Design of a Lexical Database for Sanskrit

Gérard Huet
INRIA-Rocquencourt
BP 105, 78153 Le Chesnay CEDEX
France
Gerard.Huet@inria.fr

Abstract

We present the architectural design rationale of a Sanskrit computational linguistics platform, where the lexical database has a central role. We explain the structuring requirements issued from the interlinking of grammatical tools through its hypertext rendition.

1 Introduction

Electronic dictionaries come into two distinct flavours: digital sources of dictionaries and encyclopedia, meant for human usage, and lexical databases, developed for computational linguistics needs. There is little interaction between the two forms, mostly for sociological reasons.

We shall argue, in this communication, that a lexical database may be used for both purposes, to mutual advantage. We base our thesis on a concrete experiment on the design of linguistics resources for the Sanskrit language.

2 From book form to web site

2.1 A Sanskrit-French paper dictionary

The author started from scratch a Sanskrit to French dictionary in 1994, first as a personal project in indology, then as a more structured attempt at covering Sanskrit elementary vocabulary. A systematic policy was inforced along a number of successive invariants. For instance, etymology, when known, was followed recursively through relevant entries. Any word could then be broken into morphological constituents, down to verbal roots when known. This "etymological" completeness requirement was at first rather tedious, since entering a new word may require the acquisition of many ancestors, due to complex compounding. But it appeared that the acquisition of new roots slowed down considerably after an initial "bootstrap" phase. When the number of entries approached 10000, with 520 roots, new roots acquisition became quite exceptional. This phenemenon is similar to the classical "lexical saturation" effect witnessed when one builds a lexicon covering a given corpus (Polguère, 2003). Progressively, a number of other consistency constraints were identified and systematically enforced, which proved invaluable in the long run.

At this point the source of the lexicon was a plain ASCII file in the LaTeX format. However, a strict policy of systematic use of macros, not only for the structure of the grammatical information, and for the polysemy representation, but also for internal quotations, ensured that the document had a strict logical structure, mechanically retrievable (and thus considerably easier to process without loss of information than an optically scanned paper dictionary (Ma et al., 2003)).

Indeed, around 2000, when the author got interested into adapting the data as a lexical database for linguistic processing, he was able without too much trouble to reverse engineer the dictionary into a structured lexical database (Huet, 2000; Huet, 2001). He then set to work to design a set of tools for further computer processing as an experiment in the use of functional programming for a computational linguistics platform.

The first design decision was to avoid standard databases, for several reasons. The first one is portability. Many database formats are proprietary or specific to a particular product. The second reason is that the functionalities of data base systems, such as query languages, are not well adapted to the management of lexical information, which is highly structured in a deep manner - in a nutshell, functional rather than predicative. Thirdly, it seemed best to keep the information in the concrete format in which it had been developed so far, with specific text editing tools, and various levels of annotation which could remain with the status of unanalysed comments, pending their possible later structuring. After all, ASCII is the most portable format, large text files is not an issue anymore, parsing technology is fast enough to make compilation times negligible, and the human ease of editing is the really crucial factor – any tool which the lexicographer has to fight to organise his data is counter-productive.

A detailed description of this abstract syntax is available as a research report (Huet, 2000), and will not be repeated here. We shall just point to salient points of this abstract structure when needed.

2.2 Grinding the abstract structure

The main tool used to extract information from this data-base is called the *grinder* (named after the corresponding core processor in the WordNet effort (Miller, 1990; Fellbaum, 1998)). The grinder is a parsing process, which recognizes successive *items* in the data base, represents them as an abstract syntax value, and for each such item calls a *process* given as argument to the grinder. In other words, `grind` is a parametric module, or *functor* in ML terminology. Here is exactly the module type `grind.mli` in the syntax of our pidgin ML:

```
module Grind : functor
  (Process : Proc.Process_signature)
  -> sig end;
```

with interface module `Proc` specifying the expected signature of a `Process`:

```
module type Process_signature = sig
value process_header :
 (Sanskrit.skt * Sanskrit.skt) -> unit;
value process_entry :
 Dictionary.entry -> unit;
value prelude : unit -> unit;
value postlude : unit -> unit;
end;
```

That is, there are two sorts of items in the data base, namely headers and entries. The grinder will start by calling the process `prelude`, will process every header with routine `process_header` and every entry with routine `process_entry`, and will conclude by calling the process `postlude`. Module interface `Dictionary` describes the dictionary data structures used for representing entries (i.e. its abstract syntax as a set of ML datatypes), whereas module `Sanskrit` holds the private representation structures of Sanskrit words (seen from `Dictionary` as an abstract type, insuring that only the Sanskrit lexical analyser may construct values of type `skt`).

