Formal Structure of Sanskrit Text: Requirements Analysis for a Mechanical Sanskrit Processor.

CONFERENCE PAPER · JANUARY 2008

DOI: 10.1007/978-3-642-00155-0_6 · Source: DBLP

1 AUTHOR:

Gérard P. Huet

National Institute for Research in Computer …

116 PUBLICATIONS **5,856** CITATIONS

Formal structure of Sanskrit text: Requirements analysis for a mechanical Sanskrit processor

Gérard Huet

INRIA Rocquencourt,
BP 105, 78153 Le Chesnay Cedex, France

Abstract. We discuss the mathematical structure of various levels of representation of Sanskrit text in order to guide the design of computer aids aiming at useful processing of the digitalised Sanskrit corpus. Two main levels are identified, respectively called the *linear* and *functional* level. The design space of these two levels is sketched, and the computational implications of the main design choices are discussed. Current solutions to the problems of mechanical segmentation, tagging, and parsing of Sanskrit text are briefly surveyed in this light. An analysis of the requirements of relevant linguistic resources is provided, in view of justifying standards allowing inter-operability of computer tools.
This paper does *not* attempt to provide definitive solutions to the representation of Sanskrit at the various levels. It should rather be considered as a survey of various choices, allowing an open discussion of such issues in a formally precise general framework.[1]
Keywords. Sanskrit, computational linguistics, finite-state machines, morphophonemics, dependency grammars, constraint satisfaction.

Introduction

We assume from the reader a minimal knowledge of the Sanskrit language and its grammar, some familiarity with general linguistics, and a general acquaintance with the use of computers in humanities. We shall evoke various formal structures used in current research in computational linguistics. Such material is of a mathematical nature, building on the last 40 years of research in mathematical logic (proof theory) and theoretical computer science (formal language theory, theory of computation, type theory, functional and constraint programming). Since familiarity with such formal methods is not assumed, and since their precise exposition would require more time than can be expected from most readers and more space than can be accommodated in a short paper, we shall skip technicalities and limit ourselves to rather sketchy discussions of their suitability for the representation and processing of Sanskrit text, while providing pointers to the literature for background reference.

[1] This material was presented at the Second International Symposium on Sanskrit Computational Linguistics, organized at Brown University in Providence by Peter Scharf in May 2008. Many thanks to Brendan Gillon, Amba Kulkarni and Peter Scharf for their remarks and corrections.

The general architecture of a software platform aiming at linguistic processing usually consists of processing modules which operate at various levels of representation of the target corpus, typically across linguistic layers (phonetics, morphology, syntax, semantics, pragmatics), in order to progressively massage raw input (speech or written text) into higher level abstractions where enough meaning has been extracted for the task at hand (markup, intelligent search, concordance computation, statistics extraction, posibly up to more sophisticated cognitive tasks such as question answering and translation). Such modules communicate with each other in complex ways, using various databases of lexical and grammatical information compiled from structured digital lexicons. How many representation levels are identified is often a matter of hot debate. In particular there is a tension between theoretical abstractions from general linguistic models and pragmatic methods tailored to specific language families. We propose in this paper to identify two main levels for Sanskrit text structure, respectively called *linear* and *functional* level.

The first one, as its name suggests, concerns the *linear structure* of Sanskrit text: discourse consisting of successive utterances, sentences made up of streams of word forms, words themselves represented as strings of phonemes on which morpho-phonetic transformations operate, etc. At this level we find Sanskrit text represented as sequences of inflected words (*padapāṭha*), as sandhied continuous utterances (*saṃhitapāṭha*), or in a mixed manner as sequences of chunks facilitating the recognition of word boundaries while preserving the phonetic stream; at a finer grain of linear analysis we get marked-up text represented as sequences of morphemes tagged with morphological features. We shall discuss the feasibility of the synthesis of such descriptions by semi-automatic computer tools, and consider their suitability as a substratum layer of Sanskrit digital libraries fit for philological or pedagogical applications.

The next level concerns the *functional structure*, where various linguistic and logic formalisms for hierarchical structure (the main ones being phrase-structure syntax trees, dependency graphs, and proof nets for categorial grammars) are compared and their suitability for Sanskrit discussed. A formalisation of Gillon's notation for the syntactic analysis of Apte's examples is attempted, and alternatives are discussed. At this level we also find functional representations with thematic roles à la *kāraka*, encompassing Kiparsky's analysis of Pāṇini's grammar, and we propose to view those as enrichments of dependency structures. The articulation of the two levels, specially concerning the representations of compounds, is examined.

The computational implications of such design choices are discussed. Current solutions to the problems of mechanical segmentation, tagging, and parsing of Sanskrit text are briefly surveyed in this light. An analysis of the requirements of linguistic resources such as banks of lemmatized inflected forms or tree-banks used for statistical training is provided, in view of promoting the development of standards allowing inter-operability of computer tools.

1 The linear structure

We first discuss the representation of Sanskrit sentences as a (possibly annotated) list of contiguous chunks, down to words, morphemes, syllables and phonemes.

1.1 Phonemic versus syllabic representations

First of all, we assume that the linguistic data is presented in a discretized fashion, that is we do not consider representations of continuous media such as phonetic waves, and we assume that the data is presented to us as streams of discrete characters from a finite alphabet. The main choice here is between streams of phonemes and streams of characters from some written representation. Roughly speaking, the first one is the phonemic representation, the second one the syllabic representation. The raw representation of a Sanskrit text is usually given as a syllabic representation, say in the Devanāgarī script. Digitalisation of this printed representation, by optical character recognition (OCR) or by direct keyboard input, produces strings of characters, some of which representing syllables, some of which representing punctuation symbols (*danda*, *avagraha*, spaces), possibly some extra characters such as numerals. These characters are represented in the computer in some low-level technology which is actually of little concern to us in this general discussion; the current technology uses Unicode values (code points) represented in some encoding scheme (such as UTF-8) as bytes of data (i.e. contiguous chunks of 8 bits in computer memory).

The set of phonemes of classical Sanskrit has been fixed since the seminal work of Pāṇini 24 centuries ago as a discrete collection of roughly 50 phonemes[2], which we shall simply represent as the first 50 natural numbers. These phonemes have been traditionally represented by Western sanskritists as romanized symbols with diacritics, such as *ā*, *ṭ*, *ś*, with a collating order inherited from ancient Sanskrit phonetic treatises, corresponds to the standard integer ordering of our integer codes. Actually, one small difficulty is often swept under the rug, concerning the nasalisation *ṃ* (*anusvāra*) and the final aspiration *ḥ* (*visarga*). We shall explain this point below. Finally, strings of romanized symbols are represented in the computer as transliterations, i.e. strings of ASCII characters. Popular transliteration notations are the Velthuis scheme, the Kyoto-Harvard scheme, the WX scheme used in Hyderabad and Tirupati, and the SL scheme used in the Sanskrit Library effort at Brown University. Appendix A gives the list of phonemes from classical Sanskrit under these various encodings. Many other encodings have been proposed in the past (ITRANS, CSX) but seem to become progressively obsolete.

The syllabic representation is not appropriate for basic computational linguistics tasks, and should be reserved to the Devanāgarī interface, since its granularity does not allow word segmentation of continued utterances, where syllables at word junction are often merged by sandhi. This problem is aggravated by the complex ligatures of the full Sanskrit Devanāgarī character set.

[2] in the broad sense of the term described by Scharf and Hyman [44] Section 6.1.6.

Furthermore, one should carefully distinguish between syllabic representation and Unicode representation of text. Unicode ought to be reserved for robust external exchanges, where one does not make explicit the encoding of ligatures; rather, successive consonants may be interspersed with *virāma* codes, and it is the rendition engine of the printing host which decides whether fonts available on this host may support a given ligature or not. Actually, Unicode points may be used for representing phonemes, syllables, printable letters, or control characters, and the ensuing ambiguity makes this low-level medium unfit for precise linguistic computation. Even if one chooses to represent the syllabic level, for prosody purposes for instance, a specific precise representation separate from Unicode ought to be used.

From now on, we shall assume that the basic text representation is phonemic. Thus raw output in syllabic representation has to be expanded in the finer phonemic representation. This looks straightforward enough, since every syllable is non-ambiguously translatable into a string of phonemes. However, difficulties occur when one wants to use spacing, or special symbols such as avagraha, in order to disambiguate sandhi and have a representation which makes explicit the word boundaries. Thus raw input is generally broken into *phonemic chunks* separated by blank space, each chunk formed of Sanskrit words merged with sandhi, so that a given word (*pada*) is entirely contained within one chunk. We shall discuss in the next section various representations of Sanskrit text as lists of phonemic sentences, each phonemic sentence being represented as a list of phonemic chunks.

1.2 Sandhi

We have been a little vague in referring to the process of *sandhi*, i.e. transformation of contiguous phonemic strings. Actually, there are at least two rather distinct such processes, traditionally called respectively *external* and *internal* sandhi by Western linguists. Basically, external sandhi is the transformation that occurs at word boundaries, and when merging a raw nominal stem (*prātipadika*) with an inflected form in order to compose a compound form. Whereas internal sandhi occurs between morphemes at a deeper level of morphological formation, either for derivational morphology (*kṛt* formation of nouns and participles from roots, *taddhita* secondary word formation), or for inflectional morphology (declension of substantival and adjectival stems, given gender case and number attributes, and verb conjugation given various morphological parameters). Actually, whereas external sandhi is more or less unique, internal sandhi is more a family of transformations under various circumstances. Indian grammarians do not use this terminology, which does not correspond to a precise Pāṇinian notion. Roughly speaking, the different varieties of internal sandhi correspond to various situations where a context of previous transformations determines the possible application of further rules. However, it is not usually directly possible to map a given morphological process (say conjugation) into a unique entry point in Pāṇini's *sūtras*. Furthermore, Pāṇini's *economy principle* forces the sharing of many transformation rules, which may be used for external sandhi in certain

contexts, or to varieties of internal sandhi in others, making the distinction non relevant to Pāṇinian processes.

We believe nonetheless that the distinction between internal and external sandhi is conceptually important, since their computational properties are very different. First of all, in terms of modern formal language theory, external sandhi can be defined as a *rational* transduction between phonemic strings, expressible as a regular expression. In other words, external sandhi, and its reverse operation sandhi splitting, is a local transformation which may be implemented by a finite machine (more precisely, a finite-state transducer) operating on strings of phonemes. This is not the case for internal sandhi, which has long-distance effects such as retroflexion, not easily amenable to finite state techniques. Furthermore, internal sandhi does not really operate simply on phonemic strings such as stems, but also on morphological affixes which use intermediate control characters before their erasure in the terminal surface representations. The fact that Pāṇini used actual phonemes to encode such meta-theoretic control characters (*pratyāhāra* markers, etc.) is an artifact of his system, where meta notation is immerged within the object language (the *signifiants* or phonetic realisations). There is no actual necessity to use these encodings, and it may be better to have a clearer distinction between morphologico-phonetic markers (meta-linguistic symbols) and actual phonemes.

The next level of representation of Sanskrit text is as a list of *words*, possibly annotated by the (external) sandhi rules applicable at their junction. Here by word we mean either an indeclinable form (some adverbs, prepositions, conjunctions and other particles), a declined noun stem, or a conjugated verbal form. Actually, this representation is too coarse, since it is not finite, in the sense of being finitely generated from a fixed set of primitive forms, since compounds may compose individual stems (*prātipadika*) to an arbitrary level. In order to obtain a finitely based representation, one has to break up compounds. Declined compound forms may be split as a stem form followed by a shorter declined form, which may be itself split down to a non-compound form, of which there is a finite vocabulary. Similarly an indeclinable compound (*avyayībhāva*) may be split into an adverbial prefix and a stem form. Such splitting of compounds may be done mechanically by the same process of (external) sandhi analysis (*viccheda*) that may be used to break chunks of sentences into words.

By the same token, one may decompose verbal forms into a sequence of preverb particles, followed by a root form, although in certain cases a more complex glueing may be involved, with possible retroflexion, such as *adhiṣṭhā* from preverb *adhi* and root *sthā*. Furthermore, one cannot hope to obtain such effects through just sandhi rules, which operate on contiguous morphemes, as remarked by Kiparsky in [29]. For instance, consider the perfect form *abhitaṣṭhau*: there is no hope to effect the retroflexion induced by prefixing *abhi* to the root *sthā* by a local interaction of *abhi* with the reduplicated passive form *tasthau*. Thus the interaction has to be predicted by storing forms such as *taṣṭhau*, accessible only through the relevant preverbs. Similarly, special forms have to be cooked

for the preverb ā- interacting with forms of roots starting with *i*, ī, *u*,ū, in order to segment properly say *ihehi* as *iha-ā-ihi*, while rejecting the wrong **ihaihi*.

There is no space to describe these mechanisms in detail, we just remark that modular finite-state methods [28] allow to encode such controled interactions in state transitions rather than in the conflict resolution of complex morpho-phonemics rules.

