Design of a lean interface for Sanskrit corpus annotation

Gérard Huet
Inria Paris-Rocquencourt
Gerard.Huet@inria.fr

Pawan Goyal
Indian Institute of Technology, Kharagpur
pawang@cse.iitkgp.ernet.in

Abstract

We describe an innovative computer interface designed for assisting annotators in the efficient selection of segmentation solutions for proper tagging of Sanskrit corpus. The proposed solution uses a compact representation of the shared forest of all segmentations. The main idea is to represent the union of all segmentations, abstracting on the sandhi rules used, and aligning on the input sentence. We show that this representation allows an exponential saving, both in space and time. This interface has been implemented, and has been applied to the annotation of the Sanskrit Library corpus.

1 Generalities on Sanskrit linguistics

Sanskrit is the primary culture-vehicle language of India. It has had a continuous production of literature in all fields of human endeavour over the course of four millennia, giving rise to an immense corpus which is to this date only partially digitalized. It benefits from a very sophisticated linguistic tradition stemming from the fairly complete grammar composed by Pāṇini by the fourth century B.C.E.

During the last 15 years, a significant effort at developing Sanskrit Computational linguistics has been endeavoured, and considerable progress has been achieved at providing computer assistance at Sanskrit corpus processing (Scharf and Hyman, 2009; Huet et al., 2009; Kulkarni and Huet, 2009; Jha, 2010; Kulkarni et al., 2010; Kumar et al., 2010; Kulkarni and Shukl, 2009; Goyal et al., 2009; Hellwig, 2009; Goyal et al., 2012). Nevertheless, there does not exist at this date a complete analyser for Classical Sanskrit texts able to compute reliably morphological taggings in a completely automatic way. The main difficulty concerns segmentation, since Sanskrit is represented in writing by continuous phonetic enunciation, which demands complex processing for its analysis in separate word forms. Although complete algorithms for this segmentation preprocessing have been proposed (Huet, 2005), human assistance is still needed to focus on the intended solution within all possible analyses.

We propose in this paper a new human-machine interface to help a professional annotator to decide quickly between all possible segmentations in order to select a unique morphological analysis among the many possible ones. Indeed, there exist thousands of such segmentations for simple sentences, and literally billions for complex ones. Once a sufficient amount of tagged corpus is available using such semi-automated annotation tools, it is hoped that it will be possible to use it for training a fully automated parser using statistical methods.

2 Segmentation analysis

We are going to formalize the segmentation problem at various levels of abstraction. Firstly, we assume that Sanskrit text is represented as a list of phonemes. Sanskrit may be written in all Indian scripts, most usually in the *Devanāgarī* script used by languages of North India such as Hindi, but such syllabic representation is awkward for morpho-phonetics computations, which operate at the phoneme level. It is thus preferable to translate the input into a list of phonemes, such translation being one-one. We assume the standard set of 50 phonemes, already known from the time of Pāṇini. Such low-level representation issues are discussed at length in (Scharf and Hyman, 2009; Huet, 2009).

We assume that the reader is familiar with the use of finite-state methods for morpho-phonemic computations, as explained in standard references such as (Roche and Schabes, 1997; Kaplan and Kay, 1994; Beesley and Karttunen, 2003). We also assume familiarity with the lexicon-driven Sanskrit segmenter of Huet (Huet, 2005), from which we extract the following definitions.

Definitions. A *lexical juncture system* on a finite alphabet Σ is composed of a finite set of words $L \subseteq \Sigma^*$ and a finite set R of rewrite rules of the form $[x]u|v \rightarrow w$, with $x, v, w \in \Sigma^*$ and $u \in \Sigma^+$. Note that in the Kalpan and Kay notation, the rule we write $[x]u|v \rightarrow w$ would be written as $u|v \rightarrow w/x\_\_$.

The word $s \in \Sigma^*$ is said to be a *solution* to the system (L, R) iff there exists a sequence $\langle z_1, \sigma_1\rangle; ...\langle z_p, \sigma_p\rangle$ with $z_j \in L$ and $\sigma_j = [x_j]u_j|v_j \rightarrow w_j \in R$ for $(1 \leq j \leq p)$, $v_p = \epsilon$ and $v_j = \epsilon$ for $j < p$ only if $\sigma_j = o$, subject to the matching conditions: $z_j = v_{j-1}y_jx_ju_j$ for some $y_j \in \Sigma^*$ for all $(1 \leq j \leq p)$, where by convention $v_0 = \epsilon$, and finally $s = s_1...s_p$ with $s_j = y_jx_jw_j$ for $(1 \leq j \leq p)$. We also say that such a sequence is an *analysis* of the solution word s.

In this formalization, Σ is the set of phonemes, R is the set of sandhi rules, and L is the vocabulary as a set of lexical items. As a first approximation, one may think of L as the lexicon of inflected words. In a later section, we shall partition L according to lexical sorts, some of which being morphemes such as stems and affixes, in order to segment compound words by the same method as we explain here sentence segmentation into words. This extension is necessary to keep L finite, in view of the fact that nominal compounding in Sanskrit is productive to an arbitrary depth. But all the notions defined here will apply easily to this refinement, which allows us to keep notations simple. We shall also assume the system (L, R) to be *non-overlapping*, as defined in (Huet, 2005). This assumption is met in classical Sanskrit, except for a small number of uniphonemic morphemes, which are amenable to the general treatment modulo the introduction of so-called *phantom phonemes*, as explained in (Huet, 2006; Goyal and Huet, 2013).

We may think of s as a phonetically correct utterance over vocabulary L, and its analysis $S = \langle z_1, \sigma_1\rangle; ...\langle z_p, \sigma_p\rangle$ as one of its possible segmentations. S is completely explicit, in the sense that s may be computed uniquely from S, applying sandhi rules σ_i in sequence, going from left to right. Conversely, there may be many possible segmentations S of a given utterance s, typically thousands for a moderately long sentence, although it is proven in (Huet, 2005) that they are always in finite number. We write $Segs(s)$ for the set of segmentations of s. The algorithm described in (Huet, 2005) shows how to enumerate in a complete way the set $Segs(s)$ from a given input string s. In view of its possibly enormous size, attempts have been made, e.g. (Huet, 2007), to filter out non-sensible segmentations by a semantic analysis in the manner of dependency grammars. This method works well for simple sentences, but is not sufficient for more complex sentences, specially in the presence of ellipses and other anaphoric or discourse operators where dependencies are context-sensitive. Furthermore the set $Segs(s)$ is not easily amenable to sharing, and as a consequence the segmentation cum tagging Web service of the Sanskrit Heritage site[1] has not been of practical use so far on real corpus, since it tended to generate very long Web pages, even to the point of choking the server. Wading through such long lists of segmentations was very tedious and error-prone. The new interface described in the present paper completely solves this problem. We shall now explain its main concepts.

