ISSUES AND CHALLENGES IN COMPUTATIONAL PROCESSING OF VYAÑJANA SANDHI

Dissertation submitted to Jawaharlal Nehru University

In partial fulfilment of the requirements

For the award of the

Degree of

MASTER OF PHILOSOPHY

By

DIWAKAR MISHRA

Special Centre for Sanskrit Studies

Jawaharlal Nehru University

New Delhi, INDIA

2009

JNU

विशिष्ट संस्कृत अध्ययन केन्द्र
जवाहरलाल नेहरू विश्वविद्यालय
नई दिल्ली - ११००६७

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

July 14th, 2009

CERTIFICATE

The dissertation entitled **'Issues and Challenges in Computational Processing of Vyañjana Sandhi'** submitted by **Diwakar Mishra** to **Special Centre for Sanskrit Studies, Jawaharlal Nehru University, New Delhi – 110067** for the award of degree of **Master of Philosophy** is an original research work and has not been submitted so far, in part or full, for any other degree or diploma in any University. This may be placed before the examiners for evaluation.

Sankar Basu

Prof. Sankar Basu
(Chairperson)

Chairperson
Special Centre for Sanskrit Studies
Jawaharlal Nehru University
New Delhi - 110067

Girishnath Jha

Dr. Girish Nath Jha
(Supervisor)

DR. GIRISH NATH JHA
Associate Professor
Special Centre for Sanskrit Studies
Jawaharlal Nehru University
New Delhi - 110067

विशिष्ट संस्कृत अध्ययन केन्द्र
जवाहरलाल नेहरू विश्वविद्यालय
नई दिल्ली - ११००६७

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

July 14th, 2009

DECLARATION

I declare that the dissertation entitled **'Issues and Challenges in Computational Processing of Vyañjana Sandhi'** submitted by me for the award of degree of **Master of Philosophy** is an original research work and has not been previously submitted for any other degree or diploma in any other institution/University.

Diwakar

(Diwakar Mishra)

ACKNOWLEDGEMENT

With depth of heart, I acknowledge my guru, my teacher, my guide Dr. Girish Nath Jha, who filled in me the competence of research, understanding of this technical subject and skill of language programming, who always guided me in academic and non academic matters in any circumstances. He has always paid sincere attention on us students and our work and gave time for that. He always encouraged us to proceed higher and higher in study and excellence.

After that I acknowledge for their cooperation, teachers of our centre, Prof. Shashi Prabha Kumar, Dr. Rajneesh Kumar Mishra, Dr. Hari Ram Mishra, Dr. Santosh Kumar Shukla and Dr. Ram Nath Jha. They gave me time and attention whenever I approached them. They provided me valuable suggestions and criticism which have been useful in improving my quality of research. I am also grateful to them for they teached me in M.A. and developed good understanding of different desciplines of Sanskrit. I also acknowledge Chairperson Prof. Sankar Basu for his cooperation and paying attention on our work. I also acknowledge office staff of the centre for their cooperation and help on various occasions.

My family's continuous support and cooperation has a great role in completing the research work in this way.

I am thankful to the organizers of Sanskrit Computational Linguistics Symposia, high standard papers of which helped me very much by providing quality material for research. I also thank the organizers of AICL who provide us chance to interact with great linguists.

I am very thankful to University Grants Commission for providing me fellowship which provided me necessary financial help.

Here I mention those whom I never can thank for what they have done for me. First of them is friend Surjit, who has always been with me in hot or cold days. Then I mention the sincere support of my other classmates – Neha, Debashis,

Puran, Mohan, Shashikant, Sureswar, Sanatan, Ravish, Prachi and other former classmates who left.

I acknowledge the contribution of my seniors Manji didi, Subash bhaiya, Diwakar bhaiya, Sachin ji, Muktanand ji, Kuldip ji, Surjya Kamal bhaiya, Renu didi and others. Among juniors, I would like to remember Alok, Ravi, Mukesh, Mamta, who always have been prepared for any help. I also remember Pratima, Rajnish, Baldev, Vishvesh, Jay, Devalina and Priyanka for encouraging and supporting me. Again I should never forget those, who are more like friends than junior students, and have always stood with me, Archana, Shashi, Nripendra.

I like to thank Jawaharlal Nehru University for providing such academic and intellectual platform for me. I remember one, who should have been remembered first, always with me...

Diwakar Mishra

CONTENTS

Contents

Page No

Transliteration key used in the dissertation

अ	=	a	ढ्	=	ḍh
आ	=	ā	ण्	=	ṇ
इ	=	i	त्	=	t
ई	=	ī	थ्	=	th
उ	=	u	द्	=	d
ऊ	=	ū	ध्	=	dh
ऋ	=	ṛ	न्	=	n
ॠ	=	ṝ	प्	=	p
ऌ	=	ḷ	फ्	=	ph
ए	=	e	ब्	=	b
ऐ	=	ai	भ्	=	bh
ओ	=	o	म्	=	m
औ	=	au	य्	=	y
क्	=	k	र्	=	r
ख्	=	kh	ल्	=	l
ग्	=	g	व्	=	v
घ्	=	gh	श्	=	ś
ङ्	=	ṅ	ष्	=	ṣ
च्	=	c	स्	=	s
छ्	=	ch	ह्	=	h
ज्	=	j	क्ष्	=	kṣ
झ्	=	jh	त्र्	=	tr
ञ्	=	ñ	ज्ञ्	=	jñ
ट्	=	ṭ	ऽ	=	’
ठ्	=	ṭh	ं (*Anusvāra*)	=	ṃ
ड्	=	ḍ	ः (*visarga*)	=	ḥ

INTRODUCTION

INTRODUCTION

Sandhi is an important aspect of any language because every language consists of vocal sounds and those sounds are prone to change according to the environment and context. The case of Sanskrit is somewhat special in this aspect. There are many reasons which make Sanskrit sandhi more complex. There is a famous *śloka* related to sandhi use –

saṃhitaikapade nityā nityā dhātūpasargayoḥ /

nityā samāse vākye tu vịvakṣāmapekṣate //

Sound changes according to rules are compulsory in within words but optional between words. The tendency of Sanskrit language users has been to take recourse to sandhi liberally between words where it is supposed to be optional. Thus theoretically the continuous sound sequence in Sanskrit can be infinitely long, though not practically. Some Sanskrit scholars have shown their excellence in writing very long words.

Seeing this important place of sandhi in Sanskrit, no complete language analysis system for Sanskrit can be thought of. Also it is the toughest part of analysis system in the sub-sentential category. Every morphological analyzer requires words as input to analyze, but there is no word boundary marker in continuous string. Before morphological analysis, the text needs to split sandhi and identify each and every distinct word. Thus sandhi analyzer needs to recognize all meaningful words. The task becomes more difficult when word needs to be identified not by lexical item but by its structure.

The present research aims at exploring the issues which arise in computational processing of sandhi and solutions thereof. Though the most of the issues and challenges are common to all types of sandhi processing, this research has special focus on *vyañjana* sandhi. To closely explore these issues and challenges, a small *vyañjana* sandhi processing system has been developed which has been integrated with the previously developed *svara* sandhi system.

This dissertation as the part of M. Phil. research is divided into four chapters. First chapter is titled *Sanskrit Morphonemics: a computational perspective"* discusses the concepts of morphophonemics, morphology, and phonology and general problems related to these. These concepts are more general and less Sanskrit specific. Then there is discussion of morphology from computational perspective. After that there is discussion about the formal structure of Pāṇinian system and study of it from formal and technical perspectives. There are discussed problems in computing Pāṇini in general and sandhi in particular. In the end, there is a survey of applied research and development in sandhi analysis in particular, Sanskrit language technology in general and theoretical research of formalization of Pāṇini.

Chapter two is titled *Sandhi System of Pāṇini.* First there is the general introduction of Sanskrit sandhi from the point of view of generative linguistics. The Paninian grammar formalism of Sandhi has also been presented. Finally the chapter presents a phonological representation of Paninian sandhi rules in the form phonological rule writing. After rules in that notation, there is simple explanation of rules in prose.

Third chapter is titled *Computational Processing of Sandhi: Issues and Challenges.* First of all, the chapter discusses the complexity of Sanskrit sandhi and what are the reasons which make it so complex. After that the work of Sachin (2007) has been described. Then the challenges which come in computational processing of sandhi have been discussed. In the end, an algorithm and system design is suggested to overcome these challenges.

Fourth chapter titled *Sandhi analyzer system description* illustrates the system which is developed as partial fulfilment of the research and to explore the issues closely. First the data design is explained with sample. Then the system architecture and design is explained with the sample code of JSP, HTML and Java. Finally, the introduction of implemented system is given with development information and screenshots.

CHAPTER ONE

SANSKRIT MORPHOPHONEMICS: A COMPUTATIONAL PERSPECTIVE

Chapter one

SANSKRIT MORPHOPHONEMICS:
A COMPUTATIONAL PERSPECTIVE

1.1 INTRODUCTION

This chapter presents an overview of Sanskrit morphophonemics, morphology, phonology and morphophonemics in computational perspective. A survey of research in Sanskrit morphophonemics and phonology and formalization of system of Pāṇini has been given followed by current R & D activities in the field of Language Technology for Indian languages. The survey gives special emphasis to the various projects sponsored by Indian government's Technology Development for Indian Languages (TDIL) initiative which started in 1991-92.

1.2 MORPHOPHONEMICS

Morphophonemics is the study of sound changes and functions in the environment of various word operations. This subfield involves two levels of linguistic study: study of word functions (Morphology) and the study of sound change (Phonology). The sound change due to word operations cannot be completely kept in morphology or in only phonology. The sound change in this process is not independent of words. The other name of morphophonemics is *sandhi.* Almost all kinds of morphology involve sound changes, it may be generation of forms or phrases, or it may be deriving new words from already existing words. Sometimes the sound change is so minute that it is ignored, but most of the times, the sound change is so significant that it appears clearly. In some languages, where the written form of language is dominant, many sound changes are not easily perceived by the speakers. But where the main form of the language is spoken, and writing system follows the speech, such as Sanskrit, the sound changes are easily apparent to common speakers of the language. That is why the detailed study of sandhi has been done in Sanskrit grammar tradition. In Sanskrit, sandhi is defined as 'extreme closeness of sounds'. Sandhi is mainly of two categories- external sandhi and internal sandhi. External sandhi is the sound change on the word/morpheme boundaries when the two words come together. Internal sandhi is the sound change within the word which may be caused due to combining of the words or due to any of word operations like derivation or inflection. To understand the morphophonemics, a general look on phonology and morphology is required. The Sanskrit morphophonemics will be discussed in detail in the next chapter.

1.3 MORPHOLOGY

To compute morphophonemics or sandhi, understanding morphology is necessary. Morphology deals with the word and its structure. Morphological study becomes important because what sandhi-splitting deals with is nothing but *identification of distinct words.* Jurafsky and Martin (2005:85) define morphology as "Morphology is the study of the way words are built up from smaller meaning-building units, **morphemes**. A morpheme is often defined as the minimal meaning-bearing unit in a language."

1.3.1 WORDS

Words are one of the most fundamental units of linguistic structure. They play an integral role in the human ability to use language creativity. The words are the phonetic codes to memorize the information. But the human vocabulary is not the static repository of the memorized information, it is a dynamic system. The dynamism comes in different ways, we can add words, and we can expand their meaning into new domains or change the meaning (Akmajian et. al., 2004:11).

Words are a dynamic system. No one makes effort to learn lots of words but can use infinite ideas around one. The world represented by the words is realized to be infinite in scope and vocabulary of human is finite, and very small in comparison with the scope of represented world. Human uses finite number of words to represent potentially infinite number of situations and ideas encountered in the world. This could be possible due to creativity of the words and open-endedness of the human vocabulary. The list of words for any language (though not a complete list) is referred to as its lexicon (Akmajian et. al., 2004:11).

The word in its physical speech form (that is the basic form of language) is nothing but a blur of sound; the same will be said for a sentence or a paragraph. A human, generally, does not feel difficulty in recognizing the words from a speech. The words seem to him to be natural and obvious. This observation comes because most people hear only their native language or other language which they know (have learnt). The reality comes to sight when a person hears a foreign language (which he does not understand) speaker in his natural way. Hearer is unable to recognize the words of that language. There is no realization of words for him but of a blur of sound. It means the *words are language relative*. Thus lexicon or dictionary always is of a language (or languages). The hearer can split the blur of sound into the words of foreign language speaker only when the speaker slows down a little. But physical reality of speech is that for the most part the signal is continuous, with no breaks at all between the words. Pinker (1995: 159-160) notes, "We [native speakers] simply hallucinate word boundaries when we reach the edge of a stretch of sound that matches some entry in our mental dictionary."

What is the knowledge of word? It means what do we know when we know a word? Knowing a word includes manifold information about the word. Akmajian et. al. (2004:12-13) list down five-fold information bound with the knowledge of the word.

a) *Phonetic/Phonological information-* how the sounds in the word are pronounced and in which order the sounds in the language can occur. Also, how the sounds change when a word is pronounced with other words.

b) *Lexical structure information-* what are the components of the word and how a word changes by replacing some components of word.

c) *Syntactic information-* how a word can be used in a sentence and what place can it hold in sentence structure.

d) *Semantic information-* how the meanings of components of words constitute the meaning of the word and what role words play in constituting meaning of sentence.

e) *Pragmatic information-* how a word is used in sentences and which word is used in which context.

There are some other aspects of words which linguists study. The native speaker may or may not be aware of these. Words and their usage are subject to variation across groups of speakers (language variation or dialectal study). *Aṣṭādhyāyī* records the example of usages in eastern (*prāchām*)[1] and northern (*udīcām*)[2] regions. Words and their uses are also subject to variation over time (language change or diachronic study). As *asura* word changed its usage and meaning over time and its meaning became absolutely opposite.

1.3.2 MEANS OF KNOWLEDGE OF WORDS

The knowledge of words basically comes by hearing and interaction with the other language speakers of the society. The most common means of knowledge of words is dictionary. But that is not all about the word's knowledge which we find in its lexical entry. Dictionary gives only basic information, sometimes detailed also, about the word and its meaning but we know far more than that. For example, when we know the meaning of a word through dictionary, we often get the meanings of all its forms or paradigm, and also know how different forms and their meanings are related to the base word and its meaning and also how different forms are related with each other. Besides, we also find parallels in the relations in paradigms of different words. For example we know that the relation between *run* and *runner* is same as *wash* and *washer*, but that relation does not exist between *bet* and *better.*

These, and a few others, are some extra information about the words which dictionary cannot tell. First, sometimes in dictionary, some forms are given with the lexical entry of the base word; it indicates that the two words are related. But what is the nature of relationship, dictionary does not specify. Second, the words that have similar form, lime *run-runner*, *walk-walker*, and thus have similar relationship with their base word, cannot be captured with their relatedness, because they all lie in separate entries. The separate entry approach fails to capture what all these words have in common. Third, the dictionary is a finite list and the information it contains is finite as well. How novel words behave cannot be accounted for (Amajian et. al., 2004:14). Fourth, dictionary does not tell how the language varies across the group of speakers and across the geographical area. Fifth, dictionary also cannot tell what have been the directions of the change in the meaning and usage of the words, and how a particular word emerged, also cannot be easily found in the dictionary.

The information of word variation across different areas and across groups of speakers and historic change over time mentioned above are the subject of different subfields of linguistics, namely, language variation and language change. A wide range of information associated

[1] *eṅ prācāṃ deśe* (P 1.1.75); *iñaḥ prācām* (P 2.4.60)

[2] *udīcāṃ māṅo vyatīhāre* (P 3.4.19)

with words and how new words are created, is studied in some other subfield of linguistics named morphology.

1.3.3 CONCEPTS OF MORPHOLOGY

The morphology is the identification, analysis and description of structure of words (Wikipedia). Nature of words studied in morphology tries to find out the answers of some basic questions which look simple but hard to answer. Akmajian et. al. (2004) list them as follows and mention that no completely satisfactory answer to any of them is arrived at.

- What are words?
- What are the basic building blocks in the formation of complex words?
- How are more complex words built up from simpler parts?
- How is the meaning of a complex word related to the meaning of its parts?
- How are individual words of a language related to other words of the language?

Simple and Complex Words/Morphemes

What are words? Simply a word is a complex pattern of sounds which signifies a certain meaning in the world. There is no necessary reason why the particular combination of sounds represented by a certain word should mean what it does. Hence, the relation between a word and its meaning is said to be arbitrary, though there are a few onomatopoeic words in all languages which sound like their meaning in the real world. One of the initial definitions of words can be as "A word is *an arbitrary pairing of sound and meaning*." (Akmajian et. al., 2004:16). But the authors of the book themselves call it inadequate to correctly represent the word. One other definition is given in Wikipedia that may be said to be more systematic "A **word** is a unit of language that represents a concept which can be expressively communicated with meaning. A word consists of one or more morphemes which are linked more or less tightly together, and has a phonetic value."

It has long been recognized that words must be classed into at least two categories: simple and complex. A simple word such as *tree* seems to be a minimal unit; there seems to be no way to analyze it, or break it down further, into meaningful parts (Akmajian et. al., 2004:16). On the other hand, complex words are those which are formed by two or more meaningful parts and can be analyzed or braked down into its components called morphemes. In some cases a morpheme may not have an identifiable meaning, and yet it is recognizable in other words (as *-ceive* in *receive, perceive, conceive, deceive*).

Morphemes

Morphemes are the minimal units of word building in a language; they cannot be broken down any further into recognizable or meaningful parts (Akmajian et. al., 2004:17). Simple words are also morphemes as they cannot be further broken down into meaningful parts.

Morphemes are categorized into two classes: free and bound morphemes. A free morpheme can stand alone as an independent word in a phrase or sentence such as *tree*. A bound morpheme cannot stand alone but must be attached to another morpheme – as plural morpheme *–s*. The bound morphemes can further be classified into affixes, bound bases and contracted forms. Affixes may be prefix infix or suffix. The affixes are always attached to some other morpheme, which is called a base. An affix is referred to as prefix when it is attached to the beginning of the base such as *re-* (*redo*, *recall*). Affix is an infix when it is attached within the base. Most of the languages do not have infixes. The affix attached to the right of the base morpheme (in case of left to right script languages, otherwise left of base) is referred to as suffix such as plural morpheme *–s* and past participle morpheme *–en*. Bound bases are those which cannot stand independently but should be attached to some other morpheme. For example *cran-*, it does not have its independent meaning, it must be combined with the name of some fruit *–berry*, *–apple*, *–grape* etc. Some auxiliary verbs in English get contracted or shortened when come with some other word. They cannot stand independently. They are said to be contracted forms of some words as -he*'ll* for will, -I*'ve* for have, -they*'d* for had or would.

Sometimes a distinction is made between *open-classed words* and *closed-class words*. The open class words are those belonging to the major part-of-speech classes (nouns, verbs, adjectives, and adverbs), which in any language tend to be quite large and open-ended. On the other hand closed-class words are those belonging to grammatical or functional classes (such as articles, quantifiers, and conjuncts etc.). For this, open and closed class words are also referred to as *content words* and *function words* respectively.

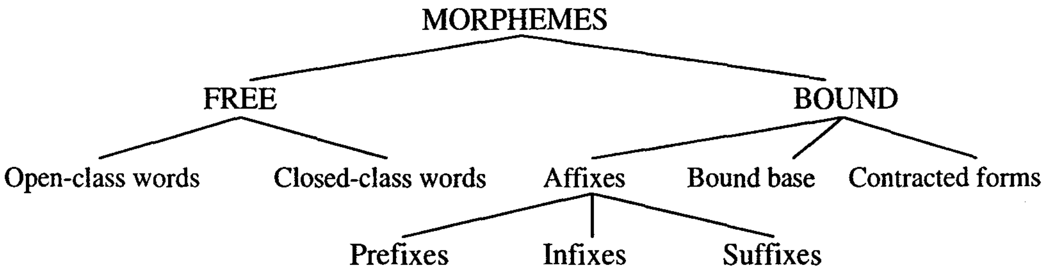


Fig. 1.1: Classification of morphemes (Akmajian et. al., 2004:24)

Parts of Speech

For linguistic purpose, all words in a language are categorized into a few grammatical categories. Each word belongs to a grammatical category. Grammatical categories and their number may differ among different languages. This is why because all languages carry some special features in them. According to Akmajian et. al., in English there are ten grammatical categories or parts-of-speech or, in short, POS. those are as following:

Noun (such as *brother*), Verb (such as *run*), Adjective (such as *tall*), Adverb (such as *quickly*); these four come under open class words. Conjunctions (such as - *and, or*), Articles (*a, an, the*), Demonstratives (*this, that*), Prepositions (such as *to, from, at*), Comparatives (such as *more, less*), Quantifiers (such as *all, some*), these six come under abovementioned closed class words. Traditionally we have learnt eight parts-of-speech in English namely; Noun (proper noun, common noun and abstract noun), Pronoun, Verb, Adjective, Adverb, Preposition, Conjunction and Interjection. But this list of eight does not show the place of some of categories like articles.

In Sanskrit, there has been long discussion about the number of categories of the words. In this discussion the number of categories of words varies from one to five. One thought says that there is only one category of words; whichever is meaningful, that is word (*arthaḥ padam*). This is the view of Indra school of *vyākaraṇa*. Other thought considers the two categories of words- noun and verb. It is the view of Pāṇini's definition *sup-tiṅantaṃ padam* (P 1.4.14), though he recognizes all four or five categories of word. Rest otherwise famous words can come under these two categories. The third thought of Nyāya philosophy in *śabda-*

śakti-prakāśikā of Jagadish accepts only three categories of words- base, suffix and particles (*prakṛti, pratyaya* and *nipāta*) (*Nirukta*, Rishi). Prefixes are one kind of particles. The fourth thought agrees with four categories of words- noun, verb, particles and prefixes. This is the view of Yāska and grammarians. The fifth thought adds one more category named as *karma-pravacanīya.* This view is accepted by later grammarians. The different views are mentioned in Durga's commentary on *Nirukta* on number of word categories.

Neologism

As earlier mentioned, according to Akmajian et. al. (2004), four grammatical categories belong to open class words. It means, in these categories, the number of words can increase and the role of existing words can change. This process of adding new words or changing the role of existing words is called neologism. Apart from other techniques to add new words, new words can enter a language through the operation of word formation rules, which is studied in the field of linguistics named as derivational morphology. Some most popular techniques of neologism are given as follows. (Amajian et. al., 2004:25-42)

Coined Words: These types of words are entirely new for the language and previously non-existent. These are often invented by the speakers.

Acronyms: acronyms are the words pronounced as common words and each of the letters that spell the word is the first letter (or letters) of some other complete word. For example, radar derives from radio detecting and ranging.

Alphabetic Abbreviations: These type of words are also formed taking first letter (or letters) of other complete words but the difference from acronyms is that acronyms are pronounced as common words while in abbreviations each letter of the word is individually pronounced. For example, CD is used for compact disc.

Clipping: Clipped abbreviations are those shortened forms of words whose spellings are shortened but the pronunciation of those is not changed. For example, Prof, MB and Dr. are shortened forms of professor, megabyte and doctor respectively and are pronounced as full words.

Blends: New words are also formed from existing ones by various blending (can say fusing) processes, for example motel from motor hotel, bit from binary digit and COLING from computational linguistics.

Generified Words: Sometimes new words are created by using specific brand names of products as name of products in general. For example Xerox is the name of the corporation that produces a photocopying machine, and now the term is used to describe photocopying process in general. Similar is the case of Dalda, the name of a brand of vegetable ghee (in many regions), and the use of word Moon for natural satellite of any planet. This time tendency is seen of using *Akash-ganga* (Hindi name of Milky Way) for galaxy in general (Hindi – *Mandakini*).

Proper Nouns: Sometimes the proper names of persons are generalized and become the part of language and are used to denote some specific behavior of the person whose name it was.

Borrowing Direct and Indirect: Sometimes the words of other languages are adopted in the language. When the word is taken as it is in its original language, then it is direct borrowing, for example, kindergarten word is borrowed in English from German. In some other way, the words of other language are literally translated into the native language. This type of borrowing is called indirect borrowing of words.

Changing the Meaning of Words: Sometimes, a new meaning becomes associated with an existing word in many ways: change in grammatical category or part of speech, vocabulary extension of one domain to another domain (metaphorical extension), broadening of scope of meaning, narrowing of scope of meaning, restricting the more general compositional meaning of complex words (semantic drift) and change of the meaning of a word to the opposite of its original meaning (reversal).

Derivational Morphology: When the previously existing words undergo under some word formation rules that a new word is created, the process is called derivational morphology. This process involves mainly two processes: Compounding and Affixation. Compounds are formed by combining two or more pre-existing words in which the meaning of the compound is related with them. In a compound, one word is the head word and the meaning of the compound is of same part-of-speech as it, and other words are modifier of the meaning of headword in compound. In affixation, several derivational affixes (mainly suffixes) are added to the word to form a new word. Many times the part-of-speech in derivation gets changed but it is not necessary for all time. The derivational morphology will be discussed in detail in some following section.

Backformation: This process is of psychological interest that some common words apparent as derived words from word formation rules are mistaken as derived words. Then the speakers of the language apply word formation rules in reverse and get the apparent base form of the original. The verb to beg is reversely derived from the previously existing word beggar. Such are the verbs to hawk, to stoke, to edit etc. This process is called backformation.

1.3.4 INFLECTIONAL AND DERIVATIONAL MORPHOLOGY

In the study of word formation, a distinction is often made between inflectional and derivational morphology. When base word goes through a process to form a final word ready to use in sentence, it is inflectional morphology. It further can be categorized as verbal and nominal. A verb root is attached to a class of eighteen suffixes together called *tiṅ*, forms verbal forms in ten *lakāra*s, three *puruṣa*s and three *vacana*s in *ātmanepada* and *parasmaipada*. The word thus formed under verbal derivational morphology is called *tiṅanta*. When a nominal base is attached to any of twenty one *sup*-suffixes to create nominal form in seven *vibhakti*s and three *vacana*s is nominal inflectional morphology. The word thus formed is called *subanta* (*sup+anta* → voicing).

When a base word goes through a process to derive another base is called derivational morphology. It is again verbal and nominal. Verbal derivational morphology involves verb to verb derivation- passive (verb root+*yak*), causative (root+*ṇic*, *ṇijanta*), desiderative (root+*san*, *sannanta*), … (root+*yaṅ*, *yaṇanta* and *yaṇluganta*). It also includes formation of new verb from nouns (*nāmadhātu*). Nominal derivational morphology includes *kṛdanta*, which is primary derivative and derives from verb root; *taddhita*, derives one nominal base to another nominal base; feminine forms, and compounds.

