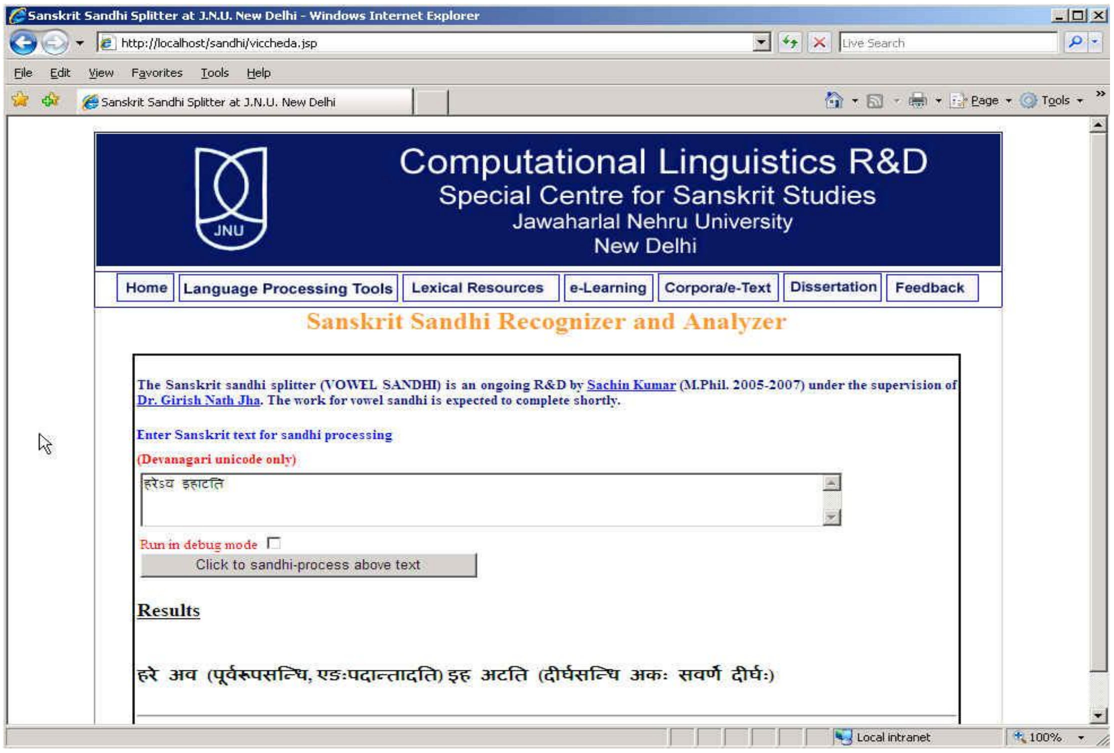
At each step of marker and pattern identification, the class will check the segmented words in the lexicon to generate the result. For this purpose, it will use MWSDD and customized Sanskrit corpora as the linguistic resources. To be a valid segmentation, both the segments must be available in either of the linguistic resources. If the word has more than one sounds marked for *sandhi*, then only the first word must be present in either of the linguistics resources. The remaining string in this case will continue with the process of rule pattern matching, splitting and search in the linguistic resources.

The splitText() function in the Viccheda class does all the processing –

```
public String splitText(String s){
          if (debug == 1)
               errmsg = errmsg +"<br><br>-----------------
START OF Viccheda.splitText()------------<br><br>";

          String tkn="";
          String ts = "";
          s = s.trim();
          String tmp = "";



if (s.length()>0){
               s = s.replace('\n',' ');

               StringTokenizer st = new StringTokenizer(s,
" "); //make  change here, add tokenization based on
sentence boundary (stop or danda)

               StringTokenizer st2 = null;

               while(st.hasMoreTokens()){
                    tkn = st.nextToken().trim();

                    tkn= preProcess(tkn);
                    if ( tkn.indexOf("PUN")==-1 &&
tkn.length() >3 ){ //not punc and a simple string
                         tkn = split(tkn);
                    }


                    ts = ts + " " + tkn;

               }

               if (debug == 1)
                    errmsg = errmsg +"<br><br>------------
----END OF Viccheda.splitText()-----------<br><br>";

               return ts;
}
```

4.4 Illustration

The intermediate result can be seen in the following format:


Sanskrit Sandhi Splitter at J.N.U. New Delhi - Windows Internet Explorer
http://localhost/sandhi/viccheda.jsp
Live Search
File Edit View Favorites Tools Help
Sanskrit Sandhi Splitter at J.N.U. New Delhi
Page Tools
Computational Linguistics R&D
Special Centre for Sanskrit Studies
Jawaharlal Nehru University
New Delhi
JNU
Home
Language Processing Tools
Lexical Resources
e-Learning
Corpora/e-Text
Dissertation
Feedback
Local intranet
100%

Sanskrit Sandhi Recognizer and Analyzer

The Sanskrit sandhi splitter (VOWEL SANDHI) is an ongoing R&D by Sachin Kumar (M.Phil. 2005-2007) under the supervision of Dr. Girish Nath Jha. The work for vowel sandhi is expected to complete shortly.

Enter Sanskrit text for sandhi processing

(Devanagari unicode only)

हरेऽव इहाटति

Run in debug mode ☐

Click to sandhi-process above text

Results

हरे अव (पूर्वरूपसन्धि, एङःपदान्तादति) इह अटति (दीर्घसन्धि अकः सवर्णे दीर्घः)

Conclusion:

The development part of the dissertation is partial at this point and is likely to be stable within a month time with extensive lexical resources. Subsequent research to make this program more efficient will always be going on. The system will be online and it can be accessed at http://sanskrit.jnu.ac.in

Conclusion

The present work is an R&D effort at the M.Phil. level for developing a vowel *sandhi* analyzer for Sanskrit. It was a two year programme: one year for course work and the next one year for R&D. Within this one year, a research on the *sandhi* rules of Aṣṭ., *Sid. Kau.* and *Kāś. vṛt* was done and the *sandhi* rules were formalized for reverse automation. Various linguistic resources for *sandhi* recognition and analysis were also developed and adapted. Besides this, the evaluation of tools and techniques- JSP for front end, Java for servlet objects and Apache Tomcat for web server was studied and an online interface was developed which is live at http://sanskrit.jnu.ac.in. The system has certain limitations which can be described as follows:

Limitations of the Recognition process (Rule Base)

- As exceptions to vowel *sandhi*, there are certain situations where no *sandhi* occurs between two immediate vowels and words may retain their position as such (*prakṛtibhāva*) either optionally or compulsorily. For example, in the input string *kavī imau*, there will be no change in input or output. So no marker has been included in rule base to process these words. But in some cases, the last long vowel of the first word, followed by a non-similar vowel, is changed to short vowel optionally. For example, in the input string *cakrī atra*, it may change to *cakri atra* or *cakrayatra*. As the input word *cakrayatra* will get processed through the marker of *yaṇ sandhi*, but the input *cakri atra* will not come across with any marker, so in the output it will reappear as it is. The logic of not storing these cases of vowel shortening in rule base is the different conditions, in which these instances appear, can not be represented in the rule base. So to process these types of words, some other approach will have to be evolved. As of now, the examples of these cases from *Sid. Kau.* have been put in the *vārttika* list.