3 Aligned segmentations

The key idea behind the new interface is to represent an abstraction of the union of all segmentation decompositions, realigned on the input utterance. This new representation is now amenable to sharing, and may thus be represented very compactly on one computer screen.

Definition. We consider a sandhi analysis S as above, generalized to allow empty sequences. It may be defined inductively, as being either empty or of the form $S = \langle z_1, \sigma_1\rangle; S'$, with S' a similar sequence. Let n be a natural number. We define the *alignment* of S *with offset* n, noted as $S \hookrightarrow n$, as a set of pairs of *aligned segments* of the form (k, z), with $k \in \mathbb{N}$ and $z \in L$, as follows. If S is the empty sequence, then $S \hookrightarrow n = \emptyset$. Otherwise, let $S = \langle z, \sigma\rangle; S'$ with $\sigma = [x]u|v \rightarrow w$. We define $S \hookrightarrow n = \{(n, z)\} \cup S' \hookrightarrow n'$, where

[1] `http://sanskrit.inria.fr`

$n' = n + |z| + |w| - (|u| + |v|)$.

If S is a segmentation analysis of utterance s, we define its corresponding *aligned segment collection* as the set of aligned segments $\overline{S} = S \hookrightarrow 0$. Note that in this new notion we are forgetting the precise sandhi rules used in the analysis S, keeping only the tabulation information that allows us to present its set of segments aligned with the original input s.

Let $\mathcal{S}$ be a set of segmentation analyses of utterance s. We define the *tabulated display* of $\mathcal{S}$, noted $D(\mathcal{S})$, as the set of aligned segments obtained as the union of all its corresponding aligned segment collections:

$$D(\mathcal{S}) = \bigcup_{S \in \mathcal{S}} \overline{S}$$

We say that an aligned segment (k, z) is *relevant* to a segmentation analysis S iff $(k, z) \in \overline{S}$. Let $\mathcal{S}$ be a non-empty set of segmentation analyses of some utterance s, and $(k, z) \in D(\mathcal{S})$. We define the *restriction* of $\mathcal{S}$ to (k, z), noted $\mathcal{S} \downarrow (k, z)$, as the set of all segmentation analyses in $\mathcal{S}$ to which (k, z) is relevant:

$$\mathcal{S} \downarrow (k, z) = \{S \in \mathcal{S} \mid (k, z) \in \overline{S}\}$$

We have of course $\mathcal{S} \downarrow (k, z) \subseteq \mathcal{S}$.

Fact 1. $(k, z) \in D(\mathcal{S}) \Rightarrow \mathcal{S} \downarrow (k, z) \neq \emptyset$.
Proof. Trivial compactness property of union.

Let $\mathcal{S}$ be a non-empty set of segmentation analyses of some utterance s, and $(k, z) \in D(\mathcal{S})$. We say that (k, z) is *critical* in $D(\mathcal{S})$ iff it is not relevant to some $S' \in \mathcal{S}$. This implies that

$$|D(\mathcal{S} \downarrow (k, z))| < |D(\mathcal{S})|$$

Thus selecting a critical segment in the interface will effectively reduce the search space. In practice, it will reduce it by half or more, and convergence will be insured in $log(N)$ steps, where N is the total number of segmentation solutions. Let us now give a sufficient condition for criticality.

Let (k, z) and (k', z') be two distinct aligned segments in some tabulated display $D(\mathcal{S})$. We say that (k, z) and (k', z') *conflict* if $k \leq k' < k + |z| - 1$ or $k' \leq k < k' + |z'| - 1$.

Fact 2. Let (k, z) and (k', z') conflict in $D(\mathcal{S})$. They are both critical, in being mutually exclusive – no segmentation may contain both.

Proof. By inspection of sandhi rules, we may check that every rule $[x]u|v \rightarrow w$ is such that $|u| + |v| \leq |w| + 1$. Thus overlapping of a segment with its successor in any segmentation is at most of length one. Since every segment is of length at least 1, overlap of a segment with some other segment in the same segmentation solution may be at most of length one. Let (k', z') be an aligned segment of $D(\mathcal{S})$ conflicting with (k, z). No segmentation analysis to which (k', z') is relevant may belong to $\mathcal{S} \downarrow (k, z)$, and thus $(k', z') \notin D(\mathcal{S} \downarrow (k, z))$.

Note that the conflicting condition is sufficient to show that two segments may not appear in a common segmentation solution, but that this is not a necessary condition, even for contiguous segments. The interest of this notion is that it is easy to check visually, whereas the necessary and sufficient criterion is not, since sandhi rules are not shown.

We now state a fact which may not be true of all lexical juncture systems, but is verified for Sanskrit sandhi, as we shall argue in section 6.2.

Fact 3. If $D(\mathcal{S})$ has no critical aligned segment, $\mathcal{S}$ is singleton.

4 A graphical interface

4.1 A user interface using aligned segments

Let s be the utterance under consideration. Initially, we compute the set $\mathcal{S} = Segs(s)$ of all its possible segmentations, and we display $D(\mathcal{S})$, where every aligned segment (k, z) is represented as the segment z displayed with an offset of k spaces from the left margin. When two aligned segments overlap, we represent them in different lines. We sort all segments, so that longer segments are listed above shorter ones. Each segment is displayed either with a blue check sign, if it does not conflict with any other segment or else with two signs, a green check sign to select this segment and a red cross sign to discard it. These green check and red cross signs are mouse sensitive, they trigger as call-back the segmentation routine, that will compute all segmentations analyses consistent with this particular choice, that is, for which all currently selected aligned segments with green check signs are present, and the segments discarded using red cross signs are absent. If s is segmentable at all, $\mathcal{S}$ is non empty, and so is $D(\mathcal{S})$. At any point in the computation, the current display $D(\mathcal{S})$ represents the union of a non-empty set $\mathcal{S}$ of segmentations of s, by repeated application of Fact 1. Consequently, selecting or discarding a segment may never fail.

