Building a Wide Coverage Sanskrit Morphological Analyzer: A Practical Approach

Akshar Bharati, Amba Kulkarni, V Sheeba
Rashtriya Sanskrit Vidyapeetha
(Deemed University)
Tirupati
{ambapradeep,v.sheeba}@gmail.com

Abstract

For an inflectionally rich language like Sanskrit, any NLP application demands a good morphological analyzer. Though Sanskrit is the best-analyzed language in the world, a good coverage morphological analyzer for it is still not available. This paper points out the complexity involved in building a wide coverage analyzer for Sanskrit and then describes a morphological analyzer that has been built using the available e-resources, based on ad-hoc principles. The coverage of this analyzer is around 95%. Though for practical applications, this is not an acceptable figure, it can however be used as a stepping-stone to develop other modules such as *sandhi* splitter, search engine, etc. At a later stage, it may be replaced by a module that is based on the classic *aṣṭādhyāyī*.

1 Introduction

Morphological analyzer is the basic tool needed for any NLP applications ranging from information retrieval, search engines, spell checkers to MT systems. A morphological analyzer takes a word (a string separated by white spaces) as an input and produces its analysis, showing the root and grammatical features such as gender, number, person, tense, aspect, modality, etc. The complexity of rules for word formation differs from language to language.

In the past decade wide coverage morphological analyzers for different languages were developed (Karp, 1992; IIIT-H; Vishvanathan S, 2003). English has the simplest morphology and hence though 2 level computational models for word recognition and production were suggested (koskennieni, 1983), a simple hash table lookup method was used to build a wide coverage analyzer (Karp, 1992).

Among modern Indian languages, Hindi has the simplest morphology and the complexity increases as we move from North to South India, with Dravidian languages having the most complex morphology. Morphological analyzers for different Indian languages were developed based on the paradigm model (Bharati Akshar et al 1995, IIIT-H). A wide coverage morphological analyzer for Tamil was developed using FST and the paradigm model (Vishvanathan et al., 2003).

Sanskrit has the richest morphology. At the same time it is the best analyzed language of all the languages in the world. Since well defined and almost exhaustive rules for word formation of Sanskrit language exist, one may think it to be a trivial task to build a morphological analyzer based on these rules.

Attempts have been made to develop morphological analyzer for Sanskrit (CDAC; ASR; Huet, 2003). However either they have limited coverage or are not available freely, making it almost unusable for any serious NLP applications. Another problem is, though well defined rules for Sanskrit morphology exist, for a typical computer science person, with no knowledge of Sanskrit, it is difficult to build a system incorporating them. At the same time, it is very rare to get a person who understands the *aṣṭādhyāyī* well and also have good knowledge of computer science. Therefore, incorporating the rules in *aṣṭādhyāyī* as it is, is a difficult task, and may take time to develop such a system. To implement the rules given in *aṣṭādhyāyī* as they are is further difficult as there are also controversies over the use of *paribhāṣā* (meta rules) to interpret the *aṣṭādhyāyī*. It is possible that it may give rise to more than one implementations. If a practical system to analyze words exists, then such a system can be used to test the performance of different systems based on different hypothesis mechanically.

If we look at the Sanskrit literature, we see that there have been many attempts to make the learning of Sanskrit word formation easier. The method adopted was a paradigm based approach where a student is taught the word forms of a common word e.g. *deva* in Sanskrit and further that it is the default paradigm for 'a' ending masculine words. Further the list of exceptional words and the forms where they differ are taught separately. Following this method, a simple algorithm was developed which is described in (Bharati Akshar et al., 1995). This algorithm has been used to develop morph analyzer for different Indian languages (IIIT-H).

2 Earlier Work

With an initiative from IIIT-H, and initial working morph analyzer developed at ASR Melkote (ASR), using lexicon from Monier William's dictionary (Monier, 1899), first morphological analyzer based on the paradigm approach, was developed(Jain, 2004). This analyzer handled only nouns. This analyzer though had a lexicon of around 0.1M *prātipadikas*, extracted mainly from the Monier-William's dictionary, had errors since the lexicon was extracted by a student who did not have much knowledge of Sanskrit.