A typical process is the printing process `Print_dict`, itself a functor. Here is its interface:

```
module Print_dict : functor
 (Printer:Print.Printer_signature)
        -> Proc.Process_signature;
```

It takes as argument a `Printer` module, which specifies low-level printing primitives for a given medium, and defines the printing of entries as a generic recursion over its abstract syntax. Thus we may define the typesetting primitives to generate a TEX source in module `Print_tex`, and obtain a TEX processor by a simple instanciation:

```
module Process_tex = Print_dict Print_tex;
```

Similarly, we may define the primitives to generate the HTML sources of a Web site, and obtain an HTML processor `Process_html` as:

```
module Process_html = Print_dict Print_html;
```

It is very satisfying indeed to have such sharing in the tools that build two ultimately very different objects, a book with professional typographical quality on one hand, and a Web site fit for hypertext navigation and search, as well as grammatical query, on the other hand[1].

2.3 Structure of entries

Entries are of two kinds: cross-references and proper lexemes. Cross references are used to list alternative spellings of words and some irregular but commonly occurring flexed forms (typically pronoun declensions). Lexeme entries consist of three components : syntax, usage, and an optional list of cognate etymologies in other Indo-European languages.

The syntax component consists itself of three sub-components: a heading, a list of variants, and an optional etymology. The heading spells the main stem (in our case, the so-called weak stem), together with a hierarchical level. At the top of the hierarchy, we find root verbs, non-compound nouns, suffixes, and occasional declined forms which do not reduce to just a cross reference, but carry some usage information. Then we have subwords, and subsubwords, which may be derived forms obtained

[1] http://pauillac.inria.fr/~huet/SKT/

by prefixing or suffixing their parent stem, or compound nouns.

Other subordinate entries are idiomatic locutions and citations. Thus we have a total of ten sorts of entries, classified into three hierarchical levels (to give a comparison, the much more exhaustive Monier-Williams Sanskrit-to-English dictionary has 4 hierarchical levels).

Let us now explain the structure of the usage component of our entries. We have actually three kinds of such usage structure, one corresponding to nouns (substantives and adjectives), another one corresponding to verbs, and still another one for idiomatic locutions. We shall now describe the substantives usage component, the verbs one being not very different in spirit, and the idioms one being a mere simplification of it.

The usage structure of a substantive entry is a list of *meanings*, where a meaning consists of a grammatical *role* and a *sense* component. A role is itself the notation for a part-of-speech tag and an optional positional indication (such as 'enclitic' for postfix particles, or 'iic' [*in initio composi*] for prefix components). The part-of-speech tag is typically a gender (meaning substantive or adjective of this gender), or a pronominal or numeral classifier, or an undeclinable adverbial role, sometimes corresponding to a certain declension of the entry. The thematic role 'agent' is also available as tag, typically for nouns which may be used in the masculine or the feminine, but not in the neuter. This results in a fairly flexible concrete syntax at the disposal of the lexicographer, put into a rigid but rigorous structure for computational use by the data base processors.

The sense component is itself a list of elementary semantic items (representing polysemy), each possibly decorated by a list of spelling variants. Elementary semantic items consist in their turn of an *explanation* labeled with a *classifier*. The classifier is either 'Sem', in which case the explanation is to be interpreted as a substitutive definition, or else it is a field label in some encyclopedic classification, such as 'Myth' for mythological entries, 'Phil' for philosophical entries, etc., in which case the explanation is merely a gloss in natural language. In every case the explanation component has type *sentence*, meaning in our case French sentence, since it applies to a Sanskrit-to-French bilingual dictionary, but here it is worth giving a few additional comments.

We remark that French is solely used as a semantic formalism, deep at the leaves of our entries. Thus there is a clear separation between a superstructure of the lexical database, which only depends on a generic dictionary structure and of the specific structure of the Sanskrit language, and terminal semantic values, which in our case point to French sentences, but could as well point to some other language within a multilingual context, or a WordNet-like (Fellbaum, 1998) pivot structure.