Furthermore, if the user selects or discards a critical segment, there is visible progress, since all conflicting segments vanish on selecting a segment and a particular segment vanishes on discarding it. This corresponds to the case when it conflicts with some other segment, which is easy to see in the visual display, since it covers a column which is strictly inside the conflicting segment.

When a segment is selected using the green check sign, both the check and cross signs are replaced by a single blue check sign, which is mouse insensitive, thus making the segment inert for the rest of the interaction. On the other hand, if a segment is discarded using the red cross sign, it vanishes and in the particular case where it conflicts only with one other segment, the other segment will become inert. Note that the user cannot select a non-critical segment since these are presented with blue check signs, which are not mouse sensitive. When there are no more critical segments, we have reached a unique segmentation solution, consistently with Fact 3.

At any moment, the user may, besides the selection of a segment, do a number of actions. Firstly, he may undo the previous selections, up to an arbitrary depth. Secondly, he may revert to the old interface giving a linear listing of all segmentations consistent with the currently selected segments. A counter indicates the size of the remaining number of distinct segmentations. Optionally, he may also use the semantic pruning mechanism in the hope to converge more rapidly with machine assistance. Finally, it is possible to send the set of remaining segmentations to the more complete dependency parser under development at University of Hyderabad (Kulkarni et al., 2010; Kulkarni and Ramakrishnamacharyulu, 2010).

4.2 Complexity analysis

The convergence of the selection is very fast. Since the method is dichotomic it converges in average in $log(N)$ steps, where N is the total number of segmentation solutions. Indeed, when the input may be split as $s = s_1 \cdot s_2$ with s_1 and s_2 independently segmentable with respectively n_1 and n_2 segmentations, presented with displays of sizes respectively d_1 and d_2, the global display has a size of $d_1 + d_2$ for a total of $n_1 \times n_2$ segmentations. This interface gives thus an exponential improvement over the recursive dove-tailing of the segmentation process. In any case the number of selections will be less than the number of words of the intended segmentation, i.e. of the order of the length of the sentence divided by the average length of a word. In practice convergence is very fast.

Theorem. Let $\mathcal{S}$ be the set of segmentation analyses of some utterance s of length n. $|\mathcal{S}|$ is of asymptotic order $O(C^n)$, whereas $|D(\mathcal{S})|$ is of asymptotic order $O(n)$.

Proof. This theorem depends on the lexicon under usage and can have at best an average complexity analysis. Let m be the length of an average segment in an utterance. For our analysis, we will also assume that each segment in a valid solution has length ≥ 2.

Consider s of length n. We will try to find an upper bound on the number of segmentation solutions for this utterance. Let us consider the i^{th} phoneme of this utterance. A valid solution can have this phoneme participating in a segment of length 2, 3, ... up to m. Analysing further, a segment of length 2 can start at 2 possible offsets, $i-1$ or i. Similarly, a segment of length 3 can start at 3 possible offsets and so on. In general, let of_j denote the number of offsets at which a segment of length j may start for the i^{th} phoneme. Then, $\mathit{of}_j \leq j$ for $j \in \{2, 3, \dots, m\}$. Every such offset k for a segment of length j defines a set with aligned segments (k, z_l) such that $|z_l| = j$. Thus, for the i^{th} phoneme, an upper bound on the number N_{ss_i} of possible such sets is:

$$\begin{aligned} N_{ss_i} &\leq \mathit{of}_2 + \mathit{of}_3 + \ldots + \mathit{of}_m \\ &\leq 2 + 3 + \ldots + m \\ &\leq \frac{m(m+1)}{2} \end{aligned} \tag{1}$$

For each of these N_{ss_i} sets, the possible number of segments depends on the sandhi rules R. For any segment in such set, permutations are possible only at the first and last phonemes because of the sandhi rules applied at the junction. Let left_w denote the number of possible v's such that $u|v \rightarrow w \in R$ for an arbitrary u. Similarly, let $right_w$ denote the number of possible u's such that $u|v \rightarrow w \in R$ for an arbitrary v. Now let *maxleft* be the maximum of all such left_w and *maxright* be the maximum of all such $right_w$. Thus, such a set can contain at most $|ss_i| = (\mathit{maxleft} \times \mathit{maxright})$ segments. Then the maximum number of segments N_i that the i^{th} phoneme can participate in is

$$N_i \leq |ss_i| \cdot N_{ss_i} \tag{2}$$

Now that we have the maximum number of possible segments for the phoneme at position i, we can use this to obtain an upper bound on the number of segments $|\mathcal{S}|$ for the utterance s. We will use the fact that the set $|\mathcal{S}|$ will be a subset of all the possible segments in which phonemes at various positions can participate. Thus

$$\begin{aligned} |\mathcal{S}| &\leq N_1 \times N_2 \dots N_n \\ &= (|ss_i|)^n \cdot N_{ss_i}{}^n \\ &= (C \cdot \frac{m(m+1)}{2})^n \end{aligned} \quad (3)$$

Similarly, an upper bound on the number of segments in the tabulated display is the sum of all possible segments at various phonemes. Thus

$$\begin{aligned} |D(\mathcal{S})| &\leq N_1 + N_2 + \cdots + N_n \\ &= (C \cdot \frac{m(m+1)}{2}) \cdot n \end{aligned} \quad (4)$$

Hence, it follows from Equations 3 and 4 that $|\mathcal{S}|$ is of asymptotic order $O(C^n)$ at worst, whereas $|D(\mathcal{S})|$ is of asymptotic order $O(n)$.