1.3 Forms of surface representation

At this level of analysis of a Sanskrit sentence, we obtain a surface representation which is linear in the sense of being a uni-dimensional list of items from a finite list of atomic forms. Let us give an example, taken from [29]. We give various forms of linear representation.

devanāgarī:

वनाद्ग्राममद्योपेत्यौदन आश्वपतेनापाचि

romanized continuous text (*saṃhitāpāṭha*):

vanādgrāmamadyopetyaudana_āśvapatenāpāci

saṃhitāpāṭha (in VH transliteration):

`vanaadgraamamadyopetyaudana_aa"svapatenaapaaci`

list of word forms in terminal sandhi (*padapāṭha*):

`vanaat graamam adya upetya odana.h aa"svapatena apaaci`

Chunk form (identical to the one given in [29]):

`vanaad graamam adyopetyaudana aa"svapatenaapaaci`

Surface representation with explicit sandhi annotation:

`vanaat`⟨t|g→dg⟩`graamam`⟨⟩`adya`⟨a|u→o⟩`upetya`⟨a|o→au⟩`odana.h`
⟨a.h|aa→a_aa⟩`aa"svapatena`⟨a|a→aa⟩`apaaci`

One may check that this surface representation is exactly the padapāṭha with spaces replaced by explicit sandhi annotation, and that the saṃhitāpāṭha is obtained from it by effecting the sandhi rewriting.

Fully segmented form (breaking down preverbs from root forms):

`vanaat`⟨t|g→dg⟩`graamam`⟨⟩`adya`⟨a|u→o⟩`upa`⟨a|i→e⟩`itya`⟨a|o→au⟩`odana.h`
⟨a.h|aa→a_aa⟩`aa"svapatena`⟨a|a→aa⟩`apaaci`

Another possible one:

`vanaat`⟨t|g→dg⟩`graamam`⟨⟩`adya`⟨a|u→o⟩`upa`⟨a|aa→aa⟩`aa`⟨aa|i→e⟩`itya`⟨a|o→au⟩
`odana.h`⟨a.h|aa→a_aa⟩`aa"svapatena`⟨a|a→aa⟩`apaaci`

The two fully segmented forms differ only by the preverb of the absolutive form *upetya*, obtainable as *upa-itya* or *upa-ā-itya*. Remark that in this distinction the verb *upe* is ambiguous in all forms. This example is interesting in several ways. First of all, it points to the fact that preverbs are glued by sandhi one at a time, left to right; it would be wrong to affix the preverb *upa* to the absolutive *etya* of verb *e* = *ā-i*, since this would lead to the wrong form **upaitya*.[3] This

[3] Unless a special rule is formulated to account for the exception as Pāṇini did in 6.1.95 *omāṅgoś ca* (P. Scharf).

has as consequence that forms of *upe=upa-ā-i* are undistinguishable from corresponding forms of *upe=upa-i*, at least at this level of phonemic distinction. This does not mean that the two verbs ought to be considered the same, since they have different shades of semantic meanings; consider *ihehi* (come here), obtained as *(iha-ā)-ihi*, and not as *iha-ihi*. Indeed Monier-Williams gives two distinct entries for *upe*. And this distinction justifies our explicit consideration of the fully segmented form, since the surface representation (or padapāṭha) does not distinguish the two.[4] In this particular example, it is the last form which represents faithfully the meaning, according to the analysis in Kiparsky [29].

1.4 Splitting compounds

The fully segmented form ought to make explicit the decomposition of compounds as well, in order to satisfy our requirement that it be finitely generable (from a finite lexicon of basic stems). Thus a compound form such as *tapovanavasitasukhā* ought to be broken up as *tapas-vana-vasita-sukhā*, possibly with sandhi-annotation between the components.

Several remarks are in order. First of all, note that the semantic understanding of such a compound demands a more complex representation than just a left-to-right list of stems. In general, a binary tree indicating the dependency ordering is needed,[5] and the more standard compounds get their main constituent to the right, such as *(((tapas<vana)<vasita)<sukhā)* in Gillon's notation. Whereas a simple list such as *tapas-vana-vasita-sukhā*, parsed from left to right, has a derivation tree going in the other direction: *(tapas-(vana-(vasita-sukhā)))*, and its sublists are not proper constituents. We should not be surprised at this. The sandhi splitting of compounds does not reflect its semantics or even its syntax, just the morphology rule stating inductively that a compound form is obtained by prefixing a bare non-compound stem to an inflected form (possibly itself compound). A more general rule, closer to the semantics, would express that the left component of a compound is a possibly compound stem (*prātipadika*). But this more general rule expresses a non-regular construction requiring non-linear representation (i.e. context-free derivation trees), and we do not want to consider it at this stage. In other words, we only keep of compound morphology what pertains to its finite-state character, and leave the rest to the non-linear structure, which accommodates syntax and possibly semantics, and which will support a better structured representation of compounds. It is exactly

[4] While it is common for editors of classical Sanskrit texts not to separate preverbs from verbs even when indicating other word divisions, and for some lexicographers, such as Monier-Williams and Apte, to list the preverb-verb unit as a headword, Vedic padapāṭhas do in fact separate preverbs and verbs as independent words. For example, RV 1.124.8 saṃhitāpāṭha *ápaity* is analyzed *ápa — eti* — in the padapāṭha. The nominal form *prétim* in RV 1.33.4 is analyzed — *prá'itim* — separating the preverb with an avagraha indicated in Roman by an apostrophe (P. Scharf).

[5] Pāṇinian compound formation rules (2.2.23-24 *śeṣo bahuvrīhiḥ*, *anekam anyapadārthe*, and 2.2.29 *cārthe dvandvaḥ*) allow the derivation of bahuvrīhi and dvandva compounds from multiple constituents (P. Scharf).

at this point that we may state a clear requirement of the interface between the linear and the functional levels: the linear structure of compounds ought to be an embedding of its functional representation. This is accomplished simply by requiring that the list of stems be the *frontier* of the binary tree. For instance, in the above example, the list *(tapas-(vana-(vasita-sukhā)))* of phoneme strings is the frontier of the binary tree *(((tapas<vana)<vasita)<sukhā)*. More generally, this will simply state that the frontier of the functional representation should be the sequence of utterances, the phonemic realisation. In other words, we are requiring the functional representation to be sequentialised on the enunciation – no movement, no erasure, no duplication. This requirement is reasonable, it amounts to requiring that the functional representation be a mark-up of the enunciation.

Another remark is that it may be undesirable to split compounds which are used as proper names, such as *devadatta*, or *rāmāyaṇa*, since their meaning is in general non-compositional. However, an automated tagger will not be able to recognize such proper names, since they are not distinguished by capitals or other marks. At best it might try not to split a compound explicitly presented in the lexicon as a proper name, but even then it will not be able to distinguish a genuine compound *devadatta* "god-given" from the proper name *Devadatta* "Theodore", anymore than it would be able to discern the color black from the name *Kṛṣṇa*. We note with interest the use of capitals as initials of proper names in Gillon's presentations or in the Clay Sanskrit Library series of books. We have adapted the same convention in our lexicon, and for this reason use for transliteration a reduced set from Velthuis'; e.g. *A* is not allowed as substitute for *aa*.

Finally, it is not clear whether privative compounds (such as *a-vyakta*), and compounds whose left component is a particle such as *vi-*, *sa-*, *su-* or a preposition ought to be split, even though their meaning is in general compositional. Such compounds may be considered under derivational morphology rather than compound formation proper, and their analysis left to lexicon lookup rather than sandhi-splitting. This is debatable of course, and ultimately too naive, since e.g. privative *a-* occurs as initial of arbitrarily long compounds. Similarly there is a trade-off between storing explicit entries in *-tā*, *-tva*, *-tṛ*, *-vat*, *-mat*, etc. in the lexicon, versus leaving them implicit from derivational morphology considered as generative. We remark *en passant* that finite state automata allow to represent implicitly an infinite number of forms, in contrast with say relational database technology, where the stored data is necessarily finite.

1.5 Automated synthesis of surface representation

This discussion calls for closer examination of the various mappings between the representations discussed above, in order to understand how to build mechanical tools that may transform one into another.

We already remarked that the saṃhitapāṭha may be obtained from the padapāṭha by applying (external) sandhi.[6] This transformation is deterministic (i.e. may be implemented by a function), except minor details such as doubling of *n* before vowels or the doubling phenomena described in [33]. It becomes fully deterministic of course if we annotate the padapāṭha with the relevant sandhi rules.

In the reverse direction, we cannot hope for a functional transformation, but since external sandhi is a rational relation [5], its inverse (*viccheda*) is too. Furthermore there is a finite number of strings which are the viccheda of a given saṃhitapāṭha, and thus all solutions may be enumerated by a terminating finite state process. Details are given in [25]. This algorithm is complete, assuming that all forms occurring in the sentence are in the database of inflected forms, i.e. that the lexicon used for morphology generation contains the vocabulary of the analysed sentence.

For instance, in the Sanskrit engine platform, I marked the stem *nṛ* as non-generative, since its nominative form *nā* was overgenerating wildly in the segmentation phase, every *nā* syllable having its chance as a nominal segment. Thus I cannot properly segment the sentence *ekonā viṃśatirnāryaḥ krīḍāṃ kartuṃ vane gatāḥ* 'one man and 20 women went to play in the woods'. But in some sense this example vindicates the decision, since this sentence is very likely to be misunderstood, being a sort of garden path sentence, as the linguists say – you are fooled by the likeness with *ekonaviṃśati* to believe that only 19 women went to the woods, and you are then puzzled by the second half of the *prahelikā*: *viṃśatirgṛhamāyātāḥ śeṣo vyāghreṇa bhakṣitaḥ* '20 came back, and the remaining one was eaten by the tiger'. The moral of this *vakrokti* is that *nā* is problematic, even for human locutors[7] since its frequency as a syllable so much exceeds its frequency as a form that it is unreasonable to handle it with the standard combinatorics, and it must be treated as an exception somehow. A similar problem occurs with *āya*, whose declined forms clash with genitive/ablative forms of substantives in *ā* and dative forms of substantives in *a*, and thus cause overgeneration. Such examples raise the general question of ambiguity between generative and inflexional morphology.

Actually, this viccheda algorithm is not usable in practice as a stand-alone segmenter without refinements, since the number of segmentations may be exponential in the length of the sentence, and thus the problem is computationally intractable for large continuous chunks of saṃhitapāṭha. Thus for the example sentence discussed in the section 1.3, with no blank at all in the input, the number of possible segmentations exceeds 10000. The standard solution is to break the input at appropriate points in order to give hints. Of course if one goes all the way to preparing by hand the padapāṭha the problem becomes trivial,

[6] We ignore exceptions, such as presented by Staal [46] p. 22, where the padapāṭha differs in the word order of the saṃhitapāṭha in a verse of the *Ṛgveda*, in order to restore the proper name *śunaḥśepa* split by particle *cid*.

[7] Admittedly, the machine would not be fooled on that particular example, because of the discrepancy between *a* and *ā*.

involving only an iteration of lookups in the database, and the only remaining non-determinism reduces to proper homonymy.

Somewhere in between, we may obtain a useful algorithm, by breaking the input with spaces at a few chosen points, like in the "chunk form" considered above. A remark is in order here. The chunk form is definitely *not* the padapāṭha, and indeed the two forms are mutually inconsistent. For instance, a chunk ended by *o* will generally correspond to a terminal sandhi form *aḥ*, as opposed to being the terminal sandhi form in *o* of e.g. the vocative of an *u* stem. Also a chunk ended in *a* in a hiatus situation will generally correspond to a terminal form in *aḥ*, *e* or *o*. Finally an avagraha may be used to signal the elision of an initial *a* after *e* or *o*.

Actually, there does not seem to exist a commonly agreed chunk representation, specially concerning the use of the avagraha. We have evolved an ad-hoc convention for chunk representation, which attempts to depart from the saṃhitapāṭha only in inserting blanks at points which must be word junctions. This seems pretty close to current digitalised Sanskrit texts, and is consistent with our sandhi splitting algorithm. The precise algorithm is given in Appendix B, and may be read as a specification of the chunk representation preprocessing.

This algorithm accepts input such as *yad iha asti tad anyatra yan neha asti na tat kvacit*, in order to pass it on as a list of final forms, to the sandhi viccheda segmenter proper. As analysed in [25], this problem is solvable by a finite-state transducer, implementing the inverse of the sandhi relation. Furthermore, for any input, there is a finite set of solutions. For instance, the transducer on the above input will infer the correct padapāṭha segmentation *yat iha asti tat anyatra yat na iha asti na tat kvacit*. However, 19 other segmentations are possible, since e.g. *neha* is a form of root *nah* in the perfect tense, and admits a total of 5 decompositions. Similarly *kvacit* admits 4 decompositions, amazingly enough. Since the combinations multiply, we get 20 decompositions. At this point the reader may wonder whether such a parser is usable in practice. The answer is that in this initial phase it is only a segmenter, piping into a semantic filter, which then attempts to build a satisfiable functional representation of each segmentation, using a kāraka-fitness test. In this precise example, this filter rejects all 20 segmentations except two, the good one and a parasitic one. Thus there is no silence, and the noise is tolerable: the precision is 94% in this case. It is rather surprising that on such an input, where most word divisions are explicitly indicated, there is still room for 20 solutions. It is interesting that if all spaces are removed, and we give the input in full saṃhitapāṭha as *yadihāstitadanyatrayannehāstinatatkvacit*, there are now 87 potential segmentations. In this case, the filter keeps 9, the good one and 8 parasitic, with precision 91%. This gives a measure of the tradeoff between undoing sandhi by hand and leaving it to the machine. If you give proper hints, a few spaces in a moderately long sentence, you get a manageable set of proposed solutions to inspect in order to select the right parse.