The strings denoting Sanskrit references are treated in a special way, since they determine the hypertext links in the HTML version of the dictionary. There are two kinds of possible references, proper nouns starting with an upper case letter, and common nouns or other Sanskrit words. For both categories, we distinguish *binding occurrences*, which construct HTML anchors, and *used occurrences*, which construct the corresponding references. In order to discuss more precisely these notions, we need to consider the general notion of scoping. But before discussing this important notion, we need a little digression about homonymy.

2.4 Homonyms

First of all, there is no distinction in Sanskrit between homophons and homographs, since the written form reflects phonetics exactly. As in any language however, there are homonyms which are words of different origin and unrelated meanings, but which happen to have the same representation as a string of phonemes (vocable). They may or may not have the same grammatical role. For such clearly unrelated words, we use the traditional solution of distinguishing the two entries by numbering them, in our case with a subscript index. Thus we distinguish entry *aja*$_1$ 'he goat', derived from root *aj* 'to lead', from entry *aja*$_2$ 'unborn', derived by privative prefix *a* from root *jan* 'to be born'.

Actually, primary derived words, such as substantival root forms, are distinguished from the root itself, mostly for convenience reasons (the usage structure of verbs being superficially different from the one of substantives). Thus the root *diś*$_1$ 'to show' is distinguished from the substantive *diś*$_2$ 'direction', and root *jñā*$_1$ 'to know' is distinct from the feminine substantive *jñā*$_2$ 'knowledge'.

2.5 Scoping

There are two notions of scoping, one global, and the other one local. First, every reference ought to point to some binding occurrence, somewhere in the data base, so that a click on any used occurrence in the hypertext document ought to result in a successful context switching to the appropriate link. Ideally this link ought to be made to a unique binding occurrence. Such binding occurrences may be explicit in the document; typically, for proper nouns, this corresponds to a specific semantic item, which explains their denotation as the name of some human or mythological figure or geographical entity. For common nouns, the binding occurrence is usually implicit in the structure of the dictionary, corresponding either to the main stem of the entry, or to some auxiliary stem or flexed form listed as an orthographic variant. In this sense a binding occurrence has as scope the full dictionary, since it may be referred to from anywhere. In another sense it is itself within the scope of a specific entry, the one in which it appears as a stem or flexed form or proper name definition, and this entry is itself physically represented within one HTML document, to be loaded and indexed when the reference is activated. In order to determine these, the grinder builds a lexical tree (trie) of all binding occurrences in the data base, kept in permanent storage. A cross-reference analysis checks that each used occurrence is bound somewhere.

Actually, things are still a bit more elaborate, since each stem is not only bound lexicographically in some precise entry of the lexicon, but it is within the scope of some grammatical role which determines uniquely its declension paradigm. Let us explain this by way of a representative example. Consider the following typical entry:

कुमार *kumāra* m. garçon, jeune homme; fils | prince; page; cavalier | myth. np. de Kumāra 'Prince', épith. de Skanda — n. or pur — f. *kumārī* adolescente, jeune fille, vierge.

There are actually four binding occurrences in this entry. The stem *kumāra* is bound initially with masculine gender for the meaning 'boy', and rebound with neuter gender for the meaning 'gold'. The stem *kumārī* is bound with feminine gender for the meaning 'girl'. Finally the proper name *Kumāra* is bound in the mythological sememe, the text of which contains an explicit reference to proper name Skanda.

3 The grammatical engine

We are now ready to understand the second stage in our Sanskrit linguistic tools, namely the grammatical engine. This engine allows the computation of inflected forms, that is declensions of nouns and finite conjugated forms of verbs. For nouns, we observe that in Sanskrit, declension paradigms are determined by a suffix of the stem and its grammatical gender. Since we just indicated that all defined occurrences of substantive stems occurring in the dictionary were in the scope of a gender declaration, this means that we can compute all inflected forms of the words in the lexicon by iterating a grammatical engine which knows how to decline a stem, given its gender.

Similary, for verbs, conjugation paradigms for the present system fall into 10 classes (and the aorist system has 7 classes). Every root entry mentions explicitly its (possibly many) present and aorist classes.