With this morphological analyzer, as a starting point, authors have taken it further to make it a full-fledged analyzer with a wide coverage. Further this is also being made available under GPL so as to avoid any duplication of work in future (anusārakā).

In the following sections, we first discuss the complexity of word formation in Sanskrit. Section four gives a brief outline of available relevant e-resources. In the fifth section we discuss the practical approach followed to make the best use of available e-resources and develop a practical system with wide coverage. Sixth section shows the results on randomly selected texts.

3 Word Formation in Sanskrit :

The finite state automaton in fig 1 describes the word formation in Sanskrit. As can be seen from the figure, theoretically it is possible to generate infinite forms from a given word. However, the ability of human mind to process a complex string puts a limit on these potentially infinite productions to a finite number and is supported by the actual data.

In the first *kānda* of rāmayayana, the longest derived word had 3 suffixes, (e.g. a *kridanta*, followed by a san suffix, followed by sup suffix). Further, for all practical purposes, just as a human being also does, the intermediate *prātipadikas*, or *dhātus* can always be kept in the lexicon.

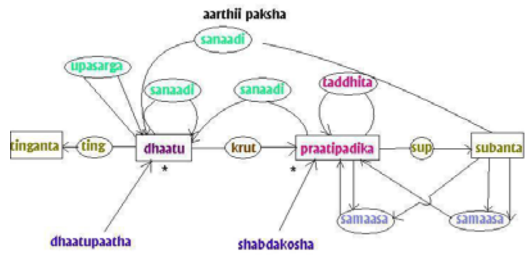

Fig 1 Word Formation in Sanskrit

The complexity in Sanskrit morphology is further aggravated because of
-- *sandhi* formation, and
-- productive *samāsa* formation.

3.1 Sandhi formation

Unlike modern Indian languages, Sanskrit uses *sandhi* extensively. *Sandhi* may be divided into two categories – external *sandhi* (rules which govern the *sandhi* making of two padas) and internal *sandhi* (rules which govern the *sandhi* within two segments of a pada). The internal *sandhi* rules are used at the morphological level. But the external *sandhi* needs to be handled separately. Sanskrit has around 50 alphabets, leading to approximately 2,500 possible combinations of two alphabets. Out of these more than 60% involve morphophonemic changes during *sandhi* formation. This naturally gives rise to multiple answers during *sandhi* splitting. For example, there are 4 possibilities a character '*ā*' can be split into:

ā -> a + a
-> a + *ā*
-> *ā* + a
-> *ā* + *ā*

and hence the word 'rāmālaya' can be split into 2 words in 8 possible ways viz:

a) ra + amālaya
b) ra + āmālaya
c) rā + amālaya
d) rā + āmālaya
e) rāma + alaya
f) rāma + ālaya
g) rāmā + alaya
h) rāmā + ālaya

A good coverage morphological analyzer can rule out half of the possibilities, leaving the last four. However to rule out these possibilities, one needs to look at the context. This second task requires a capability in machine to use the world knowledge and current technology still has limitations to handle the world knowledge.

Thus we see that a good coverage morphological analyzer needs a *sandhi* splitter and a *sandhi* splitter requires a good coverage morphological analyzer leading to a deadlock situation.

3.2 *Samāsa* formation

Another feature that increases the complexity of Sanskrit word formation is productive *samāsa* formation. The fig 2 shows different *samāsa* formations in Sanskrit.

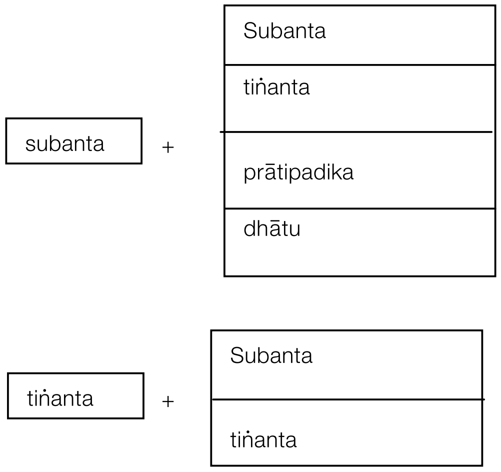

Fig 2

In case of *samāsa*'s the sandhi is a must. Two texts viz. one prose (pancatantra) and one poetry (first *kānda* of *rāmāyayana*) were selected for analysis. It was found that around 20 to 25% of the words in these texts were *samāsa*. It is not possible to list all possible *samāsa* in the dictionary.