From experimental evidence, it has been observed that the number of solutions grows exponentially with the length of the utterance and the bound $O(C^n)$ is actually reached for the real corpus. For instance, the following sentence, excerpted from the *Vikramorvaśī* play by Kālidāsa, has 6 967 296 000 ($\approx 2^{32}$) segmentations. The sentence has 240 phonemes and the desired solution has 40 segments. This sentence is manageable by our interface in 17 clicks and thus, the convergence is quite fast.

yā tapasviśeṣapariśaṅkitasya sukumāram praharaṇam mahendrasyapratyādeśaḥ rūpagarvitāyāḥ śriyaḥ alaṃkāraḥ svargasya sā naḥ priyasakhyurvaśī kuberabhavanāt pratinivartamānā samāpattidṛṣṭena keśinā dānavena citralekhādvitīyā bandigrāham gṛhītā

5 Graphical rendering of the Display

The main notion behind the interface is that of the display $D(\mathcal{S})$ for a consistent set of segmentations $\mathcal{S}$. Initially we take $\mathcal{S} = Segs(s)$, and we progressively select aligned segments $(k_1, z_1), ..., (k_n, z_n)$. The only data kept in memory is the initial sentence s, and the stack of choices $C_n = (k_1, z_1), ..., (k_n, z_n)$. The interface interaction is implemented as a CGI coroutine, which is transmitted arguments s and C_n in its invocating URL. The server recomputes at every step the sequence of all segmentations $Segs(s)$, keeping only the ones that are consistent with the stack of choices C_n, sorted by alignments into a sorted a-list of checkpoints. The display of all consistent segmentations is stored in an array 'display' of size $|s|$. The display value at index i is the list of all segments z such that (i, z) is an aligned segment of some segmentation solution consistent (i.e. not conflicting) with all the checkpoints C_n. This test is easy, since C_n is sorted. One may think of the display as a shared representation of $D(\mathcal{S})$, for $\mathcal{S}$, the set of segmentation solutions consistent with the current stack of choices. Actually the array 'display' may be thought of as a hashcoding array for the set $D(\mathcal{S})$, the hashcode of an aligned segment (k, z) being its alignment k in the input string.

What is crucial for the efficient sharing of the tree of all segmentations as a dag is the abstraction of sandhi rules. Indeed, our methodology is reminiscent of parsers based on tabulation methods which use such dynamic programming methods (Earley, 1983; Tomita, 1985; Billot and Lang, 1989; Stolcke, 1995).

Implementation of 'Undo' is trivial, since it consists in calling the interface with a stack of choices popped by the last choice.

Note the simplicity of this implementation: at every step all the information is recomputed with the standard segmenter, but since the technology is very fast this is not noticeable to the user, the reaction seems instantaneous (at least on a localhost server).

Presenting the tabulated display of the aligned segmentations in an HTML page was not entirely trivial. The segmentation analysis gives us all possible segments, appearing at various offsets. Firstly, for an arbitrary offset k_i, the number of segments may be quite large. Also, the length $|z_i|$ of the largest segment (k_i, z_i) at offset k_i might be such that it conflicts with the aligned segments at the next offset k_{i+1}. Since the objective was to have a compact display, keeping the alignment intact, it remains a problem as to where to fit the aligned segments at offset k_{i+1} in such case, once the HTML display has been populated with the segments at offset k_i. The second issue is related to the fact that while the maximum size of the display array is fixed to the length of the utterance ($|s|$), the size of an aligned segment (k_i, z_i) is $|z_i|$, a variable depending on the segment z_i. Thus, the problem is to show the aligned segment as a single entity.

Now, a simplistic implementation to keep the alignment intact would have been to list all the segments corresponding to the offset k_{i+1}, starting from the next row to where all the segments at offset k_i have been enumerated. This, obviously, would not lead to a compact display. Similarly, a very simple implementation to handle variable sized segments would be to define an array of $|s|$ columns and display each solution (k_i, z_i) in $|z_i|$ columns starting from the ${k_i}^{th}$ column. The problem with this approach is that the display of a word does not appear continuous here. Also, depending upon the transliteration scheme used, some phonemes would require more space than others, which will vary in various rows. And the segment z_i here cannot be treated as a single HTML entity in this case, which is a requirement to allow a user-friendly display of morphological tags, as well as for the callbacks, initiating user interaction.

To alleviate these problems, we sorted the segments at each offset according to their lengths. Thus, the longer segments appear on the top. Now, while filling the segment (k_{i+1}, z_{i+1}) at offset k_{i+1}, we search for the first row from the top, where the last filled segment does not conflict with (k_{i+1}, z_{i+1}) and fill this segment in that row. This gives a compact display.

Similarly, to handle the second issue, instead of using $|z_i|$ columns for an aligned segment (k_i, z_i), we used the HTML 'colspan' attribute to use variable width columns in a row. Thus, an aligned (k_i, z_i) is displayed using a $|z_i|$ width column at offset k_i. This allows us to have a continuous display of a segment, as well as to treat it as a single HTML entity.

6 Lexical categories and tagging

6.1 Dealing with lemmatized segments

Since our method is lexicon-directed, our candidate forms are generated by morphological generation, and may be kept along with their lemmas. Furthermore, we may restrict our segmenter to recognise only morphologically correct sequences, according to a regular grammar expressing morphological constraints. This refinement is also necessary because the sandhi relation after preverbs (*upasarga*) is different from the external sandhi between words or compound components. This grammar is compiled into the state-transition graph of a finite automaton/transducer, which expresses the control of our lexical scanner in the usual manner. The states of this automaton, called *phases*, correspond to the lexical categories, which we associate to colours in the interface. We may refine the above formalization to this new situation easily, replacing the notion of aligned segment (k, z) by the finer notion of *aligned lemmatized segment* $(k, (l, z))$ where l is the lemmatization of segment z.

Let us now give a concrete example. Consider the following sentence:
satyaṃbrūyātpriyaṃbrūyānnabrūyātsatyamapriyaṃpriyaṃcanānṛtambrūyādeṣadharmaḥsanātanaḥ. This is the well known saying (*subhāṣitam*): "One should tell the truth, one should say kind words; one should neither tell harsh truths, nor flattering lies; this is a rule for all times."

When entering this sentence in our Sanskrit reader, the initial display of our interface is given as Figure 1 below.