3.1 Sandhi

Given a stem and its gender, standard grammar paradigm tables give for each number and case a suffix. Glueing the suffix to the stem is computed by a phonetic euphony process known as *sandhi* (meaning 'junction' in Sanskrit). Actually there are two sandhi processes. One, called external sandhi, is a regular homomorphism operating on the two strings representing two contiguous words in the stream of speech. The end of the first string is modified according to the beginning of the second one, by a local euphony process. Since Sanskrit takes phonetics seriously, this euphony occurs not just orally, but in writing as well. This *external sandhi* is relevant to contiguous words, and compound formation.

A more complex transformation, called *internal sandhi*, occurs for words derived by affixes and thus in particular for inflected forms in declension and conjugation. The two composed strings influence each other in a complex process which may influence non-local phonemes. Thus prefixing *ni* (down) to root *sad* (to sit) makes verb *niṣad* (to sit down) by retroflexion of s after i, and further suffixing it with *na* for forming its past participle makes *niṣaṇṇa* (seated) by assimilation of d with n and further retroflexion of both occurrences of n.

While this process remains deterministic (except for occasional cases where some phonetic rules are optional), and thus is easily programmable for the synthesis of inflected forms, the analysis of such derivations is non-deterministic in a more complex way than the simple external sandhi, since it involves a complex cascading of rewrites.

3.2 Declensions

Using internal sandhi, systematic declension tables drive the declension engine. Here too the task is not trivial, given the large number of cases and exceptions. At present our nominal grammatical engine, deemed sufficient for the corpus of classical Sanskrit (that is, not attempting the treatment of complex vedic forms), operates with no less than 86 tables (each describing 24 combinations of 8 cases and 3 numbers). This engine may generate all declensions of substantives, adjectives, pronouns and numerals. It is to be remarked that this grammatical engine, available as a stand-alone executable, is to a large extent independent of the lexicon, and thus may be used to give the declension of words belonging to a larger corpus. However, the only deemed correctness is that the words actually appearing in the lexicon get their correct declension patterns, including exceptions.

This grammatical engine is accessible online from the hypertext version of the lexicon, since its abstract structure ensures us not only of the fact that every defined stem occurs within the range of a gender declaration, but conversely that every gender declaration is within the range of some defined stem. Thus we made the gender declarations (of non-compound entries) themselves mouse sensitive as linked to the proper instanciation of the grammatical CGI program. Thus one may navigate with a Web browser not only within the dictionary as an hypertext document (thus jumping in the example above from the definition of Kumāra to the entry where the name Skanda is defined, and conversely), but also from the dictionary to the grammar, obtaining all relevant inflected forms.

Similarly for roots, the present class indicator is mouse-sensitive, and yields on demand the corresponding conjugation tables. This underlines a general requirement for the grammatical tools: each such process ought to be callable from a concrete point in the text, corresponding unambiguously to a node in the abstract syntax of the corresponding entry, with a scoping structure of the lexicon such that from this node all the relevant parameters may be computed unambiguously.

In order to compute conjugated forms of non-root verbs, the list of its relevant preverbs is available, each preverb being a link to the appropriate entry (from which the etymological link provides the return pointer). Other derived stems (causative, intensive and desiderative forms) act also as morphology generators.

3.3 Inflected forms management

One special pass of the grinder generates the trie of all declensions of the stems appearing in the dictionary. This trie may be itself pretty-printed as a document describing all such inflected forms. At present this represents about 2000 pages of double-column fine print, for a total of around 200 000 forms of 8200 stems (133655 noun forms and 55568 root finite verbal forms).

3.4 Index management

Another CGI auxiliary process is the index. It searches for a given string (in transliterated notation), first in the trie of defined stems, and if not found in the trie of all declined forms. It then proposes a dictionary entry, either the found stem (the closest stem the given string is an initial prefix of) or the stem (or stems) whose declension is the given string, or if both searches fail the closest entry in the lexicon in alphabetical order. This scheme is very effective, and the answer is given instantaneously.

An auxiliary search engine searches Sanskrit words with a naive transcription, without diacritics. Thus a request for *panini* will return the proper link to *pāṇini*.

3.5 Lemmatization

The basic data structures and algorithms developed in this Sanskrit processor have actually been abstracted as a generic *Zen* toolkit, available as free software (Huet, 2002; Huet, 2003b; Huet, 2003d).