The example sentence above: *vanād grāmam adyopetyaudana āśvapatenāpāci* currently yields 928 segmentations, among which only 2 are kept, the right one

being 1st. The other one is due to the semantic ambiguity of *upetya* (with or without preverb *ā*).

1.6 Normalisation

It may be appropriate at this point to discuss the status of anusvāra coming from internal sandhi. Typically, the word *sandhi* itself comes from (obligatorily) compounding the preverb *sam* with the *kṛdanta* form *dhi* derived from the root *dhā*. And indeed the orthography *saṃdhi* is perfectly legitimate, anusvāra in this case being an alternative to nasal homo-phonic to *dh*, i.e. *n*.[8] Are we going to need to store redundant forms *saṃdhi* and *sandhi* in our lexicon, in order to store duplicate forms of all their inflected forms? In order to recognize them in all possible combinations in the input, yielding a potential exponential number of completely redundant segmentations? Of course not. We need store only one canonical form of the stem, *sandhi*[9] one canonical form of each of its inflected forms such as *sandhiḥ*, and have chunks *saṃdhiḥ* in the input normalized in their canonical form *sandhiḥ* at preprocessing time. Note that this is the right normalisation direction, since it would be incorrect, using the reverse convention, to write e.g. *sasañja* as **sasaṃja*.[10] Of course "genuine" anusvāra remains in e.g. *saṃsāra*. Symetrically, we propose to normalize *ahaḥsu* as the equivalent *ahassu*.

This normalisation of the input is one of the tricky technical details. Note that it implies a reconsideration of sandhi rules, since for instance one choice of sandhi of *sam* and *diṣṭa* produces the un-normalised *saṃdiṣṭa*. Thus there is a dilemma here. Either the lexicon has an explicit entry for the participle, normalised as *sandiṣṭa*, or the lexicon attempts to generate this form as composed of preverb *sam* and root participle *diṣṭa*, in which case the sandhi rule to be used should be *m+d* →*nd* rather than *m+d* →*ṃd*. This technical issue is tricky, since we do not want our morphological generator to overgenerate by allowing variations inconsistent with the normalisation of the input. Thus by the economy principle we prefer to interpret sūtra 8.4.59 by "is preferably replaced" rather than "is optionally replaced".

More generally, the treatment of participles is a thorny issue. There seems to be no good reason not to restrict the explicitly represented participles to the roots, and to get the other ones by preverb affixing, in the same way that we get finite verbal forms from finite root forms. But here two issues come into play. One, there are so many participial forms, and their full inclusion leads to many

[8] The phonetic alternation of *n* and *ṃ* is due to the fact that pada-final anusvāra is optionally replaced by the nasal homorganic with the following stop or semivowel in accordance with 8.4.59 *vā padāntasya*. The preverb *sam* is a pada which, as an upapada, is compounded obligatorily with the kṛdanta *dhi* in accordance with 2.2.19 *upapadam atiṅ* (P. Scharf).

[9] Note that Monier-Williams takes the opposite choice, and chooses *saṃdhi* for the corresponding entry of his dictionary.

[10] Non-pada-final anusvāra is obligatorily replaced by the nasal homorganic with the following stop or semivowel in accordance with 8.4.58 *anusvārasya yayi parasavarṇasya* (P. Scharf).

ambiguities. A typical one is the ambiguity between a present participle in the locative singular such as *bhavati* and the 3rd person singular of the corresponding present conjugation. Secondly, generating participial forms from roots is a special case of the first level of generative nominal morphology (*kṛdanta*), and thus it would seem logical to include more, or all *kṛdanta*s, or even *taddhita*s, and this looks like opening Pandora's box, with overgeneration by unattested forms on one hand, and complex internal sandhi inversion problems on the other.

1.7 Lemmatisation

We are not done yet with the linear representation, since now we may invert morphology on the individual elementary forms. Inverting morphology is not hard, if forward morphology has been already implemented, since the lexicon may be compiled into all possible forms, and this database of basic forms be stored as persistent dictionaries where they can be looked up by a stemmer. This assumes the existence of lexicons having sufficient coverage of the vocabulary of the target corpus. This requirement is not so severe, in view of the fact that our phrasal segmenter segments compounds into their constituents, so recursively only root words have to be lexicalized, besides a finite collection of irregular compounds. Besides, there exist now XML versions of the Monier-Williams dictionary,[11] and this is certainly enough to cover a significant part of the corpus. Actually, it may be too large for the task. The problem is the generation of verbal forms from such a huge dictionary. Such a dictionary covers well the nominal forms, but verbal forms have to be generated; all that is explicit are a few representative forms of a few tenses. Worse, most participles are missing, only past passive and a few passive future participles are explicitly listed. The various forms of the present participles, active past participles, and perfect participles are usually omitted as implicitly understood.

Thus forward morphology has to be implemented, not only for nouns, which is easy, but for verbs, a much harder task. This process may be more or less Pāṇinian, so to speak. For instance, in my own system, I more or less follow Pāṇini for internal sandhi, but generate paradigm tables for computing inflection in a rather ad-hoc manner percolated from mostly Western grammar descriptions. What I mean by Pāṇinian for internal sandhi is that I go through operations which mimick Pāṇini's sūtras, but using some definite choice of deterministic computation by specific rules in a sequential manner. In doing that, I allow myself the freedom of deciding which rule is *siddha*, but then everyone is in the same situation, isn't it? You have to interpret the vārttikās, decide on the proper scoping of exceptions, interpret priority of rules by *siddha* or other considerations, etc. So for me being Pāṇinian is more a reference to the spirit of a genious computational linguist precursor. And I allow myself to abstract from the straightjacket of an encoding of computation in terms of conditional string

[11] such as the one resulting from the collaboration of the Cologne Digital Sanskrit Lexicon project and the Digital Sanskrit Library Integration project at Brown University.

rewriting systems. We have general purpose programming languages, with appropriate data structures, we know how to separate clearly object data (phonemes) from meta-notation (Sanskrit phonemes used as control codes such as the markers of *pratyāhārās*).[12] Should Pāṇini be reborn as a computational linguist of our times, he would use modern tools and methodologies, of course, frame his description of the language in terms of modular high-level processes, and leave the production of compact machine code to an optimizing compiler.

One decision which may be considered cavalier with Pāṇini was to split in two the phoneme *j* in order to make internal sandhi deterministic in view of the different behaviours of say *mṛj* and *bhuj* in *kta*-suffixes, cf. their past participles respectively *mṛṣṭa* and *bhukta*. Similarly *h* has three variants, one for *duh*-like roots, one for *lih*-like roots and one for *nah*-like roots, (compare participles *dugdha*, *līḍha* and *naddha*).[13] This story is told in [27]. Being *mṛj*-like or *duh*-like is lexicalized.

The method sketched above is lexicon-based. It will fail to lemmatize forms which are not generated from the morphological closure of the root entries of the digitalized lexicon. An easy analysis may account for all irregular compounds which are explicitly listed in the lexicon. Since these irregularities are hopefully non-productive, the lexicon may be made complete since there are a finite number of attested ones. Regular compounds generable from the lexicon, at any depth, will be taken care of by the segmenter, as explained above. But there will be silence for any other form.

There exist heuristic methods to infer root stems from forms guessed as inflexions of potential words. These heuristics may be tuned by optimization on a reference tagged corpus; this is statistical training. Such heuristics allow one to lemmatize a text automatically without the need for a complete lexicon, and indeed may be used to provide suggestions for lexicon acquisition. Such methods have been used successfully by [37]. However, one should understand that Sanskrit morphology is too rich for this method to be complete, since there is a heavy non-determinism in inverting internal sandhi; for instance, should a retroflex *ṇ* or *ṣ* be reverted to *n* or *s* is not an obvious decision, and it is not clear that statistical training is sufficient in the absence of large-scale reference corpora. Furthermore, this method works pretty well when the text is presented in padapāṭha form, but it is not clear that it extends easily to chunks. In particular, external sandhi would have to be undone, but this time in a right-to-left direction, while the method given in [25] works left-to-right, on finite transducers which are compiled from the lexicon of inflected forms, so a different technology would have to be used.

[12] Abbreviatory terms formed by taking the first element of a list together with a marker placed at the end of the list. See the article by Amba Kulkarni in this volume (P. Scharf).

[13] Since *h* derives historically from any of the voiced aspirated stops in Indo-Iranian (P. Scharf).

1.8 Tagging

Once lemmatization is understood, we may proceed to the final stage of our linear representation, the mark-up of the text by its morphology: each inflected form is marked with its lemmatization.

Let us give as example what our own tagger returns on the example:
`vanaad graamam adyopetyaudana aa"svapatenaapaaci`

```
[ vanaat       { abl. sg. n. }[vana]     <>]
[ graamam      { acc. sg. m. }[graama]   <>]
[ adya         { adv. }[adya]            <a|u -> o>]
[ upaa - itya  { abs. }[upa-aa-i]        <a|o -> au>]
[ odana.h      { nom. sg. m. }[odana]    <.h|aa -> _aa>]
[ aa"svapatena { i. sg. m. }[aa"svapata] <a|a -> aa>]
[ apaaci       { aor. [1] ps. sg. 3 }[pac]
```

A few remarks are in order. The notation `<a|o -> au>` indicates sandhi. It may be omitted. The tags for substantival forms like *vanāt* are pretty obvious. Such a tag has an abstract form, such as
`<Subst case=5 number=1; gender=3; stem="vana">`, we pretty-print here an ad-hoc sugared concrete form with braces. Braces are convenient; they may be used as a set notation for *multitags*. The above example is a lucky one, because it happens that there is only one morphological production of each form. In general, ambiguities in morphology will produce multitags, like for the form *vanebhyaḥ*, which lemmatizes as `{ abl. pl. n. | dat. pl. n. }[vana]`. Actually there is even a second level of non-determinism, if we want to discriminate between homonyms, but I shall not discuss this, since it presupposes that the approach is lexicon-based.

The tag for the finite verbal form *apāci* is similarly non-mysterious, the numeral 1 standing for the aorist formation type (from 1 to 7). Similarly, the present system formations are indexed by their *gaṇa* (from 1 to 10), and passive future participal forms are similarly indexed by 1, 2 or 3, according to their formation suffix *-ya*, *-anīya*, or *-tavya*.

This extra information is important to relate modern linguistic abstractions, reflected in the tags' algebra, and actual entry points in Pāṇini's grammar. Of course a purely Pāṇinian system will replace the tags altogether by their full simulation path in the grammar. This must be understood as the problem of interface of the inflected forms with the grammar component, and with proper design a standard set of tags ought to allow interoperability between a Pāṇinian generator and a paradigm-driven one, there is no deep incompatibility between the two approaches.

A complete discussion of verbal tags is a complex matter, and their standardization will need extensive debate. For instance, passive forms may be classified as forms of the root in passive voice (in a system with 3 recognized voices, active/*parasmaipada*, middle/*ātmanepada* and passive), the option currently chosen in my system, consistent presumably with a Pāṇinian treatment, as opposed to Whitney's treatment of passive as a secondary conjugation. The granularity

of the morphological notation is also questionable. For instance, we refine the aorist tag by indicating its mode of formation (here 1 among the 7 forms of aorist), since this information is lexicalized; not all roots admit the 7 forms, so only listed modes are generated. Similarly, forms of the present system are tagged by the family class [*gaṇa*]. This is important for verbs which do not have the same regime in the various classes; e.g. the root *pac* is transitive in class 1 (*pacati*, *pacate*) and intransitive in class 4 (*pacyate*), and this information is crucial for the kāraka analysis which we shall discuss in Part 2 below.

The remaining tags concern the indeclinable forms *adya* and *upetya*. Here we make the choice of classifying such indeclinable forms according to their "part of speech" category, like `adv.` for the adverb *adya* or `conj` for the particle *ca*. The exact granularity of such categories is of course open to discussion. The form *upetya* is recognized as the absolutive of the verb *upe*, obtained from the root *i* by the sequence of preverbs *upa* and *ā*. This segmentation is effected by the segmenter, so that only forms of the roots have to be generated, and this is true as well of finite forms. Actually this example is a tricky one. Note that sandhi of preverbs *upa* and *ā*, leading to the form *upā*, has to be effected before the sandhi to the absolutive form *itya*. Otherwise, glueing the preverbs one by one in the etymological order would lead first to *etya*, then to the wrong **aitya*. This is a tough nut to crack for the segmenter, in view of the fact that the preverb *ā* is reduced to a single phoneme, allowing overlaps of left sandhi and right sandhi. This problem is solved in my implementation by a technical device called *phantom phonemes*, which is described in [26, 23]. More such technical devices are needed to account for the retroflection in forms such as *nirṇayati*, *viṣṭabdha*, etc.