- Some rules and *vārttikas* related to reduplication (*dvitva*) and elision (*lopa*) are excluded because in reverse *sandhi* segmentation, the validation of the split up forms of the string formed after reduplication and elision will not be possible. For example in the input *tānyyadhītya* (here /rÉç/ (*y*) is duplicated), the split form will be *tānyi/tānyī adhītya* and this will not be validated in the lexicon. But the words of these cases from *Sid. Kau.* have been stored in *vārttika list*. In cases where elision is made, there can be no marker for elision in the rule base. For example, the *sandhi* of *hare + ehi* may be *harayehi* or *hara ehi* (here,/rÉ/ (*ya*) is elided) To reverse it, the first case can be analyzed rightly because the marker /rÉ/ (*ya*) will give the right result for *ayādi sandhi*, but in the second case it can not be rightly analyzed as *ayādi sandhi* because it will not find the marker for it.

- There are some cases which need an explanation of context also. These cases can be easily interpreted in forward *sandhi* generation, but in reverse mode, the only possible way to analyze these words is to store markers and their equating patterns. For example, /A/ (*a*) and /G/ (*ṛ*) combined together are changed to /AUç/ (*ar*) according to the rule *ādguṇaḥ*, but if this /A/ is a last letter of a prefix and the following /G/ (*ṛ*) is the initial letter of a verb form, then both the vowels are changed to /AÉUç/ (*ār*) instead of /AUç/ (*ar*). For example: *upārcchati* = *upa+ṛcchati*. Similarly, if the word ending in /A/ (*a*) is followed by the word /LuÉ/ (*eva*) in the sense of uncertainty, then /L/ (*e*) replaces both the vowels (*pararūpa sandhi*). But if /LuÉ/ (*eva*) is used in the sense of certainty, both are replaced by /Lå/ (*ai*) (*vṛddhi sandhi*). To analyze these types of words, both the markers are stored in the rule base.

- There are some instances where the operation of a rule is optional. In such cases, the markers and patterns for both the instances are included. So the use of the marker in the input will decide which marker and pattern is applied. For example: /ऋ/(*ṛ*) /H/(*ṝ) +* /ऋ/(ṛ) /H/(*ṝ)*= H(*ṝ)* and /ऋ/(ṛ) /H/(*ṝ) +* /ऋ/(*ṛ*) /H/(*ṝ)* = U×ï(*rrṛ*)

Limitation of the analysis process

- The *sandhi* analyzer is supposed to exclude the *tiṅanta* (verb forms) found in the input text for processing. But currently, the system uses only 90,000 primary verb forms in the verb database, which are commonly found in Sanskrit literature. Although it is very unlikely that ordinary Sanskrit literature will go-around this list, yet the system is likely to start processing a verb as *subanta* if not found in the database.

- The limitation of lexical resources may affect the accuracy of the result. It is not always possible to get the segmented forms validated through the corpus.

- One more limitation of the lexicon right now is that the words in the lexicon are stored in *prātipadika* form. *Subanta* analyzer only removes the last case termination of the word, but the case termination attached in the middle of the string, is not validated. For example: in the word *munināṭāti* = *muninā* + *aṭāti*, *muninā* is not validated. To solve this problem, the option to search ending *svarānta vibhakti* (for example, *o, au, ena, eṇa, āya, sya, eṣu* for */a/* ending *prātipadika)* of LHS token of segmented string will be implemented.

- Another limitation could be the fact that this system will not segment compounds if they are found in the linguistic resources. This may mean that the larger strings may stay as un-simplified.

- In Sanskrit simple concatenation is also found. For example: *ācāryamnvagacchtām.* For accurate *sandhi* analysis, the segmentation of this simple concatenation is also necessary; otherwise it may affect the validation process. As in the above example, the *sandhi* analyzer will segment this string as *ācāryamnu* + *agacchtām* which will not be validated by lexicon check. It should have been *ācāryam* + *anu* + *agacchtām.* As the present system is not concerned with this simple concatenation, the result may be affected due to this.

- The limitations of the analysis process of the *subanta* analyzer may affect the accuracy of *sandhi* analysis because if *subanta* analyzer gives the wrong analysis of a word, the *sandhi* analyzer may not rightly validate its segmentation.

- One problem with this system may be the slow speed due to heavy processing, lexicon and corpora check. Since this system will be online and potentially multiple users could be accessing this system, speed will certainly be an issue.

Limitations of the result generating process

- There is also limitation on the validation level of segmented words. In validation process, there are two layers of searching segmented words. First, the system tries to ascertain all the segmented words from the lexicon and if all the words are not found, then the system looks for one less than the total segmented words. In this process, while splitting up the complicated long strings, the system has some ambiguities because in these strings, there may be more than one *sandhi* situations. Even to process these strings manually, a sufficient knowledge of Sanskrit lexicon, as well as a mastery over context of applying *sandhi* rules is needed. A machine has its own limitation. It can only put on the rules over the input but that may not generate right result all the time. For example in the input *sāpīhāṭati*, the user with the knowledge of Sanskrit can break the string into *sā api iha aṭati*, but the system will result into *sā apīhā aṭati* because it first split the whole string from the point it encounters with /ा/ (*ā*) and substitutes it with the corresponding pattern and generates *sā apīhā aṭati*. The system, at each step of validation, simultaneously tries to get hold of all the segmented words as well as one less than the total tokens from the lexicon. In the above example, as soon as the system finds two valid tokens *sā and aṭati* (one less than the total three tokens), it gives them back while the remaining unsegmented tokens stay as it is. The logic of getting one less segmented form of total tokens is that at least some simplification of long complicated string is done. Furthermore, the option of manually editing of these examples may be added to it. These manually edited words will be saved in the example base. Thus it will also be augmenting its example base.

Future Research and Development

The present *sandhi* analyzer has tremendous potentials in the field of Sanskrit NLP and M(A)TS. Some of the immediate and future applications of the system are discussed below –

A Complete Sandhi Analyzer for Sanskrit:

The present work only deals with the vowel *sandhi* analysis. The same methodology can be applied to build a system which can comprehensively analyze all the *sandhi* derived words in a Sanskrit text.

Samāsa analyzer for Sanskrit

As discussed in chapter I, the *sandhi* analyzer will be partially useful in *samāsa* (compounds) analysis because to analyse a compound, *sandhi* segmentation is a pre-requisite.

Self-reading and understanding

The *sandhi* analyzer can also be used for simplification of Sanskrit texts for simple reading and comprehension.

Sanskrit Search Engine and Spell Checker

Developing a Sanskrit text search engine and spell checker for Sanskrit may be the other areas where this methodology of making Sanskrit text simplified may be useful.

Support for other encoding schemes

At this point, the system takes the input in UTF-8 format but in the future, it may be upgraded to process the input in other encoding schemes also.