Thus to have a good coverage morph analyzer, again one needs a *samāsa* handler, and in *samāsa* formation *sandhi* is a must. This leads to circular dependability.

To break this circularity it was decided to pre-edit the text manually. The pre-edited text will have sandhi-splitted words and the *samāsa* tagged for their components and the type. With this pre-editing, then the problem reduces to analyzing simple padas by machine.

4 Available e-resources

Sanskrit is not only well analyzed but also contains a vast ancillary material in the form of data bases and dictionaries. Dhāturatnākara, kridantarūpamālā, vācaspatyam, rūpachandrikā, śabdakalpadruma and various monolingual and bilingual dictionaries are some of the examples of such data-bases. These data-bases are meant for humans with some exposure to grammar to assist them in understanding texts in different subjects like Ayurveda, literature, astronomy, etc.

Dhāturatnākara kosha lists the verb forms under differnt *lakāras* for all the verbs in the dhātupātha. Kridantarūpamālā gives a list of high frequency *kridanta* forms for all the verbs. **Vācaspatyam** contains derivation of different derived roots along with citations. Rūpacandrikā gives a list of different default and exceptional paradigms for all the three genders covering whole range of nouns and pronouns in Sanskrit. All these books are available in e-forms, under Sansk-Net project (SANSK-NET). However, since these books are written for humans, there are errors in the formatting of the text, which typically go unnoticed by humans (see Appendix A). However, even a small error of comma produces wrong results when processed by a machine. Further, the entries in the book were not uniform, they are not in the same order, and also sometimes, it just carries a note that entries behave like some other word X. This makes the task of extracting different forms automatically little difficult. However, with a language like Perl that has powerful support for regular expressions, the task is doable.

5 Design of morph analyzer

Separate modules have been developed to handle *subanta*, *tiṅanta* and *kridanta* words. In what follows we first describe different modules in morphological analyzer followed by an algorithm, which integrates all these modules.

5.1 *Subanta* analyzer

Subanta handler analyses nouns, pronouns, adjectives, and indeclinables. Nouns, pronouns and adjectives decline according to number and *vibhakti* in Sanskrit. Adjectives in Sanskrit, unlike English, undergo change in their roots according to gender. The *subanta* handler follows the same algorithm as described in, (Bharati Akshar et al., 1995) with a small modification in the algorithm to factor out common information about the features, so as to keep the hash table size small. The paradigms for this approach have been taken from rūpacandrikā.

A list of lexicon is extracted from Monier William's dictionary and with a simple Perl program, paradigms have been assigned automatically to each of the lexical head, based on its ending vowel and gender. Some exceptional cases need to be handled while assigning a paradigm. Words with *ra* in the *prātipadika* needed special paradigms, since in this case the *na* in suffixes changes *ṇa*. Separate paradigms are assigned to these cases. In Sanskrit adjectives also decline like nouns. Further the root forms of adjectives change according to gender. Monier William's dictionary has around 40,000 adjective entries. The head words corresponding to all these entries are for masculine/neuter gender. It also contains the information about changes required to form the feminine root form. Sanskrit grammar has well defined rules to add the feminine suffixes. Using these rules, and the information in the dictionary, feminine forms of adjectives were formed automatically followed by random checking for their correctness. A separate list of indeclinables is added to the lexicon list, to handle *avyayas*.

With a total of 222 paradigms for nouns and pronouns and 24 forms each (8 *vibhaktis* and 3 numbers) and a root dictionary of around 1,53,294 words extracted from Monier William's dictionary and vācaspatyam, the current *subanta* handler can analyze 36,79,056 word forms.

5.2 *Tiṅanta* analyzer

The *dhāturatnākara* contains different finite forms for all the verb roots in the dhātu pātha. This dictionary being meant for the human beings, all the verb forms varying with different number and person are not listed for all the verbs. So writing simple Perl programs, different forms were generated and checked manually. This database currently contains 10,13,570 verb forms corresponding to 10 *lakāras*, 3 numbers, 3 persons, *kartari*, *karmaṇi*, *Nic* and yaṅluganta forms of around 2000 verb roots and around 900 *nāmadhātus*. This data-base is used in the form of hash table for giving the analysis of verbal forms.