As indicated in the display, this diagram summarizes 120 distinct segmentations. At the right side of the diagram, one sees the long segment *sanātanas* ('eternal') and below it a messy alternative of compositions of smaller words that are obviously over-generating items. Clicking on the green sign under the blue segment *sanātanas* removes all this noise, and the number of potential solutions drops to 12, generating the display given in Figure 2 - note the blue unlinked check sign indicating the already selected segment.

Similarly one immediately notices segment *satyam* ('truth'), together with conflicting noisy alternatives. Similarly for *cana* ('and not'). These two selections will leave us with only one choice between segments *brūyāt* and *brūyām* (two forms of root *brū* ('to say') in the optative mode of the present active voice in the singular number, respectively in the 3rd and 1st person). By obvious symmetry with its other occurrences in the sentence, *brūyāt* must be chosen, obtaining the correct segmentation in a total of 4 easy clicks, as shown in Figure 3.

At this point one may click on the explicit button "Unique Solution", where fine tuning of last morphological parameters such as ambiguities of genders of substantival forms may be effected through a final user interface, shown in Figure 4. This last stage is necessary, because our lemmas label a given form with a multi-tag factoring out

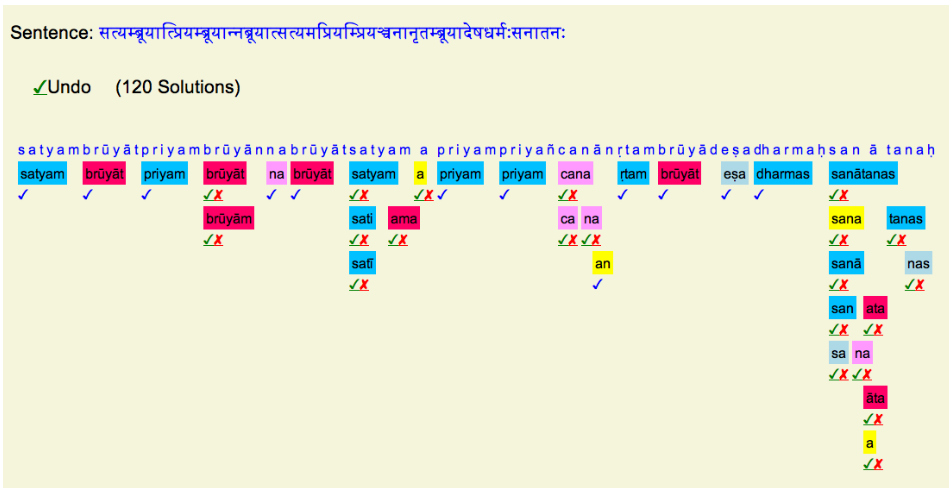

Sentence: सत्यम्ब्रूयात्प्रियम्ब्रूयान्नब्रूयात्सत्यमप्रियम्प्रियश्चनानृतम्ब्रूयादेषधर्मःसनातनः
✓Undo (120 Solutions)
s a t y a m b r ū y ā t p r i y a m b r ū y ā n n a b r ū y ā t s a t y a m a p r i y a m p r i y a ñ c a n ā n ṛ t a m b r ū y ā d e ṣ a d h a r m a ḥ s a n ā t a n a ḥ
satyam
brūyāt
priyam
brūyāt
brūyām
na
brūyāt
satyam
sati
satī
a
ama
priyam
priyam
cana
ca
na
an
ṛtam
brūyāt
eṣa
dharmas
sanātanas
sana
sanā
san
sa
tanas
nas
ata
na
āta
a

Figure 1: Initial display of the aligned segments for the sentence *satyaṃbrūyātpriyaṃbrūyānnabrūyātsatyamapriyaṃpriyaṃcanānṛtambrūyādeṣadharmaḥsanātanaḥ*

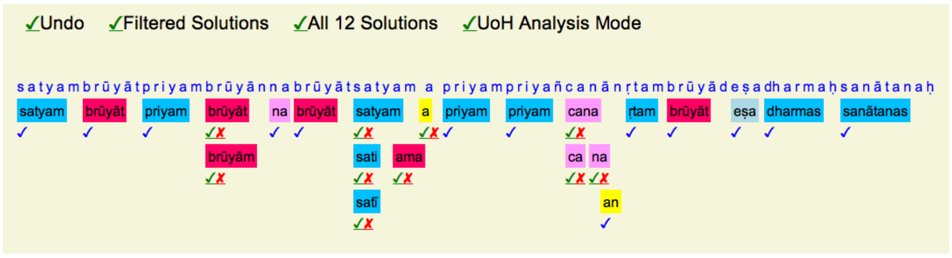

✓Undo
✓Filtered Solutions
✓All 12 Solutions
✓UoH Analysis Mode
s a t y a m b r ū y ā t p r i y a m b r ū y ā n n a b r ū y ā t s a t y a m a p r i y a m p r i y a ñ c a n ā n ṛ t a m b r ū y ā d e ṣ a d h a r m a ḥ s a n ā t a n a ḥ
satyam
brūyāt
priyam
brūyāt
brūyām
na
brūyāt
satyam
sati
satī
a
ama
priyam
priyam
cana
ca
na
an
ṛtam
brūyāt
eṣa
dharmas
sanātanas

Figure 2: Aligned segments after selection of segment *sanātanas*.

all values of morphological parameters usable to generate this form. User can select the appropriate options yielding the final unambiguous tagging of the sentence as a list of lemmas, where segments are hyperlinks to the digital lexicon, as shown in Figure 5.

This page may be stored and the next sentence may be read from the corpus input stream in order to progressively annotate the digital library.

Sometimes it is useful for the annotator to see the lemmatization of a segment in order to inform his decision with more information than its lexical category (indicated by the color code, blue for substantives, red for finite verb forms, purple for adverbs and yellow for compound). This facility is available in our interface, every segment is mouse-sensitive, and clicking on it yields its lemma, as shown in Figure 6 for the segment *brūyāt*.

Note that in this lemma the root *brū* is itself mouse-sensitive, it is a hyperlink to its lexical entry, allowing access to its meaning. We provide two aligned digital lexicons, our original Sanskrit-to-French Heritage dictionary, and optionally the more complete classical Sanskrit-to-English Monier-Williams' dictionary[2]. Thus the annotator has all available information at his disposal at any point, but with minimal cluttering of his workspace.

