SanskritTagger : a stochastic lexical and pos tagger for Sanskrit

Oliver Hellwig

SanskritTagger, a stochastic lexical and POS tagger for Sanskrit

Oliver Hellwig

Abstract

SanskritTagger is a stochastic tagger for unpreprocessed Sanskrit text. The tagger tokenises text with a Markov model and performs part-of-speech tagging with a Hidden Markov model. Parameters for these processes are estimated from a manually annotated corpus of currently about 1.500.000 words. The article sketches the tagging process, reports the results of tagging a few short passages of Sanskrit text and describes further improvements of the program.

The article describes design and function of SanskritTagger, a tokeniser and part-of-speech (POS) tagger, which analyses "natural", i.e. unannotated Sanskrit text by repeated application of stochastic models. This tagger has been developped during the last few years as part of a larger project for digitalisation of Sanskrit texts (cmp. (Hellwig, 2002)) and is still in the state of steady improvement. The article is organised as follows: Section 1 gives a short overview about linguistic problems found in Sanskrit texts which influenced the design of the tagger. Section 2 describes the actual implementation of the tagger. In section 3, the performance of the tagger is evaluated on short passages of text from different thematic areas. In addition, this section describes possible improvements in future versions.

1 Introduction

Concerning its analytical abilities, SanskritTagger is located quite at the bottom of a hierarchy of taggers. The tagger neither constructs a complete nor a partial syntactical analysis of a Sanskrit text. Instead, it only identifies the most probable lexical resolution for a given group of strings (tokenisation) and their most probable part-of-speech (POS) tags. In comparison with taggers for some European languages, this result might not seem very noteworthy. In fact, the limited abilities of this tagger are caused by the difficulties which Sanskrit poses to any tagging process especially during tokenisation and which are not encountered – at least in that degree – in the processing of European languages.

On a low phonological level, the euphonic rules called ***saṃdhi*** are certainly a serious obstacle to an easy tokenisation of Sanskrit text. While these regular phonological transformations can be resolved with automata (Huet, 2007) or by using a simple lookup strategy (see below, 2.2), they introduce a great deal of ambiguity in any analysis of Sanskrit text. Consider for example a long string where three points for *saṃdhi* splitting can be identified. Each of these *saṃdhi*s may be resolved in three different ways. Even in this simple example, $3 \cdot 3 \cdot 3 = 27$ new strings are generated by the complete resolution at the three splitting points.

The high number of candidate strings which must be checked for validity after *saṃdhi* resolution, leads directly to a group of connected phenomena which are in my opinion the central challenge for any automatic processing of Sanskrit: The extreme **morphological and lexical richness** of Sanskrit. Even if a moderately sized dictionary as in SanskritTagger is used, there exist about five million distinct inflected nominal and verbal forms which may be found in any text. (English, a language with a large vocabulary, has about one half of this number!) On one hand, opposite to languages as German and English, the rich morphology clarifies the functions of words in a phrase and therefore makes POS tagging (and parsing) easier. On the other hand, it is responsible for many analyses which are in fact just nonsensical (e.g. *āsane* ⇒ *ā - sane*, "to - in the gain"). These problems are aggravated by the peculiarities of Sanskrit lexicography. The first important lexical phenomenon is the low **text coverage** of Sanskrit vocabulary. Compare the following figures for English texts (taken from (Waring and Nation, 1997)) and for the Sanskrit corpus which I collected during the last years:

Vocabulary size	Text coverage	
	English	Sanskrit
1000	72.0	60.8
2000	79.7	70.3
3000	84.0	75.2
4000	86.8	78.2
5000	88.7	80.3
6000	89.9	81.9

Although the English corpus is certainly better balanced than the Sanskrit corpus – meaning that texts from more diverse sources are included, what should actually lead to a decrease of text coverage –, the values for text coverage are clearly higher for English than for Sanskrit. Therefore, for tokenising even a simple Sanskrit text, a tagger must take into account a considerably higher number of lexemes than in other languages. This fact excludes to some extent "easy solutions" as reduced vocabularies which proved useful in tagging (technical) texts in other languages.