1.3.5 PROBLEMS IN MORPHOLOGICAL ANALYSIS

In doing morphological analysis, we find some exceptions of many aspects of a given analysis. Three of these problems in isolating the base of a complex word involve productivity, false analysis and bound base morphemes.

Productivity

Some morphological operations are found quite frequently with one category of words, but exceptionally, some forms are also found to have similar operation. For example, able suffix frequently occurs with verbs but it is found with some nouns also, like, *knowledgeable*, *fashionable*, and *marriageable*. But exceptions are closed list words.

False Analysis

Sometimes it happens that a basic word appears similar to derived or inflected forms of other words of a word category, and the speakers wrongly take it for being the same. They take it for a derived or inflected form and wrongly analyze it and reach a new base which is not the word of that language. This happens only due to accidental phonetic resemblance with the similar forms. In these cases, the phonetic component (apparent morpheme attached to a word) similar to the regular form of some morpheme, doesn't have similar meaning as the regular morpheme. So the analysis on this basis is called false analysis.

Bound Base Morphemes

This case is slightly different from the case of false analysis that here the meaning structure of the word similar to regular form is same as regular forms. The prefix or suffix attached to the word is recognizable and has similar meaning also, but if the base is detached from that, the base is not a meaningful word of the language. These types of bases are bound and can exist only with some specific morphemes.

1.4 PHONOLOGY

Pāṇini defines sandhi as extreme closeness of sounds - *paraḥ sannikarṣaḥ saṃhitā* (P 1.4.109), and sound is subject of phonology. Phonology is the subfield of linguistics that studies the structure and systematic patterning of sounds in human language. The term phonology is used in two ways. On the one hand, it refers to a description of the sounds of a particular language and the rules governing the distribution of those sounds. That can be said phonology of a language. On the other hand, it refers to that part of the general theory of human language that is concerned with the universal properties of natural language sound system (i.e. properties related to many, if not all, human languages) (Akmajian et. al., 2004:109). Speech sound in its original form is a continuum of waves in a certain range of frequency which can be heard by human ear. We can write a language with discrete symbols. However, speech is for the most part continuous; and neither the acoustic signals nor the movement of the speech articulators can be broken into the kind of discrete units that correspond to the units represented by written symbols. What is studied in phonology is the distinctive features of speech sounds, external organization of the sounds- the syllable structure, general framework for describing the sound pattern of human language, conditions where a particular sound changes its form into another (Akmajian et. al., 2004:110).

1.4.1 DISTINCT FEATURES OF SOUNDS

Basic components of a language are the sounds (phones and phonemes). The sounds are themselves composed of several features of articulation. These features are the constituents as well as distinctive features of the sounds. These features distinguish one sound from another and also make small classes of sounds. Some of these features are very important and others are comparatively less. The importance of the features vary from one language to another on the basis of how much role does it play in the statement of phonological rules and distinguishing the different sounds.

Morris Halle and Noam Chomsky proposed a distinctive feature system to classify and distinguish the sounds of a language in their work *Sound Pattern of English* (SPE) in 1968. Their proposals in turn build on the pioneering work in distinctive feature theory carried out by Halle and Roman Jakobson in 1956. In the SPE system the articulatory features are viewed as basically binary, that is, as having one of two values: either plus value (+), which indicates the presence of the feature, or a minus value (–), which indicates the absence of the feature (Akmajian et. al., 2004:114).

The sounds are primarily classified as vowels and consonants. According to SPE, the list of distinctive features of sounds (particularly for English) is given as follows:

Distinctive Features of Vowels: Syllabic, High, Back, Low, Round, Tense or Long.

Distinctive Features of Consonants: Syllabic, Consonantal, Sonorant, Voiced, Continuant, Nasal, Strident, Lateral, Distributed, Affricate, Labial, Round, Coronal, Anterior, High, Back, Low.

Same list will not be sufficient for Sanskrit. In vowels, the additional features are rising (*udātta*), falling (*anudātta*), level (*svarita*), nasality (*anunāsikatā*), extra-long (*pluta*). In consonants, the additional features are aspirated (*mahāprāṇa*), unaspirated (*alpaprāṇa*), breath (*śvāsa*), sound (*nāda*), open (*vivāra*), close (*saṃvāra*), slight open (*īṣad-vivṛta*), slight close (*īṣat-saṃvṛta*), stop (*sparśa*). Some of the above features mentioned in SPE are not applicable on Sanskrit sounds. The list of Sanskrit sound features famous in tradition of *vyākaraṇa*[3] also may not be sufficient from linguistic point of view, though *prātiśākhyas*[4] have made deeper observation.

1.5 MORPHOPHONEMICS: COMPUTATIONAL PERSPECTIVE

Goal of morphophonemics is to study and explain the rules and patterns of sound functioning in morphological process in a language. Computational morphophonemics differs from it in many ways. The goal of computational morphophonemics is rather more practical and more applied – to prepare a text for further linguistic processing/morphological–syntactic analysis, and on other side (generative sandhi), to combine morphemes in their usual phonetic form as occurring in the language. Second thing is its input and output. In general, morphophonemics applies on a piece of a linguistic data. The studying person recognizes the words and also has in mind their meanings. On the other hand, input and output for computational morphophonemic processing, both are only strings of symbols. The third thing departing these two is the methodology of processing. In general morphophonemics, human way of processing is parallel processing. He minds the sound patterns, words, their meaning and the context of use. He can also mind the pragmatic and cultural information. In contrary, computer has to cut, paste and match the parts of string alongwith some statistical processing of the intermediate results.

Sanskrit grammar has an advantage from computational view. That is the explicitly described sandhi rules and excellent formal system of Pāṇini.

[3] *Yatno dvidhā - ābhyantaro bāhyaśca. Ādyaścaturdhā - spṛṣṭeṣatspṛṣṭa-vivṛtaisaṃvṛta-bhedāt. tatra spṛṣṭaṃ prayatanaṃ sparṣānām. īṣatspṛṣṭaṃ antaḥsthānām. vivṛtamūṣmaṇāṃ svarāṇāṃ ca. hrasvasyāvarṇasya prayoge saṃvṛtam. prakriyā-daśāyāṃ tu vivṛtameva.* (SK 10)
bāhya-prayatnastvekādaśadhā - vivāraḥ, saṃvāraḥ, śvāso, nādo, ghoṣo'ghoṣo'lpaprāṇo mahāprāṇa udātto'nudāttaḥ svaritaśceti. (SK 12)

[4] Allen, W. S., 1965; Phonetics in ancient India

1.5.1 SYSTEM OF PĀṆINI: A MODEL

The system of Pāṇini consists of five major components: rule base (*sūtra-pāṭha*), verb database (*dhātu-pāṭha*), nominal database (*gaṇa-pāṭha*), gender rules (*liṅgānuśāsana*), and special morphological rules (*uṇādi sūtra*) which work in the environment of its sound system called *Śiva-sūtra*. The rule base named *Aṣṭādhyāyī* being the main component; the integerated system Pāṇini can also be called *Aṣṭādhyāyī*.

Method of analysis in *Aṣṭādhyāyī*

The grammar of Pāṇini is a generative grammar. It does not start analysis from ready form in use to its building blocks. Starting point of Pāṇini's analysis is not meaning or intention of the speaker, but word form elements as shown in initial stage of generation process. Here morphemic elements obtained from analysis are put side by side in an order of *previous and next* from left to right. Then by applying operations to these elements a derivation process starts. The process results in a word fit for use. Thus we may say that Pāṇini starts from morphology to arrive at a finished word where no further rules become applicable (Joshi: 2009). We can recall here the basic assumption of Finite State Automata.

Organization of system of Pāṇini (Jha: 2004): Pāṇini's grammar's main and central component is *Aṣṭādhyāyī*, the linguistic *sūtra*s divided into eight chapters with four subsections each. The *sūtra*s are 4000 approximately (the actual number varying from 3975 to 4025). The components of *Aṣṭādhyāyī* can be grouped into three modules: Sound classes (*Śiva-sūtra*), Rule base (*Aṣṭādhyāyī sūtra*s and special morphological rules named as *uṇādi-sūtra*) and Lexicon (verb database- *dhātupāṭha* and nominal database- *gaṇapāṭha*). This closely corresponds to a fully functioning NLP system:

phonetic component → grammar component → lexical database.

The phonetic component- the 14 *Śiva-sūtras* are the phoneme inventory of Sanskrit with 43 elements with 9 vowels and 34 consonants including two times *'h'*. These *sūtra*s do not show the long and protracted vowels, accented vowels and nasalized vowels and some other characters. The vowels enumerated in the *sūtra*s are vowel classes which represent all its types based on length, accents and nasality. Other special characters are allophones of *anusvāra* (*ṃ*) and *visarga* (*ḥ*) and they are assumed to belong to '*aṭ*' group or sigla. The *sūtras* contain the sounds and one bounding sound which are not counted as phoneme. The sigla are formed by taking any sound as start and any of the following bounding sound. Sigla thus formed can be hundreds in number (305) but Pāṇini has used only 41 of them. Later Pāṇinian grammarians include one more '*ra*' in this list. The arrangement of the sounds is so systematic that any sigla formed has some unique property in common that puts them in a class and thus becomes useful for rule writing on the basis of sound-features (This 1.3.1).

The grammar component- the *sūtra–pāṭha* (SP) is composed of *sūtra*s in eight chapters of four subsections each. Faddegon (1936) gave a general sketch of what is covered in different sections of the grammar and also analyzed the subsections. In the arrangement of SP he noted the "tendency towards dichotomy" and divided the rules into two main sections – Ch 1-5 and 6-8 which he called analytic and synthetic parts respectively. Kapoor (1992) has reduced the treatment of subject matter into four divisions – Ch 1-2 dealing with classification an enumeration of bases and categories, Ch 3-5 consist of *prakṛti-pratyaya* enumeration, and derivation of bases, Ch 6-8.1 deal with the synthesis of *prakṛti-pratyaya*, and Ch 8.2-8.4 deal with the rules of morphophonemics (cf. Jha: 2004). Sharma (2003) describes in detail the distribution of topics in SP. The *sūtra*s are categorized into six types. It can be seven if negation is taken independent of restriction.[5]

- Operational (*vidhi*) rules which state a given operation to be performed on a given input. For example- akaH savarNe dIrghaH (P 6.1.101) – simple vowels [a i u RRi LLi] will be lengthened when they are followed by a similar (savarNa) vowel.

- Definitional (*sa~nj~nA*) rules which assign a particular term to a given entity. For example- *vṛddhirādaic* (P 1.1.1)[6] – vowels [ā ai au] are termed as *vṛddhi.*

- Meta-rule (*paribhaShA*) rules which regulate proper interpretation of a given rule or its application. For example- *tasminniti nirdiṣṭe pūrvasya* (P 1.1.66)[7] – any operation will be applicable to the component which is prior to that which is denoted by seventh case like '*in that*'.

[5] Definitions of sutras are according to Rama Nath Sharma

[6] Example different from source

[7] Example different from source

- Heading (*adhikAra*) rules which introduce a domain of rules sharing a common topic, operation, input, physical arrangement etc. For example, *kārake* (P 1.4.32)[8] states that the rules about case start from here.

- Extension (*atidesha*) rules which expand the scope of a given rule, usually by allowing the transfer of certain properties which were otherwise not available. Example- karturIpsitatamaM karma (P 1.4.49) tathA yuktaM cAnIpsitam (P1.4.50) – that which is the most desired by kartA (agent) is karma (P 1.4.49) and also undesired similarly attached to the verb (P 1.4.50).

- Restriction (*niyama*) rules restrict the scope of a given rule. For example- patiH samAsa eva (P 1.4.8) – the word '*pati*' is termed as '*ghi*' only in compound. This rule restricts its previous rule- sheShoghyasakhi (P 1.4.7).[9]

- Negation (*niṣedha*) rules which counter an otherwise positive provision of a given rule. For example- tulyAsya-prayatnaM savarNam (P 1.1.9) and nAjjhalau (P1.1.10) – savarNa is the class of sounds with comparable place and manner of articulation (P 1.1.9). This cannot be across vowels and consonants even if they bear comparable place and manner of articulation (P 1.1.10).

Ramanath Sharma mentions two more categories of rules[10]

- Optional rules (*vibhāṣā*): rules which render the provision of a given rule optional. For example- *śi tuk* (P 8.3.31) – in sandhi, '*tuk*' is augmented optionally before '*ś*'.

- Ad hoc rules (*nipātana*): rules which provide forms to be treated as derived even though derivational details are missing. For example- *kṣayya-jayyau śakyārthe* (P6.1.81) – in the sense of '*liable to*' or '*possible to*' the derived forms of '*kṣi*' and '*ji*' with '*yat*' suffix are '*kṣayya*' and '*jayya*' respectively.

The Lexicon- verb and nominal databases: first is the database of verb roots in Sanskrit. The total number of the roots listed in it is 2014 out of which about 500 are yet found in actual use in the Sanskrit corpus/literature. The roots are classified into ten classes (*gaṇa*s). One eleventh

[8] Example different from source

[9] Explanation different from source

[10] Examples not from source

class is the roots basically derived from nouns. The ten (or eleven) classes are named after the first root of the class.

Second lexical database is of primitive nominal bases. This is the lists of groups of different bases (including pronouns).

Recently Houben (2009)[11] proposed a different architecture of Panini's System keeping *dhātupāṭha* as the central component. He gives a thesis of construction grammar in which data is the central component and rules or procedure is supporting component. In support of his thesis, he quotes the derivation process which starts with selection of a *dhātu* with all its grammatical information and then the rules apply.

Technical Devices of Pāṇini

Joshi in Background of *Aṣṭādhyāyī* (2009) explains the nature of Pāṇinian derivation process as "Starting point of Pāṇini's analysis is not meaning or intention of the speaker, but word form elements as shown in the initial stage of the *prakriyā*. Here morphemic elements obtained from analysis are put side by side in an order of *pūrva* and *para* from left to right. Then by applying operations to these elements a derivation process starts. The process results in a word fit for use in *vyavahāra*, the everyday usage of an educated Brahmin. Thus we may say that Pāṇini starts from morphology to arrive at a finished word where no further rules become applicable. We have to bear in mind that Sanskrit is an inflecting language."

Pāṇini's operational rules are generally substitution rules. Here the distinction between the original (*sthānin*) and the substitute (*ādeśa*) is essential. As far as further rule application is concerned, the substitute is declared to be like the *sthānin* (P 1.1.56). An exception is made for the rules which deal with the substitution of phonemes. An ingenuous idea of Pāṇini was to extend the concept of substitution to zero-substitution (*lopa*) also. *Lopa* is defined as *adarśanam* "disappearance from sight" (P 1.1.60) (Joshi, 2009).

Three other techniques of Pāṇini have very much importance in *Aṣṭādhyāyī*. Three of those are inheritance (*anuvṛtti*), *utsarga-apavāda vidhi*, and division of *sūtra*s in *siddha* and *asiddha kāṇḍa*. Inheritance or *anuvṛtti* on one side is important from point of view of conciseness (*lāghava*), it is also important from the view of formal structure. In a program, a

[11] Third international Computational Linguistics Symposium, Hyderabad, India, 2009

definition or procedure is added some new aspects and elements by inheritance. Similarly, *anuvṛtti* brings its all aspects with it. This thing is well proved in the following example of *karma sañjñā*. In the rule *karturīpsitatamaṃ karma* (P 1.4.49), *kārake* (P 1.4.23) is inherited, thus the indicated term has two names (*sañjñās*) - *kāraka* and *karma*. Here word *karma* is explicitly used while it is used in previous rule *adhi-śīṅ-sthā-''sāṃ karma* (P 1.4.46), and it could be inherited from here. But it is not inherited from there because there the *karma-sañjñā* was assigned to a locus (*ādhāra*). If the word *karma* was inherited form there, the sense of locus also would have been inherited.

Second thing is *utsarga-apavāda-vidhi*. When two rules are applicable in same condition and have equal strength of applicability, then relatively one of them becomes *utsarga* and other *apavāda*. *Apavāda* is that which has less scope of applicability and its scope is proper subset of the scope of *utsarga sūtra*. In this condition, the *utsarga sūtra* gives up its right (*utsṛjyate*) and leaves the scope for *apavāda*.

Third technique is the division of whole *Aṣṭādhyāyī* into *siddha* and *asiddha kāṇḍa*. *Siddha kāṇḍa* is upto first quarter of eighth chapter. The rules of *siddha kāṇḍa* are recursively applicable anytime it has scope for application. But the rules of *asiddha kāṇḍa* are applicable after rules of *siddha kāṇḍa* have done their work. Technically, the result of *asiddha* rule is invisible to *siddha* rule. It means, *siddha* rule cannot be applied after the application of *asiddha* rule. In the *asiddha kāṇḍa*, every later rule is *asiddha* in respect of prior rules. The rules of *asiddha kāṇḍa*, the last three *pāda*s of last chapter (*tripādī*) need to be applied in sequential order. This system of *asiddha* regulates infinite or unnecessary recursion. Apart from these, some part of *Aṣṭādhyāyī* (6.4.22 to 6.4.175) is *asiddhavat prakaraṇa* which is not included into *siddha kāṇḍa*. The *sūtra*s of *asiddhavat prakaraṇa* that are all having same condition, can be thought of as applied simultaneously. *Siddha-asiddha* structure is graphically interpreted by Subbanna and Varkhedi (2009)[12].

1.5.2 PROBLEMS IN COMPUTING PĀṆINI

Pāṇinian grammar is said to be the most technical and formal grammar of any natural language in the world, though it is not fully computed after the researches and efforts of the decades. There are some most common problems in computerizing Pāṇini's system.

[12] Subbanna and Varkhedi, TISCLS-2009, Hyderabad

Pāṇinian grammar is a formal grammar yet not a computational grammar, but it is linguistic grammar of a natural language written in formal version of natural language. It has common weaknesses of natural language. One most common weakness of natural language is ambiguity. Cases in AD have their technical meaning but sometimes different cases have similar form and create ambiguity. For example, V and VI case singular have often same form but V case means '*after it*' whereas VI case means '*to it*' or '*of it*' or '*in the place of it*'. One such *sūtra* is *maya*$^{5/6}$ *uño*$^{5/6}$ *vo*^{1} *vā* (P 8.3.33)[13]. One such *vārtika* is *yaṇo*$^{5/6}$ *mayo*$^{5/6}$ *dve*$^{1/2}$ *vācye*$^{1/2}$ (v on P 8.2.23)[14]. In the case of this *vārtika*, the ambiguity of I case and II case is not the problem because II case is not assigned a meta-linguistic meaning and here its meaning is simply of I case (as general).

Jha and Mishra (2008)[15] explain the problems in implementation of *kāraka* formalization. The most part of the *kāraka* involves complex semantics. Semantic representation of material entities itself is very hard to implement with relation to other objects; *kāraka* rules involve semantics of abstract and subjective concepts. They classify implementation issues with respect to *kārakas* in three categories as follows:

- *Vivakṣā* dependent operations – as in *sthālyā pacati, sthālī* should be termed *adhikaraṇa* (locus) as it is the *ādhāra* (*ādhāro'dhikaraṇam*), but it is termed as *karaṇa* by rule *sādhakatamaṃ karaṇam* because the speaker thinks it is the most instrumental.

- *Vārtikas* limiting/extending rules – rule *gati-buddhi-pratyavasānārtha-śabdakarmākarmakāṇām aṇi kartā sa ṇau,* which allows *karma*, is limited by *vārtika-nīvahyorna,* which itself is limited by another *vārtika- niyantṛ-kartṛkasya vaheraniṣedhaḥ.*

- Semantic conditions necessary for implementing a rule – conditions like, *svātantrya, īpsitata/īpsitatama/anīpsita, sādhakatama, abhipraiti, dhruva, apāya, ādhāra* etc.

The AD is generative grammar and not analytical grammar. Analysis is to be derived by applying the Pāṇini's rules of substitution in reverse. Madhava M. Deshapande notes that Pāṇini's grammar "does not provide us with analytical tools to go from texts to their

[13] Superscript numbers show case number of the word.

[14] *saṃyogāntasya lopaḥ* (P 8.2.23)

[15] Second international Sanskrit Computational Linguistics Conference, Brown, (2008)

interpretation, or from sentences to morphemes and phonemes. It presupposes the result of an analytical phase of scholarship, but is itself not representative of that analytical phase." (c.f., Anand Mishra, 2009)[16].

Aṣṭādhyāyī is not a grammar in western sense of the word. It is a device, a derivational word generation device. It presupposes the knowledge of phonetics and it is based on morphemic analysis. It derives an infinite number of correct Sanskrit words, even though we lack the means to check whether the words derived form part of actual usage. Correctness is guaranteed by the correct application of rules (Joshi, 2009)[17].

The technical terms in *Aṣṭādhyāyī* (*saṃjñā*) are mostly defined but not always. The non-defined *saṃjñā*s are borrowed from various other branches of science supposed to be generally known. Joshi (2009) mentions some important of them- *mantra, yajus, napuṃsaka, liṅga, kriyā, vartamāna, vibhakti, prathamā, jāti, dravya, guṇavacana, visarga, vākya, vidhi, samartha* and *upamāna. Pratyāhāras* are also the technical terms of the *Aṣṭādhyāyī* but these are well defined (Joshi, 2009).

A wider part in Pāṇini's grammar requires the meaning information. Some scholars claim that semantics is not the subject matter of *Aṣṭādhyāyī* (Joshi, 2009) in the sense of meaning as thing meant. *Aṣṭādhyāyī* is no authority to decide that this word refers to this item. That comes from the usage. The whole of *taddhita*-section testifies the more importance of lexical meaning. Each *taddhita* suffix is used to derive a new word in a specific relation with base word. Here meaning becomes of much importance. To compute this portion requires to some extent the computing of the meaning of the suffixes. An example of complex meaning in rule is- *ṇer-aṇau yat karma ṇau cet sa kartā'nādhyāne* (P 1.3.67) (which is *karma* or object in non-causative state, if becomes *kartā* or agent in causative state, then *ātmanepada* suffix is used with the verb).

As earlier has been said that some of the technical terms of Pāṇini are not defined in *Aṣṭādhyāyī*, they are borrowed from other fields of science and are presupposed to be already known. Also there is not any medium in the system to check whether the word exists in the usage. In other words, there is no way to restrict unlimited over generation of the correct forms. Some words used in AD represent to a class of nouns (including all kinds of nominal

[16] Mishra Anand, TISCLS – 2009, Hyderabad

[17] Joshi, S. D., keynote address in TISCLS-2009, Hyderabad

bases), which are sometimes found in AD sutras itself, others are found in separately arranged lexicons- *daātupāṭha* and *gaṇapāṭha.*

When more than one *sūtra* are applicable on the same state of derivation, there is need to resolve the conflict and to determine the exact *sūtra* to be applied. There is no clear specification in the *Aṣṭādhyāyī*, but a *vārtika* specifies it- *para-nityā-ntaraṅgā-pavādānām-uttarottaraṃ balīyaḥ* – where *para* is later in terms of order in *Aṣṭādhyāyī*; *nitya* is that which is applicable in both conditions whether the conflicting *sūtra* is applied or not; *antaraṅga* is a rule about verb root and prefix (*dhātūpasargayoḥ kāryam antaraṅgam*); and *apavāda* is exception rule, which has no scope of application outside the area of main *sūtra.*

apavāda > antaraṅga > nitya > para

The rules applicable in one condition or defining one technical term together form a rule cluster or rule set.

After the computation of individual rules or rule clusters (rule sets), there is again complex system of order of application of rules. From this view, the AD is divided mainly in two sections- *siddha-kāṇḍa* and *asiddha-kāṇḍa.* In the *siddha-kāṇḍa*, there are two ranges of *sūtra*s *asiddhavat* (like *asiddha*). *Siddha* section contains *sūtra*s 1.1.1 to 8.1.74 (end of first *pāda*) minus *asiddhavat* section. *Asiddhavat* section contains *sūtra*s 6.4.22 to 6.4.175 (end of *pāda* and chapter). *Asiddha* section contains the *sūtra*s 8.2.1 to 8.4.64 (end of the chapter and the book). Subbanna and Varkhedi explain this system in somehow this way- In general, whenever conditions for a *sūtra* are satisfied, that *sūtra* is applied. Only one *sūtra* is applied at a time. In other words no two *sūtra*s can be applied simultaneously. However, the *sūtra*s of *asiddhavat prakaraṇa* that are having the same condition can be thought of as applied simultaneously. The *sūtra*s in the last three *pāda*s, that is, *tripādī* need to be applied in the sequential order.

1.5.3 CHALLENGES IN MORPHOPHONEMIC PROCESSING

Above we have seen the general problems in computing of Pāṇinian system. The computing of morphophonemic rules is more complex than other sectors of the grammar like morphology, phonology or syntax. Computing of the process of morphophonemic analysis is still more challenging. This sector requires the identification of all types of words, thus understanding of the complete morphology, though not necessarily ability of analyzing all words. Here are indicated some of the problems which will be discussed in detail in later chapters.

If we assume that the sandhi analysis system gives results perfectly according to the Pāṇinī's (including followers') sandhi rules, there are still many problems to reach the correct analysis. First and major problem is ambiguity. It simply means that a string of Sanskrit word can be split into many segments. As the number of segments increases, the number of sets of segments increases, and not linearly but geometrically. First the challenge is to through away invalid segmentation sets and then the challenge of selecting the desired result from the valid ones.

This ambiguity is caused by many factors. The most complexity generating problem arises from the nature of Sanskrit which is unique to this language, that is, theoretically, a continuous string, without pause can be infinitely long. Between the words, the sandhi is optional while it is essential within the word simple or complex. Also the minimum size of valid word is Sanskrit is very small, that is, one character. This requires searching the possibility of sandhi on each next character.

Next a few problems are related to application of rule. First thing is simple concatenation. Any word ending in consonant should be concatenated with the next word. It causes the need of judging if it is simple concatenation or sandhi, in other words, whether the sandhi rules are to be applied or not. Then, if it is decided to apply the rules, similar looking pattern of sounds (formally, a sequence of symbols) form the condition fit for application of many rules. This problem to be solved needs computation of strength of *sūtra* stated in previous heading. But this solves the problem of generation side, which is the process of Pāṇini. This is not equally efficient for analysis process. Afterwards, when a rule is decided, the same rule applies on different conditions and result in one form in generation. Thus, the same sequence of symbols in reverse sandhi process should result in multiple splits. Then it needs to select the possible ones from those multiple results. For this, the two splits are joined with left and right parts of string and thus formed segments are then checked in the dictionary to validate their existence in the used language. There is no more promising way than dictionary and corpus to verify that the word exists in the language. The finite state automata for Sanskrit words validation is too complex to implement.