It should be noted that this interface is not only easy to use, it is actually fun to play. It may be thought of as some kind of electronic game.

6.2 Justifying Fact 3

In the example just shown, we assumed implicitly that when no more choices were available to the user, there was only one segmentation solution left, and we could then proceed to the final disambiguation of the remaining multi-tags of this unique solution. This assumption is precisely what we called Fact 3 above, and that we now restate:

Fact 3. If $D(\mathcal{S})$ has no critical aligned segment, $\mathcal{S}$ is singleton.

Proof. Assume that $D(\mathcal{S})$ has no critical aligned segment. In other words, all the segments are marked with a blue mark, indicating that they

[2]The protocol for the non-trivial task of mutually linking these lexical resources has been discussed in (Goyal et al., 2012)

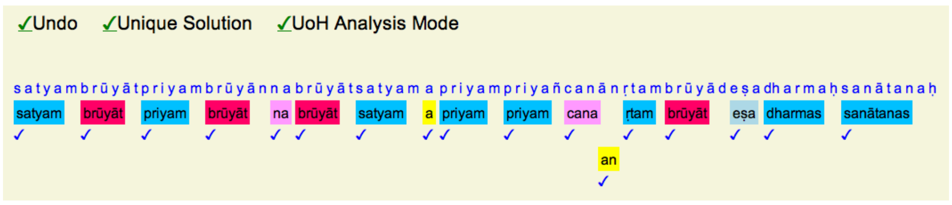

✓Undo
✓Unique Solution
✓UoH Analysis Mode
s a t y a m b r ū y ā t p r i y a m b r ū y ā n n a b r ū y ā t s a t y a m a p r i y a m p r i y a ñ c a n ā n ṛ t a m b r ū y ā d e ṣ a d h a r m a ḥ s a n ā t a n a ḥ
satyam
brūyāt
priyam
brūyāt
na
brūyāt
satyam
a
priyam
priyam
cana
ṛtam
brūyāt
eṣa
dharmas
sanātanas
an

Figure 3: Aligned segments after 4 clicks.

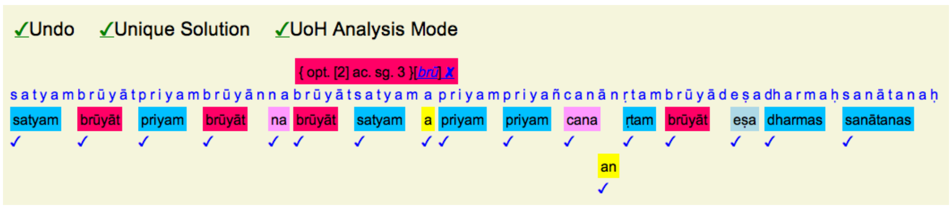

✓Undo
✓Unique Solution
✓UoH Analysis Mode
{ opt. [2] ac. sg. 3 }[brū] ✗
s a t y a m b r ū y ā t p r i y a m b r ū y ā n n a b r ū y ā t s a t y a m a p r i y a m p r i y a ñ c a n ā n ṛ t a m b r ū y ā d e ṣ a d h a r m a ḥ s a n ā t a n a ḥ
satyam
brūyāt
priyam
brūyāt
na
brūyāt
satyam
a
priyam
priyam
cana
ṛtam
brūyāt
eṣa
dharmas
sanātanas
an

Figure 6: Asking for lemma of segment *brūyāt*.

belong to all remaining solutions. Thus, all remaining solutions have the same segments. We shall need to prove that all the aligned segments are strictly ordered within one unique solution. Consider any two remaining segments (k, z) and (k', z'), where without loss of generality we may assume $k \leq k'$. If $k < k'$, the z segment must precede the z' one. Now let $k = k'$. It is not the case that both $|z| > 1$ and $|z'| > 1$, since the two segments would be conflicting with each other. Assume without loss of generality $|z| = 1$. If $|z'| > 1$, the z segment must precede the z' one. We are left to consider the case where $|z| = |z'| = 1$. The only relevant mono-phonemic segments in classical Sanskrit are the privative prefix '*a*', forming so-called *nañ-tatpuruṣa* compounds, and the preposition '*ā*', used as prefix (*upasarga*) to final (*tiṅanta*) and propositional (*kṛdanta*) verbal forms.[3] We thus only have to consider the proper ordering of co-aligned '*a*' and '*ā*' segments. The privative particle '*a*' can prefix only consonant-initial nouns, since it alternates with the form '*an*' for vowel-initial ones. The preposition '*ā*' is assumed not to be iterated, which would be redundant. Thus the only possible ordering is that an '*ā*' segment could precede an '*a*' segment (but we do not even know of a concrete example). This explanation justifies Fact 3 in the case of classical Sanskrit.

[3]Vedic Sanskrit offers additional difficulties, with autonomous prepositions and the mono-phonemic interjection *u*.

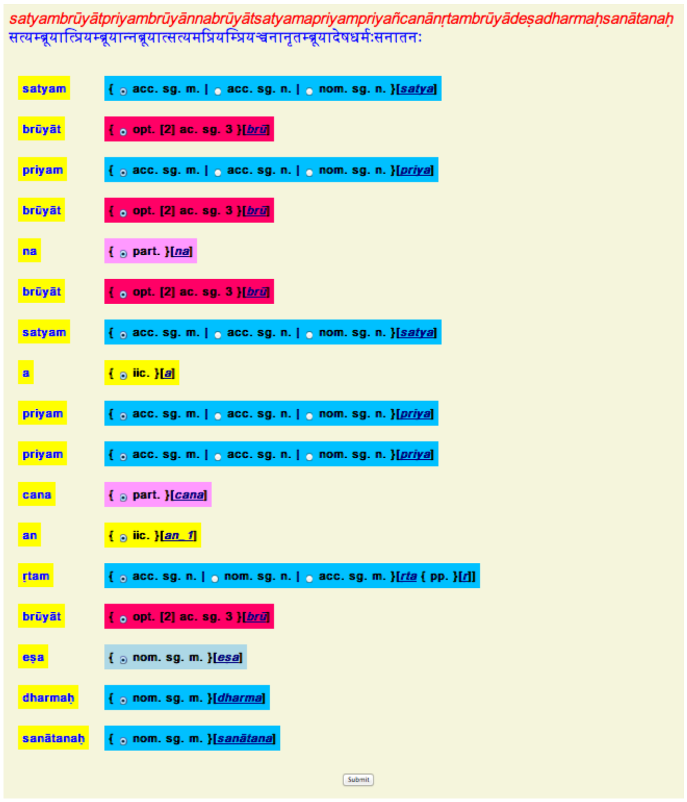
6.3 Robustness

The interface is remarkably robust for realistic sentences, as shown in the example in section 4.2. When the sentence from the *Vikramorvaśī* play by Kālidāsa, as mentioned in section 4 is processed by the Sanskrit reader, the interface shows all the 6 967 296 000 possible solutions in a compact display. The display presents various choice points to the user, and is manageable in 17 clicks.