Besides, Sanskrit has a great number of **homonyms**. A query in the program dictionary which took into account only homonymous words with the same grammatical category resulted in the following figures:[1]

nr. of homonyms	frequency
2	1949
3	112
more than 3	17

Among these homonyms, a lot of words with high frequency can be found as for example *kesara*, m. masc. with the four basic meanings "mane", "(lotus) fibre", "(a plant name, prob.) Mesua ferrea L." and (infrequent) "name of a mountain" (but see LiPur, 1, 72, 7 for a reference).

A further, often neglected difficulty is the almost total **lack of punctuation marks** in Sanskrit texts. Apart from *daṇḍa*s in narrative texts, which often mark the end of a (complex) narrative substructure, Sanskrit texts do not use any kind of reliable punctuation. *daṇḍa*s in metrical texts actually mark the end of a verse which often, but by far not regularly, coincides with the end of a syntactic structure as for example a subordinate clause. So, *daṇḍa*s may be helpful in generating hypotheses about the syntactic structure of a text, but can not be considered as punctuation marks in a strict sense. This lack has a far reaching effect on any tagging or parsing process applied to a Sanskrit text, because it can not be guaranteed that all words necessary for a complete analysis are really contained in the text delimited by these marks. Manual preprocessing of the text (e.g. insertion of a clear punctuation) can counteract this phenomenon, but certainly is contrary to the notion of natural, i.e. unpreprocessed text.

Finally, to understand Sanskrit texts correctly, it is often necessary to supplement a great deal of **implicit knowledge**. This situation occurs in two closely related areas. Firstly, texts which openly simulate speech acts as for example dialogues often necessitate the addition of central parts of speech as subject or objects. This phenomenon, which is also known from texts in other languages, is a still puzzling, but intensively studied topic in computational linguistics. Secondly, especially scientific texts in Sanskrit as commentaries or *sūtra*s frequently use a kind of prose which imitates an oral controversy between the proposers of differing opinions. Although this kind of prose is probably derived from real discussions, it uses a highly formalised language (for a description see e.g. (Hartmann, 1955)). "Sentences" in this language frequently only offer few pieces of information which must be inserted in an implicit "knowlegde frame" to be supplied by the reader. Consider for example the discussion of *sāpiṇḍya* in the Parāśarasmṛtiṭīkā (on Parāśaradharmasaṃhitā, *Ācārakāṇḍa*, 2, 15; (I.V. Vāmanaśarmā, 1893), 59). After the author has proposed the standard model of this kind of relation, which includes three generations into past and future starting from the *yajamāna*, an opponent objects that brother, uncle etc. of the *yajamāna* are not included in this model and therefore not related to the *yajamāna* by *sāpiṇḍya*. In the following reply of the author, information which is really supplied in the text is printed in bold characters:

> *maivam*
> This is **not like this**!
> *uddeśyadevataikyena kriyaikyasyātra vivakṣitatvāt*
> Brother, uncle etc. are included in this model because **the identity of the ritual is expressed by the identity of the gods invoked**.

I am not arguing that these phrases are not well formed. Nevertheless, their syntax and pragmatics can only be analysed correctly, after the pragmatics of the surrounding text has been analysed and "understood" by the computer. The same holds true for phenomena such as anaphora resolution. Actually, this is a task which in my opinion is by far too difficult for any automatical analysis of Sanskrit currently available.

[1] This is actually a quite strong constraint, as for example nouns of categories "a masc." and "a neutr.", which have the same unchangeable stem, differ only in few forms. Including these pseudo-homonymous words would increase the rate of homonyms up to ten percent of the total vocabulary.

2 Implementation of the tagger

To keep data and algorithms clearly separated, language specific information is stored in a *database*, while the *tagging routines* are implemented in `C++` with heavy use of `STL` classes. The following section describes these two central components of the tagging software.