Next few problems are related to the recognition of words. The whole exercise of sandhi processing is the recognition of the distinct words. The segment is a valid word or not, this can be judged by checking of the segment in the dictionary and corpus of the language. The dictionary contains the words in base forms while words in text are inflected. Thus, before validation, the inflectional analysis is required (both verbal and nominal). The inflectional analyzer generally should be run before seeking for application of rules because the inflection is must in Sanskrit words in sentence. But the inflections may occur within the string and may not occur (in the case of compound where the suffixes are elided). Word is to be checked in dictionary after analyzing or without analyzing? This may many times result to false analysis.

Finally the challenge is to arrive at all possible solutions and one (or a few) desired solution(s). (Sometimes, more than one segmentation is intended in cases like *śleṣa* usage.) For collecting all possible solutions, each point of segmentation needs to be split thrice-without applying any rule (for simple concatenation), applying possible and relevant rules, applying rules and then analyzing inflections. On each of three stages the segment needs to be validated. In this exercise, whole dictionary is accessed several times. When human segments sandhi, he accesses his active mental vocabulary, which is very small, so most of non-relevant words do not come to access. But in computer's case, it has very large dictionary, and whole dictionary is its active vocabulary resulting many unused and irrelevant words validated. Also, computers cannot have sense or guess of meaning while searching for validating segments whereas human has the sense earlier to applying the rule. The problem of large vocabulary can be solved by indexing the dictionary and marking the popularity index to each word. Processing the frequency of words can reduce number of solutions.

1.6 A SURVEY OF RESEARCH

1.6.1 APPLIED RESEARCH ON SANDHI ANALYSIS

Sanskrit Heritage site of Gerard Huet of INRIA, Paris, he has build segmenter which is online available on his website. This program takes a continuous string without space and segments all words whether they are simply joined or have sandhi within. This program gives many solutions. It takes input in Roman transliteration scheme which stems from Velthuis devnag TeX input convention and it displays result in IAST and Devanagari Unicode. Same site has sandhi generator module also which takes two words and returns sandhi word. It does both internal and external sandhi.

'*Sandhi Splitter and Analyzer for Sanskrit: With Special Reference to ac Sandhi*' is the M.Phil research of Sachin Kumar in supervision of Girish Nath Jha, submitted to Special Centre for Sanskrit Studies, Jawaharlal Nehru University, New Delhi (2007). A web based system is developed under this research and is available on the website of Special Centre for Sanskrit Studies. It works only for vowel sandhi splitting and takes Unicode Devanagari UTF-8 text as input. A web based sandhi generation program is also developed there as the MA course project of the students of the centre. This program takes input in i-trans encoding.

A sandhi splitter has been developed in Rashtriya Sanskrit Vidyapeetha, Tirupati and Department of Sanskrit Studies, University of Hyderabad in guidance of Amba Kulkarni and in consultancy with K. V. Ramakrishnamacharyalu. This program takes input a word in W-X notation, I-trans, Velthuis, HK, SLP and Devanagari Unicode encoding. In its older version available on the site, it gives analysis to the five levels of depth according to the wish of user. They have also developed sandhi generation program, which is online and takes input in same six notations and gives result in Unicode Devanagari with sandhi type and *sūtra*.

Sanskrit library project under the guidance of Peter M. Scharf, Classics Department, Brown University, is engaged in philological research and education in Vedic and classical Sanskrit language and literature. Current research involves linguistic issues in encoding, computational phonology and morphology. (cf. Surjit, 2008)[18].

Department of Information Technology, Ministry of Communication and Information Technology, Govt. of India had funded Jawaharlal Nehru University a CASTLE project (Computer Assisted Sanskrit Learning and Teaching Environment) under TDIL program. This is reported to have developed *sandhi-viccheda* system in DOS environment but it is not available to public access.

Software developed by Indian Heritage Group of Centre for Development in Advance Computing (C-DAC), Bangalore, DESIKA, is reported to have developed generation and analysis module for plain and accented written Sanskrit text. It has exhaustive database of

[18] *Kṛdanta Recognition and Processing for Sanskrit* M.Phil. Dissertation submitted to Jawaharlal Nehru University by Surjit Kumar Singh under supervision of Girish Nath Jha, 2008

Amarakośa, rule base using rules of *Aṣṭādhyāyī*, and heuristics based on *nyāya* and *mīmāṃsā śāstras* for semantic and contextual processing. The version of this software available on website does only nominal form generation.

1.6.2 THEORETICAL RESEARCHES IN PĀṆINIAN MORPHOPHONEMICS

Below are some theoretical researches in the direction of formal understanding of Pāṇini's phonology or sandhi. Girish Nath Jha in his M. Phil. research entitled "*Morphology of Sanskrit Case Affixes: a Computational Analysis*" under the supervision of Kapil Kapoor and G. V. Singh (in SLL&CS, JNU, 1993) introduces a formal way of writing Pāṇinian sandhi rules. Here is an example of formal sandhi rule:

> Retroflexization: ShTunAShTuH (8.4.41)
> +cons -> +cons /- +cons
> Dental retroflex retroflex
> Example: rAma-s- + ShaShThaH -> rAma-Sh-ShaShThaH

Malcom D. Hymen wrote a research paper in First International Sanskrit Computational Linguistics Symposium (2007) entitled "*From Pāṇinian Sandhi to Finite State Calculus*". In the paper he summarizes Pāṇini's handling of sandhi, his notational conventions, and formal properties of his theory. Then he introduces an XML vocabulary in which Pāṇini's morphophonemic rules can be expressed.

Cardona has made detailed and analytical survey on Pāṇinian research in his two books- '*Pāṇini- A Survey of Research*' (1976, 1980) and '*Recent Research in Pāṇinian Studies*' (1999). The earliest work recorded in the book is Kshitish Chandra Chatterji's '*Some Rules of Sandhi*' (1935). This deals with Pāṇini's morphophonemic rules and comparable rules of other descriptions. Cardona, in his two works, studied Pāṇinian morphophonemic rules in comparison with those of other treaties, seeking to extract Pāṇini's principles. His two works are respectively '*On Pāṇini's morphophonemic principle*' (published in *Language*; Journal of the Linguistic Society of America, Baltimore 1965) and '*Studies in Indian grammarians, I: the method of description reflected in śivasūtra*' (1969).

Madhav Murlidhar Deshpande in his work '*Critical Studies in Indian Grammarians, I: The Theory of Homogeneity [Sāvarṇya]*' (1975) contributed a monograph on Pāṇini's concept of homogeneity and sets of homogeneous (*savarṇa*) sounds, in which he deals also in detail with what Pāṇinīyas say. James S. Bare studied *Aṣṭādhyāyī* rules in connection with the feature system considered to be reflected in the Pāṇini's phonological system. The work of Bare, '*Phonetics and Phonology in Pāṇini: The System of Features Implicit in the Aṣṭādhyayi*' (1976) appeared in *Natural Language Studies* Vol. 21, University of Michigan. Work of Shanti S. Dighe, '*saṃhitāyām* I *A book on Pāṇinīya Rules of Sandhi*' (1997), is a short and elementary work concerning Pāṇini's sandhi rules. It is a useful presentation of a representative group of *sūtra*s that concern operations which take place, or are disallowed, when items are uttered continuously. In his article "*Did Pāṇini have a private concept of guṇa phenomenon?*" (1991-92), Gajanana Balakrishna Palsule proposes the thesis that Pāṇini held both an "official view" and a "private and intuitive view" concerning the *guṇa* class. After a long discussion, Cardona disagrees with this thesis saying, 'Pāṇini's "official view" is concerning his *guṇa* class is the only one we need deal with and that there is no cogent reason for seeking to find a "private view" concerning this class'. Sandhi splitting aims at recognizing the existing words and thus word boundary becomes important. Shivram Dattatreya Joshi in his article '*The role of boundaries in the Aṣṭādhyāyī*' (1984) discusses the boundaries which involve units Pāṇini calls *aṅga*, *bha*, and *pada*, as well as the boundary at pause (*avasāna*). Introduction of *aṅga* boundary by Pāṇini he regards as a major step in the linguistic analysis given by him.

Cardona's work '*Pāṇini: His Work and its Traditions*' has organized the rules of *Aṣṭādhyāyī* according to the category of rules and their operations. The sandhi rules are not listed in independent category because the phonological changes made by them are not independent of morphology. The sandhi rules are covered in rules of substitution, augmentation, doubling and rules of Pāṇini's derivational system.

Whitney is one of the firsts who wrote Sanskrit grammar in English. Chapter III of his grammar describes sandhi generation rules in detail. He calls sandhi '*euphonic combination*'. His organization of rules is too different from Pāṇini and his tradition. He has categorized the sandhi processes according to the sounds which have to undergo change such as 'combination of final s and r', 'conversion of s to ṣ' etc. He has also described some important aspects of Sanskrit morphology which are not commonly seen in the Sanskrit tradition such as general 'principles' of sandhi, and 'permitted finals' in Sanskrit etc.

William Sydney Allen has discussed Sanskrit sandhi in part III of his work *Phonetics in Ancient India* (1961). He considers not only Pāṇini and his followers but almost all works of phonetics and phonology like *śikṣā*s and *prātiśākhya*s. His observation is minute and his view is critical and comparative on the explanations of the Indian *śāstra*s. He does not explain those aspects, on which Sanskrit *śāstra*s do not say anything, but he discusses *śāstra*ic explanations critically and he is honest in criticizing the *śāstra*ic view and its criticism. The level of minuteness he goes into is necessary and useful for speech recognition and speech synthesis, for which, grammarians' explanation is not sufficient. All his explanation is based in Pāṇinian tradition, *prātiśākhya*s, *śikṣā*s and western grammarians who wrote Sanskrit grammar, but his critical view is his own.

1.6.3 RESEARCH ON FORMALIZING OF SYSTEM OF PĀṆINI

The following two researches are concerning with formal language and structure of *Aṣṭādhyāyī* which is much important from the view of computation. S. D. Joshi and Saroja Bhate in '*The Fundamentals of Anuvṛtti*' (1984) studied in some detail how *anuvṛtti* operates in the *Aṣṭādhyāyī*. Pierre-Sylvain Filliozat, in his article "*Ellipsis, lopa and anuvṛtti*" (1993) first published in Annals of BORI 72-73 (1991 & 1992) goes over major features of natural language observed in the *sūtra*s of the *Aṣṭādhyāyī*. Features of natural language cause difficulty in formalization of rules.

Anand Mishra (2007) proposes a model for simulating Pāṇinian System of Sanskrit grammar in FISCLS. He first defines the sets of phonemes, morphemes and lexemes as basic components. On the basis of these he defines process of forming sets of language components. After that he describes models of basic operations. Then through examples, he describes how his formal model defines different processes. In the last he gives overall layout of his Pāṇinian simulator. In his other paper (2009) he extends the same model to a complete grammatical circle formed by analysis and generation of forms. The process is completed in mainly four modules- input, analyzer, synthesizer and output. It aims at analyzing input, checking its *sādhutva* and generating the Sanskrit forms for use.

P. M. Scharf (2007 & 2009) published two papers in first and third International Symposia on Sanskrit Computational Linguistics entitled respectively, '*Modeling Pāṇinian Grammar*' and '*Levels in Pāṇini's Aṣṭādhyāyī*'. He throws light on the Pāṇinian architecture and different modern interpretation of Pāṇini. He examines the role of semantics and challenges in computerizing them and possibility of modeling Pāṇini's system in his first work. In both the works, he discusses different views of levels in Pāṇinian grammar first introduced by Paul Kiparsky and J. F. Staal and then by Huben and others. The second paper is mainly the critical evaluation of different views on levels.

Paul Kiparsky and Staal have first interpreted the Pāṇinian system from the view of the levels in 1969. Then he proposed four levels as 'Semantic representations', 'Deep structures', 'Surface structure' and 'Phonological representation'. In 1982 Kiparski revised the levels as 'Semantics', 'Abstract Syntax (e.g. kārakas)', 'Surface structure (morphology)' and 'Phonology'. In 2002, he again revised the levels as 'Semantic information', 'Morpho-syntactic representation', 'Abstract morphological representation' and 'phonological output' (c.f. Houben, 2009). Later Houben (2009, appendix) modifies this level-model of *Aṣṭādhyāyī.*

CHAPTER TWO

THE SANDHI SYSTEM OF PĀṆINI

Chapter two

THE SANDHI SYSTEM OF PĀNINI

2.1 INTRODUCTION

This chapter of the dissertation presents general concept of sandhi in Pāṇinian system. The *sūtra*s of sandhi are scattered in *Aṣṭādhyāyī. Siddhānta-kaumudī* (SK) has categorized them under *prakaraṇa*s of different types of sandhi. However, there are many instances of sound changes mostly of the internal sandhi type like *ṇatva*, are not covered in these *prakaraṇa*s. This chapter takes care of such issues and also presents sandhi rules, as much as possible, in the form of phonological rule writing.

2.2 SANDHI IN SANSKRIT

The word used for sandhi in *Aṣṭādhyayi* is *saṃhitā* and first defined in first chapter as *paraḥ sannikarṣaḥ saṃhitā* (P 1.4.109). *Paraḥ sannikarṣaḥ* is extreme closeness of sounds, that is *varṇānām atiśayitaḥ sannidhiḥ* (SK 28). In Pāṇinian grammar it is not clearly defined whether sandhi is sound change only on the boundary of morphemes or within the morpheme as well. The explanation of the rules in the tradition throws some light on it in the examples. The examples used in *Kāśikā* and SK use either complete words (*pada*) or morphemes to illustrate sound change. Thus we can say that traditionally sandhi is sound change which occurs when the two morphemes come together. This area of linguistic study is called morphophonemics. The traditional meaning of sandhi cannot satisfy the definition of morphophonemics because morphophonemics includes those sound changes also which take place in distant sounds from the point of juncture and that is within the morpheme. In other words, if we classify the processes of sound change into 'external sandhi' and 'internal sandhi', the traditional meaning of sandhi includes external sandhi only. The most important work which collects (almost all) sandhi rules together, the *Siddhānta-kaumudī* of Bhaṭṭoji-dīkṣita, also supports this hypothesis. Rules of internal sandhi are not explained in its *sandhi prakaraṇa*. There are the internal sandhi rules in *Aṣṭādhyāyī* but not in one place. SK also does not keep all of them together but in different *prakaraṇa*s and some rules are scattered in other different *prakaraṇa*s. For example two main *sūtra*s of retroflexisation of '*n*' are stated under vowel-ending masculine nominal morphology (*ajanta-pumliṅga-prakaraṇa*) – *raṣābhyāṃ no ṇaḥ samānapade* (P 8.4.1, SK 235) and *aṭkupvāṅnum-vyavāye'pi* (P 8.4.2, SK 197).

2.3 DISTRIBUTION OF SANDHI RULES

The main rule which heads the sandhi topic is in chapter six of *Aṣṭādhyāyī*, and that is – *saṃhitāyām* (P 6.1.72). It heads the rules upto P 6.1.158. Four rules (73-76) define conditions of augmentation of '*tuk*' ('*t*' attached with previous). Next few rules (77-83) are about vowel to semi-vowel change (*yaṇ* and *ayādi*). Then there are the rules for merging sounds i.e., replacement of two sounds by one sound, *ekādeśa* (84-111). *guṇa, vṛddhi* and *dīrgha* sandhi come under these. Three rules of '*u*' replacement (112-114). Then there are fourteen rules of no change or *prakṛtibhāva* (115-128). Six rules of non-category (129-134) and from 135 to 157 are the rules for augmentation of '*suṭ*' ('*s*' attached to later sound sequence)[1].

Another group of sandhi rules is in the same chapter in third *pāda* headed by similar sūtra *saṃhitāyām* (P 6.3.114). Rules in this group (P 6.3.114-135) are applicable in the juncture of words in formation of compounds. Most of rules do last vowel lengthening of the first word in compound in specific meanings; the vowels are not lengthened in other meanings. That is why SK keeps most of these rules in *samāsāśraya-prakaraṇa* and last five in *vaidika-prakaraṇa*.

The remaining clusters of sandhi rules are found in last three *pāda*s of the *Aṣṭādhyāyī*, that is *tripādī* or *asiddha-kāṇḍa*. The rules in P 8.2 are not the rules of sandhi but of morphology, but some of them apply on sandhi also. Such as *saṃyogāntasya lopaḥ* (P 8.2.23), *sasajuṣo ruḥ* (P 8.2.66). The rules of P 8.3 and 8.4 are all about sandhi. Here it is not a heading rule as in

[1] Pande, G. D. (2004:66,67)

previously mentioned rule groups, but the similar portion *saṃhitāyām* inherits from the last *sūtra* of previous *pāda* – *tayor-yvāvaci* ***saṃhitāyām*** (P 8.2.108) which goes till the end of the chapter and the book (P 8.4.68). Most of the sandhi rules are kept in *asiddha-kāṇḍa*, which are to be applied at the end and sequentially. The phonemic component gives the final form of the language. This is also discussed in the previous chapter while discussing the work of Kiparsky, Staal, Houben and others regarding levels in Paninian grammar. That is why most of these rules are supposed to be applied at the end in language generation. These rules are about consonant sandhi and the rules in chapter six are mainly of vowel sandhi. This section also has the rules of internal sandhi, retroflexisation of '*n*' and '*s*', doubling etc. The other purpose of keeping these rules in *asiddha* section may be to check unnecessary looping and backtracking. One possibility of such looping is in visarga cycle.

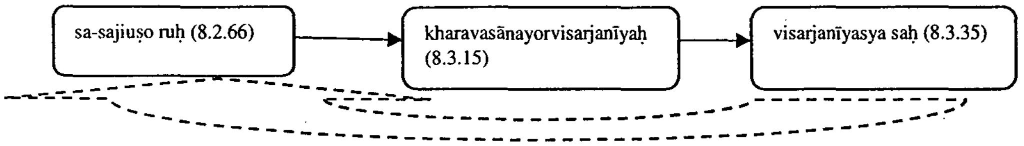

sa-sajiuṣo ruḥ (8.2.66)
kharavasānayorvisarjanīyaḥ (8.3.15)
visarjanīyasya saḥ (8.3.35)

Fig. 2.1 Visarga cycle without *asiddha*-system

The classification of SK is teaching and application oriented. Therefore many rules of sandhi are included in other different topics (*prakaraṇas*) and sandhi topic includes many of such rules which are not describing a rule for sound change, but they support the sandhi rules either by defining a term or as meta-rule.

2.4 SANDHI RULES OF SANSKRIT

Sandhi in Sanskrit is of five types – *svara* (vowel) sandhi, *vyañjana* (consonant) sandhi, *visarga* sandhi, *prakṛti-bhāva*, and *svādi* sandhi. Two more types of internal sandhi should be included as *ṇatva* and *ṣatva*. When last vowel of previous morpheme and first vowel of later morpheme come together and sound change occurs, the case is called vowel sandhi. The case of vowel sandhi may be of two types. One, in which one sound is changed and other remains the same. Second, in which both sounds are replaced with another resulting sound. The latter case is known as *ekādeśa*. First type includes *yaṇ* and *ayādi* sandhi, and second type includes *guṇa*, *vṛddhi* and *dīrgha* sandhi, *pūrvarūpa* and *pararūpa* sandhi.

When two consonants, or one consonant and one vowel come together and experience change, it is consonant sandhi. Like vowel sandhi, two sounds can result in two sounds or one sound, and also some extra sounds can be inserted as the result of morphophonemic process. The insertion of sound is called *āgama*. When *visarga* combines with some other sound, vowel or consonant, and the sounds experience change, this is *visarga* sandhi. Nominal inflected forms are formed by adding *sup* (*su, au, jas,...*) suffixes with the base. When a nominal word ending in *sup* suffix combines with other sounds following it, it is case of *svādi* sandhi.

2.4.1 VOWEL SANDHI RULES

Vowel sandhi has mainly seven sub-types mentioned above. *Prakṛtibhāva* takes place in the conditions of vowel sandhi, so SK keeps it under vowel sandhi topic. Following are the rules of vowel sandhi.

Rule A1 – vowel substituted by semi-vowel (*yaṇ-sandhi*)

The *sūtra* is *iko yaṇaci* (P 6.1.77)

[i,u,ṛ,ḷ] → [y,v,r,l](closest in place) / _+vowel

Exception: rule A7- vowel lengthening; rule C1; rule C2; P 6.1.102

When any type of a vowel among [i,u,ṛ,ḷ] is followed by a vowel, it changes to a semi-vowel [y,v,r,l] similar in place of articulation. Exceptions are – when a vowel is followed by similar (*savarṇa*) vowel; when *pluta* or *pragṛhya* is followed by a vowel, when first word is in vocative case; and morphological rule of first two cases – *prathamayoḥ pūrva-savarṇaḥ* (P 6.1.102).

Rule A2 – diphthong substituted by vowel + semi-vowel (*ayādi-sandhi*)

The *sūtra* is – *eco'yavāyāvaḥ* (P 6.1.78)

[e,o,ai,au] → [ay,av,āy,āv] (Respectively) / _+vowel

Extension rules – *vānto yi pratyaye* (P 6.1.79)

[o,au] → [av,āv]   / _[y](of suffix)

Exception – *dhātostannimittasyaiva* (P 6.1.80);

kṣayyajayyau śakyārthe (P 6.1.81); *krayyastadarthe* (P 6.1.82)

if([o,au](of dhātu) is due to suffix)

{[o,au](of dhātu) → [av,āv] / _[y](of suffix)}

[kṣi,ji](of dhātu)→ [kṣayya,jayya] (meaning *liable to*) / _[ya](of suffix)

[krī](of dhātu) → [krayya] (meaning *purpose of*)/ _[ya](of suffix)

Exception rule – *lopaḥ śākalyasya* (P 8.3.19)

[y,v] → ø (optionally) / [a,ā] _ +voiced

Exception rule – rule A6; *sarvatra vibhāṣā goḥ, avaṅ sphoṭāyanasya* (P 6.1.122, 123)

[go] → unchanged (optionally) or rule A6/ _[/a/]

[go] → [gava] (optionally) or rule A6   / _+vowel

When a vowel from [e,o,ai,au] is followed by a vowel, it is respectively changed to [ay,av,āy,āv]. [o,au] are changed to [av,āv] also when it is followed by *y*-beginning suffix, but if the suffix is after *dhātu*, then this change takes place only if it comes due to that particular *dhātu*. [kṣi,ji,krī] → [kṣe,je,kre] → [kṣay,jay,kray] if followed by *y*-beginning suffix [ya] only if the meaning is '*liable to*', otherwise [e] will remain [e].

Rule A3 – vowels substituted by *guṇa* (*guṇa-sandhi*)

The *sūtra* is – *ād guṇaḥ* (P 6.1.87)

[a] +vowel → [/a/,/e/,/o/] (closest in place)

Supporting meta-rule – *uraṇ raparaḥ* (P 1.1.51)

if([ṛ,ḷ] → [a,i,u])

{ø → [r,l] / [a,i,u] _}

Extension rule – *vā supyāpiśaleḥ* (P 6.1.92)

[a](of prefix) → [ar] (optionally) / _[/ṛ/](of nominal verb)

Exception rules – rule A4; rule A5; rule A7;

When any type of vowel [a] is followed by another vowel, both are replaced with a *guṇa* [/a/,/e/,/o/] nearest in place of articulation. When [ṛ] or [ḷ] are replaced with [a,i,u], the replacement is followed by [r] or [l] respectively. Rule A4 (*vṛddhi*), rule A5 (*pararūpa*) and rule A7 (*dīrgha*) are its exceptions. When prefix ending in [a] is followed by a nominal derived verb (*nāmadhātu*) beginning with [/ṛ/], both vowels are optionally replaced with *guṇa* [ar]. It is an exception of *vṛddhi-sandhi* (rule A4).

Rule A4 – vowel substituted by *vṛddhi* (*vṛddhi-sandhi*)

The *sūtra* is *vṛddhireci* (P 6.1.88)

[a] [e,o,ai,au] → [/ā/,/ai/,/au/] (closest in place)

Supporting meta-rule – *uraṇ raparaḥ* (P 1.1.51)

if([ṛ,ḷ] → [a,i,u])

{ø → [r,l] / [a,i,u] _}

Extension rule – *etyedhatyūṭhsu* (P 6.1.89); (v) *akṣādūhinyāmupasaṅkhyānam;*

(v) *svādīrerinoḥ;* (v) *prādūhoḍhoḍhyeṣaiṣyeṣu*

[a] [ū,e,o,ai,au] (of √iṇ, √edha, √ūṭhi]) → [/ai/,/au/]

[akṣa] [ūhinī] → [akṣauhiṇī]

[sva] [īra,īrin] → [svaira,svairin]

[a](of [pra]) vowel(of [ūha,ūḍha,ūḍhi,eṣa,eṣya]) → [/ai/,/au/] (closest)

Extension rule – (v) *ṛte ca tṛtīyā-samāse*

[a] [/ṛ/]→ [ār] /third case compound

Extension rule – (v) *pra-vatsa-tara-kambala-vasanārṇa-daśānāmṛṇe*

[a](of[pra,vatsa,vatsatara,kambala,vasana,ṛṇa,daśa]) [ṛ](of[ṛṇa])→ [ār]

Extension rule – *upasargādṛti dhātau* (P 6.1.92)

[a](of prefix) [/ṛ/](of dhātu) → [ār]

Extension rule – *vā supyāpiśaleḥ* (P 6.1.92)

[a](of prefix) [/ṛ/](of nominal verb) → [ār] (optionally)

Exception rules – rule A5; rule A7;

This is the exception of rule A3. When any type of vowel [a] is followed by vowel [e,o,ai,au], both are replaced with *vṛddhi* [ā,ai,au] closest in place of articulation. When [ṛ] or [ḷ] is replaced with closest *vṛddhi* [/ā/], it is followed by [r] or [l] respectively. Forms of [iṇ,edha] dhātus beginning with [e,ai] and *dhātu* [ūṭhi] when follow vowel [a], both vowels are replaced with closest *vṛddhi.* When [sva] is followed by [īra] or [īrin], the vowels are replaced by closest *vṛddhi.* When [pra] suffix is followed by [ūha,ūḍha,ūḍhi,eṣa,eṣya] the concating vowels are replaced with *vṛddhi,* if *eṣa* and *eṣya* are forms of *dhātu* [iṣa]. If these are forms of *dhātu* [īṣa], it will be replaced with *guṇa* [/e/]. In compound of third case, if [a] is followed by short vowel [/ṛ/], both are replaced with closest *vṛddhi.* When vowel [a] of [pra,vatsa,vatsatara,kambala,vasana,ṛṇa,daśa] is followed by [/ṛ/] of [ṛṇa], both vowels are replaced with closest of *vṛddhi.* When vowel [a] of a prefix is followed by short vowel [/ṛ/] of a *dhātu,* both are replaced by closest *vṛddhi* followed by [r] – [ār]. And if *dhātu* is nominal derived verb (*sup-dhātu*) then replacement will be optionally *vṛddhi* [ār] and *guṇa* [ar].