7 Conclusion

We have presented a new interface for interactive segmentation cum tagging of Sanskrit sentences. This technology is not limited to Sanskrit. It may be adapted to interactive feedback with a segmenter, tagger or parser, where sentences are presented as a finite collection of sequences of annotated word forms (lemmas). It may also operate at the generative morphology level, where words are presented as composition of morphemes.

This interface enables a human annotator to visualise a sentence as a sequence of words, readable from left to right in one compact hypertext page. Word forms are vertically aligned with the original input. This allows sharing of lemmas, and avoids cluttering of the visual display with redundant informations. Segments at a given offset are sorted in decreasing length order, which permits easy selection according to a heuristic of maximum overlap of segments with the input sentence. This heuristic, which tends to minimize the number of segments, is very often correct. Small word forms or morphemes, which agglutinate by chance into larger chunks of the input, get relegated as



satyambrūyātpriyambrūyānnabrūyātsatyamapriyampriyañcanānṛtambrūyādeṣadharmaḥsanātanaḥ
सत्यम्ब्रूयात्प्रियम्ब्रूयान्नब्रूयात्सत्यमप्रियम्प्रियञ्चनानृतम्ब्रूयादेषधर्मःसनातनः

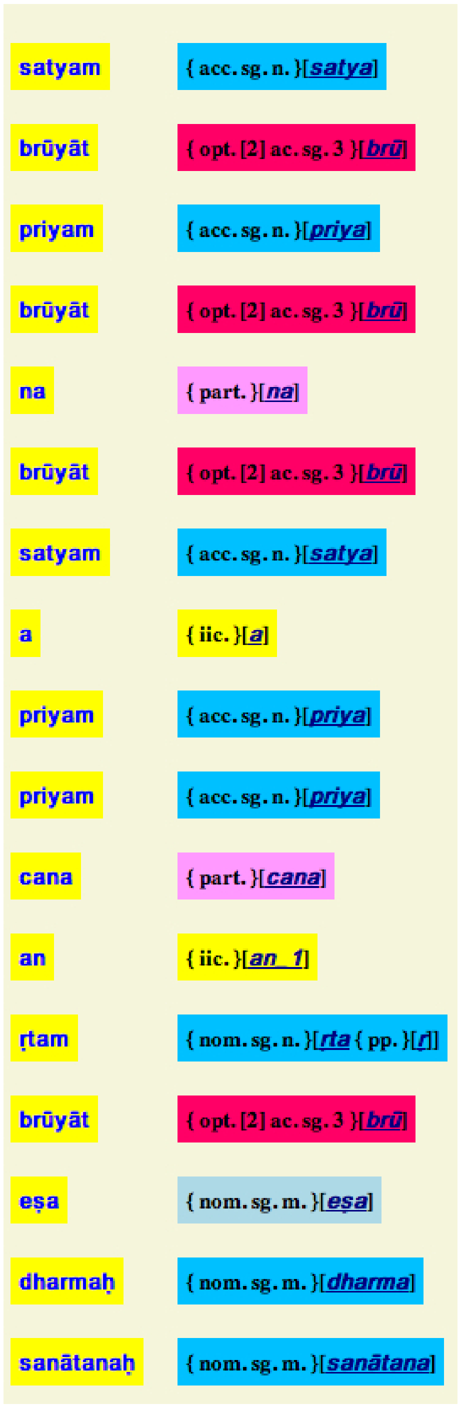
satyam	{ ◉ acc. sg. m. | ○ acc. sg. n. | ○ nom. sg. n. }[*satya*]
brūyāt	{ ◉ opt. [2] ac. sg. 3 }[*brū*]
priyam	{ ◉ acc. sg. m. | ○ acc. sg. n. | ○ nom. sg. n. }[*priya*]
brūyāt	{ ◉ opt. [2] ac. sg. 3 }[*brū*]
na	{ ◉ part. }[*na*]
brūyāt	{ ◉ opt. [2] ac. sg. 3 }[*brū*]
satyam	{ ◉ acc. sg. m. | ○ acc. sg. n. | ○ nom. sg. n. }[*satya*]
a	{ ◉ iic. }[*a*]
priyam	{ ◉ acc. sg. m. | ○ acc. sg. n. | ○ nom. sg. n. }[*priya*]
priyam	{ ◉ acc. sg. m. | ○ acc. sg. n. | ○ nom. sg. n. }[*priya*]
cana	{ ◉ part. }[*cana*]
an	{ ◉ iic. }[*an_1*]
ṛtam	{ ◉ acc. sg. n. | ○ nom. sg. n. | ○ acc. sg. m. }[*ṛta* { pp. }[*ṛ*]]
brūyāt	{ ◉ opt. [2] ac. sg. 3 }[*brū*]
eṣa	{ ◉ nom. sg. m. }[*eṣa*]
dharmaḥ	{ ◉ nom. sg. m. }[*dharma*]
sanātanaḥ	{ ◉ nom. sg. m. }[*sanātana*]

Figure 4: The interface for selecting unique tags from multi-tags

noise in the bottom of the display screen.