2.1 The database

The first main component of the program, a relational database, which can be queried via `SQL`, stores dictionary, grammatical information, and a text corpus. The original dictionary was based on the digitalised version of Monier-Williams which was parsed with regular expressions to extract lexemes, meanings and grammatical categories. These information types were stored in separate tables in the database. During the last few years, the dictionary has been extended especially in the areas of *Āyurveda* and religious philosophy. It currently contains about 178.000 lexemes (172.000 nouns and 6.000 verbs) with about 185.000 associated grammatical categories and about 325.000 meanings.

An important issue in the processing of strongly inflectional languages as Sanskrit is the correct recognition of inflected forms. Here, I chose a twofold strategy. Inflected **nominal forms** are not stored in the database, but are recognised on the fly during the tagging process. For this task, all possible endings for any nominal grammatical category are stored in a separate table. During tagging, the last few letters of a given string are compared with these endings. If an ending matches the last letters of the string, the dictionary is searched for the first part of the string in the respective grammatical category. If a matching lexeme could be found in the dictionary, a new candidate is added to the set of possible solutions. Computationally, this approach speeds up the tagging process, because the lookup of the last few letters of a given string in an efficiently organised small set of endings is much less time consuming than a query from a database of over four millions inflected forms. Though the difference in duration only amounts to a few milliseconds per operation, the performance loss sums up to several seconds for a phrase of moderate size. Figure 1 sketches an example for this approach.

On the contrary, inflected *verbal forms* are stored in the database. Currently, the database contains about 440.000 inflected verbal forms including forms derived from prefixed verbs. The decision to store the full verbal forms was not only motivated by the comparatively small number of forms, but also by the frequent irregularities and exceptions in the verbal system of Sanskrit. Of course, it is possible to construct automata which generate and accept correct verbal forms at runtime. Nevertheless, it seems to me that such an approach requires more effort than the design of a simple algorithm which generates correct verbal forms of the most typical grammatical classes and leaves the rest of the job (including correction of errors and input of rare or special forms) to the user. As in the case of nominal forms, the last few letters of possible verbs are checked before the database is queried.

*sanābhy**āṃ***

> **āṃ** = acc. sg. of declension type **ā fem.**
> dictionary lookup for (*sanābh*, **ā fem.**)
> success ⇒ 1st candidate

*sanābh**yāṃ***

> **yāṃ** = loc. sg. of declension type **i adj.**
> dictionary lookup for (*sanābh*, **i adj.**)
> success ⇒ 2nd candidate

...

Figure 1: Example for the analysis of nominal forms

Apart from the lexical and grammatical information, the database also stores a corpus of analysed Sanskrit texts which is of central importance to the tagging process. In short, every separable string of an input text is stored as a separate item in this corpus. After tagging this text, each of its strings is connected with an ordered list of references to grammatically annotated nouns from the dictionary and/or verbal forms. For example, the (fancy) string *gacchatīśvarātmamāyayā* is analysed and stored as sketched in figure 2. The figure makes clear the first important area of application of this corpus: Every string is resolved into lexemes or tokens, which are connected to the dictionary. The dictionary points in turn to further information about these lexemes as meanings etc. Of course, these relations may be inverted. Therefore, the corpus may be used to retrieve Sanskrit words efficiently in large amounts of text. Examples are the retrieval by lexeme ("Show all references in text X for the word Y!"), by meaning ("Show all references of words which have the meaning X!") or even by semantic concepts. Besides, the corpus is used to estimate the statistical parameters for the tagging process (see below). – In the moment, the corpus contains about 1.560.000 strings which are resolved into about 2.190.000 tokens. Each of these tokens is connected with a lexeme, and about 90 percent of them also have a POS annotation. Among others, the corpus includes the full analysed text of the Rāmāyaṇa, the first books of the Mahābhārata, some works of *dharma-* (e.g. Manusmṛti) and Purāṇa tradition, philosophical literature of Śaivism and many works on *Āyurveda*