Rule A5 – merger in later vowel (*pararūpa-sandhi*)

The *sūtra* is – *eṅi pararūpam* (6.1.94)

[a](of prefix) [e,o,ai,au](of dhātu) → [e,o,ai,au] (respectively)

Extension rule – (v) *śakandhvādiṣu pararūpaṃ vācyam*

Last vowel+coda (of [śaka,...]) _+vowel(x) (of [andhu,...]) → vowel(x)

Extension rule – *omāṅośca* (P 6.1.95)

[a] _+vowel (of [om,āṅ]) → vowel (of [om,āṅ])

Extension rule – *avyaktānukaraṇsyāta itau* (P 6.1.98)

Last[/a/]+coda(of onomatopoeic word) [i](of [iti]) → [i]

Exception rule – *nāmreḍitasyāntyasya tu vā* (P 6.1.99)

Doubled word [i](of [iti]) → [i] (optionally)

When [a] ending prefix is followed by a *dhātu* beginning in [e,o,ai,au], both are replaced with later sound, or both sounds merge in later sound. [śakandhu,...] is a fixed list of fourteen words in which last vowel together with its coda, called *ṭi* is merged into later vowel. [a] followed by [om] merges into its following vowel. [a] followed by prefix [āṅ] is merged into later vowel whether it is in its own form or modified into other sound. When onomatopoeic word is followed by [iti], its *ṭi* (last vowel with its coda) merges into following sound but not if the word is doubled. In doubling only last sound is merged into the following sound.

Rule A6 – merger in prior vowel (*pūrvarūpa-sandhi*)

The *sūtra* is – *eṅaḥ padāntādati* (P 6.1.109)

[e,o,ai,au]# (of inflected form) [/a/] → [e,o,ai,au]

Exception rule – *sarvatra vibhāṣā goḥ, avaṅ sphoṭāyanasya* (P 6.1.122, 123);

[o](of inflected form [go]) [/a/] → no change (optionally)

[o](of inflected form [go]) [/a/] → [o] (optionally)

[o](of inflected form [go]) → [ava] / _ [/a/] (optionally)

When vowels [e,o,ai,au] of inflected forms are followed short vowel [/a/], the later sound is merged into prior sound. Inflected form of [go] when followed by short vowel [/a/], optionally gets three forms – both unchanged, merger in prior vowel, and [o] changed to [ava], which together with [/a/] forms [avā].

Rule A7 – vowel lengthening (*dīrgha-sandhi*)

The *sūtra* is – *akaḥ savarṇe dīrghaḥ* (P 6.1.101)

savarṇa(V_1,V_2) → place(V_1)=place(V_2) && manner(V_1)=manner(V_2)

[a,i,u,ṛ,ḷ] _+vowel+savarṇa → +length+savarṇa

Exception rules – (v) *ṛti savarṇe ṛ vā;* (v) *ḷti savarṇe ḷ vā*

[/ṛ/,/ḷ/] [/ṛ/] → [/ṛ/]-length (optionally)

[/ṛ/,/ḷ/] [/ḷ/] → [/ḷ/]-length (optionally)

Extension rule – *prathamayoḥ pūrva-savarṇaḥ* (P 6.1.102)

([a,i,u,ṛ,ḷ] +vowel)first two cases → +length+savarṇa

Exception rule – *nādici* (P 6.1.104)

not(([a] _vowel-[a])first two cases → [ā])

When vowel [a,i,u,ṛ,ḷ] is followed by a vowel with similar place and manner of articulation (*savarṇa*), the both vowels are replaced by long *savarṇa* vowel. Vowels [ṛ,ḷ] are considered as *savarṇa* and [ḷ] has not long type, so these sounds together result in [ṝ].

Optionally, when [ṛ,ḷ] are followed by short [/ṛ/,/ḷ/] both are replaced with short [/ṛ/,/ḷ/] which is the following vowel. Another rule of lengthening belongs to morphology. In the formation of nominal forms of first and second case, if vowels [a,i,u,ṛ,ḷ] are followed by a vowel, both are replaced with long type of prior vowel. This is not the case when [a] is followed by other non-*savarṇa* vowel.

2.4.2 CONSONANT SANDHI RULES

Rule B1 – palatalization of dental (*ścutva-sandhi*)

The *sūtra* is – *stoḥ ścunāścuḥ* (P 8.4.40) (palatalization)

+cons+sib+stop+dental → +palatal /_ +cons+sib+stop+palatal

+cons+sib+stop+dental → +palatal /+cons+sib+stop+palatal _

Exception rule – *śāt* (P 8.4.44)

+cons+stop+dental → not(+palatal) / [ś] _

When a dental consonant except semivowel comes in contact with palatal consonant except semivowel, the dental changes to palatal closest in manner. But when [n] follows [ś], it does not change to palatal.

Rule B2 – retroflexisation of dental (*ṣṭutva-sandhi*)

The *sūtra* is – *ṣṭunāṣṭuḥ* (P 8.4.41) (retroflexization)

+cons+sib+stop+dental → +retroflex /_ +cons+sib+stop+retroflex

Exception rule – *na padāntāṭṭoranām* (P 8.4.42); *toḥ ṣi* (P 8.4.43)

+cons+stop+dental → not(+retroflex) / cons+stop+retroflex#_

[nām] → [ṇām] / cons+stop+retroflex# _

+cons+stop+dental → not(+retroflex) /_ [ṣ]

When a dental stop or sibilant consonant comes in contact with retroflex stop or sibilant, the dental changes to the palatal closest in manner. But when dental follows retroflex in the end of word, dental does not change into retroflex, but it changes when the following morpheme is [nām]. Also dental stop does not change to retroflex when followed by retroflex sibilant [ṣ].

Rule B3 – voiced assimilation and voicinng (*jaśtva-sandhi*)

The *sūtras* are – *jhalāṃ jaś jhaśi* (P 8.4.53); *jhalāṃ jaśo'nte* (P 8.2.39)

+cons+sib+stop-nasal → +stop+voiced-aspirate /_ +stop+voiced-nasal

+cons+sib+stop-nasal → +stop+voiced-aspirate /_#

Exception rule – *vāvasāne* (P 8.4.56)

+cons+sib+stop-nasal → +stop+voiced-aspirate /_# (optionally)

+cons+sib+stop-nasal → +stop-voiced-aspirate /_# (optionally)

When a consonant except nasal and semivowel is followed by voiced stop except nasal, or is in the end of word, it changes to voiced unaspirated non-nasal stop closest in the place of articulation. In the end of sentence, this changes optionally to voiced or unvoiced unaspirated stop.

Rule B4 – unvoiced assimilation (*cartva-sandhi*)

The *sūtra* is – *khari ca* (P 8.4.55)

+cons-sibilant-nasal → +stop-voiced-aspirate /_ +stop-voiced-nasal

Exception rule – *vāvasāne* (P 8.4.56) (repeated)

+cons+sib+stop-nasal → +stop+voiced-aspirate /_# (optionally)

+cons+sib+stop-nasal → +stop-voiced-aspirate /_# (optionally)

When a consonant except nasal and semivowel is followed by voiced stop except nasal, it changes to voiced unaspirated stop closest in the place of articulation. In the end of sentence, this changes optionally to voiced or unvoiced unaspirated stop.

Rule B5 – dental semivowel assimilation (t→l)

The *sūtra* is – *torli* (P 8.4.60)

+cons+stop+dental → [l] /_ [l]

When dental stop is followed by [l], it changes to [l]. When it is nasal [n], it changes to nasal [l].

Rule B6 – nasal assimilation (*anunāsika-sandhi*)

The *sūtra* is – *mo'nusvāraḥ* (P 8.3.23)

[m] → [ṃ] /_#

[m] → [ṃ] /_ +cons

Extension rule – *naścāpadāntasya jhali* (P 8.3.24)

[n] → [ṃ] /_ +cons+stop-nasal+sib, not(end of word)

Extension rule – *anusvārasya yayi parasavarṇaḥ* (P 8.4.58)

[ṃ] → +ṇasal+savarna(of following) /_ +cons+stop+semivowel

Exception rule – *vā padāntasya* (P 8.4.59)

[ṃ] → [ṃ] /_# +cons+stop+semivowel

Exception rule – *mo rāji samaḥ kvau* (P 8.2.35)

[m] → [m] / _ r(of √rāja+kvip=rāṭ)

Exception rule – *he mapare vā* (P 8.3.26)

[m] → [m] / _ [h] [m] (optionally)

Exception rule – (v) *yavalapare yavalā veti vaktavyam*

[m] → [y,v,l]+nasal / _ [h] [y,v,l] (optionally)

Exception rule – *napare naḥ* (P 8.3.27)

[m] → [n] / _ [h] [n] (optionally)

When [m] is followed by a consonant, or [n] follows a non nasal consonant except semivowels within a word, it changes to *anusvāra* [ṃ]. The *anusvāra* changes to nasal stop closer to the following consonant, but in the end of word, it may optionally remain *anusvāra* if it is followed by a sibilant or [h]. *Anusvāra* or [m] is always *anusvāra* when followed by sibilant. When [m] is followed by [h], one, it changes to *anusvāra* and optionally it changes to nasal consonant similar (*savarṇa*) to the consonant following [h]. i.e., if [h] is followed by [m,n,y,v,l], [m] changes to [m,n] or nasal [y,v,l].

Rule B7 – consonant insertion or augmentation (*āgama*)

The *sūtra* for 'k' and 'ṭ' insertion – *ṅṇoḥ kuk ṭuk śari* (P 8.3.28)

[ṅ] → + _[k] / _ +cons+sibilant (optionally)

[ṇ] → + _[ṭ] / _ +cons+sibilant (optionally)

Extension rule – (v) *cayo dvitīyāḥ śari pauṣkarasāderiti vācyam*

+cons+stop-voiced-aspirate → +aspirate / _ +cons+sib (optionally)

The *sūtra* for 'dh' insertion – *ḍaḥ si dhuṭ* (P 8.3.28)

[s] → + [dh]_ / [ḍ] _ (optionally)

Extension rule – *naśca* (P 8.3.30)

[s] → + [dh]_ / [n] _ (optionally)

The *sūtra* for 't' insertion – *śi tuk* (P 8.3.31)

[n] → + _[t] / _[ś] (optionally)

Extension rule – *che ca* (P 6.1.73)

+vowel-length → + _[t] / _[ch]

Extension rule – *āṅmāṅośca* (P 6.1.74)

[ā](of [āṅ,māṅ]) → + _[t] / _[ch]

Extension rule – *dīrghāt* (P 6.1.75)

+vowel+length → + _[t] / _[ch]

Exception rule – *padāntādvā* (P 6.1.76)

+vowel-length → + _[t] / _# [ch] (optionally)

The *sūtra* for 'ṅ', 'ṇ' and 'n' insertion – *ṅamo hrasvādaci ṅamuṇnityam* (P 8.3.32)[2]

[ṅ,ṇ,n] → + [ṅ,ṇ,n]_ / +vowel-length _ +vowel

[ṇ] → + [ṇ]_ / +vowel-length _ +vowel

[n] → + [n]_ / +vowel-length _ +vowel

The *sūtra* for *anusvāra* insertion – *anunāsikātparo'nusvāraḥ* (P 8.3.4)

+nasal → + _[ṃ] / _[ru]

[2] This rule can be regarded as the rule of doubling

When [ṅ,ṇ] is followed by sibilant, similar (*savarṇa*) unvoiced unaspirated stop is inserted attached after it. According to *pauṣkarasādi,* unvoiced unaspirated stop can optionally become aspirated. When [s] is followed by [ḍ,n], [dh] comes attached before it. When [n] is followed by [ś], [t] comes attached after it. After vowel followed by [ch], [t] is always attached to it, but after long vowel in the end of word, it is optionally inserted. After [ā] of [āṅ,māṅ] followed by [ch], it is always inserted. Nasal stops of dental, retroflex and velar group when followed by a vowel and follow a short vowel, insert a similar nasal before them. This rule can be said of doubling. When a nasal is followed by [ru], an *anusvāra* is inserted between them.

Rule B8 – consonant doubling (*dvitva*)

The *sūtra* is – *aco rahābhyāṃ dve* (P 8.4.46)

+cons-[h] → double / +vowel +[r,h] _

Extension rule – *anaci ca* (P 8.4.47)

+cons-[h] → double / +vowel _ -vowel

Extension rule – (v) *śaraḥ khayaḥ*

+cons+stop-voiced → double / +cons+sibilant _

Exception rule – *nādinyākrośe putrasya* (P 8.4.48)

[t] → not double / in [putra ādinī] (meaning *anger*)

[t] → double / in [putra ādinī] (meaning other than *anger*)

Exception rule – *triprabhṛtiṣu śākaṭāyanasya* (P 8.4.50)

+cons-[h] → double / _ +cons +cons (optionally)

+cons-[h] → double / _ +cons +cons +cons (optionally)

Exception rule – *dīrghādācāryāṇām* (P8.4.52)

+cons-[h] → not double / +vowel+length _ +cons

When a consonant except [h] comes after [r,h] followed by a vowel, it doubles. That consonant if comes directly after vowel and is followed by consonant, it doubles. In the similar environment, if there is cluster (*saṃyoga*) of three or more consonants, the consonant is optionally doubled. Voiceless stops also double after sibilants. In compound with [ādinī], the [t] of [putra] is not doubled if it gives sense of anger. In other senses, it also doubles. Same is if other word followed by [putra]. After a long vowel, a consonant is never doubled.

Rule B9 – nasal to *ru* → *visarga* (then to '*s*' and other forms)

The main *sūtra* for nasal to '*ru*' change is – *samaḥ suṭi* (P 8.3.5)

[m](of [sam]) → [ru] / _ [s](inserted suṭ before kṛ)[3]

Extension rule – *pumaḥ khayyampare* (P 8.3.6)

[m](of [pum]) → [ru] / _ +cons-voiced +voiced-(+stops-nasal)

Extension rule – (v) *saṃpuṃkānāṃ so vaktavyaḥ samo vā lopameke*

[m](of [sam,pum]) → s / _+cons-voiced

[m](of [sam,pum]) → ø / _+cons-voiced

Extension rule – *naśchavyapraśān* (P 8.3.7)

[n] → [ru] / _ +cons+stop-velar-labial-voiced +voiced-(+stops-nasal)

Extension rule – *nṝn pe* (P 8.3.10)

[n](of [nṝn]) → [ru] / _[p]

[3] *sam-paribhyām karotau bhūṣaṇe/sam-paryupebhyaḥ karotau bhūṣaṇe* (Pande:2004) (P 6.1.137)

Extension rule – *kānāmreḍite* (P 8.3.7)

[n](of [kān]) → [ru] / _[kān] (doubled word)

When [sam] is followed by inserted [s] sound, as is added to [√kṛ] after [sam,pari], the last nasal sound becomes [ru]. Last nasal of [pum] is also changed to [ru] if it is followed by unvoiced consonant followed by vowel, semivowel, nasal stop or [h]. [m] of [sam,pum] is optionally changed to [s] or elided also. [n] other than that of [praśān] becomes [ru] followed by dental, retroflex, palatal voiceless stop. Last sound of [nṝn] becomes [ru] when followed by [p]. When [kān] is doubled, its last sound becomes [ru].

Rule B10 – consonant elision (*lopa*)

The *sūtra* of ending consonant elision is – *saṃyogāntasya lopaḥ* (P 8.2.23)

+cons → ∅ / +cons _#

Exception rule – (v) *yaṇaḥ pratiṣedho vācyaḥ*

+cons-semivowel → ∅ / +cons _#

Extension rule – *lopaḥ śākalyasya* (P 8.3.19)

[v,y] → ∅ / [a] _# (optionally)

Extension rule – *oto gārgyasya* (P 8.3.20)

[v,y]-lax → ∅ / [/o/] _#

[v,y]+lax → no change / [/o/] _#

Extension rule – *uñi ca pade* (P 8.3.21)

[v,y] → ∅ / _# [u] (inflected [uñ])

Extension rule – *hali sarveśām* (P 8.3.22)

[y] → ∅ / o(of [bho,bhago,agho]) _ +cons

[y] → ∅ / [/a/] _ +cons

The *sūtra* of within-word elision is – *halo yamāṃ yami lopaḥ* (P 8.4.64)

+cons+sibilant+stop+nasal → ∅ / +cons _ +cons+sibilant+stop+nasal

Extension rule – *jharo jhari savarṇe* (P 8.4.65)

+cons+stop-nasal+sibilant → ∅ / +cons _+cons+savarṇa

When there is cluster of consonants in the end, the last sound is elided but if that sound is semivowel, it is not elided. After [a], [y,v] of word end is optionally elided if followed by voiced sound. After [bho,bhago,agho] or short vowel [/a/], the same are elided when followed by a consonant, or by inflected form [u]. After [/o/], they are elided if they are lax, otherwise they are not changed.

2.4.3 PRAKṚTIBHĀVA RULES

Rule C1 – *pluta* and *pragṛhya* followed by vowel

The *sūtra* is – *pluta-pragṛyhā aci nityam* (P 6.1.125)

+vowel+protracted → no change / _ +vowel

+vowel+pragṛhya → no change / _ +vowel

The whole *prakṛtibhāva* applies in the environment similar to required for vowel sandhi. When a *pluta* or protracted vowel, or *pragṛhya* word is followed by a vowel, both remain unchanged. *Pluta* is defined in 7 *sūtras* including *ūkālo'jjhrasvadīrghaplutaḥ* (P 1.2.27) and *pragṛhya* is defined in 7 *sūtras*.

Rule C2 – sandhi restriction and vowel shortening

The *sūtra* is – *iko'savarṇe śākalyasya hrasvaśca* (P 6.1.127)

[i,u,ṛ,ḷ] → -length / _# +vowel-savarṇa (optionally)

[i,u,ṛ,ḷ] → no change / _# +vowel-savarṇa (optionally)

Extension rule – *ṛtyakaḥ* (P 6.1.128)

[a,i,u,ṛ,ḷ] → -length / _# [/ṛ/] (optionally)

[a,i,u,ṛ,ḷ] → no change / _# [/ṛ/] (optionally)

When a vowel from [i,u,ṛ,ḷ,] at the end of word is followed by non-similar vowel, it does not conjunct and it optionally becomes short. When these vowels and [a] at the end of word are followed by short vowel [/ṛ/], then also they do not combine and first vowel becomes short. This is also applicable in compound.

Rule C3 – sandhi restriction and nasalization

The *sūtra* is – *aṇo'pragṛhyasyānunāsikaḥ* (P 8.4.57)

Primary vowel-pragṛhya → +nasal / _# (optionally)

Primary vowel-pragṛhya → -nasal / _# (optionally)

Primary vowels at the end of words other than *pragṛhya* do not combine to the later and optionally they become nasal.

2.4.4 VISARGA SANDHI RULES

Rule D1 – *visarga* to dental sibilant

The *sūtra* is – *visarjanīyasya saḥ* (P 8.3.34)

[ḥ] → [s] / _ +cons-voice

Exception rule – rule D2; rule D3; rule D5;

Extension rule – *so'padādau* (P 8.3.38)

[ḥ] → [s] / _ +cons+stop+velar+labial (not beginning of word)

Extension rule – *namaspurasorgatyoḥ* (P 8.3.40)

[ḥ](of [namaḥ,puraḥ]) → [s] / _ verb (compound)

Exception rule – *tiraso'nyatarasyām* (P 8.3.42)

[ḥ](of [tiraḥ]) → [s] / _ +cons+stop+velar+labial (optionally)

Extension rule – *ataḥ kṛkamikaṃsakumbhapātrakuśākarṇīṣvanavyayasya*(P 8.3.46)

[ḥ](not of indeclinable) → [s] / [/a/] _ [kṛ,kami ...] (in compound)

When *visarga* is followed by an unvoiced vowel, it changes to dental sibilant which later gets other formations. But when it is followed by velar, it changes to *jihvāmūlīya* and when followed by labial stop, it changes to *upadhmānīya.* When it is followed by sibilant, it optionally changes to dental sibilant and optionally it may remain *visarga.* When *visarga* is followed by a sequence of sibilant and unvoiced consonant, it is optionally elided. When *visarga* is followed by velar or labial unvoiced stops, not in beginning of word, it changes to [s]. In the same condition, *visarga* of [tiras] is optionally changed to [s] or remains *visarga. Visarga* of [namaḥ,puraḥ] change to [s] if they are *gati* i.e., indeclinable followed by verb beginning with velar or labial stop. *Visarga* of a word other than indeclinable after short vowel [/a/] is changed to [s] when followed by any word or form of [kṛ, kami, kaṃsa, kumbha, pātra, kuśā, karṇī].

Rule D2 – *visarga* to retroflex sibilant

The *sūtra* is – *iṇaḥ ṣaḥ* (P 8.3.39)

[ḥ] → [ṣ] / [i,u,ṛ,ḷ] _ +cons+stop+velar+labial (not beginning of word)

Exception rule – *idudupadhasya cāpratyayasya* (P 8.3.41)

[ḥ](not of suffix) → [ṣ] / [/i/,/u/] _ +cons+stop+velar+labial

Exception rule – *dvistriścaturiti kṛtvo'rthe* (P 8.3.43)

[ḥ](of [dvis,tris,catur]) → [ṣ] / _ +cons+stop+velar+labial

(meaning *times*) (optionally)

Exception rule – *isusoḥ sāmarthye* (P 8.3.44)

[ḥ](of [is,us]) → [ṣ] / _ +cons+stop+velar+labial (meaning *ability*)
(optionally)

Extension rule – *nityaṃ samāse'nuttarapadasya* (P 8.3.41)

[ḥ](of [is,us]) → [ṣ] / _ +cons+stop+velar+labial (in compound)
(_is, _us is not following word/is first word)

Visarga after [i,u,ṛ,ḷ] followed by a velar or labial stop only, not beginning the word changes to retroflex sibilant. If the *visarga* follows short [/i/,/u/] of a word without suffix, then it changes to [ṣ] in the same condition in beginning of word also. *Visarga* of [dvis, tris, catur] changes to [ṣ] only in meaning of repetition (how many times) optionally, otherwise remains *visarga*. *Visarga* of word ending in [is, us] changes optionally to the same in meaning of ability only. But if the word is in compound and not following another word, it always changes to [ṣ].

Rule D3 – *visarga* to *jihvāmūlīya* and *upadhmānīya* (':')

The *sūtra* is – *kupvoḥ :ka:pau ca* (P 8.3.37)

[ḥ] → [:(k)] / _ +cons+stop+velar

[ḥ] → [:(p)] / _ +cons+stop+labial

Visarga is when followed by velar stop (voiceless), it changes to *jihvāmūlīya*[4] and when followed by labial stop, it changes to *upadhmānīya*[5].

Rule D4 – *visarga* to '*r*'

The *sūtra* is – *ro'supi* (P 8.2.69)

[n](of [ahan]) → r / _ not(nominal inflection suffix)

[4] Sound before *k*, *kh* similar to half visarga

[5] Sound before *p*, *ph* similar to half visarga

Exception rule – (v) *rūparātrirathantareṣu rutvaṃ vācyam*

[n](of [ahan]) → [ru] / _ [rūpa,rātri,rathantara]

Extension rule – (v) *aharādīnāṃ patyādiṣu vā rephaḥ*

[n](of [ahan,...]) → [r] / _ [pati, ...] (optionally)

Ending of [ahan] changes to [r] when followed by a word different from nominal inflection suffix. Before nominal suffix, it changes to [ru]. When [ahan] is followed by [rūpa, rātri, rathantara] its last sound [n] changes to [ru]. When word from group of [ahan] is followed by group of [pati] etc., it is optionally changed to [r] or [ru].

Rule D5 – *visarga* unchanged

The *sūtra* is – *śarpare visarjanīyaḥ* (P 8.3.35)

[ḥ] → [ḥ] / _ +cons-voice +cons+sibilant

Exception rule – *vā śari* (P 8.3.36)

[ḥ] → [ḥ] / _ +cons+sibilant (optionally)

When *visarga* is followed by a sequence of unvoiced consonant and sibilant, it remains *visarga*, but if it is followed by sibilant, it optionally changes to dental sibilant and optionally it may remain *visarga.*

Rule D6 – *visarga* elision (*lopa*)

The *vārtika* is – (v) *kharpare śari vā visargalopo vaktavyaḥ*

[ḥ] → ø / _ +cons+sibilant +cons-voice (optionally)

When *visarga* is followed by a sequence of sibilant and unvoiced consonant, it is optionally elided.

2.4.5 SVĀDI SANDHI RULES

Rule E1 – *s* of nominal suffix to *ru*

The *sūtra* is – *sasajuṣo ruḥ* (P 8.2.66)

[s](of suffix [su]) → [ru] / _#

[ṣ](of [sajuṣ]) → [ru] / _#

Exception rule – elision of '*s*' – *so'ci lope cetpādapūraṇam* (P 6.1.134)

[s](of [tad+su]→[sas]) → ø / _# +vowel(elision required for prosody)

Exception rule – rule D4;

Sibilant endings of nominal inflectional suffix and word [sajuṣ] change to [ru] when in the end of word. If [s] of [tad s, etad s] form is required to delete to fit prosody, then it is elided. Ending of word [ahan] also changes to [ru] when followed by nominal suffix. Other places it changes to [r]. When [ahan] is followed by [rūpa, rātri, rathantara] its last sound [n] changes to [ru]. When word from group of [ahan] is followed by group of [pati] etc., it is optionally changed to [r] or [ru].