In case of Sanskrit, one or more *upasargas* (verbal prefixes) get added to the verb roots to form new verb roots. The addition of *upasarga* typically involves morphophonemic changes in the root form following the *sandhi* rules. The Dhāturatnākara contains only the forms corresponding to basic roots. The number of commonly used *upasarga* combinations is around 70.

One way to handle verb forms with *upasarga* is to generate all the forms corresponding to these *upasargas* also. Another possibility is to guess the *upasarga* and split. Third alternative is to manually split the *upasarga* till a good *upasarga* guesser is built. Currently we are following the third approach.

5.3 Kridanta analyzer

Kṛtadantarūpamālā contains a good database of commonly occuring *kridantas*. Around 135 kridantas can be formed corresponding to each verb of which around 20 are commonly occurring. Further some of the *kridantas* can take noun inflections while others are indeclinables. As experienced with other texts, kridantarūpamālā also being written for human usage, does not list all the forms for all the verbs uniformly. Sometimes the order is also different. Sometimes it just carries a note saying that the forms are similar to such and such verb. So the book, even if is available in e-form, is not readily usable. Special programs using the regular expression power of Perl are written by the authors to extract the information from this book and further this data is linked with already existing database of dhātupātha. Number of forms extracted from the book is around 42,457.

Further since some of the kridanta's also take nominal suffixes, and the feminine forms being typically different from their masculine/neuter gender counterparts, a special program was written to generate the feminine forms and assign default noun paradigms to all these kridanta pratipadikas. Just as in the case of *tiṅantas*, here also *upasargas* are treated separately. However, *upasarga* in case of a particular *kridanta* exhibit a typical behaviour. A verb with upasarga takes '*lyap*' form whereas without upasargas take '*ktvā*' form. e.g. *āgatya, gatvā*. A special module takes care of this part.

kṛtadanta analyzer algorithm

```
1. Get the word (where upasargasare split by
manually).
2. Remove the upasarga and check the word
     in ' kṛtadanta' dictionary.
   If present in the dictionary
     if kṛtadanta = lyap
       if upasarga is present
          produce the answer.
       else report an error message
```

```
 else produce the answer
else
 check the word for noun inflection.
 If noun inflection is present,
         if root is a kridanta
                 produce answer
         else produce an error
 else produce an error
```

5.4 *Samāsa* analyzer

From word formation point of view, samāsas in Sanskrit may be classified into 6 different categories (see fig 2).

Of these 6 combinations, only two are productive. The analysis of Ramayana's first kānda shows the following results:

subanta-subanta	1863
subanta- prātipadikas	763

Table 2: samāsa distribution in first *kānda.*

Instances of other samāsa are not found in the first *kānda*.

Samāsa is conjoining of more than one words giving rise to a single word. It poses two problems from analysis point of view. At morphological level, only the last component of the *samāsa* undergoes inflection and hence contains the gender, number, *vibhakti* information. The other components undergo morphophonemic changes as per the sandhi rules. For example, in case of '*rājapurushaH*', the constituents are *rājan* and *purusha*. '*rājan*' has undergone a change to become '*rāja*'. The *prātipadika* '*rājan*' will be available in the lexicon, but not '*rāja*'. While splitting the *samāsa* into constituents, a typical student of Sanskrit may not know the actual *prātipadika*. For him it is easier to split it as '*rāja-purushaH*' rather than '*rājan-puruushaH*'. To account for this, a lexicon of 'samāsa-*pūrva pada*'s is created which contains a list of words such as '*rāja*' which occur as constituents of *samāsa*.

The other problem is typical of *bahuvrīhi* compounds. In case of *bahuvrīhi*, the last component of the compound takes the gender of the referent. For example, '*ānanam*' though is neuter gender, in case of '*vikṛta-ānanām*' it is feminine. To account for this type of *bahuvrīhi* compounds, a list of commonly occurring components of *bahuvrīhi* with their pratipadika forms in other genders are listed in the '*samāsa-uttara pada*' list.