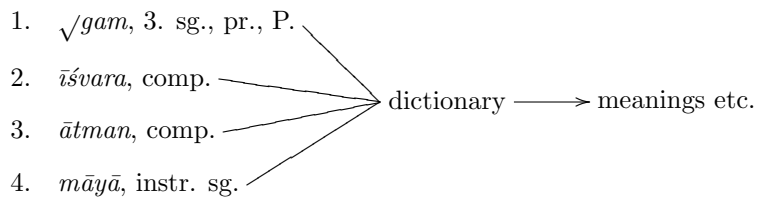

Figure 2: Analysis and storage of *gacchatīśvarātmamāyayā*

and *Rasaśāstra* (alchemy).

2.2 The tagging algorithm

The tagging software is divided into two main modules. In the first module, hypotheses about the analysis of a phrase are generated with the help of *saṃdhi* resolution and dictionary lookup. In the context of Sanskrit, a phrase does not mean a complete, self-contained syntactic structure, which may e.g. be extracted from a text with regular punctuation (see 1). Instead, a phrase $\mathcal{P}$ is a group of strings, i.e. words separated by blanks, which is terminated by a (double) *daṇḍa*. Such a group may, but needs not necessarily coincide with a complete syntactic structure. The hypotheses resulting from this first analysis are organised in an often complex tree- or rather forest-like structure. The purpose of the second module is to find the most probable lexical and morphological path through this structure given the statistical information extracted from the corpus. This path will constitute the final analysis of the phrase.

A string $\mathcal{S}_i \in \mathcal{P}$ of length L is parsed from left to right. At each position j with $1 \leq j \leq L$, a maximal number of n_{max} letters of the string are searched in a trie $\mathcal{T}$ whose nodes are sorted in binary order. $\mathcal{T}$ stores *saṃdhi* rules of the form $\mathcal{R} = \{s_{SRC}, s_1, s_2, type\}$ where s_{SRC} is the result of the *saṃdhi* between s_1 and s_2 and *type* denotes the area of application of this *saṃdhi* (word, phrase, both). In trie terminology, s_{SRC} constitutes the path which leads to the leaves consisting of the reduced rules $\mathcal{R}' = \{s_1, s_2, type\}$. Obviously, multiple leaves may be assigned to a single path, as for example $\{\bar{a}, a, both\}$ and $\{a, a, both\}$ to the single letter path $\bar{a}$. n_{max} denotes the length of the longest s_{SRC} and is precalculated at program start.

If an extract s_{ijl} of $\mathcal{S}_i$ at position j matches a path s_{SRC} from the trie, l letters are removed from $\mathcal{S}_i$ beginning at position j. $\mathcal{S}_i$ is split into two new strings $\int_{i1}$ and $\int_{i2}$ at position j and the *saṃdhi* replacements s_1 and s_2 are affixed respectively prefixed to $\int_{i1}$ and $\int_{i2}$. Thereby, the new strings $\int_{i1} + s_1 = \mathcal{S}_{i1}$ and $s_2 + \int_{i2} = \mathcal{S}_{i2}$ are created. Figure 3 shows an example for this approach.[2] To keep the *saṃdhi*-rule base simple, the program uses a recursive strategy for *saṃdhi* resolution. For example, a string $\mathcal{S}_{i1} = xxxd$ resulting from the rule $\mathcal{R} = \{dbh, d, bh, both\}$ can be transformed further by application of $\mathcal{R} = \{d, t, -, phrase\}$ into the form *xxxt*. After creation of $\mathcal{S}_{i1}$ and $\mathcal{S}_{i2}$, the *saṃdhi* routine is recursively called for these new strings which are treated in the same way as $\mathcal{S}_i$. If the running index j reaches the end of a string ($j = L$), it is checked if $\mathcal{S}_i$ is a valid Sanskrit form. The procedures for this check and the respective structures of the database were shortly described in section 2.1. If $\mathcal{S}_i$ is a valid Sanskrit form, the grammatical and lexical analysis of $\mathcal{S}_i$ are inserted into the analysis of $\mathcal{P}$.