A technical problem concerning absolutives is that the segmenter has to know that e.g. *-ya* is the form to be used with preverbs, while *-tvā* is the form to be used for plain root forms. This distinction is achieved by putting the two forms in distinct lexical categories (respectively accessed through the states Abso and Unde in the figure below), and having the segmenter operate transitions between such categories controled by a finite automaton whose states denote the phases of recognition of a simple regular grammar expressing preverb affixing, compound formation, periphrastic phrases and more generally morphological geometry. This mechanism is easily achieved via the technology of modular transducers [28]. Fig. 1 shows the current finite automaton used in my system.

A final remark is that perhaps this layer ought to be called the *rational* structure rather than the *linear* one, since its structure is not just a list of items, but a list of regularly structured items – reflecting the fact that compound formation, preverb affixing, and absolutive selection are definable as regular expressions, and solved by finite-state automata. This layer is the realm of finite-state methods, as opposed to the next layer, which represents an algebraic structure in the realm of context-free or even stronger combinatorial processes.

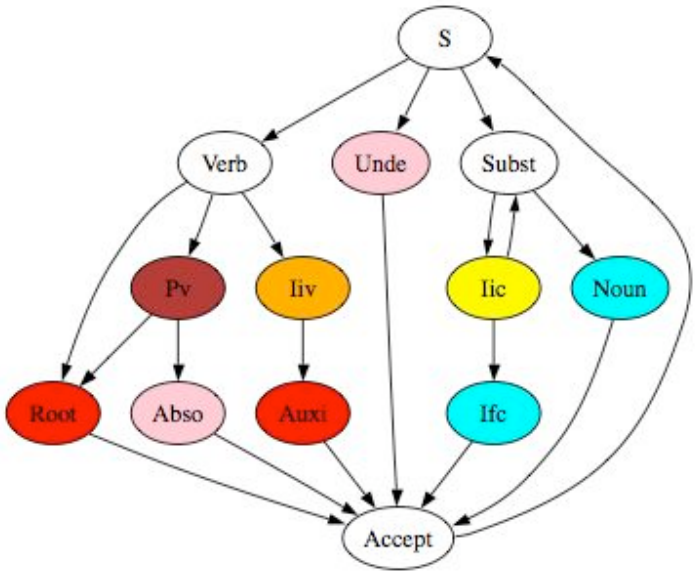

Fig. 1. The 9-phases lexical analyser.

2 The functional structure

In the tradition of Western linguistics, specially when dealing with languages with strict word order, it is customary to distinguish a layer of syntax prior[14] to a level of semantics. The combinatorial analysis of syntax, in a perspective of general linguistics which applies with more or less felicity to free-order languages, has burgeoned into a immense mass of material since the middle of the 60's, when Chomsky started his program of formal syntax. It may be worthwhile to give a short overview of these developments.

2.1 Short survey of formal linguistics

The first step in giving more structure to the sentence than the sequential order of its utterance is to give some substructure hierarchy – another partial ordering – which reflects some well-parenthesized notion of *constituent*. Thus tree-like structures are proposed as syntactic notations, subtrees being phrases, terminal nodes being the words themselves. Typically, such phrase-structure trees may represent the derivation trees of some context-free grammar. Non-terminal symbols from the grammar label the structure as mark-up tags for the phrases sorted by these non-terminals – sentences, verbal phrases, nominal phrases, etc. More abstractly, some of the recursion structure may be omitted, so that derivation

[14] in the analysis direction from utterance to sense.

trees are abstracted into terms from some abstract syntax algebra. The two levels correspond to what the logician Curry called the phenogrammatics and the tectogrammatics, to what the linguists call respectively *shallow* and *deep* structure, or the computer science distinction between *concrete syntax* and *abstract syntax* (see the discussion in [4]). Abstract syntax is closer to the semantics of the linguistic elements, since it abstracts away from operational details about parsing and word order, and keeps the essential domination ordering. In the Indian grammatical tradition, the distinction *abstract* vs *concrete* syntax corresponds more or less to the distinction *saṃbandha*/*abhisaṃbandha* (see [46] p. 13).

The problem of relating the syntax and the semantics is that usually syntax permits dislocations of the sentence, allowing arguments of a given semantic operator to appear at possibly long distance in the precedence ordering of its utterance. In order to represent the derivation of the semantic structure from the syntactic one, as some kind of compilation process, linguists have studied derivation sequences proceeding by movement of constituents, leading to so called *transformational grammars*. One proceeds from one structure to the next by some kind of computation step. This computation may be expressed by some outside process, like an interpreter, viewing each structural description as a data item. This is typical of rewriting formalisms, where each structure is like a term in some first-order logic, and computation is algebraic equality substitution in the relevant algebra. Alternatively, the structures may be represented in some higher-order formalism such as λ-terms, and the rewriting calculus is now the step by step unfolding of internal computation by λ-reduction, possibly guided by some strategy. Now non-terminals are no more atomic categories, like in phrase-structure grammars, but rather structured entities called *types*, similar to logical propositions. This last approach is characteristic of categorial grammars à la Lambek, where syntactic structures are functional expressions typed by an algebra of syntactic categories, and compile into semantic combinators in some typed λ-calculus, in the Montague tradition. Logical semantics is thus expressed as the normal form of the structure, and the combinators express the necessary movements, the precise order of evaluation, as well as the proper scoping management of quantifiers. Nowadays transformational grammars have somewhat fallen into disuse, after decades of development of complex ad-hoc devices, in favor of more uniform computation processes well studied from logic and fundamental informatics such as λ-calculus and linear logic.

The status of morphology in these various views about syntax reduces generally to annotation of the syntactico-semantic structures by morphological features. Those are used mostly for controling the constructions by consistency constraints such as agreement/concord or coordination, although some of them survive in the final semantics (typically number). These features may be organised with polarity distinctions (distinguishing between the producer of a feature, such as the case of a noun phrase, and its consumer, typically a verbal phrase which needs an argument or a complement of that case by its subcategorisation regime). These features may then be made essential components of the com-

putation process; this is the view currently held by computational versions of minimalism such as Edward Stabler's.

It is sometimes hard to dissociate general algebraico-logical frameworks from linguistic theories, such as Chomsky's $\overline{X}$ theory. And it is not always clear whether a formalism is proposed as expressive enough for modeling the right notions, or whether it is crafted in such as way that it leads to tractable parsing algorithms. Let us recall that general context-free grammars are parsable in a time proportional to the cube of the length of the candidate sentence, and that more expressive mildly context-sensitive formalisms such as Joshi's tree adjoint grammars, which seem sufficient for many natural language constructions, lead to parsers which are still bounded polynomially (although with exponent 6 this time) in the worst case.

An altogether different formalism was proposed more than 50 years ago by Lucien Tesnière as *structural syntax* [48]. This approach developed into the *dependency grammars* of the Russian school of linguistics, and is now used widely by various computational linguistics researchers. The idea is to axiomatize directly the domination ordering between words, without the mediation of an algebra of syntactic operators. In other words, the trees are not algebraic expressions with words as atomic leaves, the sentence phonetic realisation being the frontier of its constituency structure. They are now graphs whose vertices are the words, arcs denoting their mutual dependencies. Syntactic dislocations will show as crossing dependencies; the dependency graphs may not be represented as planar trees without violating the precedence ordering of the words in the sentence. Dependency graphs are fundamentally different from phrase-structure constituency structures. Note that their size is exactly bounded by the length of the sentence, since every constituent is explicitly anchored on an actual word phonetically realized in the sentence. There is no non-overt copula construct, for instance, a nominal sentence must have a head predicate and a dependent subject. This has advantages – every construction step must consume a terminal – and drawbacks – it is not possible to delay movement in the right argument position by some intermediate computation on the algebraic structure of syntax. Also there is no room for fitting covert quantification.

We remark that the linguistic notion of *head* is extremely overloaded. In dependency grammars, the head of a constituent is the word which has minimal dependency (it labels the root of the tree), whereas in phrase structure formalisms or in minimalism the notion of head is more complex, and is encapsulated in an algorithm that computes inside the constituent in order to project to its head. Thus the head is not a simple uniform notion of the formal apparatus, but rather a specific notion associated with a particular linguistic theory such as $\overline{X}$ theory or minimalism. At the other extreme, a very general framework such as HPSG (Head Phrase Structure Grammars) is rather transparent to linguistic theories, since it consists essentially in first order terms with attribute annotations. It can be considered as a very loose algebraic markup of linguistic data, with all the computing power relegated to the interpreter of the formalism. This neutrality with respect to linguistic theories is good for expressivity, since any

Turing complete computational model may be associated with it – the drawback being that it is not logically controled, and thus no complexity analysis is readily available.

2.2 Short survey of linguistic theories of Sanskrit syntax

In contradistinction to the immense amount of more or less interesting material published in formal general linguistics over the last 50 years, there is a relative scarcity of material available on Sanskrit syntax. Thus Pāṇini's grammar is very extensive on phonology and morphology, but says little on syntax. It however develops a theory of *kārakās* which may be viewed as an interface between syntax and semantics, and is therefore relevant to functional structure. But this theory is sketched only for simple sentences with at most one finite verb, and it is not even clear how it accounts say for relative clauses.[15] Later grammarians of the ancient Indian tradition, starting with Bhartṛhari, elaborated further on semantic and pragmatic linguistic analysis, as did Navya Nyāya which introduced a formal language of semiotics in Sanskrit. Other schools such as exegesis (*mīmāṃsā*) developed alternative methods of semantic analysis, but syntax *per se* does not seem to have been a major concern, although systematic ways of paraphrasing linguistic constructions such as compounding may be thought of as transformational theories to a certain extent.

A peculiarity of Sanskrit is the important tradition of commentarial work. Commentaries on terse formulaic styles of teaching (*sūtra*) were necessary for their understanding. Such commentaries deal with several levels of analysis, starting from sandhi viccheda, explained as a prose markup of the original work [50]. The commentarial tradition gives more or less standardized analyses of the commented text, which could be exploited for mechanical extraction. Furthermore, stylistics and poetics was theorized very early by poets such as Daṇḍin and an important Kashmirian school of literary criticism (Ānandavardhana, Abhinavagupta, Mammaṭa, etc.) It is to be expected that this tradition of esthetics, in as much as it influenced the literary production, will be of help ultimately when our computational linguistics tools will be advanced enough to attack the problems of metaphoric expression.

The first general studies of Sanskrit syntax came thus rather late, in India with Apte's manuel [1], and in the West with Speijer treatise [45]. Both build on analogies between Sanskrit and the Western classical languages (Latin and Greek) in order to adapt the then current linguistic terminology to Sanskrit, not entirely convincingly. In these, and most Western Sanskrit grammars until recently, the topic of the proper order of the words in a Sanskrit sentence was dealt with in a rather vague manner; Sanskrit was deemed more or less free-order, but a more detailed analysis was lacking. Besides these two isolated attempts, linguistic studies of Sanskrit consisted mostly of very specialised monographs on specific constructions, often in the context of Vedic literature [39, 49, 19–21].

[15] Cardona, Pāṇini: His Work and its Traditions, pp. 167-178 discusses the derivation of complex sentences, and *Recent Research*, 189-197 surveys research on Pāṇinian syntax other than on kāraka relations (P. Scharf).

The first attempt to give a formal model of Sanskrit syntax was due to Frits Staal in the 60's [46]. What he proposes is a Calder-mobile-like theory of syntax, where oriented trees of first-order logic are replaced by non-oriented trees. In other words, trees which are arguments of a given functor are organised as a multiset rather than as a list. The built-in associativity and commutativity of such a collection allow for arbitrary rotation of arguments, similar to the rotations in a Calder mobile. This gives a precise handle on allowed freedom in ordering of the words of a sentence: any scrambling is allowed, in so far as it is consistent with the domination ordering, in the sense that if a word precedes another, then all the words it dominates must precede the words dominated by the other one. In terms of dependency analysis, there should be no crossing of dependencies. As Gillon expresses it [8]: "[Staal thinks that] word order is free up to preserving constituency."

This immediately gives the limit of the formalism: dislocated sentences cannot be accounted for, unless some prior movement unscrambles long-distance dependencies. Also, it ignores the fact that quantification may be expressed by word order: compare *bālakaḥ gṛhe asti* "the boy is in the house" and *gṛhe bālakaḥ asti* "a boy is in the house".[16] Nevertheless, it gives a simple framework, which may be refined further with dislocation annotations. The crux of the matter is to replace the notion of oriented tree (the trees of concrete syntax and even abstract syntax seen as a construction from constituents ordered by enunciation) by a notion of tree in the algebraic sense of graph theory, where arcs are not necessarily ordered consistently with the enunciation ordering; in other words, the graph may not be representable as a planar graph, with no crossings.