Rule E2 – *ru* to *y* or elision → *u*

The *sūtra* is – (*ru* → *y*) – *bhobhagoaghoapūrvasya yo'śi* (P 8.3.17)

[ru] → [y] / [o](of [bho,bhago,agho]) _ +voiced

[ru] → [y] / [/a/] _ +voiced

Extension rule – *vyorlaghuprayatnataraḥ śākaṭāyanasya* (P 8.3.18)

[v,y] → [v,y]+lax / _# (optionally)

[v,y] → [v,y]-lax / _# (optionally)

Extension rule – *lopaḥ śākalyasya* (P 8.3.19) (repeated from rule B10) [v,y] → ø / [a] _# (optionally)

Extension rule – *oto gārgyasya* (P 8.3.20) (repeated from rule B10)

[v,y]-lax → ø / [/o/] _#

[v,y]+lax → no change / [/o/] _#

Extension rule – *uñi ca pade* (P 8.3.21) (repeated from rule B10)

[v,y] → ø / _# [u] (inflected [uñ])

Extension rule – *hali sarveśām* (P 8.3.22) (repeated from rule B10)

[y] → ø / [o](of [bho,bhago,agho]) _ +cons

[y] → ø / [/a/] _ +cons

When [ru] follows [bho, bhago, agho] words or [/a/] of a word, and is followed by voiced sound, it changes to [y]. When it is followed by a consonant, it is elided. After [a], [y,v] of word end is optionally deleted if followed by voiced sound. In the end of inflection, [v,y] optionally become lax, and after [/o/], lax [y,v] is elided. [y,v] in the end of word is also elided when followed by inflected word [u].

Rule E3 – *a+ru* to *a+u* → o

The *sūtra* is – *ato roraplutādaplute* (P 6.1.113)

[r](of [ru]) → [u] / [/a/] _ [/a/]

Extension rule – *haśi ca* (P 6.1.114)

[r](of [ru]) → [u] / [/a/] _ +cons+voiced

After short vowel [/a/], [r] (of [ru]) is changed to [u] when followed by short [/a/] or a voiced consonant.

CHAPTER THREE

COMPUTATIONAL PROCESSING OF SANDHI: ISSUES AND CHALLENGES

Chapter three

COMPUTATIONAL PROCESSING OF SANDHI: ISSUES AND CHALLENGES

3.1 INTRODUCTION

This chapter discusses the complexities inherent in the sandhi analysis of Sanskrit language. Some of them may be applied to language in general but most are specific to Sanskrit because of the unique properties of this language. The chapter also describes the basic algorithm for reverse sandhi computation to develop a minimum working system for it. Thereafter certain issues are presented which arise while processing sandhi through computer. An advanced algorithm is also presented to meet these challenges. Finally, the chapter presents some still un-resolved challenges.

3.2 COMPLEXITY OF SANSKRIT SANDHI

As as a language becomes more and more systematic, it tends to become more complex. The sandhi formulations of Sanskrit as described by Panini are very complex from the point of view of computer processing. Most of the problems of sandhi processing stem from the very nature of Sanskrit language and the remaining are related to the limits of computer processing. The significant problems in the sandhi processing by computers are the following.

3.2.1 INFINITELY LONG STRING

Sanskrit morphology allows very long words, which could, from the point of computer algorithm, be infinitely long. . Sandhi will take place in most cases where more than one word is used in continuum. There are however cases where just string concatenation occurs without sound modification (sandhi). In many places, sandhi is compulsory but it is optional in sentence.. Some prose texts of Sanskrit as *Kādambarī* and *Daśakumāracaritam* are good examples of this feature of Sanskrit. A single sentence in *Kādambarī* can run into several pages. Some examples from Kādambarī are as follows-

जलावगाहनागतजयकुञ्जरकुम्भसिन्दूरसंध्यायमानसलिलयोन्मदकलहंसकुलकोलाहलमुखरीकृतकूल
या (78 char) वेत्रवत्या परिगता विदिशाभिधाना नगरी राजधान्यासीत् । (pg 11)

काव्यनाटकाख्यानकाख्यायिकालेख्यव्याख्यानादिक्रियानिपुणैरतिकठिनपीवरस्कन्धोरुबाहुभिरसकृदव
दलितसमदरिपुगजघटापीठबन्धैः (111 char) ... सुखमतिचिरमुवास । (pg 12)

निर्दयश्रमच्छिन्नहारविगलितमुक्ताफलप्रकरानुकारिणीभिर्ललाटपट्टकेऽष्टमीचन्द्रशकलतलोल्लसदमृतबि
न्दुविडम्बिनीभिः (106 char) ... । (pg 30)

शुककुलदलितदाडिमीफलद्रवार्द्रीकृततलैरतिचपलकपिकम्पितकक्कोलच्युतपल्लवफलशबलैरनवरतनि
पतितकुसुमरेणुपांसुलैः (100 char)
पथिकजनरचितलवङ्गपल्लवसंस्तरैरतिकठोरनालिकेरकेतकीकरीरबकुलपरिगतप्रान्तैस्ताम्बूलीलताव
नद्धपूगखण्डमण्डितैर्वनलक्ष्मीवासभुवनैरिव (121 char) विराजिता लतामण्डपैः ... । (pg 38)

3.2.2 VERY SMALL SIZE OF WORDS

The Sanskrit language has a large number of small words. which makes the identification of such words in continuous string very difficult. Almost all verbs are of single syllable and so are many particles, prefixes etc of one letter only .· And these are often widely used cases making it difficult to decide if they are independent words or part of other words. That a word has to be of certain minimum length to qualify for Sandhi processing can be made in to a useful criterion. But where the smallest wprd is one character long, the criterion will become void every character in the the string will be tried for a word. Verb roots like *गम्, पठ्, स्था* etc. and pronouns like *स:, सा, तत्, न:, मे* etc. are of one syllable but many are of one character as the *dhātu*s *इ, ऋ* and indeclinables *च, न, आ, उ* etc.. Some longer *dhātu*s get reduced to one character as *ग, ज, द* etc. There are more interesting cases in which more than one morphemes appears as one character after sandhi for example, prefix *आ* + *dhātu* *इ(ण्)*. Their separate recognition is required in the places like *śiva+ehi* = *śivehi*. Here '*e*' sound is result of three sounds combined into one.

3.2.3 MULTIPLE COMBINATIONS LEADING TO ONE RESULT

There are many combinations and contexts of sounds where the resultant sounds appear similar. One sound sequence is result of many sound orders. In the way of arriving basic sounds from resultant sounds, many sound sequences correlate it. For example, a voiced unaspirated stop results if a non-nasal stop consonant or a homorganic sibilant is followed by a voiced non-nasal stop according to *jhalāṃ jaś jhaśi* (P 8.4.53). According to *stoḥ ścunā ścuḥ* (P 8.4.40), the palatal sound results if a dental is followed by a palatal. Now both these rules cam apply in the generation of a sound. For example, /j/ can be a result of 8.4.53 as well as 8.4.40. Third consonant of palatal stop class also can be result of four consonants of dental class and dental sibilant. This can lead to several many-to-one cases and therefore make reverse processing more complex.

3.2.4 SIMPLE CONCATENATION OF SOUNDS

Generally sandhi is compulsory between morphemes within the word and optional between words in the sentence. But there are the cases where there is simple concatenation. This case is different from *prakṛtibhāva*. In *prakṛtibhāva*, there is a possibility of sandhi but it is restricted by rules while in this cases of simple concatenation, the words are joined together without sandhi. Mostly this is the case with consonant ending words followed by vowel in the next word – *ajjhīnaṃ pareṇa saṃyojyam.* This condition requires checking the possibility of occurrence of words in such environments before trying any sandhi rule. And this has to be done after each character due to the fact that minimum possible length of the word could be just a single letter. This is the problem of deciding whether a rule is to be applied here or not. For example, *तत्वं* has two words (*तत्वं पूषन् अपावृणु* (*īṣāvāsyopaniṣad*)) and there is no sandhi in *सुखमतिचिरमुवास* but simple concatenations of constituents *सुखम् अति-चिरम् उवास.*

3.2.5 DEVANAGARI UNICODE IS NOT PHONEMIC

Unicode writing system is syllabic and not phonemic. There are some sounds which appear differently, like vowel as in their nominal form and *mātrā.* Vowel /a/ not in the beginning of the word does not appear as separate character. Pure consonants and consonants with /a/ sound have reverse situation in mapping the writing and the sound systems. Pure consonants parallel to one phoneme appears as two characters, a nominal character with *halanta*; whereas consonant with /a/ sound, two phonemes appear as single character, a nominal consonant character. When a consonant is concatenated with consonant, it is convenient that though not in display, previous consonant is separately recognizable by computer in the form of two characters. But when consonant is concatenated by vowel sound, pure consonant with *halanta* does not exist. In case of vowels except /a/, the presence of vowels is indicated by its *mātrā*, but in the case of vowel /a/, no such marker exists. For example, in the examples above, unlike त् त् in 'तत्वं', म् अ and म् उ do not exist explicitly in म and मु respectively.

3.2.6 INFLECTIONS WITHIN THE SANDHI STRING

Identification of word is made by vocabulary, both by human beings and machines. For a machine, it is the lexicon given to it or automatically generated. Unlike humans, a machine cannot have all words with inflected forms in its lexicon. To recognize inflected form, recognition of inflection is required. Here again there are two possibilities - whether there will be inflections or not. For example, तस्मिन्नपि and काशीवासी. If the sandhi is within the word, like in a compound, there may be inflection (as अन्तेवासी) or may not be (as काशीवासी). But if the sandhi is between the words, it is more likely that there is a suffix. The case of prefix is similar. This requires any sandhi processing to identify the inflections prefixes/suffixes etc before validating the word in case the word is not found in the dictionary.

3.2.7 VARYING SIZE OF AFFIXATION

When suffix identification is required, then the varying size of suffix causes difficulty. Size of suffix can vary from zero to length one less than the word, because there is a possibility of a chain of derivational and inflectional suffixes. Sometimes the length of suffix can be said to be negative because they do not add anything after the base but reduce some sound or sounds like in *n*-ending bases in neuter gender- ब्रह्मन्, प्रेमन्. There is no such problem where the suffix length is zero. Sometimes it is confusing how much of the end of the word is suffix. For example in a *bhyām*-ending words, the possible suffix may appear to be any of *bhyām* (like in *haribhyām*), *yām* (like in *ramāyām*), *ām* (like in *satām*) and *m* (like in *vidyām*). All possible lengths are to be checked for identification of suffix to validate the segment. Identification of negative length suffixes mentioned above is rather more difficult. Problems in suffix identification are discussed in work of Subash (2006).

3.2.8 VALIDATION OF UNDESIRED SEGMENTS

When a rule is applied at a point, it is divided into two segments and both are to be validated through dictionary and affix check. There are further possibilities of more than one point of sandhi like सच्चिदानन्दरूपम्. In that case, the later segment is again processed for sandhi and this process goes longer as the string goes longer. Thus many words will be validated if they exist in the lexicon and all of them must not be desirable. The word may again be required to go in to the loop for finding the word without sandhi, applying rule and validating, and if not validated then identifying affixation and validating. This is to be continued till the end of the string and till all the segments are validated.

3.2.9 MANY LOOPS CAUSING OVER-GENERATION

The identification process of words in string needs to run in at least three ways - finding words without sandhi analysis, finding words by applying rule and validating segments, and identifying affixation to validate word. Applying a number of rules, several possible sandhi points, many rounds of processing, validation of undesired words - all these lead to a large set of validated segmentations. The number of sets is directly proportional to the length of the input string. The systems awaiting the output of sandhi analyzer, like morph-analyzers, need one output to further analyze the text. Among the numbers of sets, one set of segments is to be selected on the basis of relevance and context. Larger the number of sets, bigger the challenge to choose one set. See a simple example of over validation. A simple string सन्नाम can be split and validated as सद् नाम; सन् नाम; सद् न आम; सन् न आम; सन् आम. All are correct according to dictionary, but not desired.

3.2.10 CONVENTION OF CONTINUOUS (SAṀHITA) WRITING

The Sanskrit writing system is more phonetic and in this, the script follows the speech. Humans never speak word by word. Instead they speak in continuum. The Sanskrit writers followed this convention of writing as the language is spoken. That is why the rules of sandhi in Sanskrit are explicitly described and are applied in writing system also. On the one hand, it makes thelanguage more compact and precise, on the other, it poses difficulties in formally processing the written text. But from the view of speech processing, the correspondence of speech and writing system has obvious advantages.

3.3 BASIC ALGORITHM OF SANDHI PROCESSING

Sachin (2007) presents a basic algorithm of sandhi analysis and its system constituents. The modules of this system includes rule base, verb database, dictionary or lexicon, proper names database, *avyaya* database and sandhi example base. It also includes supporting systems like *subanta* analyzer, verb analyzer. Other constituents are technical – basic software and programming language and objects. He has implemented only vowel sandhi analysis system. Process flow of his system (as described in the dissertation) 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)

Fig. 3.1 – vowel sandhi analysis system design by Sachin

Sachin (2007) in his dissertation under supervision of Dr. Girish Nath Jha describes the implemented analysis procedure as following.

> “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 string in sandhi example base and subanta-analysis takes place respectively. Pre-processing will mark punctuations 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 from processing. The sandhi recognition and analysis will be according to the process outlined in the chapter III.”

His rule base is in the form of

Marker=pattern:(sandhi name, sandhi *sūtra*)

Some example from his rule base

s= +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=ु+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);ार्= +ऋ:(वृद्धिसन्धि उपसर्गादृति धातौ/वा सुप्यापिशलेः); ार्=ा+ऋ:(वृद्धिसन्धि उपसर्गादृति धातौ/वा सुप्यापिशलेः);र्= +ऋ:(गुणसन्धि आद् गुणः);ा=ा+आ:(दीर्घसन्धि अकः सवर्णे दीर्घः); ा=ा+अ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ा= +आ:(दीर्घसन्धि अकः सवर्णे दीर्घः);ा= +अ:(दीर्घसन्धि अकः सवर्णे दीर्घः);

In the above process, first input is made ready for sandhi processing by pre-processing. This step checks if the text is in proper encoding and Devanagari script. It recognizes punctuations and removes extra, possibly undesired characters from within the strings and makes proper string. Then it takes words one by one and first checks them in the example base where the complex and less common sandhi words are stored with their correct manual segmentation. Before segmentation, it is fruitful to decide what is not to be segmented. The sandhi analyzer system takes help of *subanta* analyzer which, before suffix identification, prepares the string as told above. Then it identifies verbs from database of simple verb forms and indeclinables from *avyaya* database. These words are identified as single words, therefore not required for segmentation. Then it analyzes the remaining words assuming them to be nominal forms. By analyzing the nominal suffix, the validation of the last segment of string becomes easy.

Now the system applies rules, finds sandhi marker, replaces with corresponding pattern, splits the string into two segments, and sends each segment for validation. For validation, a segment is searched into the MWSDD (Monier William Sanskrit Digital Dictionary). If it is not found there, it is searched into proper names database, which he calls place name list and noun list. If there also it is not found, it is searched in Sanskrit corpus for validation. The purpose of validation is to check if the segment is a meaningful word. This is not the only way of word validation, but other reliable ways like finite state automata are too complex to implement.

3.4 CHALLENGES AND EFFORTS TO MEET THEM

In the above quoted primary algorithm for sandhi analysis, some problems and errors are recorded.

3.4.1 SERIES OF MORE THAN ONE RULE APPLIED AT ONE POINT

Rules of sandhi often do not give the final form on applying one rule. Many sandhi rules apply serially one by one to give resultant sound. For example, सत् जन becomes सज्जन applying two rules *stoḥ ścunāścuḥ* (P 8.4.40) and *jhalam jash jhashi* (P 8.4.53). For their reverse application also, the rules in reverse will apply in reverse order. For rule writing, all the rules which apply one by one at the same point should be collectively regarded as a single rule.

3.4.2 OVER-GENERATION NECESSARY NOT TO MISS DESIRED RESULT

In generative sandhi, when rules are given with context, they result in one result; also if they result in many results, all of them are possibly correct and desired provided rule is correct. In contrast, in reverse sandhi rule, a resultant sound may be result of many sound combinations and all those sound sequences should be recorded in rules as possible splits of resultant sound, but, only a few of them will be correct and in most cases, only one will be desired. This will certainly over generate the results otherwise there will be chances to miss the desired result if all possibilities are not considered.

3.4.3 DEFICIENCIES IN RULE WRITING SYSTEM

In the rule writing method mentioned above, it is not clear how context is defined. More problems are there where the presence of vowel /a/ is effective. Here the rules are written to find the marker of sandhi, the resultant sound, in the string as it appears. In the Devanagari Unicode writing system, vowel /a/ is not represented by any sign except in the beginning of the word, though absence of any vowel can be identified by *halanta*, and other vowels are identified by their *mātrā*. In this rule writing system, /a/ is represented by space or nothing. This problem of identifying each phoneme is tried by small embedded java program of phoneme splitting which splits the vowels, consonants, *anusvāra* and *visarga*s from a Devanagari Unicode string. Now the string is transformed parallel to phonemic representation. If the rules are written in phoneme split form and input string is split into phonemic representation, each sound of rule can be found in the string if it occurs there. For validation, the segments can be rejoined into Devanagari orthography. Phoneme splitter splits Devanagari combinations as following:

Input: सह नाववतु

Output: स् अ ह् अ न् आ व् अ व् अ त् उ

Some rules are such that one marker is a substring of the marker of another rule. Some rules are such that markers of both are similar and context of application of one rule is simpler and that of another is complex. Thus scope of application of one rule is overtaken by another. In cases like these two, if rule with wider scope is applied first, it leaves no scope for applying rule of narrower scope. For these reasons, rule application ordering is to be carefully defined keeping in mind their scope of application. One criterion is to keep rules with bigger marker prior to rules with smaller marker. Another criterion is to keep rules of narrower scope prior to rules of wider scope, in other words, *apavāda* prior to *utsarga* or general rule.

3.4.4 HORIZONTAL VS. VERTICAL PROCESSING

One question about rule application is horizontal or vertical processing. Here horizontal means application of one by one rule along the string on all possible sandhi points of that rule. In this method, one rule is taken and its marker is searched for application along the string left to right, and it is broken according to rules at those points, and then the next rule is taken. Vertical means application of all possible rules one by one on a possible point of sandhi. In this method, string is taken, its first possible point of sandhi (a sound sequence) is considered and rules are checked for application one by one. Whichever rules are applicable are applied All the remaining rules are memorized/stored for further consideration. Then the next possible point of sandhi is taken for the same vertical processing. Both of these methods have pluses and minuses. In the horizontal method, if the same rule is applicable on all points, all the segments are likely to be validated if it breaks at correct points. But if there are more later (which are to applied after the present rule according to the sequence of application) rules applicable, and that in between two points of the present rule, the segment within would not be validated. The invalidated string would again be considered for search and application of later rules. This was the case where the sandhi is broken on desired or correct points. But it also may be broken on undesired or incorrect point. That undesired break will make the desired word unavailable even if the correct rules are applied on both edges of that word (that is wrongly broken). Wrongly broken word (not full word separated with space, but word which had to be a segment) neither will be sent for validation with its broken parts, nor will it be found in the dictionary or corpus to be validated. Solution for this problem can be found in two ways, either by rejoining (optionally) segments on break points of prior rules before applying the present rule or by trying the new rule on fresh unbroken string. This solution also can have problems. If the new rule is applied on fresh unbroken string, no word would be validated which has two rules applicable on both of its edge points. Only first and final word and the word with same rule applicable on both of its edge points will be validated. If the breaks of prior rules are optionally rejoined, then a rule will be applied taking each point of break one by one as broken and unbroken. Then it would be a problem identify the sequence of rule application, breaking string, rejoining previous breaks to cover all possibilities of segmentations. No doubt, there is possibility of huge over generation.

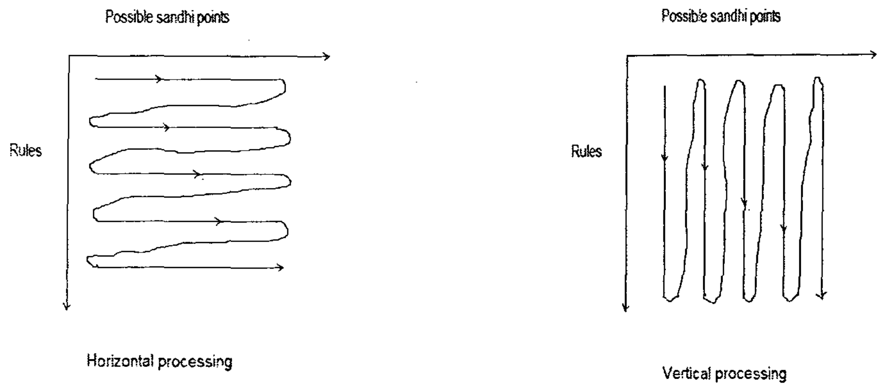


Fig. 3.1: horizontal and vertical rule application

3.4.5 COVERING ALL POSSIBLE SEGMENTATIONS

We have seen above that there are many reasons for generating many sets of segmentations. There are many rules applicable on similar sound sequences. In a single rule, same sound sequence may be split into more than one pair of sounds. Among the possible points of sandhi splitting, some may be desirable and some may not be. If a point undesirably splits, it would not let the correct word to be found. So in application of every new rule, the segmentation by previous rules should be taken as done and undone optionally. Also not all can be taken as done at a time and undone at another time, but one by one with combination of all others as done and undone. That is the way to collect and recall all the permutations (and combinations with order) of segments that correct combination should not miss. Taking each point of break in two possible states – done or undone, the number of permutations will rise at least upto 2^N, where 'N' is the number of points of segmentation in the string. If this is counted by the combination method, the number of segmentations remains the same. Assuming that each possible point of break has one rule applicable, not overlapping and has one solution, the number of permutations will be as following:

$$C(N,0) + C(N,1) + \ldots\ldots\ldots\ldots\ldots + C(N,N-1) + C(N,N)$$

Formula 3.1: calculating number of segmentation with one break at a point

Here 'N' is the number of possible points of break and 'C(N,r)' is the number of possible combinations of 'N' items taking 'r' at a time. This is calculated by taking possibilities of zero points broken, one point broken, two points broken ...... all points broken. Here the number of permutations is calculated by combination because the order of segments is unchangeable. The assumptions, on which the above cālculation of segmentations is based, are rare. In fact, sandhi markers (resultant sounds of sandhi) can overlap, more rules can be applicable on one marker and one rule can break a marker into many. So, the real number of possible segmentations will be far more than this calculation. The calculation mentioned above is illustrated here with an example of a string with five possible sandhi break points. In the example, states of possible points of sandhi are indicated by 0 or 1 where 0 indicates point unbroken and 1 indicates point broken. Thus the original string is 00000. Then the possible segmentations including original one will be as following:

C(5,0) = 1 → 00000,

C(5,1) = 5 → 10000, 01000, 00100, 00010, 00001,

C(5,2) = 10 → 11000, 01100, 00110, 00011, 10100,
01010, 00101, 10010, 01001, 10001,

C(5,3) = 10 → 11100, 01110, 00111, 11010, 01101,
10110, 01011, 11001, 10011, 10101,

C(5,4) = 5 → 11110, 01111, 10111, 11011, 11101,

C(5,5) = 1 → 11111,

Fig. 3.2: structure of segmentation sets of a string with five sandhi points

Thus the total number of segmentation sets in a string with five possible points of sandhi under assumed conditions is 32 which is equal to 2^5. In this calculation the possibilities of simple concatenations are not calculated. The number of segmentations goes to 2^5 when there are two conditions for each point possible. When there adds a third condition of break without sound change, the number will go to $3^5 = 243$. Here again it is assumed that simple concatenation also may be only on possible points of sandhi.

Where the number of breaks on each break point is one or more than one, the calculation formula for number of segmentations will be different.

$$(N(B_1)+2)\ .\ (N(B_2)+2)\ .\ \ldots\ldots\ .\ (N(B_{n-1})+2)\ .\ (N(B_n)+2)$$

Formula 3.2: calculating number of segmentation with many breaks at a point

Here ‘N’ is the number of possible breaks at a break point and ‘n’ is the number of possible break points and ‘B’ indicates break point. ‘+2’ indicates two conditions of that point – unbroken and broken without sound change (to resolve simple concatenation).

3.4.6 SCREENING OF INVALID AND UNDESIRED SEGMENTATIONS

After collecting the permutations of segments, next step is the screening of the permutations. First to select the correct sets and then to find relevant, desired and popular permutation/set of segments. The goal is to arrive at one desired solution, and then it is success of system. Many of the sets would be screened off due to having invalidated segments.

3.4.7 DIFFERENCE IN APPROACH OF HUMAN AND MACHINE

Difference in the process of analyses by human and computer and computer's inability to guess and intuition makes the task more challenging. The method of processing by the computer is different from human's method in many ways.

- First, human simultaneously applies rules and recognizes meaningful words while processing of computer is linear. Computer can apply rules one by one and cannot recognize words simultaneously but after that.
- Second, human when sees a string, he applies all his linguistic knowledge including the contextual and world knowledge simultaneously and therefore is able to see the parts of the word instantly. On the other hand, computer sees string as a sequence of symbols and it neither has intuition nor knows or can guess context.
- Third, the size of active vocabulary of computer is different from that of human. Human recognizes a meaningful word by his active vocabulary which is generally smaller compared to what a computer can hold in its active memory. Human probably organizes information based on popularity, frequency of use and pragmatics. He can expand it according to need. Probably therefore, humans are able to recognize words never before. In fact, human's active vocabulary is set in active morphological knowledge. On the other side, computer's vocabulary can be potentially very large, theoretically containing all base words of the language. Also a computer's entire vocabulary can be considered 'active'. By plugging in statistical data information with respect to popularity, frequency etc can also be stored. The multimodal systems can even have algorithms to accumulate new words (as humans do). But all this knowledge is not available to the machine simultaneously as it is to humans who have common sense reasoning and unbelievable fast parallel processing capability.