Machine Translation System (MTS)

Major goal of this and other R&D currently in progress at the SCSS , JNU is to design an M(A)TS from Sanskrit to Indian languages (right now Hindi), which can not be achieved without reverse analysis of *sandhi*. To process Sanskrit for this purpose, an algorithm of Sanskrit analysis system was presented by Dr. Jha and all students (2006) as follows[102] –

Input text

[upload file or cut/copy-paste text in text-area component]

|

pre-processing

|

charset check → non Unicode → discontinue

|

language check → not Sanskrit → discontinue

|

[LIGHT SANDHI PROCESSING]

|

connect to tiṅanta db → export verbs into a text file

|

[TINANTA MODULE]

tag verbs→ verbs not found →proceed to next step

[reverse derivation/duplication for complex verb forms]

|

[SUBANTA MODULE]

connect to subanta db → export avyaya data

|

check for avyayas → tag them

|

connect to subanta db → export vibhakti data

|

[102] Jha, Girish Nath. 2006, "**Towards a Computational analysis system for Sanskrit"** in the proceeding of first National symposium on Modelling and Shallow parsing of Indian Languages at Indian Institute of Technology Bombay pp. 25-34

check vibhaktis→tag nps for vibhaktis

|

[SAMASA MODULE]

|

[TADDHITA MODULE]

|

[STRI PRATYAYA MODULE]

|

[KRDANTA MODULE]

|

[AVYAYA MODULE]

|

[KARAKA MODULE]

|

yogyata check

|

karaka semantics check

|

karaka-vibhakti analysis

|

output text display/download/email

APPENDICES

APPENDIX- I

<u>Sample data of the rule base of reverse vowel sandhi:</u>

ऽ= +अ:(पूर्वरूपसन्धि, एङःपदान्तादति);ाय=ै+ :(अयादिसन्धि एचोऽयवायावः);ाय=ै+अ:(अयादिसन्धि एचोऽयवायावः);य=े+ :(अयादिसन्धि एचोऽयवायावः);य=े+अ:(अयादिसन्धि एचोऽयवायावः);य=ी+ :(यण् सन्धि इकोयणचि);य=ी+अ:(यण् सन्धि इकोयणचि);य=ि+ :(यण् सन्धि इको यणचि);य=ि+अ:(यण् सन्धि इकोयणचि);ाव=ौ+ :(अयादिसन्धि एचोऽयवायावः);ाव=ौ+अ:(अयादिसन्धि एचोऽयवायावः);व=ो+ :(अयादिसन्धि एचोऽयवायावः);व=ो+अ:(अयादिसन्धि एचोऽयवायावः);व=ू+ :(यण् सन्धि इको यणचि);व=ू+अ:(यण् सन्धि इको यणचि);व=ु+ :(यण् सन्धि इको यणचि);व=ु+अ:(यण् सन्धि इको यणचि);

APPENDIX-II

Sample data of the example base of reverse vowel sandhi:

i) example list:

तद्यथानुश्रूयते=तद्+यथा+अनुश्रूयते;तेऽपि=ते+अपि;तान्यधीत्य=तानि+अधीत्य;स्वजनोऽपि=स्वजनः+अपि;यदपूज्योऽपि=यद्+अपूज्यः+अपि;यदगम्योऽपि=यद्+अगम्यः+अपि;इत्येवं=इति+एवम्;केनापि=केन+अपि;तत्रैकस्य=तत्र+एकस्य;यथेच्छया=यथा+इच्छया;अतोऽहं=अतः+अहम्;अपृष्टोऽत्राप्रधानः=अपृष्टः+अत्र+अप्रधान;ममाकुलं=मम+आकुलम्;कुलीनोऽपि=कुलीनः+अपि;व्यवहारोऽयं=व्यवहारः+अयम्;योऽनाहूतः=यः+अनाहूतः;ततोऽहं=ततः+अहम्;सोऽत्र=सः+अत्र;तेनादिष्टः=तेन+आदिष्टः;नास्त्येकमपि=न+अस्ति+एकम्+अपि;तेनापि=तेन+अपि;नागच्छामि=न+आगच्छामि;अपृष्टोऽपि=अपृष्टः+अपि;सोऽस्ति=सः+अस्ति;दुष्टोऽस्ति=दुष्टः+अस्ति;महाकायोऽयं=महाकायः+अयम्;यतोऽयं=यतः+अयम्;प्रीतोऽस्मि=प्रीतः+अस्मि;धर्मोऽयं=धर्मः+अयम्;स्वभावोऽत्र=स्वभावः+अत्र;ह्यविज्ञातं=हि+अविज्ञातम्;ज्ञायतेऽस्य=ज्ञायते+अस्य;ततोऽहमत्रागतः=ततः+अहम्+अत्र+आगतः;कोलाहलोऽश्रावि=कोलाहलः+अश्रावि;ममोपरि=मम+उपरि;प्रत्ययोऽत्र=प्रत्ययः+अत्र;प्रणम्योपविष्टः=प्रणम्य+उपविष्टः;दृष्टोऽसि=दृष्टः+असि;जीवन्तोऽपि=जीवन्तः+अपि;सेवकोऽपि=सेवकः+अपि;चाल्पानि=च+अल्पानि;मोदकेनापि=मोदकेन+अपि;मृदुलापि=मृदुल+अपि;इत्याह=इति+आह;नाधिकारेण=न+अधिकारेण;प्रकोपितोऽपि=प्रकोपितः+अपि;

ii) vārttika list

गवेन्द्रः=गो+इन्द्रः:इन्द्रेच;क्षय्यम्=क्षि+यम्:क्षय्यजय्यौ शक्यार्थे;जय्यम्=जि+यम्:क्षय्यजय्यौ शक्यार्थे;क्रय्यम्=क्री+यम्:क्रय्यस्तदर्थे;अक्षौहिणी=अक्ष+ऊहिनी:अक्षादूहिन्यामुपसंख्यानम् (वा.);स्वैर=स्व+ईर:स्वादीरेरिणोः(वा.);स्वैरिणी=स्व+ईरिन्:स्वादीरेरिणोः(वा.);मद्ध्वरिः=मधु+अरिः:इको यणचि;मध्वरिः=मधु+अरिः:इको यणचि;धात्रंशः=धातृ+अंशः:इको यणचि;धात्त्रंशः=धातृ+अंशः:इको यणचि;हर्य्यनुभवः=हरि+अनुभवः:अचो रहाभ्यां द्वे;नह्य्यस्ति=नहि+अस्ति:अचो रहाभ्यां द्वे;

APPENDIX-III

<u>Sample data of MWSDD</u>

अयथपुरम्;अयथबलम्;अयथामात्रम्;अयथायथम्;अयथावत्;अयथोक्तम्;अया;अयि;अयुगपद्;अयु
तशस्;अये;अरण्यवत्;अरम्;अररे;अरे;अरेरे;अर्च्य;अर्जुनतस्;अर्थकारणात्;अर्थतस्;अर्थिसात्;अर्ध
शस्;अर्प्य;अर्वाक्;अलकम्;अलग्लम्;अलम्;अलंकृत्वा;अलंतराम्;अलले;अल्पशस्;अव;अवतरम्;
अवक्षायम्;अवगृह्य;अवघ्रायम्;अवचक्षणम्;अवज्योत्य;अवधूय;अवप्लुत्य;अवमत्य;अवमन्य;अव
मर्शम्;अवयवशस्;अवरतस्;अवरस्तात्;अवरार्धतस्;अवरोप्य;अवष्टभ्य;अवसलवि;अवसवि;अवस
ल्ल;अवसाय्य;अवाक्;अवाग्नाभि;अवाप्य;अविघ्नतस्;अविचार्य;अविजित्य;अविद्यमानवत्;अवि
धा;अविधानतस्;अविधिपूर्वकम्;अविपर्यासम्;अविभज्य;अविरामम्;अविलग्नम्;अविवेचम्;अविवे
नम्;अविशेषतस्;अविश्रामम्;अविस्वरम्;अवृथा;अव्यतिषङ्गम्;अव्यवानम्;अव्याप्य;अशम्;अशे
षतस्;अश्रुतवत्;अष्टकृत्वस्;अष्टधा;असंव्यवहितम्;असंश्रवणे;असंश्रावम्;असंस्वादम्;असकृत्;अस
त्कृत्य;असद्यस्;असमक्षम्;असमवहितम्;असमिध्य;असमीक्ष्य;असंप्रति;असंप्राप्य;असंभव्यम्;अ
संभ्रान्तम्;असम्यक्;असर्वशस्;असाक्षात्;अस्यसि;अस्ति;अस्तिनास्ति;अस्था;अस्मत्रा;अस्मद्वत्;
अह;