We profit of this view of commutative-associative formalisation of family descriptors to revisit the categorial grammars school. Categorial grammars originated from the work of Ajdukiewicz in the 30's and Bar-Hillel in the 50's, where syntax operators were viewed in analogy with arithmetical fraction notation. The ensueing systems of combinators, with complex notions of raising allowing a more flexible notion of order of evaluation of constituency structure, was simplified by Lambek in the 60's with his L system [41], which allows deduction under hypothesis, and is to categorial combinators what Church's λ-calculus is to Curry combinators [2, 18], or in logic what Gentzen's natural deduction is to Hilbert axiomatic systems [31]. But all the structural rules of sequent calculus are not allowed, and in particular thinning and contraction are not allowed. This is in line with the economy of linguistics, which is different from the economic regime of logic, where hypotheses may be ignored or duplicated, whereas words in a sentence have to be used exactly once in the derivation of its syntactic tree. Furthermore no permutation is allowed either, since word order is assumed to be rigid. Thus Lambek's L system is very close to Girard's linear logic [14], except that Girard's logic is commutative. This connection was noticed in the 90's, when linear logic was firmly established [15]. But if we believe in Staal's model, it is natural to restore commutativity, and try to adapt directly the proof-nets

[16] this example is borrowed from [16].

of linear logic to the syntactic descriptions of Sanskrit. We leave this as a hint for future research at this stage.

Let us note further that recent elaborations of dependency grammars add information on the arcs in order to denote functional relationships. For instance, the node headed by some finite form of a transitive verb will typically have as immediate successors two nodes headed by substantival forms, one arc being labeled 'Subject' and one labeled 'Object', or one labeled 'Agent' and the other one labeled 'Goal' if one prefers a more semantic representation. The verbal node may be itself colored as verbal phrase or action. In any case, the structure `[Verb "v" [Subject "s"] [Object "o"]]` may now be twisted as a Calder mobile in order to project its frontier to the enunciation order, be it SOV, SVO, or whatever. Such structures are being used in the Mel'čuk linguistics school in Montreal, in order to represent lexical semantics as semantic graph schemas [35].

2.3 Gillon's contributions to Sanskrit syntax

We shall now turn to a more recent attempt at designing a formal notation for Sanskrit syntax, taking into account dislocations. This paragraph will sketch a family of proposals by Brendan Gillon for formalising Sanskrit syntax, consistently with Western theories of linguistics. He made a comprehensive study of Classical Sanskrit syntax, and presented his analyses in a systematic system of notation. This notational system was put to the test on hundreds of characteristic examples from Apte's manual, plus a fair selection from typical sentences extracted from the classical Sanskrit corpus. Let us consider an example. The foundational article is [8], in which he makes a systematic classification of extrapositions in Classical Sanskrit, and in which a first version of his notational system is presented by way of examples. Let us look at a typical one, originally from Kālidāsa's *Mālavikāgnimitra*, given as exemple 2 in Apte's lesson XIV on past participles:

$$[_S\ [_{NP6}\ [_{AP6}\ \text{tayoḥ}\ ]\ \text{baddhayoḥ}\ ]\ [_{VP}\ 0\ [_{AP1}\ (\text{kim}\ -\ \text{nimittaḥ})\ ]\ ]$$
$$\text{the}\qquad\text{prisoner}\qquad\qquad\text{what}\ -\ \text{basis}$$

$$[_{NP1s}\ [_{AP1}\ \text{ayam}\ ]\ \_\_\ \text{mokṣaḥ}\ ]\ ]$$
$$\text{the}\qquad\text{release}$$

What basis does the release of the two prisoners have?

Gillon explains that the genitive NP6 is extraposed from the subject NP1s. Which he renders by the dislocation notation __. We shall not concentrate on his linguistic analysis, but rather look at the formal descriptive apparatus. The first striking feature is that, despite its phrase-structure early-Chomskian look, it is pretty close to a dependency structure. The NP6 node is indeed an NP6 because it has an immediate constituent which is terminal (it corresponds to the phonetic realisation *baddhayoḥ*), and recognized as an inflected form in the genitive case of a nominal lexical item, namely *baddha*. Thus there is an implicit assumption, namely that each syntax node (a matching pair of square brackets) has an atomic head from which it inherits its features. This assumption is exactly what

dependency structures demand: each piece of the construction consumes overt phonetic material. The head, here *baddhayoḥ*, is actually an inflected form whose full morphological tagging is `<N case=6 number=2 gender=? stem="baddha">`. Note how the N and the 6 percolated to its governing left bracket. Note also that we might as well percolate the dual number, which is needed for the gloss 'two prisoners'. Concerning the gender, it is not just undecided, but not needed for the immediate analysis, so I kept it as an unknown. Note also that we made a choice between genitive and locative, a more automated analysis would have to consider the two cases. Finally, a more refined morphological analysis would analyse further *baddha* as a kṛt formation, i.e. past passive participle of root *bandh*. Anyway, such details of granularity notwithstanding, we arrive at a formal structure of the kind $[_N$ case=6 number=2 [...] "baddha"], where the substructure [...] stands itself for a similar $[_N$ case=6 number=2 [...] "tayoḥ"]. Actually, Gillon labels this substructure $AP6$, meaning that pronoun *tad* is used adjectivally to form the noun phrase 'the two prisoners'. We deem this choice dependent on a linguistic model where *adjective* is a clear-cut part of speech category. For Sanskrit, the case is not so clear it seems, specially in view of bahuvrīhi compound formation. Actually, *baddha* itself, in the sense of prisoner meant in this sentence, is but the coercion to a substantive of the adjective 'bound' issuing from its participial nature. A slightly more primitive analysis would factor together the two nominal forms as $[_N$ case=6 number=2 "tad" "baddha"], leaving to a deeper analysis phase the actual dependency of *tad* on *baddha*.

Let us return to the full example. We note that the subject is indicated by functor $NP1s$. We believe that this should be taken similarly as a feature structure $[_N$ case=1 role=subject ...]. But now we have a feature which is not morphological and synthesized from its head, but rather syntactical and inherited from the main clause. This with a particular intended linguistic model recognizing a role for subjects. Actually, another model would recognize this nominative as merely a naming device for the agent, properly carried by the predicate, here the adjective *kiṃnimitaḥ*. This would have the advantage of getting rid altogether of the copula construction $[_{VP}$ 0], an anathema for dependency structure since it rests on thin air (0 indicating the lack of phonetic realisation). This construction would be replaced by the predicate marking of *kiṃnimitaḥ*, corresponding to selection of the agent (*kartṛ*). While not strictly speaking Pāṇinian, since Pāṇini restricts kāraka analysis to sentences with finite verbs, this analysis is consistent with Kiparsky's presentation of the Pāṇinian model [29].

The reader may be confused at this stage, because while I am explaining Gillon's notation I am proposing to possibly use it in other ways. More precisely, I am suggesting that his meta-formalism could be slightly modified in ways which are suggested by dependency structures, and thus better adapted to automated synthesis, while making it less dependent on a particular linguistic model, with the possible benefit that it could hopefully be used with Pāṇinian models such as Kiparsky's. What we wish to keep from the meta-notation is the fact that it uses bracketing to denote substructure with percolation of some morphological annotations of their heads, and that a system of references makes explicit the

extrapositions/dislocations. In the above example, this suggests replacing the __ place-holder by some integer or label index, with an explicit annotation for its antecedent (here the genitive phrase). This indexing would be similar to λ notation, except that it is linear and bi-directional (i.e. the binder may precede or follow the unique bound occurrence). In other words, this is the information which completes the tree into a dependency graph, crossings being replaced by indirect adressing. What remains a fundamental contribution of [8] is the classification of extrapositions in classical Sanskrit, and the corresponding amendment to Staal's principle.

Over the years, Gillon modified slightly his notation, and extended it to compound analysis [6, 9, 7, 10, 12, 11, 13].

2.4 Kiparsky's analysis of the Pāṇinian syntax-semantics interface

Staal's monograph dealt mostly with the problem of word ordering in the sentence, but did not account for a number of Sanskrit linguistic constructions, and stayed at a shallow level of syntactic analysis. His approach was revisited in a joint paper with Kiparsky [30], reprinted in [47]. In this paper, the authors show that Pāṇini's kāraka theory gave a proper account of the interface between shallow syntax (morphological cases) and deep syntax (thematic roles). This investigation was the start of a deeper analysis of Pāṇini's devices and their precise relation with linguistic phenomena such as agreement, ellipsis, anaphora, etc. by Kiparsky, culminating in his survey on the architecture of Pāṇini's grammar [29].

There is not proper space in the present paper to present Kiparsky's analysis in any detail, which anyway appears as a chapter in this volume. But the striking feature, for which Pāṇini departs radically from the usual syntactic treatment of languages such as Latin, French and English, is the elimination of the notion of subject in favor of the notion of agent (more precisely *kartṛ*), and furthermore the attribution of this role, in the case of an active sentence, *not* to the substantival 'subject' (when there is one), but rather to the inflection suffix of the finite verb. Pāṇini's requirement, that every role should have a unique expression in the sentence, deprives the nominative of its elevated status of subject, and makes it a kind of dummy in the underlying semantical binding theory, relegating it to an optional naming role, rather than having it act a proper semantic role. This is quite revolutionary – even if it is anachronistic to say that an ancient grammarian makes a revolutionary proposal to contemporary linguistics! What is most disturbing is that this view shatters the dichotomy between substantivial forms – thought of as having a positive polarity of actors – and verbal forms – thought of as having a negative polarity of roles (i.e. their valency, obtained from their sub-categorization frame by mediation with the voice). Thus the underlying graph matching of positive actors with negative roles, in order to express the semantic constraints, is now replaced by a more symmetric situation calculus, where a sentence may express one of three abstract modes, namely dynamic Action and Reaction, and static Situation. The choice of the mode is conditioned by the verb (in case of verbal sentences), which denotes respectively the Agent, the Goal, or the Process. This model has many advantages. First of all, it gives a

proper status to passive forms of intransitive verbs, where an impersonal sentence such as *mayā supyate* lacks a subject, and is not readily translatable in English in a different way than *ahaṃ svapimi*, "I sleep". Secondly, it permits better accommodation of elliptic constructions. If the subject is missing, it is either because it is the deictic you or I when the agentive *tiṅ* denotes the second or first person, and thus optional in a pro-drop language such as Sanskrit, or because it is ellipsed, being the discourse theme, topic or whatever, and is only retrievable from the discourse context.

Kiparsky shows in his survey that this linguistic model accounts for all the linguistic facts such as agreement, provided a proper treatment of co-indexation (corresponding to the Pāṇinian notion of *samānādhikaraṇa*) is effected. This treatment accounts for agreement of verb and subject, as well as concord of substantive and adjective.

We believe that this model is the correct one to analyse the Sanskrit corpus at this functional level which acts as an interface between syntax and semantics. We further believe that the linguistic constraints can be expressed in an explicit computational model which refines the morphological analysis, and is thus consistent with our requirement of proper embedding of the linear and the functional levels. This computational level must make explicit some hierarchical dependency structure, some complementary dislocation notation, and a proper co-indexation equivalence, needed for agreement/concord as well as corelatives and anaphora. We remark that such co-indexicals appear in more recent variants of Gillon's notation [9].

What is very important is that this functional structure be applied at the level of discourse, rather than at the level of individual sentences. This contextual calculus will be appropriate for instance to analyse a *śloka* in a modular manner, the second half-*śloka* being analysed in the context of the first one, a much needed reduction in complexity. Rather than focusing on the notion of sentence, which will demand global treatment of complex sentences, while not being sufficient for the contextual treatment of anaphora and ellipses, it is much wiser to accommodate directly discourse, and to build a more modular engine able to consider long statements piecewise. Furthermore this "continuous" model accounts easily for chains of absolutives, which typically share their common agent with the main clause. This problem of control is specifically addressed by Gillon in his study of implicit roles in auxiliary clauses [9].

Viewing a text as a continuous discourse, in the tradition of discourse representation structures, has several advantages. Rather than a flat sequence of closed sentence structures [$_S$ ...], we get a sequence of dependent open structures [Agent = x; Goal = y [...] S] where x and y are co-indexicals, the inner [...] is the dependency structure of the current frame, and S is the next frame, in which co-indexicals x and y may appear – x and y are bound in S. Thus [...] may contain co-indexicals x and y, as well as previous ones coming from the context. The scoping of these existentials will be convenient to express quotations: the tool word *iti* marks the closing of some frame, hiding from the rest of the discourse the bindings of the quotation. This structure should also be adequate for repre-

senting stories embedded inside stories, in Pañcatantra style. The beginning of a new sub-story will hide the current protagonists, one of which tells the story. When the inner story is finished, its current characters are erased (their scope is ended), and the outer protagonists will pop up again in the discourse context.

In the rest of this section, we shall examine a few recent efforts at computational modeling of the functional level of Sanskrit. We start with our own effort at building a Sanskrit Engine.

2.5 Parsing Sanskrit text with a constraint processor

The method we have been following for parsing Sanskrit text is the following. First of all, we designed a Sanskrit lexicon format accommodating grammatical annotations [24]. Then we implemented internal sandhi and a paradigm-based morphology generator. This morphology generator is applied iteratively to all lexicon entries, yielding a digitalised inflected forms database (implemented as a collection of finite-state automata, one for each lexical category) [22, 23]. We then construct mechanically from this data a modular finite-state transducer inverting external sandhi, using the algorithms developed in [25, 28]. This transducer is used as a non-deterministic tagging segmenter, which we apply to Sanskrit text, preprocessed with the algorithm given below in Appendix B. This lexicon-directed methodology has been sketchily explained in [26].