One important data structure is the *revmap*, which allows to store inflected forms as an invertible morphological map from stems, with minimal storage. The Sanskrit platform uses this format to store its inflected forms in a in such a way that it may directly be used as a lemmatizer. Each form is tagged with a list of pairs *(stem, features)*, where *features* gives all the morphological features used in the derivation of the form from root *stem*. A lemmatization procedure, available as a CGI executable, searches this structure. For instance, for form *devayos* it lists:

```
{ loc. du. m. | gen. du. m. |
  loc. du. n. | gen. du. n. }[deva]
```

where the stem *deva* is a hyperlink to the corresponding entry in the lexicon. Similarly for verbal forms. For *pibati* it lists:
`{ pr. a. sg. 3 }[paa_1]`, indicating that it is the 3rd person singular present form of root $pā_1$ in the active voice.

We end this section by remarking that we did not attempt to automate derivational morphology, although some of it is fairly regular. Actually, compound formation is treated at the level of segmentation, since classical Sanskrit does not impose any bound on its recursion depth. Verb formation (which sequences of preverbs are allowed to prefix which root) is explicit in the dictionary structure, but it is also treated at the level of the segmentation algorithm, since this affix glueing obeys external sandhi and *not* internal sandhi, a peculiarity which may follow from the historical development of the language (preverbs derive from postpositions). At present, noun derivatives from verbal roots are explicit in the dictionary rather than being computed out, but we envision in some future edition to make systematic the derivation of participles, absolutives, infinitives, and periphrastic future and perfect.

4 Syntactic analysis

4.1 Segmentation and tagging

The segmenter takes a Sanskrit input as a stream of phonemes and returns a stream of solutions, where a solution is a list of (inflected) words and sandhi rules such that the input is obtainable by applying the sandhi rules to the successive pairs of words. It is presented, and its completeness is proved, in (Huet, 2004). Further details on Sanskrit segmentation are given in (Huet, 2003a; Huet, 2003c).

Combined with the lemmatizer, we thus obtain a (non-deterministic) tagger which returns all the (shallow) parses of an input sentence. Here is an easy example:

```
# process "maarjaarodugdha.mpibati";

 Solution 1 :
[  maarjaaras
 < { nom. sg. m. }[maarjaara] >
   with sandhi as|d -> od]
[  dugdham
 < { acc. sg. m. | acc. sg. n. |
     nom. sg. n. }[dugdha] >
   with sandhi m|p -> .mp]
[  pibati
 < { pr. a. sg. 3 }[paa#1] >
   with sandhi identity]
```

This explains that the sentence *mārjārodugdhaṃpibati* (a cat drinks milk) has one possible segmentation, where *maarjaras*, nominative singular masculine of *maarjara* (and here the stem is a hyperlink to the entry in the lexicon glosing it as *chat* i.e. cat) combines by external sandhi with the following word by rewriting into *maarjaro*, followed by *dugdham* which is the accusative singular masculine of *dugdha* (draught) or the accusative or nominative singular neuter of *dugdha* (milk - same vocable), which combines by external sandhi with the following word by rewriting into its nasalisation *dugdhaṃ*, followed by *pibati* ... (drinks).

4.2 Applications to philology

We are now at the stage which, after proper training of the tagger to curb down its overgeneration, we shall be able to use it for scanning simple corpus (i. e. corpus built over the stem forms encompassed in the lexicon). The first level of interpretation of a Sanskrit text is its word-to-word segmentation, and our tagger will be able to assist a philology specialist to achieve complete morphological mark-up systematically. This will allow the development of concordance analysis tools recognizing morphological variants, a task which up to now has to be performed manually.

At some point in the future, one may hope to develop for Sanskrit the same kind of informative repository that the Perseus web site provides for Latin and Classical Greek[2]. Such resources are invaluable for the preservation of the cultural heritage of humanity. The considerable classical Sanskrit corpus, rich in philosophical texts but also in scientific, linguistic and medical knowledge, is an important challenge for computational linguistics.

Another kind of envisioned application is the mechanical preparation of students' readers analysing a text at various levels of information, in the manner of Peter Scharf's Sanskrit Reader[3].