APPENDIX-IV

i) Sample data of place name list

अयोध्या;अनुराधापुर;अजन्ता;अमरावती;अङ्ग;अनूप;अवन्ति;अवन्तिपुर;आम्रकूट;अरावलि;अर्बुदाचल;अरुणाचल;अयोमुख;आभीर;अरिणिका;अर्जुनायन;आनर्त्त;आरिय;इन्द्रप्रस्थ;उच्छ;उदयपुर;उज्जयिनी;उडुपि;उत्कल;उच्छल;उदयगिरि;उशीनर;एरिकेण;एलोरा;ओंकारेश्वर;कुरुक्षेत्र;कपिलवस्तु;कौशाम्बी;कुरु;कुशीनगर;कांची;कामरूप;कोणार्क;कल्याणी;कन्धार;काम्पिल्य;कंटकशिला;काशी;कुशावती;किरात;कुशस्थली;कर्णावती;कोशल;कनखल;क्रौञ्च;कैलाश;किष्किन्धा;कर्णूल;काञ्ची;कामगिरि;कलिङ्ग;कपिशा;कम्बोज;कुशीनारा;कान्यकुब्ज;काश्मीर;कर्णसुवर्ण;कुलुट;कुशद्वीप;कैकेय;कदम्ब;कुण्डिनपुर;खण्डगिरि;गान्धार;गौड;गुरुग्राम;गन्धवती;गया;गिरिव्रज;गोमेदद्वीप;गिरिनगर;घण्टशाला;चन्द्रगिरि;चित्रकूट;चम्पा;चेदि;चीनभुक्ति;जनकपुर;जम्बूद्वीप;तक्षशिला;तोशलि;ताम्रलिप्ति;तुरुष्क;तुर्वश;ताम्रपर्णी;थानेश्वर;द्वारिका;दशपुर;देवगिरि;दशार्ण;दण्डकारण्य;दक्षिणापथ;धनुष्कोटि;धार;नृसिंहपुर;नैमिषारण्य;नीचैर्गिरि;नवद्वीप;नन्दिवर्धन;नासिक;पृथूदक;पञ्चवटी;पुरुषपुर;पाञ्चाल;पाटलिपुत्र;प्रयाग;प्रभास;पावा;

ii) Sample data of noun list

आदिनाथ;आदिकवि;आदित;आदितेय;आदित्य;आदित्यनन्दन;आदित्यवर्द्धन;आद्रिपति;अद्वैत;अद्वय;आफ़ताब;आगस्ति;अगस्त्य;अघ;अघत;अघर्ण;आग्नेय;अग्निकुमार;अग्निपर्व;अग्रिम;अग्रीय;अहसान;ऐजाज़;ऐमान;ऐनेश;अजातशत्रु;अजय;अजेन्द्र;अजिन्क्य;अजित;अजिताभ;अजितेश;अजमल;अजामिल;अकल्मष;आकश;अकबर;अखिल;अखिलेश;अकमल;अकरम;अक्रूर;अक्ष्ण;अक्षर;अक्षत;अक्षय;अक्षित;आकुल;अलगन;अलगरसु;आलम;आलमगीर;आलिम;अलहद;अलि;आलोक;अमानत;अमल;अमलेन्दु;अमलेश;अमन्द;अमर;अमर्त्य;अम्बर;अम्बरीष;अम्बुज;अमेय;अमिल;आमीन;अमीर;आमिर;अमिश;अमित;अमितविक्रम;अमितज्योति;अमिताभ;अमितेश;अमित्रसुदन;अमिय;अम्लान;अम्लानकुसुम;अमर;आमोद;अमोघ;अमोह;अमोल;अमोलिक;अमोलक;अमृत;अमृक;अमूल्य;अनादि;अनल;अनमित्र;अहिल्या;ऐशानी;ऐश्वर्य;ऐश्वर्या;अजन्ता;अखिल;अक्षय;अक्षित;अलकनन्दा;अलका;अल्का;अल्मस;अल्पना;अमल;अम्ल;अम्बिका;अम्बुजा;अमिता;अमित;आमोदिनी;आम्रपाली;

APPENDIX-V

i) Sample data of Verb database:

आसन्;उपकरोति;अनिच्छातः;इच्छन्ति;जागर्ति;विशदीकरोतु;सन्तु;परित्यजति;उपकरोति;रक्ष
न्तु;अरोचत;प्रचलति;पश्यतु;याति;आसीत;प्रासारयत्;उपाविशत्;आदिशत्;उत्पतत्;उदपतन्;अधा
वत्;आगच्छन्;न्यवर्तत्;कर्ति ष्यति;अकृन्तत्;प्रत्यागच्छत्;आलोकयन्;लिम्पन्ति;भूषयन्ति;प्रेष
यन्ति;प्रज्वालयन्ति;अपनयति;धारयन्ति;भक्षयन्ति;समायोजयन्ति;प्रभवति;कथयन्ति;मन्यन्ते
;पूजयन्ति;उपयुज्यते;उत्पादयति;सञ्चालयति;प्राविशत्;अपतत्;अव्रजत्;अभणत्;त्रस्यत;अस्मि;
परिपालय;अकथयन्;आज्ञापयति;आकर्णयत्;अकुर्वन्;अलभत्;असि;संजाय
ते;अजायत;प्रारभत;प्राशंसत्;अलङ्करोति;प्रयतेमहि;भ्रमति;उद्भवन्ति;हरति;धारयन्ति;भ्रम
न्ति;गुञ्जन्ति;प्रसीदन्ति;समायोज्यते;धारयन्ति;खादन्ति;प्रसीदन्ति;अनुभवन्ति;रञ्जयति;रोच
ते;नास्ति;वर्जयेत;जयते;प्रविचलन्ति;प्रविशति;नमस्करोमि;इच्छसि;ददामि;याचे;प्रदास्ये;ददामि;
पिबामि;इच्छामि;आरोहामि;नेच्छामि;श्रूयताम्;आस्ताम्;आगच्छताम्;कुरु;चालय;स्वीकरोतु;अक्र
न्दत्;अवदत्;जीवतु;अर्हति;ग्रहीष्यति;अक्षाम्यत्;असहत;जीवति;अतिष्ठन्;अहरत्;मरिष्यन्ति;आ
गमिष्यति;

ii) Sample data of avyaya database

अ;कश्चित्;सदैव;अकस्मात्;अकाण्डे;अग्निसात्;अग्नी;अघोः;अङ्ग;अजस्रम्;अञ्जसा;अतः;अति;
अतीव;अत्र;अथ;अथकिम्;अथवा;अथो;अद्धा;अद्य;अद्यापि;अधरात्;अधरेद्युः;अधरेण;अधः;अधस्ता
त्;अधि;अधिहरि;अधुना;अधोऽधः;अध्ययनतः;अनिशम्;अनु;अनेकधा;अनेकशः;अन्तः;अन्तरा;
अन्तरेण;अन्यतः;अन्यत्;अन्यत्र;अन्यथा;अन्यदा;अन्येद्युः;अन्वक्;अप;अपरेद्युः;अपलुपम्;अपि;
अपिवा;अभि;अभितः;अभीक्षनम्;अमा;अमुत्र;अम्;अयि;अये;अरम्;अरे;अरेरे;अर्जुनतः;अलम्;अ
ल्पशः;अवगाहे;अवचक्षे;अवदत्तम्;अवरतः;अवः;अवश्यम्;अव्यथिष्यै;अष्टधा;असकृत्;असाम्प्रतम्;
अस्तु;अह;अहह;अहो;अह्नाय;अअ;आण;आतः;आतृद;आदह;आदितः;आम्