It should be noted that this *samāsa* analyzer just gives the morphological analysis of the padas involved in the *samāsas*, and does not declare the type of the *samāsa*. The latter part requires a world knowledge and also sometimes context.

As one can see, these are just temporary and ad-hoc solutions. Better solutions need to be worked out. Till then these ad-hoc solutions help in understanding the complexities involved.

***Samāsa* handler algorithm**

1. Split the given word into different constituent members.
2. Ensure that each constituent member except the last one is either in the lexicon database or is a kridanta or a *tiṅanta* or in the list of samāsa puurvapada list, or has a valid morphological analysis.
3. Check whether the morph analysis of the last word exists. For this use the normal lexicon database as well as the samāsa uttarapada list.
4. If both 2 and 3 are satisfied, produce the answer.

5.5 Overall Algorithm

Each word is filtered through all the four handlers, viz. subanta handler, *tiṅanta* handler, kridanta handler and the samāsa handler, and all possible answers are produced.

Advantages of this approach

In this approach, the programs are independent of the data. Hence, a linguist can handle the morphological analyzer with ease. He can update the data without disturbing the programs. Secondly, since different modules handle different categories of words, different people can work simultaneously without affecting each other's work. More modules to handle other derivational suffixes such as *taddhitas* can be plugged in when they are ready.

6 Results

The following table shows size of the data bases in the current implementation of the morphological analyzer.

Noun/Adjective lexicon	153294

Finite verb forms	1013510
Non finite verb forms	42457
Samāsapūrva pada	275
Samāsauttara pada	300

Table 3: Current Data Size

This analyzer was developed and improved to handle the words from the first kānda of rāmayana(R). Naturally the performance on the rāmāyana text is good. Same analyzer is also tested on pancatantra(P) text, sample from harṣa carita(H) and elementary Sanskrit reader(E). The analysis is shown below.

	R	P	H	E
Total words	21270	498	231	240
Unrecog-nised words	500 (2.5%)	28(6%)	7(4%)	0

Table 4: Results

Analysis of unrecognized words

Around 50% of the unrecognized words were typos or cases of splitting errors. Remaining were the cases of missing lexicons. In rāmayana around 15% of the unrecognized words were 'non-pāninian' usages.

7 Conclusion

This paper describes a practical approach to build a morph analyzer. Though it does not give any linguistic insight, it provides a practical tool, which can be used to build other computational resources such as sandhi splitter, spell checker etc. to analyze the Sanskrit text.

The analyzer being modular, it is easy to add new modules to it. For example, the current analyzer does not have a module to handle the 'taddhita' suffixes that produce derived nouns. There are quite a few *taddhita* suffixes that are very productive, and are in common usage, such as '*tasil*', '*matup*', etc.

Further, the current analyzer, separates the data from programs, making it easy to update the database without changing the programs. Any language person with good knowledge of Sanskrit grammar can handle this data with ease. A program to assign the paradigms automatically also makes the updation/addition of lexicon further.

The current coverage of the morphological analyzer on unknown text is around 95%. Further, as is observed during the analysis of different unrecognized words, mainly the words were not recognized because the *prātipadika* was not available in the lexicon. It is very straight-forward to add the *prātipadika*s in the lexicon. So if this morphological analyzer is put to actual use, incrementally its performance can be improved further. This system may also be used to mechanically test different interpretations of *aṣṭādhyāyī* leading to different implementations.

Acknowledgement

Authors thank Prof. K. V. Ramkrishnamacharyulu, Vice chancellor of Shri Ramananda Rajasthan Sanskrit University for giving useful suggestions in handling *samāsās*. Thanks are also due to Sri. Srinivas Varakhedi and the students of M.A. *śābdabōdha* course for supplying paradigms of numerals, and providing feminine forms of adjectives.