The next step consists in filtering, out of the possibly thousands of possible segmentations, the few that pass some semantic test inspired from kāraka satisfaction. Basically, this proceeds as follows. For a simple clause with a finite verb form, the valency of the verb[17], together with its voice, determine the cases of nominal phrases which are necessary in order to fulfill the semantic roles demanded by its valency. The sentence is explored in a right to left fashion, looking for the needed phrases. Nominatives are searched with given number and person, thus constraining subjects to agree with their purported verb. Extra nominatives are factored with chosen ones as much as possible, thus building concording nominal phrases (possibly dislocated). Extra oblique cases, as well as extra accusatives, are just ignored as adverbs. Thus instrumentals are deemed needed in passive constructions, as denoting agents, while being ignored in active constructions, as denoting instruments or adverbs of manner. This extremely naive constraint processing is nevertheless quite effective, in filtering out more than 95% of erroneous segmentations in average on a simple training corpus [42], while at least generally retaining the right parse.

An extra mechanism of tag stream transducers, associated with tool words, has been proposed to solve problems such as coordination. Thus 'ca' (in the case it coordinates nominals) operates on the stream of tags of preceding noun phrases, and attempts to replace them with a coordinated tag, effecting proper addition of their numbers (in a non-standard arithmetic model with 3 numbers,

[17] Actually, the valency may depend on the particular verbal form. At present, we use the *gaṇa* of present system forms as a parameter. Thus the verb *pac*, to cook, is marked as transitive for class 1 (*pacati*), and intransitive for class 4 (*pacyate*).

1, 2 and many), as well as proper dominance of their person and gender. This mechanism is sketched in [27], and may be tested as a Sanskrit Reader Companion web service at `http://sanskrit.inria.fr/DICO/reader.html`. We shall in the following refer to this shallow parser as the Sanskrit Engine.

We do not claim that this crude constraint processor is a full-fledged Sanskrit parser. It is merely a nonsense filter, usable as a front-end to a more sophisticated analyser. It does not deal at present with co-relatives, does not attempt to solve anaphora, tends to overgenerate with alleged vocatives, and deals poorly with ellipsed elements. We believe that, specially concerning this last problem, a proper solution involves an analysis closer to Kiparsky's on a continuous discourse flow (as opposed to individual sentences), as sketched at the end of the previous section. Such a more robust parser ought to generate dependency structures annotated with co-indexed semantic roles, in the manner suggested above. Conversely, it ought to be usable as a deterministic checker for a corpus annotated with such representations. It could then learn from a properly annotated corpus (tree bank). It is reasonable to hope that the corpus of Apte's examples annotated by Gillon, and comprising several hundreds of sentences characteristic of Sanskrit constructions, could be adapted as a training treebank - possibly complemented by longer pieces of text, usable as a discourse treebank. Learning would allow the system to tune its parameters in order to improve its precision, and ultimately yield a deterministic parser usable for fast automatic tagging of digital libraries of a real corpus. Due to the great diversity of styles of classical Sanskrit, specifically tuned parsers, adapted to specific prosodic styles, will probably need to be tuned separately, and meter indications should help the segmenting process.

2.6 Other efforts at Sanskrit parsing

Early work on mechanical Sanskrit processing was reported in [51, 38]. We now turn to on-going efforts.

The work reported in [37] on analysing Sanskrit text by the IIT Kanpur team is very promising. Its methodology is actually rather close to the Sanskrit Engine one reported in the previous section. It uses finite-state methods as well. Parsing is guided by a pattern-matching basis of rules with certain priorities. Some guessing is done, instead of exploring systematically all branches of ambiguity. In a certain sense this priority selection algorithm could be seen as a strategy for solving the kāraka constraints. It is not entirely clear how the strategy described agrees with Kiparsky's analysis, though. Impressive exemples of parsed Sanskrit text are given, showing that the approach is sound, even if it cannot yet be used to process satisfactorily a large corpus.

The work reported in [17] presents a Sanskrit tagger, using statistical techniques. It is a POS tagger if you agree on a notion of part of speech which reflects the morphological overt structure, pretty much as assumed in the first part of this paper. Such a statistically trained tagger is an obvious boon for non-deterministic approaches such as the Sanskrit engine methodology. A natural solution would be to incorporate the underlying hidden Markov model in

the complete non-deterministic finite-state tagger, in the manner of stochastic automata, in order to guide the search towards the most plausible solution first. Statistical methods are extremely important to tune the performance of linguistic processors, at any level. However, they are not a panacea. They can only cluster the data in categories that are assumed *a priori* by the formal model - one will find lines or shapes in a digital bitmap image only if one looks for them, i.e. measures the data against segment or shape representations. Similarly, a statistical linguistic tool should be seen as an optimizer of formal recognition. Thus statistical methods and algebraico-logical methods have to work hand in hand.

Of course all these approaches need a reliable morphological database. There are currently several teams which have developed significant such databases, for instance in the Sanskrit Library project of P. Scharf and M. Hyman, in the University of Hyderabad project of A. Kulkarni, in the JNU project of G. Jha. The generators are fairly compliant with the Pāṇinian methodology, and thus seem to agree on their common root coverage. This has been checked at least on the noun morphology tables of Kulkarni, Scharf and Hyman, and Huet. A more systematic evaluation will be possible, when a consensus for a tag set emerges. Further progress towards interoperability will need consensus on root naming standardization. Such naming will specify homonymy indexing, and details such as what should be the main stem of pronouns and numbers.

Other methods have been experimented with, not relying on a full lexicon. This has the advantage of not requiring at the start a full coverage of the target corpus' vocabulary by the morphological closure of the root lexicon. But the problem of lemmatizing a conjectured form without a handle on possible stems is very hard. It involves undoing internal sandhi, a non-rational transducing problem in view of retroflexion, thus not easily amenable to finite state methods. Furthermore, if one wants to do segmentation at the same time, one will have to face undoing external sandhi modulo undoing internal sandhi, a problem very hard to solve exactly, although heuristics may help in a fair number of cases.

2.7 Designing a functional level formalism

This paper has sketched a general framework for representing Sanskrit text. The main framework uses discourse representation structures, in order to manage scoping and naming of indexicals. The functional structure may be thought of as a situation calculus, in the tradition of Montague semantics (higher order type theory in the sense of Church, possibly enriched as a dependent type framework in the sense of Martin-Löf [34]). Indexicals will serve to group together actors of the same function in a phrase, sharing the features attributed to the forms it groups. This gives the treatment of both agreement and concord. Note that sharing may affect a gender feature to a verbal form, if it has an overt subject in the active voice (respectivally an overt object in the passive voice).

It should be understood that the functional level, where kāraka satisfaction takes place, is *not* yet the semantic level. For instance, it does not need to specify the analysis of compounds. The only really necessary distinction among compounds is between *avyayībhāva* (adverbials) and *tatpuruṣa*/*karmadhāraya*,

which play a unique role, consistent with their case endings. There is no real need to explicitate the possible adjectival role of a bahuvrīhi compound. Rather, such a compound will be co-indexed with other concording substantives. It is left to a further level of semantics to figure out the needed coercions between individuals, characteristic properties and classes which are needed to make sense of the gliding between adjectival and substantival uses.

Similarly, genitives do not really need to be explicitly linked to the actual word they complement. They can "float around" so to speak, without clear dependency or constituency specification. This is consistent with Pāṇini, who does not associate a kāraka with genitives. Deciding the role of genitives in a phrase may thus be left to a semantic layer, where common sense reasoning (i.e. hard deduction) might be needed for their disambiguation. For instance, even a very simple sentence [42] such as: *tiṣṭhan bālaka upādhyāyasya praśnānām uttarāṇi kathayati* seems to require for its understanding that answers belong to questions, and that questions belong to the teacher.

In the same vein, rather than requiring a dative argument in the subcategorisation frame of verbs such as *dā*, it may be better to consider it a complement rather than a needed argument. Thus the functional modeling may operate at various granularity levels, between a crude consistency check and a deeper semantic analysis, and consequently subcategorization may be more or less coarse. Nevertheless, we may expect problems at the right handling of verbs with two accusatives, since knowing which is which often demands some common sense reasoning.

What distinguishes clearly the functional level from the linear level is that it is branching in a tree-like constituency, or rather dependency structure. Thus in addition to the total ordering of enunciations (i.e. the temporal sequence of the phonemic chunks), there is a partial ordering of dependencies, where bracketing indicates scoping. Thus for instance absolutive clauses depend on the rest of the sentence, with at the top the main finite verbal phrase, which itself depends on the surrounding discourse. Here dependency runs contrary to enunciation, so that the intermediate situations may get their ellipsed arguments from the future to a certain extent. We see this dependency structure as the analysis of discourse, linking situations in some kind of discourse Montagovian semantics. A particle such as *iti* is typically a discourse operator, closing a quotation, i.e. a subdiscourse, which may span several sentences. Similarly, the *daṇḍa* punctuation symbol should be treated as a discourse operator, allowing the left half-*śloka* to be analysed piecewise, while having access to roles expressed in the second half. In contrast, the double danda is a fuller break in the discourse. Having an explicit handle on the discourse will facilitate the analysis of deictics through an explicit representation of elocution situations ('I' refers to the speaker in the current sub-discourse, and its gender/number consistency may be checked, etc.)

We see kāraka satisfaction following Kiparsky's analysis, with the selection of a head, its coercion to one of the three modes, and the verification that roles are fulfilled, by co-indexed complexes visible in the discourse dependency structure. The proof of consistency may or may not be explicit in the functional structure,

there may be several levels of partial specification. In case of participial heads, the lexicon must give an indication of their prefered voice – Passive for passive participial forms, for instance. The search for a consistency proof, when the corpus is tagged only with the dependency bracketing, will look for roles in the appropriate periphery of their predicate, according to analyses such as Gillon's on Bhartṛhari's rule [9].

We stop here our speculations about the precise specification of the functional level, since as we emphasized there is a space for design which needs concrete experimentation to validate precise data structures and algorithms. As already said, we are still far from a semantic level, where the notation will express logical deductions which are not verbalized, and thus by necessity the algebraic structure is richer than just a dependency graph between phonetic realisations. Formalisms are ready to be used for this modeling, through the Montague semantics, which has been revived lately with material coming from the denotational semantics of programming languages. We have rich type theories in which to embed logical deductions and reasoning in ontological structures with grammars of descriptions, etc, etc. Nonetheless, all this logical tradition is still a little bit naive with respect to understanding natural language, it somehow assumes some universal structure of thoughts and concepts on which natural language productions should project, once the syntax is deciphered. But there are so many intermediate levels which can be considered. For instance, style. Style is a fundamental parameter which should accompany the discourse structure. Here actually, we are rather fortunate in Sanskrit, since there is a very important tradition of literary criticism and esthetics of poetry. This tradition spells out particular styles, such as more or less intensive use of compounds. Corpus material which obeys the canons of style praised by the masters will tend to conform to certain recognizable patterns, thus helping the understanding of the text by mechanical means. Furthermore a lot of *śāstra* literature is itself strongly structured; for instance, the *Dhvanyāloka* consists in a dense teaching in a form of terse *kārikā* verses, interwoven with a prose commentary (*vṛtti*), which is itself commented upon by further authors. Proper understanding of such texts will need a precise grasp on this commentary structure [50]. We get discourse within discourse in similar ways as the *Pañcatantra* tells us tales within tales, and this suggests a continuum between the discourse structure within one text and the intertextual structure of a Sanskrit digital library.

A closely related phenomenon is prosody. Knowing the meter of metrical poetry will be a tremendous help to the segmenter. Knowing the accent will permit to recognize bahuvrīhi compounds. The enormous Indian tradition of prosody, *chandas*, has to be exploited for proper computational modeling. Finally, coming to semantics, it is not so clear that the Western logical apparatus which is the substrate of theoretical investigations of semantics in Western linguistics is entirely appropriate for Sanskrit. It corresponds more or less to a mathematical abstraction of *prācīnanyāya*, 'old' logic. Whereas Indian investigations in the theory of language, by Gaṅgeśa and his successors, developed a new logic *navyanyāya* which is a kind of sophisticated formal semiotics. Fur-

thermore, there is an important Indian tradition of pragmatics, arising from the philosophical school of *mīmāṃsā*. When speech is ritualised, it means it contains overt structure that may be recognized. Mantras are not random nonsense, they are actually strongly structured patterns [52]. If we take the extreme view that speech is driven by communication intention, then meaning must depend on will. We have to look for the fruit of the speech act, in pretty much the same way that the policeman looks for the motive of the criminal.

2.8 A roadmap for future work

The task is formidable indeed, and even if all investigating teams join forces in collaborating on open standards for interoperability of the various available tools, it will take many years to build Sanskrit processing computational platforms that will process usefully a significant part of a giant corpus, and we must be modest in our goals. But it seems we can at least sketch a roadmap of plausible stages.

One problem that has been only tangentially addressed is the structure of the lexicon. This is however of paramount importance, whatever methodology is followed. The problem of completeness of the lexicon is a very complex one, since the computerized lexicon need not only be a static data base of stems and forms. It is more appropriate to look at the lexicon as a generative device - for instance *kṛt* and/or *taddhita* formation could be expressed implicitly as a grammar, and dictionary lookup implemented as parsing.