APPENDIX-VI

i) Sample data of Subanta Example base:

सर्वः=सर्व+सु प्रथमा एकवचन;सर्वौ=सर्व+औ प्रथमा द्वितीया द्विवचन;सर्वे=सर्वा+जस प्रथमा द्वितीया द्विवचन (स्त्री.) ,सर्व+जस प्रथमा बहुवचन (पु.), सर्व+जस प्रथमा द्वितीया द्विवचन (नपु.); ना=नृ+सु प्रथमा एकवचन;नरः=नर+सु प्रथमा एकवचन/नृ+जस् प्रथमा बहुवचन;नरौ=नर+औ/औट् प्रथमा, द्वितीया द्विवचन/नृ+औ/औट् प्रथमा, द्वितीया द्विवचन;नृन्=नृ+जस् बहुवचन;सः=तद्+सु प्रथमा एकवचन;एतं=एतत+सु अम्, प्रथमा द्वितीया एकवचन;नरम्=नर+अम्, प्रथमा द्वितीया एकवचन/नृ+अम्, प्रथमा द्वितीया एकवचन;नरं=नर+अम्, प्रथमा द्वितीया एकवचन, नृ+अम्, प्रथमा द्वितीया एकवचन;प्रशामः=प्रशाम्+जस्/शस्/ङसि/ङस् प्रथमा/द्वितीया बहुवचन, पञ्चमी/षष्ठी एकवचन;

ii) Sample data of Subanta Rule base:

ः=+सु प्रथमा एकवचन;ा=ा+सु प्रथमा एकवचन;ाभ्याम्=+भ्याम् तृतीया चतुर्थी पञ्चमी द्विवचन;ाभ्यां=+भ्याम् तृतीया चतुर्थी पञ्चमी द्विवचन;भ्याम्=+भ्याम् तृतीया चतुर्थी पञ्चमी द्विवचन;भ्यां=+भ्याम् तृतीया चतुर्थी पञ्चमी द्विवचन;ेभ्यः=+भ्यस् चतुर्थी पञ्चमी बहुवचन;भ्यः=+भ्यस् चतुर्थी पञ्चमी बहुवचन;ानाम्=+आम् षष्ठी बहुवचन;ानां=+आम् षष्ठी बहुवचन;ाणाम्=+आम् षष्ठी बहुवचन;ाणां=+आम् षष्ठी बहुवचन;नाम्=+आम् षष्ठी बहुवचन;नाम्=+आम् षष्ठी बहुवचन;णाम्=+आम् षष्ठी बहुवचन;णां=+आम् षष्ठी बहुवचन;स्य=+स्य ङस्, षष्ठी एकवचन;े=+ङि सप्तमी एकवचन;ी=ी+सु प्रथमा एकवचन;

APPENDIX-VII

Punctuation and other Tags[103]

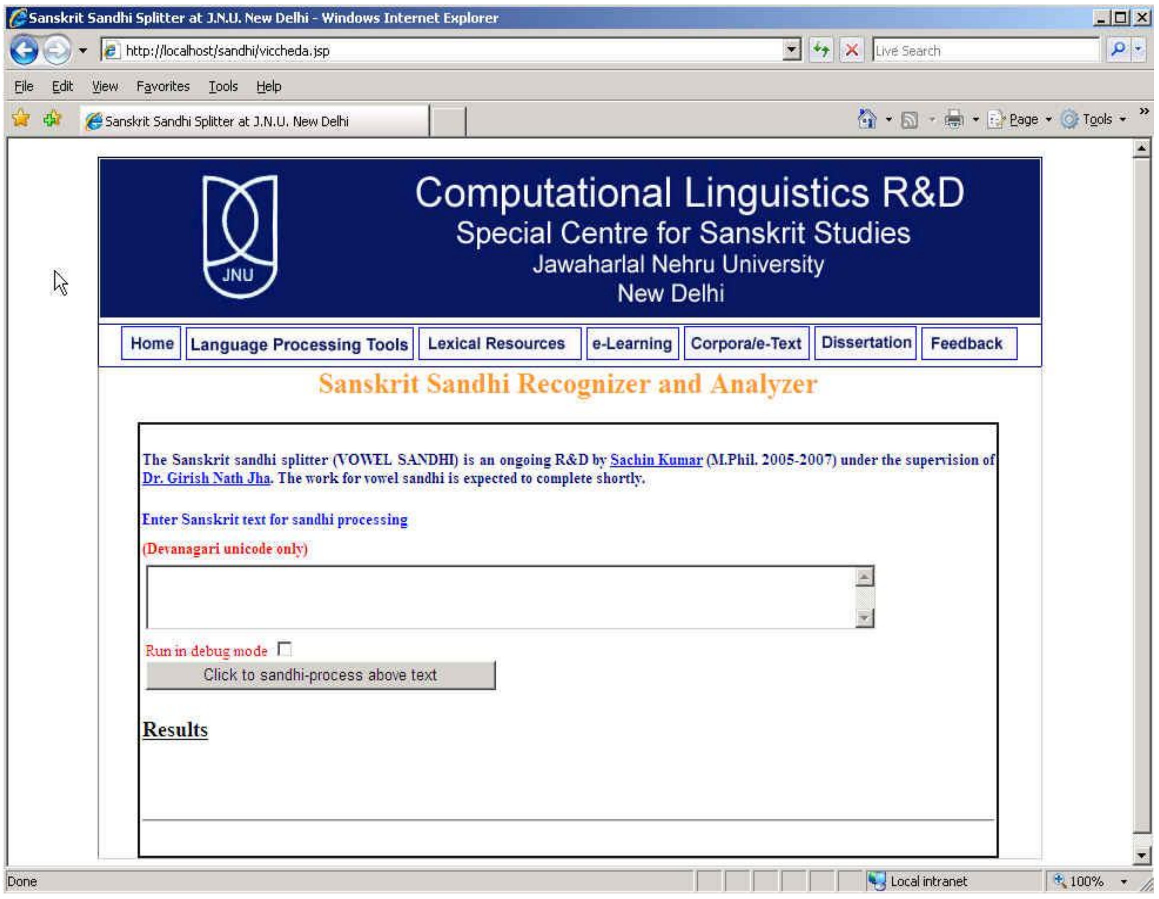
|	**PUN_VV** - Punctuation Vākya Virāma - sentence end marker, half shloka marker, etc
||	**PUN_SA** - Punctuation Śloka Anta - shloka end marker
,	**PUN_LV** – Punctuation Laghu Virāma - comma
?	**PUN_PC** – Punctuation Praśna Cihna - question mark
!	**PUN_AC** – Punctuation Āścarya Cihna - exclamatory mark
‘	**PUN_UC** – Punctuation Uddharana Cihna 1 - quote open
’	**PUN_SC** – Punctuation Samvarana Cihna 1 - quote close
“	**PUN_UCd** – Punctuation Uddharana Cihnadvaya - double quote open
”	**PUN_SCd** – Punctuation Samvarana Cihnadvaya - double quote close
(	**PUN_VU1** – Punctuation Valaya-Uuddharana Cihna 1 - open braces
)	**PUN_VS1** – Punctuation Valaya-Samvarana Cihna 1 - close braces
[	**PUN_VU2** Punctuation Valaya-Uddharana Cihna 2 - open square bracket
]	**PUN_VS2** Punctuation Valaya-Samvarana Cihna 2 - close square bracket
{	**PUN_VU3** – Punctuation Valaya-Uuddharana Cihna 3 - open flower bracket
}	**PUN_VS3** Punctuation Valaya-Samvarana Cihna 3 - close flower bracket
-	**PUN_DS** dash
:	**PUN_CL** colon
;	**PUN_VA** – Vākya-anga Anta - semi-colon
/	**PUN_BS** – Punctuation back slash