References

ANUSAARAKA, http://ltrc.iiit.net/~anusāraka

ASR Melkote, http://www.sanskritacademy.org/

Bharati Akshar, Vineet Chaitanya, Rajeev Sangal. 1995, *Natural Language Processing: A Paninian Perspective,* Prentice-Hall of India (Chapter 3)

CDAC-B http://www.cdac.in/html/ihg/ihgidx.asp

Huet Gerard, 2003, *Towards computationsl Processing of Sanskrit*, Proceedings of ICON 2003

Huet, http://sanskrit.inria.fr/

Jain Vinish 2004, *Sanskrit-English Anus¡raka: Morphological Analyzer and Dictionary Component, IIIT-Hyderabad*

Karp Daniel, Yves Schabes, Martin Zaidel, Dania Egedi, 1992*, 'A Freely Available Wide Coverage Morphological Analyser for English', Proc of Coling, 1992.*

Koskennieni kimmo, 1983, Two level morphology: a general computational model for word-form recognition and production, Technical report, University of Helsinki, Helsinki,Finland

Monier 1899: Sanskrit-English Dictionary, Oxford.

Muni Lāvanya Vijaya Suri, 1867, *Dhaturatn¡kar¡H,* Navarang Booksellers and publishers, New Delhi.

RSVP, http://rsvidyapeetha.ac.in/~anus¡raka

SANSK-NET, http://www.sansk-net.org

Vasu Srisa chandra , 1962, *The aṣṭādhyāyī of Pānini,* Motilal Banarasi Das, Delhi

Vishvanathan S, Ramesh Kumar S, Kumar Shanmugham B, Arul Mozi S, Vijay Shankar K, 2003, *A Tamil morphological Analyzer*, ICON 2003

Appendix A

Sample page from kridanta ruupamaalaa

<ējṛ dīptau--1-s-a.>
(131) "ējṛ dīptau" (I--bhvādiḥ--179. aka. sēṭ. ātma.)
`dīptau śapyējatē tatra, bhavēdējati kampanē .." (ślō. 58) iti dēvaḥ .
ējakaḥ--jikā, ējakaḥ--jikā, ējijiṣakaḥ--ṣikā ;
ējitā--trī, ējayitā--trī, ējijiṣitā--trī ;
--- ējayan--ntī, ējayiṣyan--ntī--tī ;
ējamānaḥ, ējayamānaḥ, ējijiṣamāṇaḥ ;
ējiṣyamāṇaḥ, ējayiṣyamāṇaḥ, ējijiṣiṣyamāṇaḥ ;
{1}ēk_--ēg_--ējau--ējaḥ ; {1.`cōḥ kuḥ' (8-2-30) iti kutvam .} --- ---
{A}ējitaḥ--tam,
{A."amuñcanairmañcitacittamañcitatrayīmatapra stucitaiḥ śubhārjakaiḥ . samṛñjitaṃ kandamabhṛktasatphalānyadadbhirēkatra yadējitaṃ janaiḥ .." dhā. kā. 1-24.} ējitaḥ--tam, ējijiṣitaḥ--tavān ;

Appendix B

Sample entries in kridanta database corresponding to the entries shown in Appendix A:

"ējaka","ējṛ","1","ṇvul","0","noun_m"

"ējaka","ējṛ","1","ṇvul","0","noun_n"

"ējikā","ējṛ","1","ṇvul","0","noun_f"

"ējaka","ējṛ","1","ṇvul","ṇic","noun_m"

"ējaka","ējṛ","1","ṇvul","ṇic","noun_n"

"ējikā","ējṛ","1","ṇvul","0","noun_f"

"ējijiṣaka","ējṛ","1","ṇvul","san","noun_m"

"ējijiṣaka","ējṛ","1","ṇvul","san","noun_n"

"ējijiṣikā","ējṛ","1","ṇvul","san","noun_f"

"ējitṛ","ējṛ","1","śatṛ","ṇic","noun_m"

"ējitṛ","ējṛ","1","śatṛ","0","noun_n"

"ējitrī","ējṛ","1","śatṛ","0","noun_f"

"ējayitṛ","ējṛ","1","śatṛ","ṇic","noun_m"

"ējayitṛ","ējṛ","1","śatṛ","ṇic","noun_n"

"ējayitrī","ējṛ","1","śatṛ","ṇic","noun_f"

"ējanīya","ējṛ","1","anīya","0","noun_m"

"ējanīya","ējṛ","1","anīya","0","noun_n"

"ējanīyā","ējṛ","1","anīya","0","noun_f"