The next stage of analysis will group together tagged items, so as to fulfill constraints of subcategorization (accessible from the lexicon) and

[2]`http://www.perseus.tufts.edu/`

[3]`http://cgi-user.brown.edu/Departments/Classics/Faculty/Scharf/`

agreement. The result ought be a set of consistent dependency structures. We are currently working, in collaboration with Brendan Gillon, on the design of an abstract representation for sanskrit syntax making explicit dislocations and anaphora antecedents, with the goal of building a consistent tree bank from his work on the analysis of the examples from Apte's manual (Apte, 1885; Gillon, 1996).

An interesting piece of design is the interface between lexicon citations and the corpus. An intermediate structure is a virtual library, acting as a skeleton of the corpus used for indexation. This way citations in the lexicon are mere pointers in the virtual library, which acts as a citations repository, but also possibly as a citation server proxy to the actual corpus materal when it is actually available as marked-up text. For lack of space, we omit this material here.

5 Conclusions

The computational linguistic tools should be modular, with an open-ended structure, and their evolution should proceed in a breadth-first manner, encompassing all aspects from phonetics to morphology to syntax to semantics to pragmatics to corpus acquisition, with the lexical database as a core switching structure. Proper tools have to be built, so that the analytic structure is confronted to the linguistic facts, and evolves through experimentally verifiable improvements. The interlinking of the lexicon, the grammatical tools and the marked-up corpus is essential to distill all linguistic information, so that it is explicit in the lexicon, while encoded in the minimal way which makes it non-redundant.

We have argued in this article that the design of an hypertext interface is useful to refine the structure of the lexicon in such a way as to enforce these requirements. However, such a linguistic platform must carefully distinguish between the external exchange formats (XML, Unicode) and the internal logical structure, where proper computational structures (inductive data types, parametric modules, powerful finite-state algorithms) may enforce the consistency invariants.

References

Vāman Shivarām Apte. 1885. *The Student's Guide to Sanskrit Composition. A Treatise on Sanskrit Syntax for Use of Schools and Colleges.* Lokasamgraha Press, Poona, India.

Christiane Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database.* MIT Press.

Brendan S. Gillon. 1996. Word order in classical Sanskrit. *Indian Linguistics*, 57,1:1–35.

Gérard Huet. 2000. Structure of a Sanskrit dictionary. Technical report, INRIA. `http://pauillac.inria.fr/~huet/PUBLIC/Dicostruct.ps`

Gérard Huet. 2001. From an informal textual lexicon to a well-structured lexical database: An experiment in data reverse engineering. In *Working Conference on Reverse Engineering (WCRE'2001).* IEEE.

Gérard Huet. 2002. The Zen computational linguistics toolkit. Technical report, ESSLLI Course Notes. `http://pauillac.inria.fr/~huet/ZEN/zen.pdf`

Gérard Huet. 2003a. Lexicon-directed segmentation and tagging of Sanskrit. In *XIIth World Sanskrit Conference, Helsinki.*

Gérard Huet. 2003b. Linear contexts and the sharing functor: Techniques for symbolic computation. In Fairouz Kamareddine, editor, *Thirty Five Years of Automating Mathematics.* Kluwer.

Gérard Huet. 2003c. Towards computational processing of Sanskrit. In *International Conference on Natural Language Processing (ICON), Mysore, Karnataka.*

Gérard Huet. 2003d. Zen and the art of symbolic computing: Light and fast applicative algorithms for computational linguistics. In *Practical Aspects of Declarative Languages (PADL) symposium.* `http://pauillac.inria.fr/~huet/PUBLIC/padl.pdf`

Gérard Huet. 2004. A functional toolkit for morphological and phonological processing, application to a Sanskrit tagger. *Journal of Functional Programming*, to appear. `http://pauillac.inria.fr/~huet/PUBLIC/tagger.pdf`.

Huanfeng Ma, Burcu Karagol-Ayan, David Doermann, Doug Oard, and Jianqiang Wang. 2003. Parsing and tagging of bilingual dictionaries. *Traitement Automatique des Langues*, 44,2:125–149.

G. A. Miller. 1990. Wordnet: a lexical database for English. *International Journal of Lexicography*, 3,4.

Alain Polguère. 2003. *Lexicologie et sémantique lexicale.* Presses de l'Université de Montréal.