[103] Chandrashekhar, R. 2006, ‘*Part-of-Speech Tagging for Sanskrit*’, submitted for Ph.D degree at SCSS, JNU.

+	**PUN_PL** - Punctuation plus sign
=	**PUN_EQ** - Punctuation equal sign
.	**PUN_BIN –** bindu - Punctuation dot
*	**PUN_LAG -** Laghvikarana - abbreviation marker (पं* for पंडित)
AB	**AB** AnyaBhāṣā - foreign word
SAM-SAM	**SAM-SAM** hyphenated number (१९४७-२००६)

APPENDIX- VIII

Input screen


Sanskrit Sandhi Splitter at J.N.U. New Delhi - Windows Internet Explorer
http://localhost/sandhi/viccheda.jsp
Live Search
File Edit View Favorites Tools Help
Sanskrit Sandhi Splitter at J.N.U. New Delhi
Page Tools
JNU
Home
Language Processing Tools
Lexical Resources
e-Learning
Corpora/e-Text
Dissertation
Feedback
Run in debug mode
Click to sandhi-process above text
Done
Local intranet
100%

Computational Linguistics R&D
Special Centre for Sanskrit Studies
Jawaharlal Nehru University
New Delhi

Sanskrit Sandhi Recognizer and Analyzer

The Sanskrit sandhi splitter (VOWEL SANDHI) is an ongoing R&D by Sachin Kumar (M.Phil. 2005-2007) under the supervision of Dr. Girish Nath Jha. The work for vowel sandhi is expected to complete shortly.

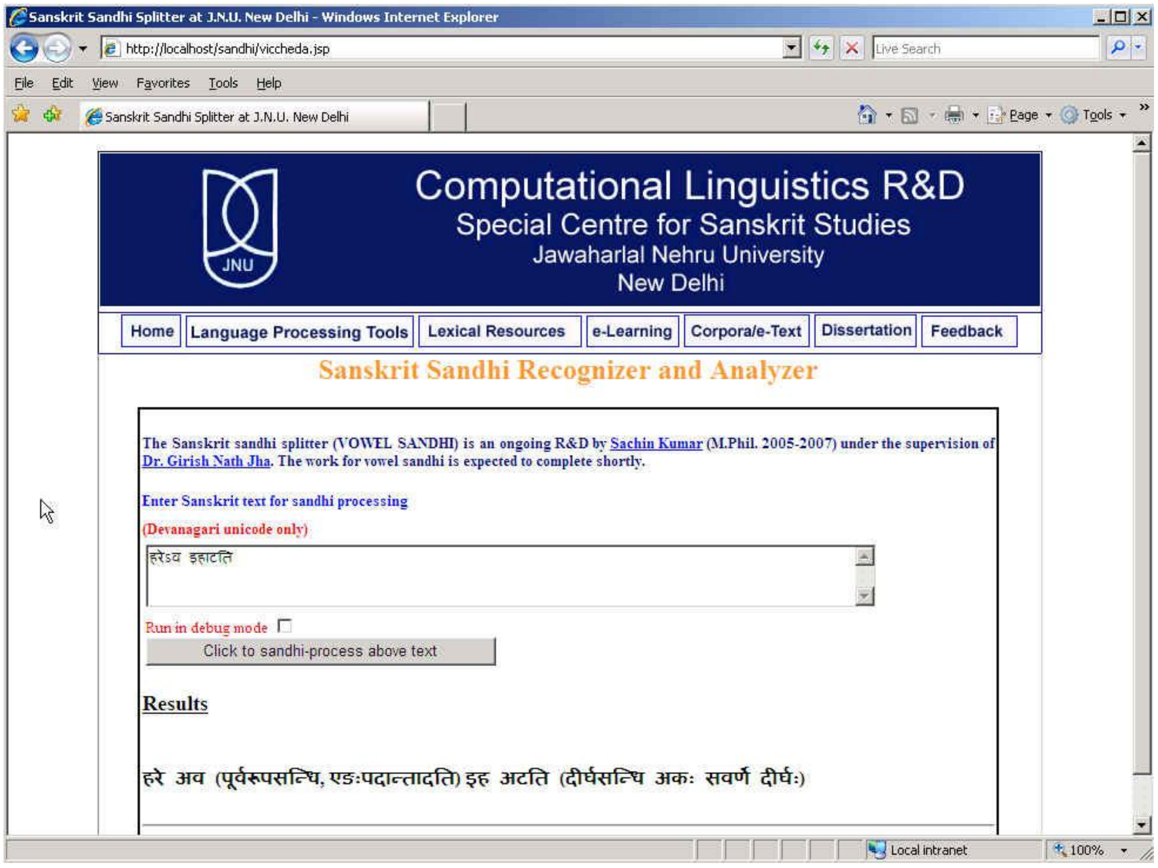
Enter Sanskrit text for sandhi processing

(Devanagari unicode only)

Results

APPENDIX-IX

screen shot of *svara sandhi* analysis


Sanskrit Sandhi Splitter at J.N.U. New Delhi - Windows Internet Explorer
http://localhost/sandhi/viccheda.jsp
Live Search
File Edit View Favorites Tools Help
Sanskrit Sandhi Splitter at J.N.U. New Delhi
Page Tools
Computational Linguistics R&D
Special Centre for Sanskrit Studies
Jawaharlal Nehru University
New Delhi
JNU
Home
Language Processing Tools
Lexical Resources
e-Learning
Corpora/e-Text
Dissertation
Feedback
Local intranet
100%

Sanskrit Sandhi Recognizer and Analyzer

The Sanskrit sandhi splitter (VOWEL SANDHI) is an ongoing R&D by Sachin Kumar (M.Phil. 2005-2007) under the supervision of Dr. Girish Nath Jha. The work for vowel sandhi is expected to complete shortly.