3.4.8 VALIDATION OF COMPLEX WORDS

Validation of complex words/segments is also a challenge. Simple words can be validated by finding their existence in the dictionary or corpus. In a complete system of Sanskrit analysis, morphological analyzing tools require individual words to be identified, i.e., sandhi processed or sandhi free text. And if sandhi analysis system depends on them, it is a vicious circle of interdependency. Previously this chapter has discussed that what the problems in identifying the suffixes are. When the word is with prefix and more than one suffix, identification and validation becomes more difficult. There are more problems in identifying derived words. They can be compound of two or more words, can have serial siffixation, prefixation etc. Though they have a system and cannot come in random order, but their possible structures are various. Complexity of derived words is shown in the work of Surjit (2008) as follows -

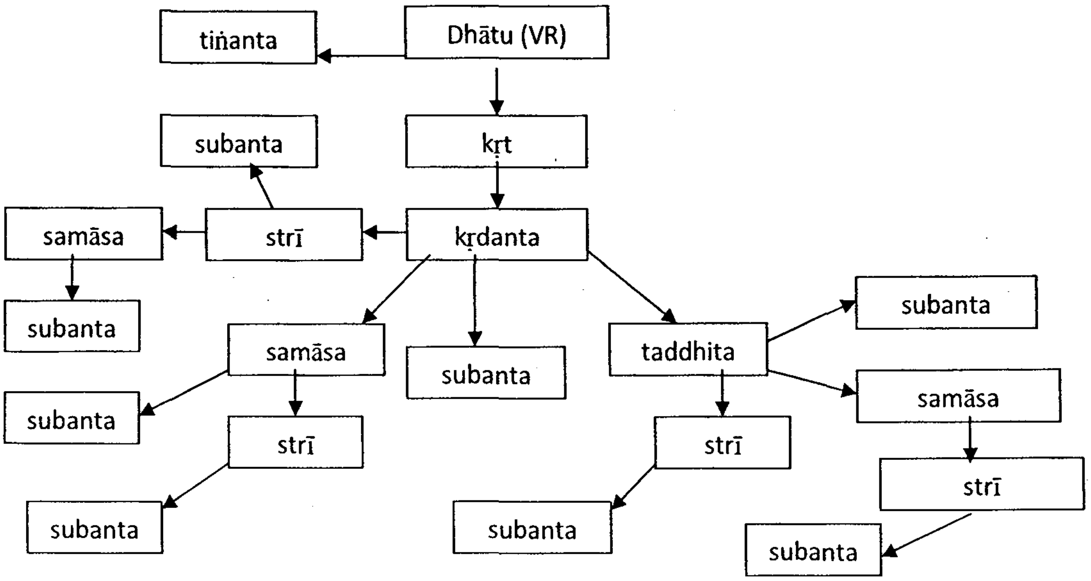


Fig 3.3: Nominal derivation from verb root[1]

In the diagram above, only variety of nominal derivation from verb root is shown. Besides there are verbal derivatives derived from both nominal bases and verb roots.

3.5 SUGGESTED ALGORITHM FOR SANDHI PROCESSING

New algorithm will consider the problems and challenges discussed above. It would have strategy of two levels - wider and deeper. Wider level algorithm is called macro algorithm and the deeper level algorithm is called micro algorithm. Macro algorithm deals with overall

[1] Surjit (2008: 26)

applications of rules, how to prepare text for sandhi processing, how to find possibility of rule application, how and in which order to apply rules, how to store all possible segmentations and how to screen them and find the solution. Micro algorithm deals with application of an individual rule on its possibility of application. Screening or validating is again a complex process which also needs a separate micro algorithm.

3.5.1 MACRO ALGORITHM FOR SANDHI PROCESSING

This algorithm controls overall process of sandhi splitting. The process is as follows-

Pre-processing

- Pre-processing or preparing of the text for analysis. In this step, the text is checked for proper encoding – whether the text is in Devanagari UTF-8 encoding. If it is, then undesirable symbols like punctuation marks etc. Pass the text from *subanta* analyser which analyses nominal inflections and identifies regular verb forms, indeclinables and proper names from the verb database, *avyaya* database and names database.
- Exclude verb forms, avyaya and proper nouns from sandhi processing. Bases of *subanta*, unanalyzed *subanta*s and unidentified words will be considered for sandhi analysis.

Prepare each word for sandhi analysis

- Take first/next string.
- Split phoneme as in 3.4.3 above.
- Count and mark the sandhi markers in the string with their serial number. Marker is the set of sounds/sound sequence which appears as the result of a sandhi. Markers are easy to count in phoneme split string than in continuous spelling (because each phoneme is separately visible in phoneme split form. Markers are listed in a list in which each marker is associated with a separate rule and they themselves are in phoneme split form. More markers can be associated with one rule and more rules can be associated with one sandhi marker but there should be one to one mapping not list to one, one to list or list to list mapping.

- While counting, first consider larger markers then smaller ones. Even if markers overlap partly or fully, each marker should be marked. One marker may be substring of other; even then it is to be marked. The serial number of every marker should be different and should be memorized in all steps till the segmentation process ends (before validation).
- Counting of marker should start from second phoneme, though first phoneme can have sandhi. Next round of marker counting should start from phoneme next to one from which previous round has started.
- Serial number of marker is in the form of 1a, 1b, 1c, 2a, 2b... constitutes of round number as natural number (1, 2, 3...) and marker number in that round as alphabet (a, b, c...).
- Marker should be taken as prefix of the part of string starting from the phoneme of counting.
- Start a list of segmentations. First segmentation is string itself. Segmentation means a set of possible segments in order after sandhi analysis of the string. List should have all the possible segmentations of the string applying rules of Sanskrit sandhi.
- List of segmentations has a serial number constituting of marker number and rule number of applied rule in the form of 1a.(rule number).

Applying the rules

- Take first sandhi marker.
- Apply first rule associated with current marker and add segmented string to the list of segmentations. If there are more than one result on applying the rule, add all of them to the list. Other later sandhi markers marked on string will remain marked.
- If there is any rule associated with the current marker, apply the rule on that marker, not on other possible markers. Add the segmentation(s) to the list of segmentations.
- If there is no other rule associated with the current marker, take the next sandhi marker.

Loop: applying rule on next marker and add segmentation to the list

- Repeat the process mentioned in above three points – applying rule on marker, and add segmentations in the list of segmentations.
- This process should be applied on all segmentations associated with prior round of markers. Remember that marker serial number constitutes of round number and marker number in a round of sandhi marker counting.
- In all segmentations to be applied a rule, rule should be applied on the marker of same serial number at a time.
- In each round (not in each marker) one segmentation should be such that breaking the string into two parts without any sound change. This is done to explore possibility of simple concatenation.
- All new segments thus found should be added in the list of segmentations.
- List completes when there are no more markers and no other rule left to be applied on the string.

Screening or validation of segmentations

- Start validation process with verb database, customized corpus and customized Monier Williams Sanskrit Digital Dictionary (MWSDD). They are customized by tagging popularity of every word entry.
- Remove the sandhi marker marks from all the segmentations.
- Generate written forms of the word from phoneme split forms of segments by reverse of phoneme splitter.
- In all segmentations, take each segment and search in verb database, MWSDD and customized corpus and tag with its popularity index (1 – 10) tagged with the entry in the lexical source (database, dictionary and corpus).
- The segments which are not found in the lexical sources and are with reasonable length are sent for suffix identification (not analysis) and suffix removed segment is again sent for validation to the lexical resources.
- If validated in inflection suffix identification, then the segment (without removing suffix) is tagged with popularity index one less than in the source because the identification of suffix is probable to be wrong. If it is validated in derivation suffix identification, then the segment is tagged with popularity index two less than in the source because here is more probability of false analysis.

- The segments which are not validated till this stage should be tagged with popularity index -1.

Statistical calculation for screening

- In the following steps, the data given in number is not static but can change depending on tests and experiments.
- In the list of segmentations, if contents are more than 50, reject all the segmentations which have any of segments with -1 popularity index. If they are 10 to 49, then reject all the segmentations which have at least two segments with -1 popularity index. Don't reject any segmentation at this stage if they are less than ten.
- If no segmentation has negative index or some of segmentations have one negative index in their segments, take the average of popularity index of segments of all segmentations. If the difference between highest and second highest average index is more than a reasonable gap (here consider 0.6, which can change depending on tests of its effect on correctness of selection), choose the segmentation of highest average as correct segmentation.
- If more than one segmentation come in the range of highest average and less than it by reasonable gap mentioned above, choose by higher frequency of higher popularity index. For it, increase the weightage of index by adding to each the mean deviation – difference from 5.
- Where there are segmentations with more than negative index, which may be the case of less than ten segmentations, there will be different formula for selecting correct and desirable solution. Here the reasonable gap is more than previously mentioned condition (here consider 1).
- Take the average of the popularity index of segments in all segmentations. If no other segmentation comes in the range of highest average and less than it by reasonable gap, choose the segmentation with highest average as the correct segmentation.

3.5.2 MICRO ALGORITHM FOR SANDHI ANALYSIS

Macro algorithm is applicable on individual rule so it can be somehow different for different rules depending upon the nature of application and condition of the rule. But there is some common process applicable on all rules. The rule specific process is to be defined in the rule itself. This algorithm may need illustration with an example of a specific rule. The common algorithm is as follows:

- Take the string marked with the sandhi markers marked with their serial numbers on which the rule is applied.
- Select the first/next marker to apply the rule. The marker should be that with the serial number provided by macro algorithm.
- Select the first/next rule associated with that marker. The rule to be applied is also decided at the macro level.
- Obtain the definitions of technical terms used in the rule. Technical terms like *pratyāhāra*s, vowel, consonant, voiced, unvoiced, velar, prefix, verb etc. are separately defined as closed lists.
- Check the conditions or environment of the sandhi point if they satisfy the conditions as mentioned in the rule.
- If yes, then make the replacement of sounds according to rule. If there is more than one optional replacement, make all replacements and regard them all as separate segmentations. Remove the mark on current marker from split string and leave the other markers marked. Add this/these to the list of segmentations.

3.5.3 ILLUSTRATION OF A VYAÑJANA SANDHI RULE

Rule is in the form that may be applicable for both generative and analytical purpose. The difference in both approaches will be that in the generative approach, LHS will be replaced by RHS if the condition of LHS satisfy while in the analytical approach, RHS will be replaced by LHS if the condition of RHS satisfies.

Marker: [ज् ज], Rule: (झलां जश्झशि + स्तोः श्चुनाश्चुः)

Rule: [त्,थ्,द्,ध्,स]+[ज्,झ]=[ज]+[ज्,झ]: (झलां जश्झशि + स्तोः श्चुनाश्चुः)

Obtain the definitions of technical terms – not applicable here

Make pairs of original and replacements

[त्:ज्/थ्:ज्/द्:ज्/ध्:ज्/स्:ज्]

Check the condition for replacement (of RHS)

ज् is followed by ज् or झ्

If satisfies, (as in स् अ [ज् ज्] अ न् अ) replace replacement by corresponding original

Replace first j with all 5 corresponding originals in above pairs

Thus obtain 5 segmentations and add them to list of segmentations

[स् अ [त् ज्] अ न् अ]

[स् अ [थ् ज्] अ न् अ]

[स् अ [द् ज्] अ न् अ]

[स् अ [ध् ज्] अ न् अ]

[स् अ [स् ज्] अ न् अ]

Split string from start of marker and add it to the list of segmentations

[स् अ - [ज् ज्] अ न् अ]

Stop.

Rejoining of phonemes into spelling and validation/screening are part of macro algorithm. Here illustrated algorithm shows two *sūtra*s combined as one *sūtra*. Those two *sūtra*s also are independent rules from this combined rule. Purpose of keeping combined rule different from constituent rules is that once a rule is applied, it should not leave scope of applying other rule at the same point. There are already many points of rule applications causing wide over-generation. If analyzed points are again explored for new rule application, the process will become endless. For example, in the above illustrated example, one replacement of [j j] is [j j] itself. If it is again applied this rule, it will start uncontrolled chain reaction and infinite recursion.

3.6 RESEARCH METHODOLOGY OF THE CURRENT RESEARCH

The present work is interdisciplinary in nature. The research methodology used in this research involves the methodologies of linguistics, computational linguistics and software development. Though these methodologies are interwoven and distinct portions of the dissertation under each methodology cannot be clearly separated, they can be identified in different tasks of research.

Linguistic methodologies – for understanding sandhi analysis, the study of Sanskrit sandhi is required. As sandhi *viccheda* aims at recognizing all the distinct words in a continuous string. Words in sandhi do not necessarily occur in base form. They can occur with all their complexity of structure. So, for the recognition of the words, the recognition of word structure is required. The thorough study of Sanskrit morphology is required, also the understanding of complexity of morphology. Detailed concepts of morphology is given in the first chapter which is less Sanskrit specific and more general. For the computational application of sandhi, not only understanding of rules is sufficient, but to understand them from the perspective of linguistics. In the second chapter, the sandhi system of Sanskrit is introduced and sandhi rules are classified in different order from *Aṣṭādhyāyī* and *Siddhānta-kaumudī* and written in phonological rule writing system as much as possible. All rules are categorized into five categories and twenty eight rules.

Methodologies of Computational Linguistics – these are not independent from linguistic methodologies but there is slight difference from computational perspective. Before writing code of rules in formal language, they are to be written in formal way, which is closer to the format which computer understands. Rules of AD are more formal than any documented grammar of any natural language but they are not absolutely formal to write in formal language. Study of possibility and problems in rule formalization with theories of computational linguistics is needed. To explore the process of analysis to overcome the problems, data designing, dictionary adapting, corpus customization according to the planned strategy for sandhi analysis are also part of computational linguistics methodology. This is discussed and presented in the third chapter of the dissertation.

Methodologies of software development – these involve algorithm development, tools selection, data formatting, writing code of program, test and analysis of the system. The algorithm is dived into two levels - macro and micro. It mentions and requires different databases in a certain format. Both algorithms and data structure is given in the third chapter. System is planned to develop in web architecture. The system tools are Java based. Front end is done in Java Server Pages (JSP) running on the webserver called Apache Tomcat. The processing databases are in text format and the programming has been done in the Java environment. The system design, modules etc are illustrated in the fourth chapter of the dissertation.

CHAPTER FOUR

SANDHI ANALYZER SYSTEM DESCRIPTION

Chapter four

SANDHI ANALYZER SYSTEM DESCRIPTION

4.1 INTRODUCTION

This chapter presents the implementation part of the research. The topic of the research is to explore the issues and challenges. To explore these challenges, a web based system has been developed which will be illustrated here in this chapter. *Vyañjana* sandhi processing system is integrated with *ac*-sandhi analysis system developed by Sachin as his M. Phil. research (2007). This is a partial implementation of the algorithm discussed in chapter three. That algorithm, especially the screening section is subject to test and experiment. The numerical values are even more changeable. The developed computational model uses Java based web technology. The system depends upon the formalization of *Pāṇinian* rules and their description in *Siddhānta-kaumudī.* The system accepts the Sanskrit text in Devanagari Unicode UTF-8 format and returns output in similar format. The system takes the input in text area, prepares it for processing, identifies possibilities of sandhi *viccheda* and segments from those points. The segmentation is displayed as output only if all the segments are validated either by their occurrence in the dictionary or occurrence of their stem after *subanta* analysis. The present system also inherits the 'Subanta Recognition and Analysis System for Sanskrit' developed as part of M.Phil. research by Subash (2006). The chapter also describes the codes of the modules of the system.

4.2 DATA DESIGN

The system consists of front end of Apache Tomcat server, programming is done in Java and back-end is in form of different data files. Here the formats of data files are briefly explained with their sample from file.

4.2.1 EXAMPLE-BASE OF THE SYSTEM

First data file is the ‘example_base’ . This contains uncommon sandhi words with their segmentation. There are some data which cannot be explained by general rules but there are special rules for them. There are some special rules which apply for less data. Some rules are such that their implementation is too hard and it is better to declare direct result than explain the process. Such data from computational purpose are easier and comprehensive to process through example based or statistical techniques. The *nipātana* technique used in Pāṇinian system does the same by directly declaring the result of process instead of explaining. There are some rules in Pāṇinian system as *kṣayya-jayyau śakyārthe* (P 6.1.81); *krayyastadarthe* (P 6.1.82) and *vārtika*s as *śakandhvādiṣu pararūpaṃ vācyam* which directly declare the word forms.

4.2.2 FORMAT OF RULES

Second data file is ‘viccheda_pattern’ . This contains the formal rules of the *vyañjana*-sandhi in a specific format. Rules of *svara*-sandhi from the system developed by Sachin (2007) are also inherited and adapted in this format. These rules are of two types, static and dynamic. Static rules have sounds themselves and dynamic rules consist of some technical terms also which are in the form of Roman alphabet which have some meaning interpreted in Java classes. The rule consists of ‘marker’, ‘replacement’ and sandhi name and *sūtra.* Marker and replacements together called ‘pattern’ are separated with each other by ‘=’ sign and together separated from sandhi name and *sūtra* by ‘:’ sign. The patterns are in phoneme split form. The sample from viccheda_patterns is given as follows with both types of rules.

ग् ग्=क्+ग्:(झलां जश् झशि);ग् ग्=ग्+ग्:(झलां जश् झशि);ग् घ्=क्+घ्:(झलां जश् झशि);ग्
घ्=ग्+घ्:(झलां जश् झशि);ज् ज्=च्+ज्:(झलां जश् झशि);ज् ज्=ज्+ज्:(झलां जश् झशि);ज्
झ्=च्+झ्:(झल़ां जश् झशि);ज् झ्=ज्+झ्:(झलां जश् झशि);ज् ज्=त्+ज्:(झलां जश् झशि + स्तोः
श्चुनाश्चुः);ज् ज्=द्+ज्:(झलां जश् झशि + स्तोः श्चुनाश्चुः);ज् झ्=त्+झ्:(झलां जश् झशि + स्तोः श्चुनाश्चुः);ज्
झ्=द्+झ्:(झलां जश् झशि + स्तोः श्चुनाश्चुः);ड् ड्=ट्+ड्:(झलां जश् झशि);ड् ड्=ड्+ड्:(झलां जश् झशि);ड्
ढ्=ट्+ढ्:(झलां जश् झशि);ड् ढ्=ड्+ढ्:(झलां जश् झशि);ड् ड्=त्+ड्:(झलां जश् झशि + ष्टुनाष्टुः);ड्

ड्=द्+ड्:(झलां जश् झशि + ष्टुनाष्टुः);ड् ड्=ध्+ड्:(झलां जश् झशि + ष्टुनाष्टुः);ड् ढ्=त्+ढ्:(झलां जश् झशि + ष्टुनाष्टुः);ड् ढ्=द्+ढ्:(झलां जश् झशि + ष्टुनाष्टुः);

अ ङ् ङ् s=अ ङ्+s:(ङमो ह्रस्वादचि ङमुण्नित्यम्);इ ङ् ङ् s=इ ङ्+s:(ङमो ह्रस्वादचि ङमुण्नित्यम्);उ ङ् ङ् s=उ ङ्+s:(ङमो ह्रस्वादचि ङमुण्नित्यम्);ऋ ङ् ङ् s=ऋ ङ्+s:(ङमो ह्रस्वादचि ङमुण्नित्यम्);ऌ ङ् ङ् s=ऌ ङ्+s:(ङमो ह्रस्वादचि ङमुण्नित्यम्);अ ण् ण् s=अ ण्+s:(ङमो ह्रस्वादचि ङमुण्नित्यम्);इ ण् ण् s=इ ण्+s:(ङमो ह्रस्वादचि ङमुण्नित्यम्);उ ण् ण् s=उ ण्+s:(ङमो ह्रस्वादचि ङमुण्नित्यम्);b छ् s=b+श् s:(शश्छोऽटि);p छ् s=p+श् s:(शश्छोऽटि);b छ् a=b+श् a:(शश्छोऽटि);p छ् a=p+श् a:(शश्छोऽटि);

Here s=svara, v=vyañjana, a=semivowel, p=unvoiced stop, b=voiced stop

4.2.3 CUSTOMIZING 'MWSDD'

Third data file is th ecustomized Monier Williams Sanskrit Digital Dictionary (MWSDD). Dictionary items have their popularity index from 1 to 10 to be marked with each as discussed in the third chapter. This index depends upon the frequency of the lexical item in the corpus and the complexity of word structure. This popularity index is needed in the screening of undesired results. Current implementation does not use popularity index and dictionary is not marked with that. The sample of dictionary is as follows.

उत्तरंग;उत्तरंगय;उत्तरल;उत्तरलाय;उत्तरलित;उत्तरलीकृ;उत्तर्जन;उत्तान;उत्ताप;उत्तार;उत्ताल;उत्तालीभवन;उत्तिङ्ग;उत्तिज्;उत्तेजक;उत्तेजन;उत्तेजित;उत्तीर्ण;उत्तु;उत्तुङ्ग;उत्तुङ्गता;उत्तुङ्गत्व;उत्तुण्डित;उत्तुद्;उत्तुद;उत्तुल्;उत्तोलन;उत्तोलित;उत्तुष;उत्तृद्;उत्तृ;उत्तर;उत्तरण;उत्तरिका;उत्तार;उत्तारक;उत्तारण;उत्तारिन्;उत्तार्य;उत्तार्य;उत्तितीर्षु;उत्तीर्ण;उत्तीर्णविकृति;उत्तीर्य;उत्तेरित;उत्तोरण;उत्तोरणपताक;उत्तोलन;उत्त्यज्;उत्त्यक्त;उत्त्याग;उत्त्रस्;उत्त्रस्त;उत्त्रास;उत्त्रासक;उत्त्रिपद;उत्त्रुट्;उत्त्रुटित;उत्था;उत्तिष्ठासा;उत्थ;उत्थातव्य;उत्थातृ;उत्थान;उत्थानयुक्त;उत्थानवत्;उत्थानवीर;उत्थानशील;उत्थानशीलिन्;उत्थानहीन;उत्थानैकादशी;उत्थानीय;उत्थापक;उत्थापन;उत्थापनीय;उत्थापयितृ;उत्थापित;उत्थाप्य;उत्थाप्य;उत्थाय;उत्थायोत्थाय;उत्थायम्;उत्थायिन्;उत्थायित्व;उत्थित;उत्थितता;उत्थिताङ्गुलि;उत्थिति;उत्पक्ष;उत्पक्ष्मन्;उत्पक्ष्मल;उत्पच्;उत्पचनिपचा;उत्पचिष्णु;उत्पाचित;उत्पट्;उत्पट;उत्पाट;उत्पाटयोग;उत्पाटक;उत्पाटन;उत्पाटित;उत्पाटिन्;उत्पाट्य;उत्पत्;उत्पत;उत्पतन;उत्पतनिपता;उत्पतित;उत्पतितव्य;उत्पतितृ;उत्पतिष्णु;उत्पात;उत्पातक;उत्पातिक;उत्पित्सु;

4.2.4 DATA FILES OF THE INHERITED SYSTEM

Besides vowel sandhi, the system inherits *subanta* analyzer system developed by Subash (2006). It is used in validating segment by analyzing *subanta* if it is not validated in its original form. That system has two data files included in the present system – ‘sup_EB’ (*subanta* example base) and ‘sup_RB’ (*subanta* rule base). Uncommon and irregular nominal forms are analyzed by *subanta* example base. *Subanta* rule base contains patterns of nominal endings and their analysis which are identified in each word. Sample of ‘sup_EB’ is as follows:

अवतारः=अवतार+सु, प्रथमा एकवचन;जलमुक्=जलमुच्+सु, प्रथमा एकवचन;जलमुचा=जलमुच्+टा, तृतीया एकवचन;धावन्=धावत्+सु प्रथमा एकवचन;स्=स्+स्;धीः=धी+सु, प्रथमा, एकवचन;ह्रीः=ह्री+सु, प्रथमा, एकवचन;श्रीः=श्री+सु, प्रथमा, एकवचन;भीः=भी+सु, प्रथमा, एकवचन;वृश्चिकभीः=वृश्चिकभी+सु, प्रथमा, एकवचन;भूः=भू+सु, प्रथमा, एकवचन;सूः=सू+सु, प्रथमा, एकवचन;जूः=जू+सु, प्रथमा, एकवचन;भ्रूः=भ्रू+सु, प्रथमा, एकवचन;सुभ्रूः=सुभ्रू+सु, प्रथमा, एकवचन;भूनाम्=भू+आम्, षष्ठी, बहुवचन;स=तद् सु प्रथमा एकवचन;मम=अस्मद् ङस् षष्ठी एकवचन;विषये=विषय+ङि सप्तमी एकवचन;सा=तद्+सु , प्रथमा एकवचन;ना=नृ+सु , प्रथमा एकवचन;नरः=नर+सु , प्रथमा एकवचन/नृ+जस् प्रथमा बहुवचन;

Sample of ‘sup_RB’ is as follows:

स्यां=+ङि्, सप्तमी विभक्ति एकवचन);ात्मा=ात्मन्+सु,प्रथमा विभक्ति एकवचन;राजः=राजन्+सु प्रथमा विभक्ति एकवचन;ौ= +औ(प्रथमा विभक्ति/द्वितीया विभक्ति द्विवचन);ं= +अम्(द्वितीया विभक्ति एकवचन);ेषु= +षु(सप्तमी विभक्ति बहुवचन);ापो=प्+जस्(प्रथमा बहुवचन);ायं=ायं+अम्(प्रथमा विभक्ति एकवचन);ैनं=ैनं+अम्(प्रथमा विभक्ति एकवचन);ः= +सु(प्रथमा विभक्ति एकवचन);स्मिन्=स्मिन्+अस्मिन्(सप्तमी विभक्ति एकवचन);ो=+सु(प्रथमा विभक्ति एकवचन); ृत्=ृत्ृ (पुल्लिङ्ग) + सु, प्रथमा, एकवचन;ृतौ=ृत्+औ/औट्, प्रथमा/द्वितीया, द्विवचन;ृतः=ृत्+जस्/शस्/ङसि/ङस्, प्रथमा/द्वितीया, बहुवचन, पञ्चमी/षष्ठी, एकवचन;ृतम्=ृत्+अम्, द्वितीया, एकवचन;ृता=ृत्+टा, तृतीया, एकवचन;ृद्भ्याम्=ृत्+भ्याम्, तृतीया/चतुर्थी/पञ्चमी,

द्विवचन;ृद्भिः=ृत्+भिस्, तृतीया, बहुवचन;ृते=ृत्+ङे, चतुर्थी, एकवचन;ृद्भ्यः=ृत्+भ्यस्, चतुर्थी/पञ्चमी, बहुवचन;ृतोः=ृत्+ओस्, षष्ठी/सप्तमी, द्विवचन;ृताम्=ृत्+आम्, षष्ठी, बहुवचन;ृति=ृत्+ङि, सप्तमी, एकवचन;ृत्सु=ृत्+सुप्, सप्तमी, बहुवचन;ित्=ित्+सु, प्रथमा, एकवचन;

4.3 SYSTEM MODULES

System is developed in multi-tier web-architecture. Its front end is in JSP, a java based server language. Back-end is data files in proper format containing data in UTF-8 Devanagari and Roman characters and symbols. Programming is done in a few Java objects. Here it is illustrated how different modules of the system function with the help of the sample code.

4.3.1 FRONT-END

Front end is Java Server Pages file named `viccheda.jsp`. The program is hosted on Apache Tomcat 4.0, a java based web server. The page contains codes of HTML, JSP, Java and Javascript languages. Following is the sample code of this page.