Fast recomputation of solutions respecting selection or rejection of a given segment achieves exponential convergence rate. Even for long sentences admitting over billions of solutions, the effect of these selections appear instantaneous. Selection mistakes may be fixed rapidly using the Undo facility. Morphological information is hidden in order not to clutter the screen, since appropriate use of colors for lexical categories usually suffices for making the right decision. In case of doubt, the annotator may click on any puzzling segment and get instantly its full lemmatization, including possible lexicon access for checking its meaning.

The main concept behind the data structure holding the display information is dynamic programming, i.e. sharing a tree structure into a dag, a standard technique in tabulated parsers. However, here the originality of our approach is that the tree structure is not the forest of parse trees, but the union of all possible segmentation solutions, in which sandhi justification has been erased. This representation allows an exponential saving, both in space (the displayed graph) and in time (the number of disambiguation operations).



satyam	{ acc. sg. n. }[*satya*]
brūyāt	{ opt. [2] ac. sg. 3 }[*brū*]
priyam	{ acc. sg. n. }[*priya*]
brūyāt	{ opt. [2] ac. sg. 3 }[*brū*]
na	{ part. }[*na*]
brūyāt	{ opt. [2] ac. sg. 3 }[*brū*]
satyam	{ acc. sg. n. }[*satya*]
a	{ iic. }[*a*]
priyam	{ acc. sg. n. }[*priya*]
priyam	{ acc. sg. n. }[*priya*]
cana	{ part. }[*cana*]
an	{ iic. }[*an_1*]
ṛtam	{ nom. sg. n. }[*ṛta* { pp. }[*ṛ*]]
brūyāt	{ opt. [2] ac. sg. 3 }[*brū*]
eṣa	{ nom. sg. m. }[*eṣa*]
dharmaḥ	{ nom. sg. m. }[*dharma*]
sanātanaḥ	{ nom. sg. m. }[*sanātana*]

Figure 5: Final tagging.

The main ideas of this interface have been reused for summarizing all possible dependencies between word forms in the dependency parser developed at the Sanskrit Studies Department of the University of Hyderabad[4]. This parser may be accessed as a second pass of our segmenter, leading to a smooth composition of the two processes -

[4]`http://sanskrit.uohyd.ernet.in/scl`

the user switches seamlessly between tagging and parsing. When to call the parser is actually an interesting trade-off. If we call it too early, it will just choke under the enormous number of possible taggings. On the other hand, should we use our manual interface till the obtention of a unique tagging, we would be loosing important opportunities of automation, since the dependency analysis will discard many inconsistent word combinations.

Our interface has been tested successfully by the Sanskrit Library team[5] for the annotation of a variety of classical Sanskrit texts. It was used as filter to an annotator's interface, with great savings in manual labour, for about 800 sentences of the *Pañcatantra.*

References

Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite-State Morphology*. CSLI Publications, The University of Chicago Press.

Sylvie Billot and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th annual meeting on Association for Computational Linguistics*, ACL '89, pages 143–151, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jay Earley. 1983. An efficient context-free parsing algorithm (reprint). *Commun. ACM*, 26(1):57–61.

Pawan Goyal and Gérard Huet. 2013. Completeness analysis of a Sanskrit reader. In *Proceedings, 5th International Symposium on Sanskrit Computational Linguistics*. D. K. Printworld.

Pawan Goyal, Vipul Arora, and Laxmidhar Behera. 2009. Analysis of Sanskrit text: Parsing and semantic relations. In Huet et al. (Huet et al., 2009), pages 200–218.

Pawan Goyal, Gérard Huet, Amba Kulkarni, Peter Scharf, and Ralph Bunker. 2012. A distributed platform for Sanskrit processing. In *Proceedings of COLING 2012*, pages 1011–1028, Mumbai, India. COLING.

Oliver Hellwig. 2009. SanskritTagger, a stochastic lexical and POS tagger for Sanskrit. In Huet et al. (Huet et al., 2009), pages 266–277.

Gérard Huet, Amba Kulkarni, and Peter Scharf, editors. 2009. *Sanskrit Computational Linguistics 1 & 2*. Springer-Verlag LNAI 5402.

Gérard Huet. 2005. A functional toolkit for morphological and phonological processing, application to a Sanskrit tagger. *J. Functional Programming*, 15,4:573–614.

[5]`http://sanskritlibrary.org/`

Gérard Huet, 2006. *Themes and Tasks in Old and Middle Indo-Aryan Linguistics, Eds. Bertil Tikkanen and Heinrich Hettrich*, chapter Lexicon-directed Segmentation and Tagging of Sanskrit, pages 307–325. Motilal Banarsidass, Delhi.

Gérard Huet. 2007. Shallow syntax analysis in Sanskrit guided by semantic nets constraints. In *Proceedings of the 2006 International Workshop on Research Issues in Digital Libraries*, New York, NY, USA. ACM.

Gérard Huet. 2009. Formal structure of Sanskrit text: Requirements analysis for a mechanical Sanskrit processor. In Huet et al. (Huet et al., 2009).

Girish Nath Jha, editor. 2010. *Sanskrit Computational Linguistics 4*. Springer-Verlag LNAI 6465.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20,3:331–378.

Amba Kulkarni and Gérard Huet, editors. 2009. *Sanskrit Computational Linguistics 3*. Springer-Verlag LNAI 5406.

Amba Kulkarni and K. V. Ramakrishnamacharyulu. 2010. Parsing Sanskrit texts: Some relation specific issues. In Jha (Jha, 2010).

Amba Kulkarni and Devanand Shukl. 2009. Sanskrit morphological analyser: Some issues. *Indian Linguistics*, 70(1-4):169–177.

Amba Kulkarni, Sheetal Pokar, and Devanand Shukl. 2010. Designing a constraint based parser for Sanskrit. In Jha (Jha, 2010).

Anil Kumar, Vipul Mittal, and Amba Kulkarni. 2010. Sanskrit compound processor. In Jha (Jha, 2010).

Emmanuel Roche and Yves Schabes. 1997. *Finite-State Language Processing*. MIT Press.

Peter Scharf and Malcolm Hyman. 2009. *Linguistic Issues in Encoding Sanskrit*. Motilal Banarsidass, Delhi.

A. Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational linguistics*, 21(2):165–201.

M. Tomita. 1985. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*, volume 8 of *The Springer International Series in Engineering and Computer Science*. Springer.