Enter Sanskrit text for sandhi processing

(Devanagari unicode only)

हरेऽव इहाटति

Run in debug mode ☐

Click to sandhi-process above text

Results

हरे अव (पूर्वरूपसन्धि, एङःपदान्तादति) इह अटति (दीर्घसन्धि अकः सवर्णे दीर्घः)

APPENDIX- X

Debugging process for the input word हरेऽव

हरे अव (पूर्वरूपसन्धि, एङः पदान्तादति)

```
-----------------START OF Viccheda.splitText()------------



-----------------START OF Viccheda.preProcess()------------

input=हरेऽव

output=हरेऽव

-----------------END OF Viccheda.preProcess()------------



-----------------START OF Viccheda.split()------------

input=हरेऽव

output= हरे  अव (पूर्वरूपसन्धि, एङः पदान्तादति)


-----------------END OF Viccheda.split()------------
```

```
-----------------END OF Viccheda.splitText()------------



----------------------------START OF Segmenter.segment()-----------------------


----------------------------START OF Segmenter.checkPatterns()-----------------------

------------starting to process (पूर्वरूपसन्धि, एङःपदान्तादति)  sandhi----------

rs =हरेऽव

VP=ऽ= +अ:(पूर्वरूपसन्धि, एङःपदान्तादति)

leftOfVP=ऽ

rightOfVP= +अ:(पूर्वरूपसन्धि, एङःपदान्तादति)

token Count =2

next token=हरे

token=हरे  count=2 tkn#1

next token=व

token=व  count=2 tkn#2


----------------------------START OF Segmenter.validateSplit()-----------------------


after validation total tokens=हरे  अव(2) validated = 2

validated=true
```

```
----------------------------END OF Segmenter.validateSplit()------------------------

------------result of processing (पूर्वरूपसन्धि, एङः पदान्तादति)  sandhi =हरे  अव

iResult= हरे अवfResult=

----------------------------END OF Segmenter.checkPatterns()------------------------

segmented token= हरे  अव (पूर्वरूपसन्धि, एङः पदान्तादति)



----------------------------END OF Segmenter.segment()------------------------
----------------START OF Viccheda.splitText()-----------
```

Bibliography

Primary Sources:

Chaturvedi, Giridharsharma (ed.). 2004, ***Vaiyakaraṇasiddhāntakaumudī with Bālamanorama and Tattvabodhini tikā***, Delhi: Motilal Banarasidass.

Jigyasu, Pt. Brahmadatta (ed.). 1998, ***Pāṇini-Aṣṭādhyāyī***, Sonipat: Ramlal Kapoor Trust.

Louis Bontes Digital MW Sanskrit-English Dictionary at http://members.chello.nl/l.bontes (accessed on 10.12.2006)

M. Monier Williams, 2004, ***Sanskrit- English Dictionary***, New Delhi: Munshiram Manoharlal Publishers Pvt. Ltd.

Mishra Narayan (ed). 1996. ***kāṣikā of Pt.Vamana and Jayaditya,*** Varanasi : Chaukhamba Sanskrit Sansthan.

Panashikar, Vasudev Lakshman Shastri (ed). 1994, ***Siddhāntakaumudī***, New Delhi: Chaukhamba Sanskrit Pratisthan.

Pandey, G.D., (ed.). 2003, ***Vaiyākaraṇa Siddhāntakaumudī,*** Varanasi: Chaukhamba Surbharati Prakashana.

Secondary Sources:

Books

Allen, W. Sidney. 1965, ***Phonetics in Ancient India***, London: Oxford University Press.

Allen, W. Sidney. 1972, ***Sandhi: The Theoretical, Phonetic, and Historical Bases of Word- Junction in Sanskrit***, The Hauge: Mouton & Co. Publishers.

Bakharia, Aneesha. 2001, ***Java Server Pages,*** New Delhi: Prentice Hall of India Private Limited.

Bharati, Akshar, Rajeev Sangal and Vineet Chaitanya. 2004, **Natural language Processing: A Paninian Perspective**, New Delhi: Prentice Hall of India Private Limited.

Bhat, D. N. Shankara. 1972, ***Sound Change***, Poona : Bhasha Prakashan.

Bird, Steven. 2003, '***Phonology***', in '***The Oxford Handbook of Computational Linguistics'***, Edited by Ruslan Mitkov, New York: Oxford University Press, pp. 3-24.

Bucknell, Roderic S. 1994, ***Sanskrit Manual***, Delhi: Motilal Banarsidass Publisher Pvt. Ltd.

Cardona, George. 1997, ***Pāṇini: His Work and its Traditions*** **(Vol. I)**, New Delhi: Motilal Banarasidass.

Jurafsky, Daniel., James H. Martin. 2005, ***Speech and Language Processing: An Introduction of Natural Language Processing, Computational Linguistics and Speech Recognition***, Delhi: Pearson Education.

Kale, M.R. 1972, ***A Higher Sanskrit grammar***, Delhi: Motilal Banarasidass.

Kapoor, Kapil. 2005, ***Dimensions of Panini Grammar: The Indian Grammatical System***, New Delhi: D. K. Printworld (P) Ltd.

Kaundinnayaayana, Shivram Acharya. 2004, ***Pāṇinianśikṣā,*** Varansi: Chaukhamba VidyaBhawavan.

Kushvaha, Mahesh Singh. (ed.) 1999, ***Laghusiddhāntakaumudī of Varadaraja***, Varanasi: Chaukhambha Vidyabhawan.

Macdonell, A.A. 1997, ***A Sanskrit Grammar for Students***, New Delhi: D.K. Printworld (P) Ltd, pp. 1-32.

Max Muller, F. 1977, ***A Sanskrit Grammar***, New Delhi: Asian Educational Services.

Mishra, Vidhata. 1972, ***A Critical Study of Sanskrit Phonetics***, Varanasi: The Chowkhamba Sanskrit Series office.

Mishra, Vidya Niwas. 1966, ***The Descriptive Technique of Pāṇini: An Introduction***, The Hague: Mouton & Co. Publishers.

Pandey, Shrishyamacarana, 1975, ***Pañcatantra of Viṣṇuśarman***, Delhi: Motilal Banarsidass Publishers Pvt. Ltd.

Pathak Manish Kumar. 2004, ***An Introduction to Sanskrit Grammar***, Delhi: Bharatiya Kala Prakashan.

Russell, Joseph P. 2002, ***Java Programming***, New Delhi: Prentice Hall of India Private Limited.

Sharma, Rama Nath. 2002, ***The Aṣṭādhyāyī of Pāṇini*** **(Vol. I)**, New Delhi: Munshiram Manoharlal Publishers Pvt. Ltd.

Shastri, Achrya Ram. 2002, ***Sanskrit śikṣaṇa saraṇī***, Delhi, Acharya Ram Shastri Gyanpeetha.

Shastri, Kapildev & Shrikant Pandey. 2001, ***Hindi-Mirukta :Pratham Adhyāya (In Hindi)***, Meerut: Sahitya Bhandar.

Troast, Harald. 2003, '***Morphology'***, *in* ***The Oxford Handbook of Computational Linguistics***, Edited by Ruslan Mitkov, New York: Oxford University press, pp. 25-47.

Varshney, Radhey L. 2001, ***An Introductory Textbook of Linguistic and Phonetics***, Bareilly: Student Store.

Whitney, William Dwight. 1995, ***Sanskrit Grammar***, Delhi: Low Price Publications.

Williams, M. Monier. 2000, ***A Practical Grammar of the Sanskrit Language***, New Delhi: Munshiram Manoharlal Publishers Pvt. Ltd.

Articles

Agrawal, Muktanand, 2006, '***Computational Identification and Analysis of Sanskrit Verb-forms***', In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, pp.126-127.

Bhowmik, Preeti & Girish Nath Jha. 2006, '***Sanskrit Language Pedagogy: an e-learning approach***', In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, p.150.

Jha, Girish Nath, et al., 2006, '***Towards a Computational Analysis System for Sanskrit***', In the Proceedings of First National Symposium on Modelling and Shallow Parsing of Indian Languages, IIT-B, Mumbai, pp.25-34.