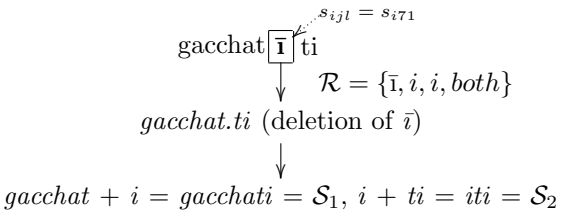

Figure 3: *saṃdhi* resolution for the string *gacchatīti*

During *saṃdhi* resolution and dictionary lookup, each $\mathcal{S}_i \in \mathcal{P}$ may have been resolved into m different subsets $\{\mathcal{S}_{i11}, \mathcal{S}_{i12}, \ldots, \mathcal{S}_{i1n_1}\}$, $\ldots$, $\{\mathcal{S}_{im1}, \ldots, \mathcal{S}_{imn_m}\}$ due to different *saṃdhi* resolution (SR). Furthermore, each of the substrings $\mathcal{S}_{ijk}$ has at least one grammatical and lexical analysis A_{ijkl} attached to it. Here, j is the number of the current SR, k the position of the substring in SR_j and l the index of the analysis for substring $\mathcal{S}_{ijk}$. Take for example the string $\mathcal{S}_1$ = *devadattakṛtakriyā*. (Parts of) two possible solutions (SR_1 and SR_2) are shown in figure 4 and 5. As indicated by the lines connecting the partial solutions A_{ijkl}, the second step of the tagging consists in finding the most probable path which runs through all $\mathcal{S}_i \in \mathcal{P}$. This step is again divided into two substeps. In the first substep, the

[2] The expression s_{i71} in figure 3 is not a mistake. Because *ch* is treated as a single phoneme, it is replaced with a single letter in the internal representation.

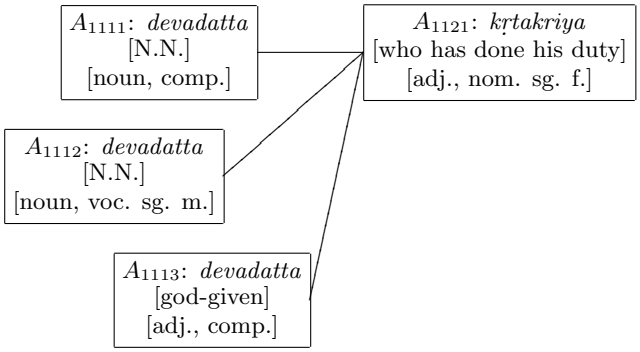

Figure 4: A possible analysis of the string *devadattakṛtakriyā*

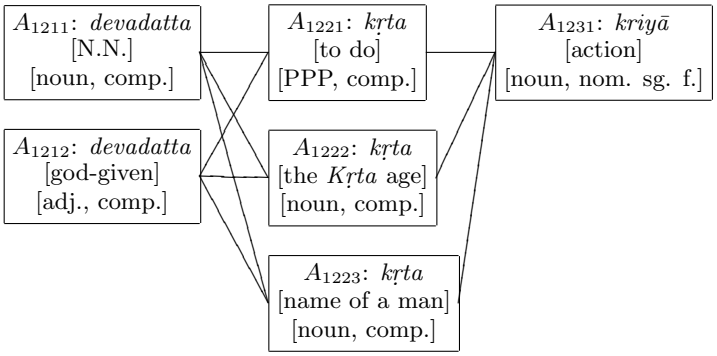

Figure 5: Another possible analysis of the string *devadattakṛtakriyā*

most probable lexical path is searched with the help of a Markov model (MM). This path is fixed as the tokenisation of $\mathcal{P}$. In the second substep, the most probable syntactical analysis of this path is searched with a HMM.