The following code sets the language, encoding and content type of the page and imports package java.util.

```
<%@ page

     language="java"

     pageEncoding="utf-8"

     contentType="text/html; charset=utf-8"

     import="java.util.*"

%>
```

The following code imports user defined java package for sandhi processing named SandhiAPI

```
<%@ page import="SandhiAPI.*" %>
```

The following code obtains values from checkbox and text area, to run in debug mode decision from checkbox and input text from text area.

```
<%

     request.setCharacterEncoding("UTF-8");
```

This code initializes input variables.

```
     String itext = request.getParameter("itext");

     int dbg = 1;
```

This code assigns values to input variables.

```
     if  (request.getParameter("debug") == null)

          dbg = 0;

     String ch = "checked";

     if (dbg == 0)

          ch = "";

     if (itext==null)

          itext = "";
```

This code calls the main class 'Viccheda' to analyze sandhi.

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

The following is the code of the form and text area to enter the input text, check debug mode and submit button.

```
<FORM   METHOD=get   ACTION=viccheda.jsp#results   name="iform"
accept-Charset="UTF-8">

<TEXTAREA name=itext COLS=40 ROWS=9><%=itext %></TEXTAREA>

<font color=red size=2>Run in debug mode</font>

<input type=checkbox name="debug"  <%= ch %> value="ON">

     <input type=submit value="Click to sandhi-split (संधि-विच्छेद करें) ">
```

This code calls main function splitText(itext) of main class Viccheda and displays the sandhi segmentation result.

```
     <% if (itext.length()>0) { %>

          <%=v.splitText(itext) %>

     <% } %>
```

The following code displays the process of analysis of the text if the checkbox "Run in debug mode" is checked.

```
     <% if (itext.length()>0) { %>

          <%=v.printErr() %>

     <% } %>
```

4.3.2 CORE PROGRAM STRUCTURE

Programming part of the system is in Java objects packaged in SandhiAPI. Main class is Viccheda and other classes are Preprocessor, Segmenter, RSubanta, SupAnalyzer, MatraVowel, and STokenizer. RSubanta and SupAnalyzer classes are for analyzing subanta of a segment to get its stem to validate if it is not validated.

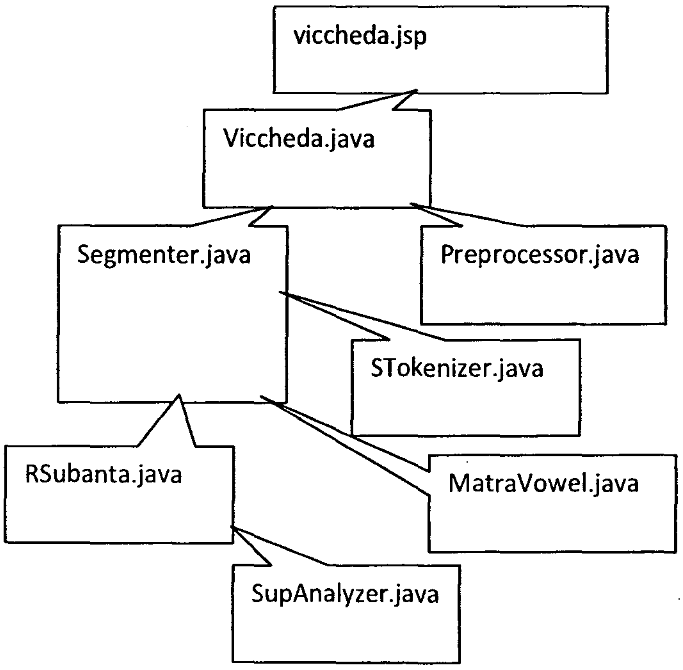


Fig. 4.1: Structure of core Java program for sandhi analysis

The following is the description of sample code of the main class.

The following code makes the class part of package SandhiAPI

```
package SandhiAPI;
```

The following code imports java packages to be used in the class.

```
import java.util.*;

import java.io.*;
```

The class starts

```
public class Viccheda{
```

Declaration and initialization of different variables

```
Hashtable lex =null;

BufferedReader br1 = null;

BufferedReader br2 = null;

BufferedReader br3 = null;

Preprocessor pre = null;

Segmenter s = null;

StringBuffer examples = null;

StringBuffer viccheda_patterns = null;

StringBuffer lexicon = null;

String errmsg="";

int debug = 0;
```

This is the class constructor, creates new instance of the class when called.

```
public Viccheda(int dbg){

    debug = dbg;
```

Within the class constructor, three buffered readers read three data files and assign the data to three variables *examples*, *viccheda_patterns* and *lexicon.*

```
    try{
```

```
               br1 = new BufferedReader( new
InputStreamReader(new FileInputStream("----"),"utf-8") );

               br2 = new BufferedReader( new
InputStreamReader(new FileInputStream("----"),"utf-8") );

               br3 = new BufferedReader( new
InputStreamReader(new FileInputStream("----"),"utf-8") );

               examples = new StringBuffer(br1.readLine());

               viccheda_patterns = new
StringBuffer(br2.readLine());

               lexicon = new StringBuffer(br3.readLine());

               br1.close();

               br2.close();

               br3.close();
```

Within the class constructor, this code calls class Preprocessor and Segmenter

```
               pre = new Preprocessor(debug);

               s  =  new  Segmenter(viccheda_patterns,  lexicon,
examples, debug);

          }

     }// end of class constructor
```

This is the main function splitText(s) of the class

```
     public String splitText(String s){
    }
```

The following code obtains each word as separate token.

```
          StringTokenizer st = new StringTokenizer(s, " ");
```

While loop to continue process till last token

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

Calling preProcess function which prepares the raw text for segmentation by the class Preprocessor.

```
                    tkn= preProcess(tkn);
```

When token is in appropriate format, then calling function split(tkn) which analyzes sandhi by function segment(tkn) of class Segmenter.

```
          if ( ........... ){ //not punc and a simple string
                    tkn = split(tkn);
          }
```

Definition of function preProcess(tkn) which calls function preProcess(tkn) of Preprocessor.

```
     private String preProcess(String tkn){
          if (tkn.length()>0){
               tkn = pre.preProcess(tkn);
          }
```

```
        return tkn;

    }
```

Definition of function split(tkn) which calls function segment(tkn) of class Segmenter.

```
    private String split(String tkn){

        if (tkn.length()>0)

            tkn = s.segment(tkn);

        return tkn;

    }
```

Function printErr() which prints the process of analysis when run in debug mode.

```
    public String printErr(){

        return errmsg + "<br>"+ s.printErr();

    }
```

Besides main class, the most of the work of sandhi splitting is done by the ‘Segmenter’ class. Here is brief description of functions of the ‘Segmenter’ class-

The following code is class constructor which carries inherited values of ‘viccheda_patterns’, lexicon and example (example base). This also calls two classes ‘RSubanta’ and ‘MatraVowel’.

```
    public  Segmenter(StringBuffer  vp,  StringBuffer  lex,
StringBuffer ex, int dbg){

        viccheda_patterns=vp;

        lexicon=lex;
```

```
                examples = ex;

                debug = dbg;

                subanta = new RSubanta();

                mw = new MatraVowel();

          }
```

Following is the main function of the class which checks token in example base. If not found, it checks patterns in rule base. If segmented, it validates the segments by different function.

```
          public String segment(String tkn){

          }
```

Following code checks if token has an analysis in the example base and if found, gives result.

```
          private String checkExampleBase(String tkn){

          }
```

The following function takes the token, splits into phonemes, checks applicability of rules from rule base. If any rule is applied, it splits the string and sends for validation.

```
          private String checkPatterns(String rs){

          }
```

Following code inside checkPatterns(rs) calls function of class MatraVowel to split token into phonemes.

```
          String ors = mw.splitPhoneme(rs);
```

Following code takes the string with sign of segmentations, and picks one by one token to validate by dictionary, otherwise by analyzing subanta and validating its stem. If all tokens are validated, then only allows display as output.

```
private String validateSplit(String rs){

}
```

Following code inside validateSplit(rs) calls function of class MatraVowel to rejoin split phonemes into Sanskrit spelling.

```
String ors = mw.joinSounds(rs);
```

Following code is inside the function validateSplit(rs) and gets subanta analyzed by getStem(t1) function of class RSubanta.

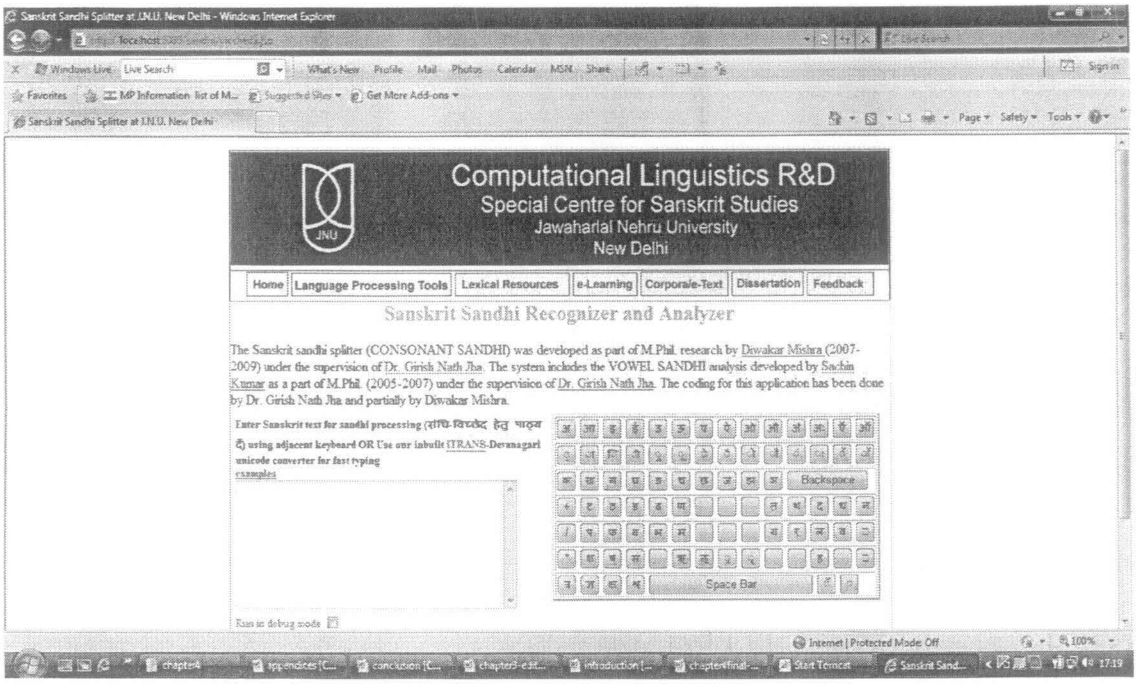
```
t1=subanta.getStem(t1);
```

4.4 INTRODUCTION OF THE SYSTEM

The system developed as the partial fulfilment of the research, to explore the issues and challenges in *vyañjana* sandhi processing, is available on the internet. The current web address of the system is http://sanskrit.jnu.ac.in/sandhi/viccheda.jsp and can be tested there. The system includes and extends the *ac*-sandhi analysis system developed by Sachin as his M. Phil. research (2007) and subanta analysis system developed by Subash as his M. Phil research (2006). The coding of the system has been done by Girish Nath Jha. Partial coding and design change and adapting of vowel sandhi system has been done by Diwakar Mishra. The Devanagari input mechanism has been developed in Javascript by Satyendra Kumar Chaube, Dr. Girish Nath Jha and Dharm Singh Rathore.

4.4.1 HOW TO USE THE SYSTEM

The Sanskrit text for splitting can be entered in the text area as illustrated below. The text has to be utf-8 Devanagari.


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
Backspace
Space Bar

Sanskrit Sandhi Recognizer and Analyzer

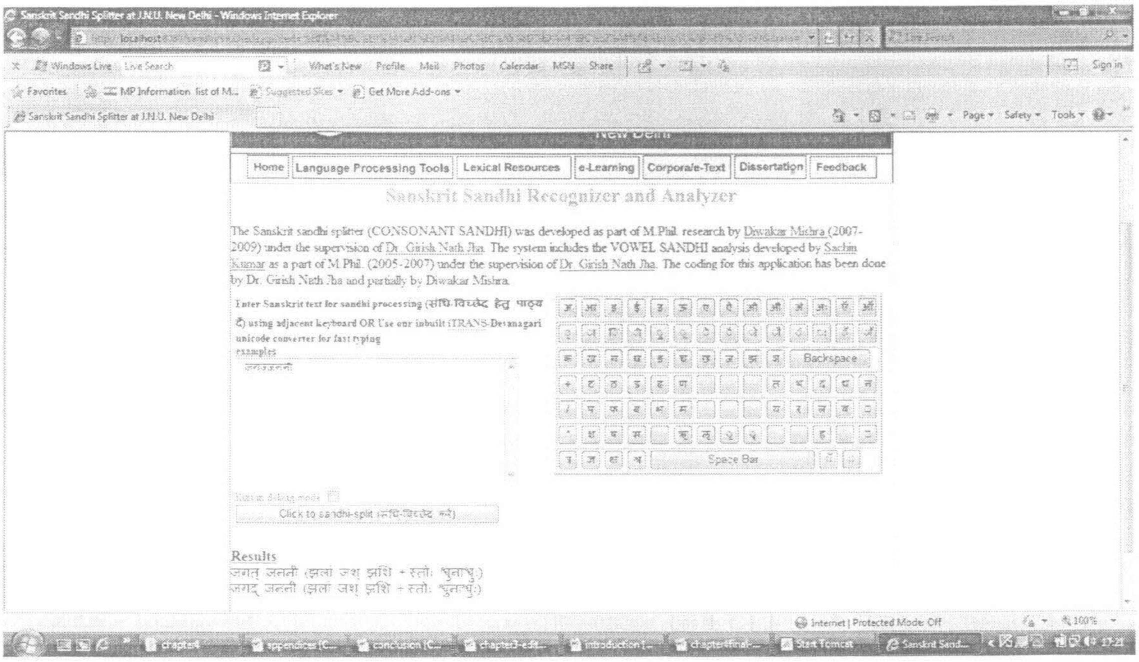
The Sanskrit sandhi splitter (CONSONANT SANDHI) was developed as part of M.Phil. research by Diwakar Mishra (2007-2009) under the supervision of Dr. Girish Nath Jha. The system includes the VOWEL SANDHI analysis developed by Sachin Kumar as a part of M.Phil. (2005-2007) under the supervision of Dr. Girish Nath Jha. The coding for this application has been done by Dr. Girish Nath Jha and partially by Diwakar Mishra.

Enter Sanskrit text for sandhi processing (संधि-विच्छेद हेतु पाठ्य दें) using adjacent keyboard OR Use our inbuilt ITRANS-Devanagari unicode converter for fast typing
examples

Run in debug mode □

Screenshot 4.1: main page of sandhi analysis system

On clicking the button 'Click to Sandhi split', the system will generate results for the input text for those words which have sandhi situation inside them –



Sanskrit Sandhi Recognizer and Analyzer

The Sanskrit sandhi splitter (CONSONANT SANDHI) was developed as part of M.Phil. research by Diwakar Mishra (2007-2009) under the supervision of Dr. Girish Nath Jha. The system includes the VOWEL SANDHI analysis developed by Sachin Kumar as a part of M.Phil. (2005-2007) under the supervision of Dr. Girish Nath Jha. The coding for this application has been done by Dr. Girish Nath Jha and partially by Diwakar Mishra.

Enter Sanskrit text for sandhi processing (संधि-विच्छेद हेतु पाठ्य दें) using adjacent keyboard OR Use our inbuilt ITRANS-Devanagari unicode converter for fast typing
examples

जगज्जननी

Run in debug mode □

Click to sandhi-split (संधि-विच्छेद करें)

Results
जगत् जननी (झलां जश् झशि + स्तोः श्चुनाश्चुः)
जगद् जननी (झलां जश् झशि + स्तोः श्चुनाश्चुः)

Screenshot 4.2: result from sandhi analysis system

The Devanagari text can be either typed using the onscreen keyboard or by the Roman keyboard of the computer in the ITRANS scheme.

Before clicking the 'Click to Sandhi split' button, if the user checks 'Run in debug mode' checkbox, then the system displays all intermediate steps leading to the final results (please refer to appendix for the result).

CONCLUSION

CONCLUSION

The research work had to explore the issues and challenges in computational processing of *vyañjana* sandhi. It is an important aspect of language processing because in case of Sanskrit, sandhi has a very important place. Though, mostly, the issues of *vyañjana* sandhi are common to the other types of sandhi, yet it has some different issues too. To understand the issues in computational processing, a close study of sandhi is required. Seeing the importance of phonology and morphology in sandhi, both topics are studied from linguistics perspective. Papers from first, second and third Sanskrit Computational Linguistics Symposia have been very useful in understanding and conceptualizing the computational and mathematical aspects of linguistics.

Before computation, formalization is a must step. In the second chapter, not only *vyañjana* sandhi rules, but also the rules of all five types of sandhi are written in phonological rule writing system. Internal sandhi is discussed in the chapter but rules are not written. Some issues are discussed in brief in first chapter and in detail they are explained in chapter three. There are some issues which arise before starting formalization. Those are either in the nature of sandhi or in the nature of Sanskrit language. Other challenges arise when one tries to implement the system. Some of them can be guessed and studied before development but some of them cannot come to sight until the system is implemented.

The main problem with sandhi analysis is to identify the correct constituent words. The words can be of any of the structures. The interdependency of sandhi system and morphological analyzer system is a major problem. On one side, morphological analyzer system requires sandhi free text, on the other hand, sandhi analyzer needs morphological analyzers to identify correct words which cannot be identified by dictionary. Inclusion of *subanta* analyzer in sandhi system is a good example of this. Other major problem is to select or identify the desired result out of many correct answers. There are other ways of context recognition and sense disambiguation, but they make the task more complex and difficult.

The usefulness of sandhi analysis system is for any Sanskrit language processing tool. It may be on morphological, sentential or discourse levels. Syntax can only be parsed after morph-analysis which in turn requires sandhi processing. Besides this, the system can be used for text simplification, self reading and learning also.

The third chapter not only finds challenges but suggests methodology and algorithm also to meet them. The suggested system is though not yet fully implemented but shows the path to overcome the problems. The suggested system also does not claim to completely solve the problems and to be a fool proof system. Some parts of the algorithm, especially the screening of the segmentation could be subjective. Even there, the numerical values are not very objective. This part is subject to change depending on the results of the tests and experiments.

APPENDICES

Appendix-1

Sandhi Splitting System Result

Results without debug mode

Input =

जगज्जननी

Output =

Results

जगत् जननी (झलां जश् झशि + स्तोः श्चुनाश्चुः)

जगद् जननी (झलां जश् झशि + स्तोः श्चुनाश्चुः)

Input =

जगदीशः

Output =

Results

जगत् ईशः (झलां जशोऽन्ते)

जगद् ईशः (झलां जशोऽन्ते)