Another problem is the exact representation of parameters, both for morphology (e.g. intercalating *i* in the future stem, *gaṇa* class of verbs, *parasmaipada* vs *ātmanepada* use, etc.) and for the functional structure (subcategorization regime of verbs, etc.) A close look at the information encoding in *dhātupāṭhas* and *gaṇapāṭhas* is a good starting point. More parameters will progressively percolate from the constraint satisfaction algorithms. The general point is that all this information ought to be lexicalized, rather than being hard-wired in the processing programs.

Still another problem is how to organize the diachronic character of a language that spans 3 milennia and one subcontinent. The combinatorics of Vedic are likely to be different from the ones of Classical Sanskrit, the vocabulary is differently distributed, the morphological operations have evolved - some tenses have vanished, while compound formation increased, and prepositions got more rigidly affixed as preverbs. This is clear from Pāṇini's grammar, which goes at length to describe exceptions in order to cover the attested Vedic corpus. It seems that the digital lexicon ought to have diachronic annotations which will turn on or off certain generative processes, according to the intended corpus. This area of research has hardly been touched yet it seems (but see [36] for epic Sanskrit, also [43] for Vedic).

The basic assumption of the two-level model which we presented is that the linear treatment (tagging) is complete, so that we may make the more semantic analysis without need to backtracking into the search space for segmentation/lemmatisation. We remind the reader that quality of non-deterministic

search to solve any problem is expressed by two parameters: precision and recall, measuring respectively the lack of extraneous solutions, and the covering of the right solutions by the search space. Poor precision means many answers are returned, there is noise. Poor recall is worse, there is silence - i.e. a legitimate enunciation is not recognized. Good design means no silence, and noise minimised. Typically, several answers are potentially produced, but the intended one comes very early in the list. If it always come first, then we may use it as a function, determinism is restored. But this ideal situation is not always possible, since we know there are actual ambiguities, such as *śvetodhāvati*, where the white (horse) coexists so to speak with a dog, in the absence of a disambiguating context.

We claim that segmentation/tagging is complete in a strong sense - every Sanskrit sentence has a finite number of interpretations modulo *sandhiviccheda* and morphological analysis. Thus, if the lexical basis is sufficient to cover the vocabulary of the target corpus, we are left with an optimisation problem, of computing the most probable functional/semantic structure in the finite space of all possible taggings. This is where statistical methods enter into play. We shall look optimally for the solution in choosing the one maximizing some likeliness criterion in the stochastic functional space.

In order to train the statistical optimizer, we need to build a treebank for learning its parameters. Building a treebank involves two steps. First, a complete specification of the datastructure for the functional structure, since its formal design has been only sketched above. Secondly, one must prepare a fully tagged corpus, an expensive task, since it involves tedious work by a human expert - someone understanding both Sanskrit, linguistics, and informatics. One obvious path to take would be to build on the work of Brendan Gillon's tagging of Apte's sentences, by adapting his parses to a non-ambiguous representation of dependencies and indexicals. However, we shall need an explicit schematic representation of the context, in order to get the right discourse structure, and not just successive parses of sentences. Once a sufficient training basis will be available, the system should be able to bootstrap, since we shall be able to use the functional satisfaction algorithm on untagged corpus, in order to present a small number of potential structures to the human expert analysing raw corpus, and thus considerably speed up the treebank production. Specialized treebanks will be used to train specialised recognizers, for various styles, various periods, etc. Failures of proper recognition will turn into opportunities for lexicon acquisition. Here problems of maintenance and versioning of the lexical database, in a possibly distributed manner, will arise.

At some point in the future, when available software will be robust enough to be usable for philological applications, the need will arise to inter-operate with digital library formats, such as Perseus [`http://www.perseus.tufts.edu/`] or IntraText [`http://www.intratext.com/`].

Conclusion

We have argued in this paper that a formal linguistic model for Sanskrit, usable for computational treatment and ultimately understanding by computers, should distinguish 3 levels, called linear, functional and semantical. The linear level is appropriate to treat by finite-state methods the morpho-phonetic processes of inflexion, sandhi, and their inverses segmentation and lemmatisation. The functional level is seen as dependency graphs applied to discourse structure. This level should make explicit the scoping structure and a co-indexation relation, sufficient to express and or verify control such as kāraka consistency. This functional level has not been fully specified, since it may be more or less detailed, but its design space has been sketched, and it has been argued that parsing prototypes such as [37, 27] operate on this level. The semantical level is left for long term research, since it involves complex common sense reasoning, the use of metaphors and other figures of style, all things which a computer is notoriously bad at. Thus a roadmap has been sketched for future research, in which many opportunities of cooperation by the various teams working on computational Sanskrit linguistics will arise.

References

1. V. S. Apte. *The Student's Guide to Sanskrit Composition. A Treatise on Sanskrit Syntax for Use of Schools and Colleges.* Lokasamgraha Press, Poona, India, 1885.
2. H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North Holland, 1984.
3. A. Bharati, V. Chaitanya, and R. Sangal. *Natural Language Processing. A Paninian Perspective.* Prentice-Hall of India, New Delhi, 1995.
4. D. Dowty. *The nature of Syntactic Representation; Eds. Pauline Jacobson and Geoffrey K. Pullum*, chapter Grammatical relations and Montague Grammars. Reidel, 1982.
5. S. Eilenberg. *Automata, Languages, and Machines, volume A.* Academic Press, 1974.
6. B. S. Gillon. Bartṛhari's solution to the problem of asamartha compounds. *Études Asiatiques/Asiatiche Studien*, 47,1:117–133, 1993.
7. B. S. Gillon. Autonomy of word formation: evidence from Classical Sanskrit. *Indian Linguistics, 56 (1-4)*, pages 15–52, 1995.
8. B. S. Gillon. Word order in Classical Sanskrit. *Indian Linguistics*, 57,1:1–35, 1996.
9. B. S. Gillon. *Indian linguistic studies: Festschrift in honour of George Cardona; Eds. Madhav M. Deshpande et Peter E. Hook*, chapter Bartṛhari's rule for unexpressed kārakas: The problem of control in Classical Sanskrit. Motilal Banarsidass, Delhi, 2002.
10. B. S. Gillon. Null arguments and constituent structure in Classical Sanskrit. Private communication, 2003.
11. B. S. Gillon. *Language and grammar: studies in mathematical linguistics and natural language; Eds. Philip Scott, Claudia Casadio and Robert Seely*, chapter Subject predicate order in Classical Sanskrit, pages 211–225. Center for the Study of Language and Information, 2005.

12. B. S. Gillon. Exocentric (bahuvrīhi) compounds in classical Sanskrit. In G. Huet and A. Kulkarni, editors, *Proceedings, First International Symposium on Sanskrit Computational Linguistics*, pages 1–12, 2007.
13. B. S. Gillon. Pāṇini's *aṣṭādhyāyī* and linguistic theory. *J. Indian Philos.*, 35:445–468, 2007.
14. J.-Y. Girard, Y. Lafont, and L. Régnier, editors. *Advances in Linear Logic*. London Mathematical Society Lecture Notes 222, Cambridge University Press, 2005.
15. J.-Y. Girard, Y. Lafont, and P. Taylor, editors. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science 7, Cambridge University Press, 1988.
16. P. Goyal and R. M. K. Sinha. Translation divergence in English-Sanskrit-Hindi language pairs. In *Proceedings, 3th International Symposium on Sanskrit Computational Linguistics*. Springer-Verlag Lecture Notes, 2009.
17. O. Hellwig. SanskritTagger, a stochastic lexical and pos tagger for Sanskrit. In G. Huet and A. Kulkarni, editors, *Proceedings, First International Symposium on Sanskrit Computational Linguistics*, pages 37–46, 2007.
18. J. R. Hindley and J. P. Seldin. *Introduction to Combinators and λ-Calculus*. Cambridge University Press, 1986.
19. H. H. Hock. The Sanskrit quotative: a historical and comparative study. *Studies in the Linguistic Sciences*, 12,2:39–85, 1982.
20. H. H. Hock, editor. *Studies in Sanskrit Syntax*. Motilal Banarsidass, Delhi, 1991.
21. K. Hoffmann. *Der Injunktiv im Veda. Eine synchronische Untersuchung*. Karl Winter Universitätsverlag, 1967.
22. G. Huet. The Zen computational linguistics toolkit: Lexicon structures and morphology computations using a modular functional programming language. In *Tutorial, Language Engineering Conference LEC'2002*, 2002.
23. G. Huet. Towards computational processing of Sanskrit. In *International Conference on Natural Language Processing (ICON)*, 2003. `http://yquem.inria.fr/~huet/PUBLIC/icon.pdf`
24. G. Huet. Design of a lexical database for Sanskrit. In *Workshop on Enhancing and Using Electronic Dictionaries, COLING 2004*. International Conference on Computational Linguistics, 2004. `http://yquem.inria.fr/~huet/PUBLIC/coling.pdf`
25. G. Huet. A functional toolkit for morphological and phonological processing, application to a Sanskrit tagger. *J. Functional Programming*, 15,4:573–614, 2005. `http://yquem.inria.fr/~huet/PUBLIC/tagger.pdf`.
26. G. Huet. *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, 2006.
27. G. Huet. 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, 2007. ACM. `http://yquem.inria.fr/~huet/PUBLIC/IWRIDL.pdf`
28. G. Huet and B. Razet. The reactive engine for modular transducers. In K. Futatsugi, J.-P. Jouannaud, and J. Meseguer, editors, *Algebra, Meaning and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, pages 355–374. Springer-Verlag LNCS vol. 4060, 2006. `http://yquem.inria.fr/~huet/PUBLIC/engine.pdf`
29. P. Kiparsky. On the architecture of Pāṇini's grammar. In *International Conference on the Architecture of Grammar, Hyderabad*, 2002.
30. P. Kiparsky and J. F. Staal. Syntactic and semantic relations in Pāṇini. *Foundations of Language*, 5:83–117, 1969.

31. S. C. Kleene. *Introduction to Metamathematics*. North Holland, Amsterdam, 1971 8th reprint (1st ed. 1952).
32. M. Kracht. The combinatorics of case. *Research on Language and Computation*, 1,1/2:59–97, 2003.
33. M. Kulkarni. Phonological overgeneration in paninian system. In G. Huet, A. Kulkarni, and P. Scharf, editors, *Topics in Sanskrit Computational Linguistics*. Springer-Verlag LNCS, this volume, 2009.
34. P. M. Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.
35. I. Mel'čuk. *Dictionnaire explicatif et combinatoire du français contemporain. Recherches lexico-sémantiques IV*. Les Presses de l'Université de Montréal, 1999.
36. T. Oberlies. *A Grammar of Epic Sanskrit*. De Gruyter, Berlin, 2003.
37. V. A. Pawan Goyal and L. Behera. Analysis of Sanskrit text: Parsing and semantic relations. In G. Huet and A. Kulkarni, editors, *Proceedings, First International Symposium on Sanskrit Computational Linguistics*, pages 23–36, 2007.
38. P. Ramanujan. Computer processing of Sanskrit. In *Computer Processing of Asian Languages Conference 2*. IIT Kanpur, 1992.
39. L. Renou. *La valeur du parfait dans les hymnes védiques*. Honoré Champion, Paris, 1925; 2ème édition étendue, 1967.
40. L. Renou. *Terminologie grammaticale du sanskrit*. Honoré Champion, Paris, 1942.
41. C. Rétoré. The logic of categorial grammars. Technical report, INRIA Rapport de recherche 5703, 2005. `http://www.inria.fr/rrrt/rr-5703.html`
42. V. Sastri. *Samskrita Bālādarśa*. Vadhyar, Palghat, 2002.
43. P. Scharf. Pāṇinian accounts of the vedic subjunctive. *Indo-Iranian Journal*, 48,1-2:71–96, 2005.
44. P. Scharf and M. Hyman. *Linguistic Issues in Encoding Sanskrit*. Motilal Banarsidass, Delhi, 2009.
45. J. S. Speijer. *Sanskrit Syntax*. E. J. Brill, Leyden, 1886.
46. J. F. Staal. *Word Order in Sanskrit and Universal Grammar*. Reidel, Dordrecht, 1967.
47. J. F. Staal. *Universals - Studies in Indian Logic and Linguistics*. The University of Chicago Press, 1988.
48. L. Tesnière, editor. *Éléments de Syntaxe Structurale*. Klincksieck, Paris, 1959.
49. B. Tikkanen. *The Sanskrit Gerund: a Synchronic, Diachronic and typological analysis*. Finnish Oriental Society, Helsinki, 1987.
50. G. A. Tubb and E. R. Boose. *Scholastic Sanskrit*. Columbia University, New York, 2007.
51. A. Verboom. Towards a sanskrit wordparser. *Literary and Linguistic Computing*, 3:40–44, 1988.
52. R. A. Yelle. *Explaining mantras*. Routledge, New York and London, 2003.

Appendix A : Phonemes from classical Sanskrit

The various transliteration schemes are designated as VH for the adaptation of the Velthuis scheme used in [27], KH for Kyoto-Harvard, WX for the Hyderabad-Tirupati scheme, SL for the one used by the Sanskrit Library effort.