The first substep, i.e. the **tokenisation** can be modelled with a discrete, first-order Markov chain (see e.g. (Rabiner, 1989) for a readable introduction). The MM is based on the concept of conditional probability, which is the probability of event B given the occurrence of another event A: $P(B \mid A) = \frac{P(A \cap B)}{P(A)}$. In the given context, if $x_{[a,b]}$ denotes an ordered sequence of lexemes with decreasing indices i ($a \geq i \geq b$), the probability that phrase $\mathcal{P}$ of length L is tokenised into lexemes $x_1, x_2, \ldots x_L$ is given as

$$\begin{aligned} \underbrace{p(x)}_{P(A \cap B)} &= \underbrace{p(x_L \mid x_{[L-1,1]})}_{P(B|A)} \cdot \underbrace{p(x_{[L-1,1]})}_{P(A)} \\ &= p(x_L \mid x_{[L-1,1]}) \cdot p(x_{L-1} \mid x_{[L-2,1]}) \\ &\quad \cdot p(x_{[L-2,1]}) \quad \text{etc.} \end{aligned} \tag{1}$$

Under the assumption, that the probability of each lexeme depends only on its direct predecessor (first-order model), the formula is simplified to

$$\begin{aligned} p(x) &= p(x_L \mid x_{L-1}) \cdot p(x_{L-1} \mid x_{L-2}) \cdot \ldots \cdot p(x_1) \\ &= p(x_1) \Pi_{i=2}^{L} p(x_i \mid x_{i-1}) \end{aligned} \tag{2}$$

For reasons of floating point accuracy, equation 2 is transformed into

$$p(x) = \log p(x_1) + \sum_{i=2}^{L} \log p(x_i \mid x_{i-1}) \tag{3}$$

with $\log(a \cdot b) = \log a + \log b$. To find the most probable path, a modified form of the well known *Viterbi algorithm* is used. This algorithm was actually designed for HMMs, but can be applied to the given problem due to its similar structure. Before applying this algorithm to the data, it should be taken into consideration, that any string $\mathcal{S}_i$ may have been resolved in subsets S_{ix} and S_{iy} with different sizes $|S_{ix}| \neq |S_{iy}|$. Therefore, the algorithm can not be applied naively to the hypotheses generated in the first step. Instead, for each Phrase $\mathcal{P}$ which contains N strings $\mathcal{S}_1, \ldots, \mathcal{S}_N$, a vector of length N is allocated which stores for every string $\mathcal{S}_i$ the currently checked index j of its *saṃdhi* resolutions. So, a vector beginning in $\boxed{1}\boxed{2}\boxed{1}\ldots$ means, that currently the first resolution of $\mathcal{S}_1$, the second resolution of $\mathcal{S}_2$ and the first resolution of $\mathcal{S}_3$ are checked. Such a combination of the analyses of successive strings will be called *path subset*

(PS). (Note that the diagrams which show possible resolutions of *devadattakṛtakriyā* each constitute one path subset!) If n_i denotes the number of SRs for $\mathcal{S}_i$, there exist $\Pi_{i=1}^{N} n_i$ different PSs. Now, among all PSs, the PS with the best path regarding lexical probability is searched with a modified version of the Viterbi algorithm. Obviously, some PS consist of shorter paths because the strings in these combinations were split at fewer *saṃdhi* points. To treat all PS equally, they are filled up with "dummy probabilities" which are calculated as the mean of the probabilities constituting the best path $\pi_{opt\,u} \in PS_u$. If l_{max} denotes the length of the longest PS, l_u the length of the current PS_u and $\bar{p}(\pi_{opt\,u})$ the average transition probability between all elements constituting the optimal path in PS_u, the value $\sum_{r=l_{max}-l_u}^{l_{max}} \log \bar{p}(\pi_{opt\,u})$ is added to the probability of $\pi_{opt\,u}$.