Appendix-2

Result in debug mode showing process of sandhi splitting

Input = जगन्माता

Output (in debug mode) =

```
Results
जगत् माता (यरोऽनुनासिकेऽनुनासिको वा)
जगद् माता (यरोऽनुनासिकेऽनुनासिको वा)
जगन् माता (यरोऽनुनासिकेऽनुनासिको वा)



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



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

input=जगन्माता
output=जगन्माता

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


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

input=जगन्माता
output= जगत् माता (यरोऽनुनासिकेऽनुनासिको वा)
जगद् माता (यरोऽनुनासिकेऽनुनासिको वा)
जगन् माता (यरोऽनुनासिकेऽनुनासिको वा)

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


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

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

---------------------------START OF Segmenter.checkExampleBase()-----------------------

ts=जगन्माता

---------------------------END OF Segmenter.checkExampleBase()-----------------------

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

Split input rs=जगन्माता-----------starting to process (पूर्वरूपसन्धि, एङःपदान्तादति) sandhi---------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ऽ= +अ:(पूर्वरूपसन्धि, एङःपदान्तादति)
leftOfVP=ऽ
rightOfVP= +अ
-----------starting to process (अयादिसन्धि एचोऽयवायावः) sandhi---------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ाय=ै+ :(अयादिसन्धि एचोऽयवायावः)
leftOfVP=ाय
rightOfVP=ै+
-----------starting to process (अयादिसन्धि एचोऽयवायावः) sandhi---------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ाय=ै+अ:(अयादिसन्धि एचोऽयवायावः)
leftOfVP=ाय
rightOfVP=ै+अ
-----------starting to process (अयादिसन्धि एचोऽयवायावः) sandhi---------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ये=े+ए:(अयादिसन्धि एचोऽयवायावः)
leftOfVP=ये
rightOfVP=े+ए
-----------starting to process (अयादिसन्धि एचोऽयवायावः) sandhi---------
rs =ज् अ ग् अ न् म् आ त् आ
VP=वे=ु+ए:(अयादिसन्धि एचोऽयवायावः)
leftOfVP=वे
rightOfVP=ु+ए
-----------starting to process (अयादिसन्धि एचोऽयवायावः) sandhi---------
rs =ज् अ ग् अ न् म् आ त् आ
```

```
VP=य=ै+ :(अयादिसन्धि एचोऽयवायावः)
leftOfVP=य
rightOfVP=ै+
------------starting to process (अयादिसन्धि एचोऽयवायावः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=य=ै+अ:(अयादिसन्धि एचोऽयवायावः)
leftOfVP=य
rightOfVP=ै+अ
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=यु=ी+उ:(यण् सन्धि इको यणचि)
leftOfVP=यु
rightOfVP=ी+उ
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=य=ी+अ:(यण् सन्धि इको यणचि)
leftOfVP=य
rightOfVP=ी+अ
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=यु=ि+उ:(यण् सन्धि इको यणचि)
leftOfVP=यु
rightOfVP=ि+उ
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=य=ि+अ:(यण् सन्धि इको यणचि)
leftOfVP=य
rightOfVP=ि+अ
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=् य् ए=् इ+ए:(यण् सन्धि इको यणचि)
leftOfVP=् य् ए
rightOfVP=् इ+ए
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=् य् ऐ=् इ+ऐ:(यण् सन्धि इको यणचि)
```

```
leftOfVP=◌् य् ऐ
rightOfVP=◌् इ+ऐ
------------starting to process (लोपः शाकल्यस्य) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=य= + :(लोपः शाकल्यस्य)
leftOfVP=य
rightOfVP= +
------------starting to process (अयादिसन्धि एचोऽयवायावः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=◌ाव=◌ौ+ :(अयादिसन्धि एचोऽयवायावः)
leftOfVP=◌ाव
rightOfVP=◌ौ+
------------starting to process (अयादिसन्धि एचोऽयवायावः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=◌ाव=◌ौ+अ:(अयादिसन्धि एचोऽयवायावः)
leftOfVP=◌ाव
rightOfVP=◌ौ+अ
------------starting to process (अयादिसन्धि एचोऽयवायावः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=व=◌ो+ :(अयादिसन्धि एचोऽयवायावः)
leftOfVP=व
rightOfVP=◌ो+
------------starting to process (अयादिसन्धि एचोऽयवायावः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=व=◌ो+अ:(अयादिसन्धि एचोऽयवायावः)
leftOfVP=व
rightOfVP=◌ो+अ
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=व=◌ू+ :(यण् सन्धि इको यणचि)
leftOfVP=व
rightOfVP=◌ू+
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=व=◌ू+अ:(यण् सन्धि इको यणचि)
leftOfVP=व
```

```
rightOfVP=◌ू+अ
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=व=◌ु+ :(यण् सन्धि इको यणचि)
leftOfVP=व
rightOfVP=◌ु+
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=व=◌ु+अ:(यण् सन्धि इको यणचि)
leftOfVP=व
rightOfVP=◌ु+अ
------------starting to process (लोपः शाकल्यस्य) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=व= + :(लोपः शाकल्यस्य)
leftOfVP=व
rightOfVP= +
------------starting to process (वान्तो यि प्रत्यये/अध्व परिमाणे च) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=व्=◌ो+ :(वान्तो यि प्रत्यये/अध्व परिमाणे च)
leftOfVP=व्
rightOfVP=◌ो+
------------starting to process (वान्तो यि प्रत्यये) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=व्=◌ौ+ :(वान्तो यि प्रत्यये)
leftOfVP=व्
rightOfVP=◌ौ+
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=◌े= +ई:(गुणसन्धि आद् गुणः)
leftOfVP=◌े
rightOfVP= +ई
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=◌े=◌ा+इ:(गुणसन्धि आद् गुणः)
leftOfVP=◌े
rightOfVP=◌ा+इ
```

```
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=े=ा+ई:(गुणसन्धि आद् गुणः)
leftOfVP=े
rightOfVP=ा+ई
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=े= +इ:(गुणसन्धि आद् गुणः)
leftOfVP=े
rightOfVP= +इ
------------starting to process (पररूपसन्धि एङि पररूपम्/ओमाङोश्च) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=े= +ए:(पररूपसन्धि एङि पररूपम्/ओमाङोश्च)
leftOfVP=े
rightOfVP= +ए
------------starting to process (पररूपसन्धि एङि पररूपम्/ओमाङोश्च) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=े=ा+ए:(पररूपसन्धि एङि पररूपम्/ओमाङोश्च)
leftOfVP=े
rightOfVP=ा+ए
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ो= +ऊ:(गुणसन्धि आद् गुणः)
leftOfVP=ो
rightOfVP= +ऊ
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ो= +उ:(गुणसन्धि आद् गुणः)
leftOfVP=ो
rightOfVP= +उ
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ो=ा+ऊ:(गुणसन्धि आद् गुणः)
leftOfVP=ो
rightOfVP=ा+ऊ
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
```

```
rs =ज् अ ग् अ न् म् आ त् आ
VP=ो=ा+उ:(गुणसन्धि आद् गुणः)
leftOfVP=ो
rightOfVP=ा+उ
------------starting to process (पररूपसन्धि एङि पररूपम्/ओमाङोश्च) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ो= +ओ:(पररूपसन्धि एङि पररूपम्/ओमाङोश्च)
leftOfVP=ो
rightOfVP= +ओ
------------starting to process (पररूपसन्धि एङि पररूपम्/ओमाङोश्च) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ो=ा+ओ:(पररूपसन्धि एङि पररूपम्/ओमाङोश्च)
leftOfVP=ो
rightOfVP=ा+ओ
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ल=ऌ+ :(यण् सन्धि इको यणचि)
leftOfVP=ल
rightOfVP=ऌ+
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ल=ऌ+अ:(यण् सन्धि इको यणचि)
leftOfVP=ल
rightOfVP=ऌ+अ
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ल्= +ऌ:(गुणसन्धि आद् गुणः)
leftOfVP=ल्
rightOfVP= +ऌ
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ल्=ा+ऌ:(गुणसन्धि आद् गुणः)
leftOfVP=ल्
rightOfVP=ा+ऌ
------------starting to process (वृद्धिसन्धि उपसर्गादृति धातौ/वा सुप्यापिशलेः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
```

```
VP=◌ार्= +ॠ:(वृद्धिसन्धि उपसर्गादृति धातौ/वा सुप्यापिशलेः)
leftOfVP=◌ार्
rightOfVP= +ॠ
------------starting to process (वृद्धिसन्धि उपसर्गादृति धातौ/वा सुप्यापिशलेः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=◌ार्=◌ा+ॠ:(वृद्धिसन्धि उपसर्गादृति धातौ/वा सुप्यापिशलेः)
leftOfVP=◌ार्
rightOfVP=◌ा+ॠ
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=र्= +ॠ:(गुणसन्धि आद् गुणः)
leftOfVP=र्
rightOfVP= +ॠ
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=र्= +ॠ:(गुणसन्धि आद् गुणः)
leftOfVP=र्
rightOfVP= +ॠ
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=र्=◌ा+ॠ:(गुणसन्धि आद् गुणः)
leftOfVP=र्
rightOfVP=◌ा+ॠ
------------starting to process (गुणसन्धि आद् गुणः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=र्=◌ा+ॠ:(गुणसन्धि आद् गुणः)
leftOfVP=र्
rightOfVP=◌ा+ॠ
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=र=◌ृ+ :(यण् सन्धि इको यणचि)
leftOfVP=र
rightOfVP=◌ृ+
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=र=◌ृ+अ:(यण् सन्धि इको यणचि)
```

```
leftOfVP=र
rightOfVP=ृ+अ
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=र=ृ+ :(यण् सन्धि इको यणचि)
leftOfVP=र
rightOfVP=ृ+
------------starting to process (यण् सन्धि इको यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=र=ृ+अ:(यण् सन्धि इको यणचि)
leftOfVP=र
rightOfVP=ृ+अ
------------starting to process (वृद्धिसन्धि वृद्धिरेचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ै= +ऐ:(वृद्धिसन्धि वृद्धिरेचि)
leftOfVP=ै
rightOfVP= +ऐ
------------starting to process (वृद्धिसन्धि वृद्धिरेचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ै=ा+ऐ:(वृद्धिसन्धि वृद्धिरेचि)
leftOfVP=ै
rightOfVP=ा+ऐ
------------starting to process (वृद्धिसन्धि वृद्धिरेचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ै= +ए:(वृद्धिसन्धि वृद्धिरेचि)
leftOfVP=ै
rightOfVP= +ए
------------starting to process (वृद्धिसन्धि वृद्धिरेचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ै=ा+ए:(वृद्धिसन्धि वृद्धिरेचि)
leftOfVP=ै
rightOfVP=ा+ए
------------starting to process (वृद्धिसन्धि वृद्धिरेचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ौ= +औ:(वृद्धिसन्धि वृद्धिरेचि)
leftOfVP=ौ
```

```
rightOfVP= +औ
------------starting to process (वृद्धिसन्धि वृद्धिरेचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ौ= +ओ:(वृद्धिसन्धि वृद्धिरेचि)
leftOfVP=ौ
rightOfVP= +ओ
------------starting to process (वृद्धिसन्धि वृद्धिरेचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ौ=ा+औ:(वृद्धिसन्धि वृद्धिरेचि)
leftOfVP=ौ
rightOfVP=ा+औ
------------starting to process (वृद्धिसन्धि वृद्धिरेचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ौ=ा+ओ:(वृद्धिसन्धि वृद्धिरेचि)
leftOfVP=ौ
rightOfVP=ा+ओ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ी=ी+ई:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ी
rightOfVP=ी+ई
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ी=ी+इ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ी
rightOfVP=ी+इ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ी=ि+ई:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ी
rightOfVP=ि+ई
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ी=ि+इ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ी
rightOfVP=ि+इ
```

```
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ू=ू+ऊ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ू
rightOfVP=ू+ऊ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ू=ू+उ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ू
rightOfVP=ू+उ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ू=ु+उ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ू
rightOfVP=ु+उ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ू=ु+ऊ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ू
rightOfVP=ु+ऊ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ृ=ृ+ॠ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ृ
rightOfVP=ृ+ॠ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ृ=ृ+ऋ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ृ
rightOfVP=ृ+ऋ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ृ=ृ+ऋ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ृ
rightOfVP=ृ+ऋ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
```

```
rs =ज् अ ग् अ न् म् आ त् आ
VP=ॄ=ृ+ऋ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ॄ
rightOfVP=ृ+ऋ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ॄ=ॢ+ऋ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ॄ
rightOfVP=ॢ+ऋ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ा=ा+आ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ा
rightOfVP=ा+आ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ा=ा+अ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ा
rightOfVP=ा+अ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ा= +आ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ा
rightOfVP= +आ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ा= +अ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=ा
rightOfVP= +अ
------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=आ=आ+आ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=आ
rightOfVP=आ+आ
left token =ज् अ ग् अ न् म्
right token = त् आ
```

```
marked position =19
left token =ज् अ ग् अ न् म् आ+आ त्
right token =
marked position =26

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

tokens to validate = ज् अ ग् अ न् म् आ+आ त् आ+आ
joined tokens to validate = ज् अ ग् अ न् म् आ+आ त् आ+आ
'जगनआ त् आ' not validated
validated=false

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

------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=आ=आ+अ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=आ
rightOfVP=आ+अ
left token =ज् अ ग् अ न् म्
right token = त् आ
marked position =19
left token =ज् अ ग् अ न् म् आ+अ त्
right token =
marked position =26

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

tokens to validate = ज् अ ग् अ न् म् आ+अ त् आ+अ
joined tokens to validate = ज् अ ग् अ न् म् आ+अ त् आ+अ
'जगनआ त् आ' not validated
validated=false

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

------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=आ=अ+आ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=आ
rightOfVP=अ+आ
left token =ज् अ ग् अ न् म्
```

```
right token = त् आ
marked position =19
left token =ज् अ ग् अ न् म् अ+आ त्
right token =
marked position =26

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

tokens to validate = ज् अ ग् अ न् म् अ+आ त् अ+आ
joined tokens to validate = ज् अ ग् अ न् म् अ+आ त् अ+आ
'जगन्मआ त् अ' not validated
validated=false

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

------------starting to process (दीर्घसन्धि अकः सवर्णे दीर्घः) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=आ=अ+अ:(दीर्घसन्धि अकः सवर्णे दीर्घः)
leftOfVP=आ
rightOfVP=अ+अ
left token =ज् अ ग् अ न् म्
right token = त् आ
marked position =19
left token =ज् अ ग् अ न् म् अ+अ त्
right token =
marked position =26

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

tokens to validate = ज् अ ग् अ न् म् अ+अ त् अ+अ
joined tokens to validate = ज् अ ग् अ न् म् अ+अ त् अ+अ
'जगन्मआ त् अ' not validated
validated=false

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

------------starting to process (वा.ऋति सवर्णे ऋ ॄ वा) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ॄ=ृ+ऋ:(वा.ऋति सवर्णे ऋ ॄ वा)
leftOfVP=ॄ
rightOfVP=ृ+ऋ
```

```
------------starting to process (वा. ळृति सवर्णे ळृ वा) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=्ल्ळृ=ृ+ळृ:(वा. ळृति सवर्णे ळृ वा)
leftOfVP=्ल्ळृ
rightOfVP=ृ+ळृ
------------starting to process (यण संधि इकः यणचि) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=ये=ि+ए:(यण संधि इकः यणचि)
leftOfVP=ये
rightOfVP=ि+ए
------------starting to process (यरोऽनुनासिकेऽनुनासिको वा) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=न् न्=त्+न्:(यरोऽनुनासिकेऽनुनासिको वा)
leftOfVP=न् न्
rightOfVP=त्+न्
------------starting to process (यरोऽनुनासिकेऽनुनासिको वा) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=न् न्=द्+न्:(यरोऽनुनासिकेऽनुनासिको वा)
leftOfVP=न् न्
rightOfVP=द्+न्
------------starting to process (यरोऽनुनासिकेऽनुनासिको वा) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=न् न्=न्+न्:(यरोऽनुनासिकेऽनुनासिको वा)
leftOfVP=न् न्
rightOfVP=न्+न्
------------starting to process (यरोऽनुनासिकेऽनुनासिको वा) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=न् म्=त्+म्:(यरोऽनुनासिकेऽनुनासिको वा)
leftOfVP=न् म्
rightOfVP=त्+म्
left token =ज् अ ग् अ
right token = आ त् आ
marked position =15

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

tokens to validate = ज् अ ग् अ त्+म् आ त् आ
```

```
joined tokens to validate = ज् अ ग् अ त्+म् आ त् आ
'जगत्' validated
'माता' validated
validated=true
----------------------------END OF Segmenter.validateSplit()------------------------

जगत् माता validated
------------starting to process (यरोऽनुनासिकेऽनुनासिको वा) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=न् म्=द्+म्:(यरोऽनुनासिकेऽनुनासिको वा)
leftOfVP=न् म्
rightOfVP=द्+म्
left token =ज् अ ग् अ
right token = आ त् आ
marked position =15
----------------------------START OF Segmenter.validateSplit()------------------------

tokens to validate = ज् अ ग् अ द्+म् आ त् आ
joined tokens to validate = ज् अ ग् अ द्+म् आ त् आ
'जगद्' validated
'माता' validated
validated=true
----------------------------END OF Segmenter.validateSplit()------------------------

जगद् माता validated
------------starting to process (यरोऽनुनासिकेऽनुनासिको वा) sandhi----------
rs =ज् अ ग् अ न् म् आ त् आ
VP=न् म्=न्+म्:(यरोऽनुनासिकेऽनुनासिको वा)
leftOfVP=न् म्
rightOfVP=न्+म्
left token =ज् अ ग् अ
right token = आ त् आ
marked position =15
----------------------------START OF Segmenter.validateSplit()------------------------

tokens to validate = ज् अ ग् अ न्+म् आ त् आ
```

```
joined tokens to validate = ज् अ ग् अ न्+म् आ त् आ
'जगन्' validated
'माता' validated
validated=true

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

जगन् माता validated

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

segmented token= जगत् माता (यरोऽनुनासिकेऽनुनासिको वा)
जगद् माता (यरोऽनुनासिकेऽनुनासिको वा)
जगन् माता (यरोऽनुनासिकेऽनुनासिको वा)


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

BIBLIOGRAPHY

Bibiliography

Books

- Abhyankar, K.V., 1961, ‘*A Dictionary of Sanskrit Grammar*’, Gaekwad’s Oriental Series, Baroda.
- Bakharia, Aneesha. 2001, ‘*Java Server Pages*’, Prentice Hall of India Private Limited, New Delhi.
- Ballantyne, James R., 1967, ‘*Laghukaumudī of Varadarāja*’, Chaukhamba Sanskrit Pratisthan, MLBD. Delhi.
- Bhandarkar, R.G., 1924, ‘*First book of Sanskrit*’, Radhabai Atmaram Sagoon, Bombay.
- Bhandarkar, R.G., 1924, ‘*Second book of Sanskrit*’ Radhabai Atmaram Sagoon, Bombay.
- Bharati, Akshar, Vineet Chaitanya and Rajeev Sangal, 1995, ‘*Natural Language Processing: A Paninian Perspective*’, Prentice-Hall of India, New Delhi.
- Cardona, George, ‘*Pāṇini: his work and its traditions*’, Motilalal banarasidass, New Delhi.
- Cardona, George, 1980, ‘*Panini- A Survey of Research*’, Motilal Banarasidass, New Dalhi, First Indian Edition; First published: The Hague, 1975
- Cardona, George, 1999, ‘*Recent Research in Pāṇinīan Studies*’, Motilal Banarasidass, New Dalhi.
- Date, C.J., 1987, ‘*Introduction to Database Systems*’, Addison-Wesley, Reading.
- Fromkin & Rodman, 2003, ‘*An Introduction to Language*’, Thomson Wadsworth.
- Guru Prasad Shastri (ed). 1999, ‘*Vyakarana-Mahābhāsya with Pradipoddhat tikā*’ Pratibha Prakashan, Delhi.
- Iyer, K.A.Subramania, 1969, ‘*Bhartṛhari : A study of the vākyapadīya in the light of ancient commentaries*’ Deccan College, Poona.
- Joshi, Sivaram Dattatray & Roodbergen J. A. F., 1998, ‘*The Aṣṭādhyāyi of Pāṇini*’, Sahitya Akademi, New Delhi.

- Jurafsky, Daniel & Martin, 2005, *'Speech and languages processing',* Pearson Education Pvt. Ltd., Singapore.
- Kale, M.R., 1972, *'A Higher Sanskrit grammar'*, Motilal Banarasidas, Delhi.
- Kapoor Kapil, 2005, '*Dimensions of Pāṇini Grammar: the Indian Grammatical System',* D.K. Printworld (P) Ltd., New Delhi.
- Katre, S.M., 1968, *'Dictionary of Pāṇini',* Deccan College, Poona.
- Kielhorn, F., 1970, '*Grammar for Sanskrit Language*' Chowkhamba Sanskrit Series office, Varanasi.
- Macdonell, A. A., 1997, '*A Sanskrit Grammar for Students*', D. K. Printworld (P) Ltd., New Delhi.
- Mishra, Narayan (ed.), 1996, *'Kāśikā of Pt.Vāmana and Jayāditya',* Chaukhamba Sanskrit Sansthan, Varanasi.
- Muller, F. Max, 1983, '*A Sanskrit grammar*' Asian Educational Services, Delhi.
- Nautiyal, Chakradhar Hans, 1995, *'Brhada-anuvād-candrikā',* Motilal Banarasidass, Delhi.
- Panashikar, Vasudev Lakshman Shastri (ed), 1994, '*Siddhāntakaumudī',* Chaukhamba Sanskrit Pratisthan, Delhi.
- Pandey, G. D., (ed.), 2003, *'Vaiyākarana Siddhāntakaumudī',* Chaukhamba Surbharati Prakashana, Varanasi.
- Pt.Brahmadatta jigyasu (ed.), 1998, '*Pāṇini-Astādhyāyi*', Ramlal kapoor trust, Sonipat.
- Reyle, U. and C. Rohrer (eds.), 1988, *'Natural Language Parsing and Linguistic Theories',* D. Reidel, Dordrecht.
- Rishi, Uma Shankar Sharma (ed.); '*Yaska-pranitam niruktam',* Vol. I- chapters 17; Varanasi, Chowkhamba Vidyabhawan; reprint 2005; Vidyabhawan Sanskrit Granthamala-57
- Russell, Joseph P. 2002, '*Java Programming*', Prentice Hall of India Private Limited, New Delhi.
- Sharma, Rama Nath, 2003, '*The Astādhyāyi of Pāṇini*', Munshiram Manoharlal Publishers Pvt. Ltd., Delhi.

- Shastri Charu Deva, 1991, *'Pāṇini : Re-interpreted'* Motilal Banarasidass, Delhi.
- Singh, Jag Deva, 1991, *'Pāṇini: His Description of Sanskrit (An Analytical Study of the Aṣṭādhyāyī)'*, Munshiram Manoharlal, Delhi.
- Stall, J. F., 1985, *'A Reader on Sanskrit Grammarians'*, Motilal Banarsidas, Delhi.
- Troast, Harald. 2003, *'Morphology', in 'The Oxford Handbook of Computational Linguistics'*, Edited by Ruslan Mitkov, Oxford University press, New York, pp. 25-47.
- Vasu, Srisa Chandra (ed. & Translated into eng.), 1982, *'The Siddhānta Kaumudī of Bhaṭṭoji Dīkṣita, Vol. II*, MLBD. Delhi.
- Whitney, William Dwigth, 1983, '*Sanskrit Grammar*', MLBD, Delhi.
- Whitney, William Dwigth, 2004, '*Sanskrit Grammar: Including both the Classical Language, and the older Dialects, of Veda and Brahmana*', Munshiram Manoharlal Publishers, Delhi, Reprint from the second edition of 1889.
- Williams, Monier, '*A Practical Grammar of the Sanskrit Language*', Clarendon Press, Oxford
- Wilson, H. H., 1841, '*An Introduction to the Grammar of the Sanskrit Language*', J. Madden & Co., London.

Articles and Papers

- Bharati Akshar, Amba P. Kulkarni Vineet Chaitanya, 1996, '*Challenges in developing word analyzers for Indian languages*', Presented at Workshop on Morphology, CIEFL, Hyderabad.
- Bharati, A., Sangal R., 1990, '*A karaka based approach to parsing of Indian languages*', proc of the 13th COLING vol 3, pp 30-35, Finland.
- Bharati, Akshar and Rajeev Sangal, '*Parsing free word order languages using the Paninian framework*', In ACL93: Proc.of Annual Meeting of Association for

Computational Linguistics, Association for Computational Linguistics, New York, 1993.

- Bharati, Akshar, Vineet Chaitanya and Rajeev Sangal, '*A computational framework for Indian languages*', Technical Report TRCS-90-100, Dept. of CSE, IIT Kanpur, July 1990b. (Course Notes for Intensive Course on NLP for Linguists, Vol.1)
- Bharati, Akshara, Amba Kulkarni and V. Sheeba, *'Building a wide Coverage Sanskrit Morphological Analyzer: A Practical approach,* MSPIL-06.
- Bhate, Saroja and Subhash Kak, 1993, '*Pāṇini's Grammar and Computer Science*' Annals of the Bhandarkar Oriental Research Institute, vol. 72, pp. 79-94.
- Cardona, George, 2004, *'Some Questions on Pāṇini's Derivational system'* In SPLASH proc. of iSTRANS, pp. 3.
- G.V. Singh, Girish Nath Jha, '*Indian theory of knowledge: an AI perspective*' proc. of seminar, ASR, Melkote, Mysore, 1994
- Houben, Jan E. M., '*Panini's Grammar and its Computerization: A Construction Grammar Approach',* in Third International Symposium of Sanskrit Computational Linguistics, Springer, LNAI-5406, p.p. 6-25
- Huet, Gerard, '*Towards Computational Processing of Sanskrit*'
- Hymen, Malcolm D., 2007, '*From Paninian Sandhi to Finite State Calculus'* in proceedings of FISSCL, INRIA, Paris, p.p. 13-21
- Jha Girish N, 1994, '*Indian theory of knowledge: an AI perspective*' (proc. of national seminar on "Interface Mechanisms in Shastras and Computer Science", Academy of Sanskrit Research, Melkote, Mysore, April, 1994)

- Jha Girish N, 1995, '*Proposing a computational system for Nominal Inflectional Morphology in Sanskrit*' (Proc. of national seminar on "Reorganization of Sanskrit Shastras with a view to prepare their computational database", January, 1995)
- Jha Girish Nath, Mishra, S K, Chandrashekar R, Subash, August, 2005, '*Developing a Sanskrit Analysis System for Machine Translation*' presented, at the National Seminar on Translation Today: state and issues, Deptt. of Linguistics, University of Kerala, Trivandrum.
- Jha Girish Nath, November, 2005 '*Language Technology in India: A survey*' Issue of C.S.I. magazine
- Jha Girish Nath, October 2003, '*A Prolog Analyzer/Generator for Sanskrit Subanta Padas*', Language in India, Volume 3: 11.
- Jha Girish Nath, February 2004, '*The System of Panini*', Language in India, volume 4:2
- Jha Girish Nath, March 2004, '*Generating nominal inflectional morphology in Sanskrit*' SIMPLE 04, IIT-Kharagpur Lecture Compendium, Shyama Printing Works, Kharagpur, WB,
- Jha, Girish Nath, December, 2003, '*Current trends in Indian languages technology*', Langauge In India, Volume December.
- Jha, Girish Nath. '*Regional & linguistic perspective on internationalization: the case of Hindi/Sanskrit*', 2007.
- Jha, Girish Nath. 2007, '*Introduction to Computational Morphology*', Lecture delivered on 5 January 2007 at CDAC, Noida.

- Joshi, Shivram Dattatreya, '*Background of Ashtadhyayi*' in Third International Symposium of Sanskrit Computational Linguistics, Springer, LNAI-5406, p.p. 1-5
- Kapoor, Kapil, 1996. '*Panini's derivation system as a processing model*' (to appear in the proc. of "A Symposium on Machine Aids for Translation and Communication, 11-12 April, School of Computer & Systems Sciences, J.N.U. New Delhi, 1996)
- Kiparsky, P. and Stall, J. F., 1969, '*Syntactic and Semantic Relation in Panini*' (Foundations of Language, Vol.5, 83-117).
- Mishra, Anand, 2009, '*Modelling the Grammatical Circle of the Paninian system of Sanskrit Grammar*' in Third International Symposium of Sanskrit Computational Linguistics, Springer, LNAI-5406, p.p. 40-55
- Mishra, Anand; 2007, '*Simulating the Paninian System of Sanskrit Grammar*' in proceedings of FISSCL, INRIA, Paris, p.p. 89-95
- R.M.K. Sinha, 1989, '*A Sanskrit based Word-expert model for machine translation among Indian languages*', Proc. of workshop on Computer Processing of Asian Languages, Asian Institute of Technology, Bangkok, Thailand, Sept.26-28, 1989, pp. 82-91.12.
- Ramakrishnamacharyulu, K.V., '*Paninian Linguistics and Computational Linguistics*', Samvit, Series no. 27. Pp. 52-62, Academy of Sanskrit Research, Melkote, Karnataka (India), 1993.
- Scharf, Peter M., 2007, '*Modeling Paninian Grammar*' in proceedings of FISSCL, INRIA, Paris, p.p. 77-87

- Scharf, Peter M., 2009, '*Levels in Panini's Aṣṭādhyāyi'* in Third International Symposium of Sanskrit Computational Linguistics, Springer, LNAI-5406, p.p. 66-77.

Thesis and Dissertations

- Agrawal, Muktanand, 2007, '*Computational Identification and Analysis of Sanskrit Verb-forms of bhvādigana*', submitted for M.Phil degree at SCSS, JNU.
- Bhadra, Manji, 2007, '*Computational Analysis of Gender in Sanskrit Noun Phrases for Machine Translation*', submitted for M.Phil degree at SCSS, JNU.
- Chandra, Subash, 2006, '*Machine Recognition and Morphological Analysis of Subanta-padas*', submitted for M.Phil degree at SCSS, JNU.
- Chandrashekhara, R., 2006, '*POS Tagging for Sanskrit*', submitted for Ph.D degree at SCSS, JNU.
- Jha Girish N, 1993, '*Morphology of Sanskrit Case Affixes: A Computational analysis*' Dissertation of M.Phil submitted to Jawaharlal Nehru University, New Delhi-110067.
- Kumar, Sachin, 2007, '*Sandhi Splitter and Analyzer for Sanskrit' (with special reference to aC sandhi)*, submitted for M.Phil degree at SCSS, JNU.
- Mishra, Sudhir Kumar, 2007, '*Sanskrit Karaka Analyzer for Machine Translation*', submitted for Ph.D. degree at SCSS, JNU.
- Singh, Surjit Kumar, 2008, '*Krdanta Recognition and Processing for Sanskrit'*, submitted for M.Phil degree at SCSS, JNU.

Web References

- Academy of Sanskrit Research, Melkote, *http://www.sanskritacademy.org/About.htm* (accessed: 22 April 2009).
- Anusaaraka, *http://www.iiit.net/ltrc/Anusaaraka/anu_home.html* (accessed: 10April, 2009).
- AU-KBC Research Centre - *http://www.au-kbc.org/frameresearch.html* (accessed: 25 April 2009).
- Baraha, *http://www.baraha.com/BarahaIME.htm* (accessed: 6 July 2009).
- Brown University research http://research.brown.edu/research/ (accessed: 15 June, 2009).
- C-DAC, *http://www.cdac.in/html/ihg/activity.asp* (accessed: 20 April 2009).
- Computational Linguistic R&D, J.N.U., *http://sanskrit.jnu.ac.in/index.jsp* (accessed: 2 May 2009).
- Department of Sanskrit UOHYD >> sandhi http://sanskrit.uohyd.ernet.in/~anusaaraka/sanskrit/samsaadhanii/sandhi/index.html (accessed: 20 April, 2009).
- Desika, *http://tdil.mit.gov.in/download/Desika.htm* (accessed: 10 May, 2009).
- *http://en.wikipedia.org/wiki/Clay_Sanskrit_Library* (accessed: 2 July 2009).
- *http://tdil.mit.gov.in/languagetechnologyresourcesapril03.pdf* (accessed: 20 April 2009).
- *http://www.acroterion.ca/Morphological_Analysis.html* (accessed: 15 April 2009).
- *http://www.cfilt.iitb.ac.in/wordnet/webhwn/wn.php* (accessed: 25 April 2009).
- *http://www.comp.lancs.ac.uk/ucrel/claws* (accessed: 20 April 2009).

- *http://www.languageinindia.com/feb2004/panini.html* (accessed: 2 May 2009).
- *http://www.sas.upenn.edu/~vasur/project.html* (accessed: 18 April 2009).
- *http://www.sil.org/pckimmo/* (accessed: 18 April 2009).
- IIT, Bombay, *http://www.cse.iitb.ac.in* (accessed: 25 April 2009).
- Java Server Pages, *http://java.sun.com/products/jsp/* (accessed: 5 July, 2009).
- Java, Servlet, *http://java.sun.com/products/servlet/* (accessed: 5 July, 2009).
- John Clay, *http://www.claysanskritlibrary.org/* (accessed: 2 July 2009).
- Language Processing Tools: TDIL website, *http://tdil.mit.gov.in/nlptools/ach-nlptools.htm* (accessed: 20 April 2009).
- Oflazer,Kemal,*http://folli.loria.fr/cds/2006/courses/Oflazer.ComputationalMorphology.pdf* (accessed: 15 April 2009).
- Peter M. Scharf and Malcolm D. Hyman, *http://sanskritlibrary.org/morph/* (accessed: 18 April, 2009).
- RCILTS, School of Computer & System Science, *http://rcilts.jnu.ac.in* JNU, New Delhi. (Accessed: 20 April 2009).
- RCILTS, Utkal University, *http://www.ilts-utkal.org/nlppage.htm* (accessed: 15 April 2009).
- RSV Tirupati, *http://rsvidyapeetha.ac.in* and *http://www.sansknet.org* (accessed: 22 April 2009).
- Sanskrit heritage site >> Sanskrit Reader http://sanskrit.inria.fr/DICO/reader.html (accessed: 10 April, 2009).
- The Web server, Apache Tomcat, *http://www.apache.org/* (accessed: 5 July 2009).
- Wikipedia, *http://en.wikipedia.org/wiki/Natural_language_processing* (accessed: 15 April 2009).