Kumar, Sachin & Girish Nath Jha. 2006, '***Issues in sandhi processing of Sanskrit***', In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, p.129.

Kumar, Sachin, Girish Nath Jha. 2005, "***A Pāṇinian Sandhi Analyzer for Sanskrit***" In the Souvenir Abstracts of Platinum Jubilee International Conference of the Linguistic Society of India, University of Hyderabad, Hyderabad, pp. 36-37.

Mani, Diwakar, & Girish Nath Jha. 2006, '***Online indexing of Ādiparva of Mahābhārata'***, In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, p. 125.

Mishra, Sudhir Kumar & Girish Nath Jha. 2004, '***Kāraka Based Sentence Validation System***', In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, pp. 130-131

Mishra, Sudhir Kumar & Girish Nath Jha. 2004, '***Sanskrit Karaka Analyser for Machine Translation***', In the ***Proceedings of iSTRANS-2004***, New Delhi, pp.224-225.

Mishra, Sudhir Kumar & Girish Nath Jha. 2005, '***Identifying Verb Inflections in Sanskrit Morphology***', In the ***Proceedings of SIMPLE-05***, IIT-Kharagpur, pp.79-81.

Singh, Surjit Kumar & Girish Nath Jha. 2006, '***Strategies for Identifying and Processing Derived Nouns in Sanskrit***', In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, p. 131.

Subash & Girish Nath Jha. 2006, '***Sanskrit Subanta Recognizer and Analyzer for Machine Translation***' In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, p. 130.

Vyas, Pankaj K & Sivaja S Nair. 2006, '***Computer Implementation of Sanskrit Sandhi-rules (Version 0.1)'***, In the Souvenir Abstracts of 28th AICL, BHU, Varanasi, p.128.

Theses and Dissertations

Chandra, Subash. 2006, '***Machine Recognition and Morphological Analysis of Subanta-padas***', M.Phil dissertation submitted to SCSS, JNU

Chandrashekar, R. 2007, '***Part-of-Speech Tagging for Sanskrit***', Ph.D. thesis submitted to SCSS, JNU.

Jha, Girish Nath. 1993, ***Morphology of Sanskrit Case Affixes: A Computational Analysis***, M.Phil dissertation submitted to JNU, pp.17-24.

Mishra, Sudhir Kumar. 2007, ***Sanskrit Kāraka Analyzer for Machine Translation,*** Ph.D. thesis submitted to SCSS, JNU.

Websites

Academy of Sanskrit Research, Melkote, http://www.sanskritacademy.org/About.htm (accessed on 10.02.2007).

Anglabharti, IIt,Kanpur, http://www.cse.iitk.ac.in/users/langtech (accessed on 20.02.2007).

Anubharti, IIt,Kanpur, http://www.cse.iitk.ac.in/users/langtech (accessed on 20.02.2007).

Anusaaraka, http://www.iiit.net/ltrc/Anusaaraka/anu_home.html (accessed: 10.01.2007).

AU-KBC Research Centre, http://www.au-kbc.org/frameresearch.html (accessed: 15.02. 2007).

Bharati, Akshar, Vineet Chaitanya, Amba P. Kulkarni, Rajeev Sanghal. ‘***Anusaaraka: Overcoming the Language Barrier in India***’ http://www.iiit.net/techreports/2001_1.txt (accessed on 21.12.2006)

Bharati, Akshar, Vineet Chaitanya, Rajeev Sanghal. ‘***Computational Linguistics in India: An Overview***’ http://acl.ldc.upenn.edu/P/P00/P00-1077.pdf (accessed on 21.12.2006)

Bhate Saroja & Subhash Kak, ‘***Pāṇini’s Grammar and Computer Science’*** http://www.ece.lsu.edu/kak/bhate.pdf (accessed on 15.07.2007)

Dash, Anirban. 2005, ‘***Pāṇini’s Grammar: A Few Characteristics***’ http://www.languageinindia.com/feb2005/anirbanpanini1.html (accessed on 23.12.2006)

Dash, Anirban. 2006, ‘***Computational Analysis of Sanskrit Language***’ http://www.languageinindia.com/may2006/anirbancomputational.html (accessed on 23.12.2006)

Desika, http://tdil.mit.gov.in/download/Desika.htm, (accessed on 10.020.2007).

Ganakashtadhyayi, www.taralabalu.org/panini (accessed on 10.02.2007).

Hindi Wordnet http://www.cfilt.iitb.ac.in/wordnet/webhwn/wn.php (accessed on 17.07.2007)

Huet, Gerard. ‘***Design of a Lexical Database for Sanskrit***’ http://pauillac.inria.fr/~huet/PUBLIC/coling.pdf (accessed on 20.12.2006)

Huet, Gerard. ‘***Towards Computational Processing of Sanskrit’*** http://pauillac.inria.fr/~huet/PUBLIC/icon.pdf (accessed on 20.12.2006)

Jha, Girish Nath. 2004, ‘***The System of Panini***’ http://www.languageinindia.com/feb2004/panini.html (accessed on 23.12.2006)

Language Processing Tools: TDIL website, http://tdil.mit.gov.in/download/Shabdbodha.htm (accessed on 20.02.2007).

Oflazer,Kemal.http://folli.loria.fr/cds/2006/courses/Oflazer.ComputationalMorphology.pdf (accessed on 10.04.2007).

RCILTS, JNU, http://tdil.mit.gov.in/SanskritJapaneseChinese-JNUJuly03.pdf (accessed on 20.02.2007).

RCILTS, Utkal University. http://www.ilts-utkal.org, (accessed on 20.02.2007).

RSV, Tirupati, http://rsvidyapeetha.ac.in, (accessed on 20.02.2007).

S. Aparna & M. Ingle. 2004, '***Morphological Parsing of Sandhi Based Words in SanskritText***'http://www.languageinindia.com/dec2004/aparnasanskritparsing.html (accessed on 20.12.2006)

S'eaghdha, Diarmuid'O. '***Object-Language and Metalanguage in Sanskrit Grammatical Texts***' http://www.cl.cam.ac.uk/~do242/metaprogress.pdf (accessed on 15.07.2007)

Sanskrit Studies Links and Information, http://www.sanskritlinks.blogspot.com/ (accessed on 05.07.2007)

Shakti, LTRC, IIIT, Hyderabad, http://www.iiit.net/ltrc/index.html (accessed on 20.02.2007).

TDIL, MIT, GOI website, http://tdil.mit.gov.in/nlptools/ach-nlptools.htm (accessed on 10.02.2007).

The Sanskrit Heritage Site, Huet, Gerard. http://sanskrit.inria.fr/ (accessed on 10.02.2007).

The Sanskrit Library, http://sanskritlibrary.org/ (accessed: 20.05.07)

<u>e-Resources</u>

Apache Tomcat Servlet, http://tomcat.apache.org/ (accessed: 07.12.2006)

Baraha, Software, http://www.baraha.com/BarahaIME.htm (accessed: 07.04.2006)

Java Server Pages, http://java.sun.com/products/jsp/ (accessed : 15.12.2006).

JAVA, Servlet, http://java.sun.com/products/servlet/ accessed : 15.12.2006).

<u>Dictionary/Encyclopedia</u>

Collins Internet-linked Dictionary of Synonyms & Antonyms, 2005, Glasgow: Harper Collins Publishers.

Crystal, David. 1985, A Dictionary of Linguistics and Phonetics, Oxford: Basil Blackwell Ltd.