The most probable path found with this algorithm is considered as the tokenisation of $\mathcal{P}$, which is then annotated morphologically with the help of an HMM. The POS tagset used for this annotation contains the 136 items shown in table 1. To understand the relation between this tagset and the actual morphological analysis of a word, consider the string *gacchati*. After the program has fixed the lexeme *gam* ("to go") as its most probable lexical analysis, there exist three possible morphological resolutions: "he/she/it goes" (3rd, sg., pres., P.) and "in the going ..." (loc. sg., masc./neutr., part. pres., P.). The solution "he ... goes" is mapped to the POS tag ["present tenses", 3rd, sg.], while the nominal solutions are mapped to ["present participles", loc., sg., masc.] and ["present participles", loc., sg., neutr.], respectively. Note, that a good deal of information is lost during this mapping process. For example, no distinction is made between different present tenses. Instead, forms like *gacchati* and *gacchatu* are considered to be syntactically equivalent and are therefore mapped to the same POS tag. This loss of information reflects the decision between the granularity of the tagset and the amount of text from which the probabilities of the tags can be estimated – the more text is available, the finer a granularity can be chosen.

The HMM $\lambda = \{A, B, \pi\}$ used to simulate the syntactical structure of $\mathcal{P}$ consists of the probability distribution A of transitions between tags (= states), the observation symbol probability distribution B, which records the probabilities $p(x \mid T)$ that a lexeme x is emitted given the tag T, and the initial state distribution π, i.e. the probabilities with which a tag opens a phrase (cmp. (Rabiner, 1989), 260/61). The values in A can be estimated from the corpus. The probabilities in B can be calculated using *Bayes theorem* $p(T) \cdot p(x \mid T) = p(x) \cdot p(T \mid x)$, where $p(x)$, $p(T)$ and $p(T \mid x)$ again can be estimated from the corpus. As indicated above, the optimal path with regard to POS probability is again searched with the Viterbi algorithm. This path is finally presented to the user as the most probable resolution of a phrase, which may be accepted and stored in the database or (manually) replaced with a better solution.

3 Performance and improvements

This section gives the results of tagging some short passages of text. The results are in no way representative. They are only meant to demonstrate the performance of the algorithm on different types of Sanskrit text. Three types of errors are distinguished. A saṃdhi *error* (e_S) occurs if a string is split at wrong splitting points. This error invalidates the results for the whole string. A *lexical error* (e_L) indicates that a string was split at correct *saṃdhi* points, but that a wrong lexeme was activated during tokenisation. Finally, a *POS error* (e_{POS}) occurs if a wrong POS tag was assigned to a correct token. Furthermore, the value r_{SL} gives the ratio of strings to lexemes, i.e. $r_{SL} = \frac{\text{nr. of strings}}{\text{nr. of lexemes}}$. A high value of r_{SL} indicates that the passage uses few composite words. – The following five passages were analysed:

1. Liṅgapurāṇa, 2, 20, 1-10: A *Purāṇic* text which treats a Śivaite topic. Easy verses.
2. Viṣṇusmṛti, 63, 35-50: An example of the scientific style. Many supplements are needed to get the full meaning of the passage.
3. Mūlamadhyamakārikā of Nāgārjuna, 12, 1-10: Easy Buddhist prose (from a linguistic point of view!).
4. Gītagovinda of , 1.2-5: Poetry with many unusual words.
5. Kāmasūtra, 2, 1, 1-12: Scientific prose.

The results, which are displayed in table 2, support the assumptions about the problems which are encountered in tagging natural Sanskrit text (see section 1). The tagger performs best on texts which are written in an easy style and come from "well known" areas of knowlegde (1, 3). On the contrary, a difficult vocabulary (5) and demanding syntactical structures (4) introduce a great deal of *saṃdhi* (4, 5) and POS (5) errors. The comparatively high number of POS errors in 3 is above all caused by confusion between nom. and acc. sg. neutre and could certainly be reduced by training the tagger with only a few similar texts.