In the table below, Code 14 is anusvāra, indicated by *bindu*, while code 15 is *anunāsika*, indicated by *candrabindu*. Code 50 is used to note the hiatus. It is needed for schemes VH and KH, since they are not prefix codes. Thus

a_i is needed to represent the hiatus without confusion with *ai*. Similarly for *a_u*, for instance in word *tita_u*. See Appendix B for another use of this code, independently from transliteration.

Vedic *l* is not accommodated at this point. It is proposed to reserve code 0 for hyphen and code -1 for avagraha. Other codes will be needed for extra punctuation, numerals, the marking of accent, optional marking of proper names, vocatives and other interjections, etc.

code	*deva*	*roma*	*VH*	*KH*	*WX*	*SL*	*code*	*deva*	*roma*	*VH*	*KH*	*WX*	*SL*
1	अ	a	a	a	a	a	26	ञ्	ñ	~n	J	F	Y
2	आ	ā	aa	A	A	A	27	ट्	ṭ	.t	T	t	w
3	इ	i	i	i	i	i	28	ठ्	ṭh	.th	Th	T	W
4	ई	ī	ii	I	I	I	29	ड्	ḍ	.d	D	d	q
5	उ	u	u	u	u	u	30	ढ्	ḍh	.dh	Dh	D	Q
6	ऊ	ū	uu	U	U	U	31	ण्	ṇ	.n	N	N	R
7	ऋ	ṛ	.r	R	q	f	32	त्	t	t	t	w	t
8	ॠ	ṝ	.rr	RR	Q	F	33	थ्	th	th	th	W	T
9	ऌ	ḷ	.l	L	L	x	34	द्	d	d	d	x	d
10	ए	e	e	e	e	e	35	ध्	dh	dh	dh	X	D
11	ऐ	ai	ai	ai	E	E	36	न्	n	n	n	n	n
12	ओ	o	o	o	o	o	37	प्	p	p	p	p	p
13	औ	au	au	au	O	O	38	फ्	ph	ph	ph	P	P
14	ं	ṃ	.m	M	M	M	39	ब्	b	b	b	b	b
15	ँ	ṁ	"m	M	z	~	40	भ्	bh	bh	bh	B	B
16	ः	ḥ	.h	H	H	H	41	म्	m	m	m	m	m
17	क्	k	k	k	k	k	42	य्	y	y	y	y	y
18	ख्	kh	kh	kh	K	K	43	र्	r	r	r	r	r
19	ग्	g	g	g	g	g	44	ल्	l	l	l	l	l
20	घ्	gh	gh	gh	G	G	45	व्	v	v	v	v	v
21	ङ्	ṅ	"n	G	f	N	46	श्	ś	"s	z	S	S
22	च्	c	c	c	c	c	47	ष्	ṣ	.s	S	R	z
23	छ्	ch	ch	ch	C	C	48	स्	s	s	s	s	s
24	ज्	j	j	j	j	j	49	ह्	h	h	h	h	h
25	झ्	jh	jh	jh	J	J	50	ऽ	ऽ	_	_	_	_

Appendix B : Input preprocessing

We sketch an algorithm for transforming a list of chunks separated by blanks into a list of proper forms fit for segmentation processing. The specification is the following. We want the input list of chunks to be obtainable from the phonemic form of the continuous saṃhitapāṭha of the considered Sanskrit text by mere insertion of blanks at the sandhi junction of the words (or at the necessary initial spacing in case of hiatus). We want the proper forms to be in final sandhi. For instance, the padapāṭha is a list of such proper forms, but it is not necessary that a proper form consists of just one *pada*, and indeed e.g. the form *tacchrutvā* is

proper. The form *viṣṇo* is a proper (vocative) form, whereas *devo* is not, although it is in final sandhi form as a string of phonemes, but the proper (nominative) form is *devaḥ*.

Finally, we want the algorithm to be deterministic, while being complete. That is, we want all segmentation solutions of the candidate sentence to be obtainable by concatenating individual solutions to the separate segmentation of the proper forms. Thus all non-deterministic search is done within the sandhi segmentation of the separate proper forms, but the production of these forms is a deterministic preprocessing. This last requirement poses a problem for certain hiatus situations, since a chunk ending in vowel *a* preceding a chunk starting with say vowel *u* could come by hiatus from a stem ending in *aḥ* as well as from a stem ending in *e*, as well as from a stem ending in *o*. In order to solve this dilemma, we shall not process these two chunks *... a u ...* as two separate forms, but rather keep them glued within one form *... a_u ...*, where the underscore indicates an explicit hiatus situation which will be handled by the sandhi splitter in a non-deterministic fashion. This is the reason why we added code 50 above in our tables, even though it is not a phoneme proper, but a glottal stop. We want to emphasize that this hiatus symbol is necessary in the internal representation of sandhi and its inversion, while it is neither necessary nor useful in the external input form – chunks do not contain underscores – except that they are needed exceptionally in order to input words which have internal hiatus such as *pra_uga* when the transliteration scheme is not a prefix code (this is the case for the VH and KH schemes above, whereas WX and SL are prefix codes, and indeed express a same-length transduction, admittedly a slight advantage). A final remark is that the full padapāṭha presentation is *not* in general a *bona fide* input form, since for instance neither *kutrāpi* nor *anyokti* nor *tacchrutvā* are decomposable into chunks obeying our requirements.

Let us now turn to the algorithm. We shall describe it in phases. It takes as input a list of chunks, and produces as output a list of proper forms. When the input is the empty list, so is the output. When the input is a unit chunk, we return it as a unit list of forms. Now assume the input is the non-unit list `[ chunk :: chunklist ]`, where we have recursively pre-processed `chunklist` into a non-empty list of proper forms `formlist`. Let `c` be the first character of the first form in `formlist`. In the general case, given `c` we adjust `chunk` into a proper form `form`, and we return the list `[ form :: formlist ]`. This adjustment is by cases on the last character `x` of `chunk`. If `x` is anusvāra, `form` is `chunk` where the last character is replaced by *m*. If `x` is *o*, then we replace it by the sequence *aḥ* whenever `c` changes accordingly this sequence into *o* by sandhi. This includes the possible case when it is avagraha, in which case it is replaced by *a* in the output form. If `x` is *d*, *n*, *c* or *l*, then we revert it to *t* whenever `c` changes accordingly *t* into `x` by sandhi. Finally, when `x` is *a*, if `c` is not a vowel different from *a* there is no adjustment, otherwise we return `formlist`, where the first form is changed by prefixing it with `chunk` followed by underscore. Similarly, when `x` is *ā*, if `c` is not a vowel there is no adjustment, otherwise we return `formlist`, where the first form is changed by prefixing it with `chunk` followed

by underscore, recognizing again a hiatus situation. To complete the algorithm, we only need to take care to rewrite an initial avagraha into its original *a*. The avagraha notation is necessary, by the way, as we shall see.

The reader will have understood that our algorithm implements simple cases of sandhi inversion, in order to interpret for instance the chunks list *tad api* as the forms list *tat api*, *tan matra* as *tat matra*, *tac ca* as *tat ca*. Thus the sentence *yad iha asti tad anyatra yan neha asti na tat kvacit* may be understood as a chunks list, and preprocessed to the forms list *yat iha asti tat anyatra yat neha asti na tat kvacit* where now each of the 11 forms may be separately un-sandhied, for instance using the algorithm given in [25], in order e.g. to segment further *neha* into the primitive forms *na iha*. Each of the forms is in terminal sandhi, like *tat*, which may be recognized as the final sandhi form of either the nominative or the accusative singular of the neuter pronoun *tad*. However, remark that we do the transformations only in cases where it is safe to be applied in a deterministic fashion. For instance, the *subhāṣita* input as the list of 8 chunks: *na hi kukkuṭyā ekadeśaḥ pacyata ekadeṣaḥ prasavāya kalpate* is preprocessed into the list of 7 forms *na hi kukkuṭyā ekadeśaḥ pacyata_ekadeṣaḥ prasavāya kalpate* so that it is left to the non-deterministic unsandhier to select among the 2 potential solutions of sandhi segmentation of the composite form *pacyata_ekadeṣaḥ* the right one, viz. *pacyate ekadeṣaḥ*. Note that underscore acts nonetheless as a segmentation hint, no segmentation overlapping with underscore will be attempted.

Maybe some extra argumentation of the prediction of *o* as *aḥ* in favorable contexts is needed. Care must be taken for completeness, since after all *o* is a permitted final, of vocative forms of *u* stems for instance. Thus the (admittedly ad-hoc) saṃhitapāṭha sentence *viṣṇodadhidadhyāt* may only be presented in padapāṭha form as the two chunks *viṣṇodadhi dadhyāt* but definitely not as the three chunks *viṣṇo dadhi dadhyāt*, since the preprocessing would yield the fautive form **viṣṇaḥ*. This is implied from the fact that sandhi of *aḥ* and *d* yields *od*. Furthermore, if one wants to force a segmentation hint at the juncture of *viṣṇo* and *dadhi*, it is easy to allow underscore to appear in the form *viṣṇo_dadhi*. All is needed is to add to the finite-state description of sandhi a few extra rules such as *o+d* →*o_d*, and the sandhi splitting will do the segmentation correctly into the forms *viṣṇo* and *dadhi*. Note that this does not overgenerate, since no other sandhi rules may produce *o_d*. Finally, an adaptation of this example to an initial *a* yields *viṣṇo'mṛtaṃdadhyāt*, which may be presented as the chunks list *viṣṇo'mṛtaṃ dadhyāt*. This example shows the necessity of the avagraha convention for completeness. Finally, we remark that anusvāra coming from sandhi of final *n* is *not* assimilated to ṃ by preprocessing, since it is followed by a sibilant and thus it is not at a place legal for segmentation. Consider for instance *devāṃśca*, which we cannot present as *devāṃ śca* any more than we can break *tacchrutvā*.

The preprocessing algorithm is very useful because it allows easy insertion of segmentation hints by placing extra spaces, without having to rewrite say *amṛtaṃdadhyāt* into the two final sandhi forms *amṛtam dadhyāt*: preprocessing will do that for you automatically from the chunks list *amṛtaṃ dadhyāt*.

Although this proprocessing seems to be consistent with the usual ways of presenting Sanskrit text in transliteration, there is no uniform convention, especially concerning the use of avagraha. Very often texts for students follow rather a padapāṭha way of presentation, so that beginners don't face the full complexity of sandhi. In such texts [42] we find: *kṛṣṇaḥ uttiṣṭhatu* instead of *kṛṣṇa uttiṣṭhatu*. Indeed such notation eases reading for the beginner, since it prevents his trying all segmentations with forms *kṛṣṇe* and *kṛṣṇo*. This difficulty turns into overgeneration of our Sanskrit processing tools. Even if the segmenter is driven by a lexicon of inflected forms, so that he will not consider *kṛṣṇo*, he will still have to consider *kṛṣṇe* besides *kṛṣṇaḥ*. These 2 choices, when cumulated in a long sentence, yield exponential behaviour in the number of chunks in hiatus situations. Furthermore, elimination of the *kṛṣṇe* branch will have to do some semantic evaluation, for all the morphological solutions (in this case, the 9 different morphological productions of *kṛṣṇe* from *kṛṣṇa* in the 3 genders). This is indeed a heavy price to pay. Of course, if we accept entering *kṛṣṇaḥ uttiṣṭhatu*, the segmentation will be unique. Note however that the form with vocative *kṛṣṇa* is not presentable with our convention as: *kṛṣṇa uttiṣṭha* (which would correspond actually to wrong padapāṭha as *kṛṣṇe* | *uttiṣṭha*). It must be presented in sandhied form as *kṛṣṇottiṣṭha*. But if we want to allow the absence of sandhi after an initial vocative, we have to invent some extra separator such as | in order to accept as input *kṛṣṇa* | *uttiṣṭha*. This separator will then allow a less ambiguous padapāṭha input, specially if it is completed by a similar convention for compound viccheda.

Similarly, remark that the part of our preprocessor which infers *yat* from chunk *yan* preceeding chunk *neha* is only an incomplete heuristics, which will prevent to put a space after a genuine *n* form such as *brahman* in a similar context. Note also that chunking such as *ity ukta* for *ityukta* is not recognized. In our interface we allow such chunking (as well as similar transformation of *u* into *v* before vowels), but in this case the effect of the preprocessor is just to glue the two chunks into one, losing the benefit of the segmentation information. The tradeoff here is between glueing for completeness, but not profiting of the segmentation hints, and applying some incomplete un-sandhi heuristics, obliging to forbid spaces in the cases where solutions would be missed. There is no completely optimal solution to such tradeoff. The most reasonable attitude at this point is to evaluate variations of the chunking preprocessing, in order to determine what is the best fit on a representative sample of the existing digitalised corpus. The transliteration parameter is irrelevant, since going from one column to another in Table I is an easy rational transduction, which can normalize a corpus in whatever phonemic transliteration or syllabic notation, in whatever encoding, ASCII or Unicode-UTF8 or whatever. The only difficulty will be to give codes to extra notation, dandas, other punctuation symbols, vedic letters, numerals, accents, prosody, etc. If some consensus can be reached for a few digital libraries, then the preprocessing of spaces and *avagraka*-like notations may be quickly standardized, allowing the design of inter-operable software.