At the moment, three main areas for **improvement** of this tagger can be discerned. Firstly, a reliable **estimation of probability values** for rare

Verbal forms		
Finite verbal forms		
present tenses (incl. imperative and opt.)	×9: person, number	9
past tenses	×9: person, number	9
future tenses	×9: person, number	9
other tenses	×9: person, number	9
Infinite verbal forms		
absolutive		1
infinitive		1
past participle, gerund	×24: case, nr., gender	24
present participles	×24: case, nr., gender	24
other participles	×24: case, nr., gender	24
Nominal forms		
indeclinable		1
nouns in composite words		1
nouns, adjectives	×24: case, nr., gender	24
		136

Table 1: Tagset used for POS tagging of Sanskrit text

nr.	P	S	L	r_{SL}	e_S	e_L	e_{POS}	corr. phrases
1	22	89	139	0.64	4	6	2	10
2	17	48	88	0.55	2	4	6	5
3	20	135	157	0.86	0	2	9	13
4	8	49	86	0.57	8	4	1	0
5	24	123	167	0.74	8	5	15	10

Table 2: Error rates of the tagger in five short passages – Abbreviations: **nr.**: number of the passage, **P**: number of phrases, **S**: number of strings, **L**: number of lexemes

lexemes and for infrequent POS combinations is the central step in improving the tagger. Application of the classical forward-backward-reestimation actually lead to degradation of the probability values (cmp. (Abney, 1996), 3). Although many other methods such as smoothing probabilities or the use of neural networks were proposed, Sanskrit offers a (partial) solution of this problem which is totally based on its lexicography. Sanskrit not only possesses a high number of homonymous, but also of synonymous words. Many of these words are already integrated in a semantic network (based on the `OpenCyc` ontology), which is contained in the program database. To estimate probabilities, groups of synonyms can be identified which designate the same sememe with a high degree of probability. In this sense, the group "horse" is constituted by {*aśva*, *turaga*, *turaṃga*, *turaṃgama*, *vājin*, *haya*}, but not *hari*, which means "Viṣṇu" in most cases. If one of the words which is included in the group "horse" is met in an unknown context (either lexical or POS/morphological), the respective probabilities can be estimated from the values given for other members of the group.

Secondly, **integration of rules** can certainly improve analysis. Due to lack of punctuation (see 1), these rules should not describe well-formed and complete phrases, but only check the coherence of few members of a phrase $\mathcal{P}$, i.e. a syntactic substructure delimited by *daṇḍa*s (chunk parsing). Some preliminary tests with rules which reject paths during POS tagging on the base of simple syntactic criteria turned out to be successful.

Thirdly and finally, the strict **separation of tokenisation and POS tagging** is a constant source of errors. Consider, for example, the simple sentence *brahmā varaṃ te dāsyati* ("Brahmā will give you a boon."). Although the POS sequence [dat. sg.] - [3. sg. fut.] is well established and would always be preferred to the incongruent [nom. pl.] - [3. sg. fut.], during tokenisation *te* is interpreted as the nom. pl. masc. of the pronoun *tad* due to its enormous frequency. Because the content of the best lexical path can not be changed during POS analysis, the correct analysis for *te* (dat. sg. of *tvad*) will never be activated. Possible workarounds for this problem are a more flexible POS analysis, which takes into account e.g. the first five lexical resolutions, or a combination of tokenisation and POS tagging in one procedure.

References

1893. *Parāśaradharmasaṃhitā. Ācārakāṇḍam.*

Hartmann, Peter. 1955. *Nominale Ausdrucksformen im wissenschaftlichen Sanskrit.* Carl Winter Universitätsverlag, Heidelberg.

Lawrence R. Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286.

Steven Abney. 1996. In *Corpus-Based Methods in Language and Speech*, Dordrecht. Kluwer Academic Publishers.

Robert Waring and Paul Nation. In *Vocabulary: Description, Acquisition and Pedagogy.* pages, 6–19, Cambridge. Cambridge University Press.

Hellwig, Oliver. 2002. *Sanskrit und Computer.* Ph.D. thesis, Freie Universität Berlin.

Gérard Huet. 2007. Shallow syntax analysis in sanskrit guided by semantic nets constraints. In *International Workshop on Research Issues in Digital Libraries.*