Converting Phrase Structures to Dependency Structures in Sanskrit

Pawan Goyal
Department of CSE
Indian Institute of Technology
Kharagpur, India – 721302
pawang@cse.iitkgp.ernet.in

Amba Kulkarni
Department of Sanskrit Studies
University of Hyderabad
Hyderabad, India – 500046
apksh@uohyd.ernet.in

Abstract

Two annotations schemes for presenting the parsed structures are prevalent viz. the constituency structure and the dependency structure. While the constituency trees mark the relations due to positions, the dependency relations mark the semantic dependencies. Free word order languages like Sanskrit pose more problems for constituency parses since the elements within a phrase are dislocated. In this work, we show how the enriched constituency tree with the information of displacement can help construct the unlabelled dependency tree automatically.

1 Introduction

Sanskrit has a rich tradition of linguistic analysis with intense discussions and argumentations on various aspects of language analysis ranging from phonetics (śikṣā), grammar (vyākaraṇa), logic (nyāya), ritual exegesis (karmamīmāṃsā), and literary theory (alaṃkāraśāstra) which is not only useful for analysing Sanskrit but it also has much to offer computational linguistics in these areas. The series of symposia in Sanskrit Computational Linguistics (Huet et al., 2009; Kulkarni and Huet, 2009), the consortium project sponsored by the Technology Development for Indian Languages (TDIL) and the research of individual scholars and the collaborations (Goyal et al., 2012) among them resulted into a) development of several tools ranging from segmenters (Huet, 2009), morphological analysers (Kulkarni and Shukl, 2009), parsers (Goyal et al., 2009; Hellwig, 2009; Kumar, 2012; Kulkarni, 2013) to discourse annotators, b) lexical resources ranging from dictionaries, WordNet (Kulkarni et al., 2010) to Knowledge-Nets (Nair, 2011), and c) annotated corpora [http://sanskrit.uohyd.ernet.in/scl].

Pāṇinian grammar, the oldest dependency grammar, provides a formalism for annotation of the sentences. While the Sanskrit consortium has annotated a few thousand sentences following the dependency grammar, we also came across a very valuable source of annotation of Sanskrit sentences following the constituency structure (Gillon, 1996). The constituency structure was enriched to suite the requirements of Sanskrit. This aroused our curiosity to study the equivalence of the two annotation schemes.

The importance of dependency structure has been well recognised by several computational linguists (Culotta and Sorensen, 2004; Haghighi et al., 2005; Quirk et al., 2005) in the recent past. The dependency format is preferred over the constituency not only from evaluation point of view (Lin, 1998) but also because of its suitability (Marneffe et al., 2006) for a wide range of NLP tasks such as Machine Translation (MT), information extraction, question answering etc.. This has upsurged several works on converting a constituency structure into dependency. The parsers for English now produce the dependency parse as well. Xia and Palmer discuss three different algorithms to convert dependency structures to phrase structures for English (Xia and Palmer, 2001). Magerman gave a set of priority lists, in the form of a head percolation table to find heads of constituents (Magerman, 1994). Yamada and Matsumoto modified these head percolation rules further (Yamada and Matsumoto, 2003). Their method was reimplemented by Nivre, who also defined certain heuristics to infer the arc labels in the dependency tree produced (Nivre, 2006). Johansson and Nugues used a richer set of edge labels and

Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 1834–1843, Dublin, Ireland, August 23-29 2014.

introduced links to handle long-distance phenomena such as wh-movement, topicalization, expletives and gapping (Johansson and Nugues, 2007). Their conversion procedure made use of this extended structure in Penn Treebank. De et al. described a system for generating typed dependency parsed from the phrase structure parses (De Marneffe et al., 2006). (Palmer et al., 2009; Xia et al., 2009; Bhatt et al., 2009) discuss a multi-layered representation framework for Hindi and Urdu, where the information from syntactic as well as dependency parse is presented together.

In this work, we explore the relationship between enriched constituency structures and dependency structures for Sanskrit language, with main emphasis on the conversion from constituency to dependency structures. This work aims not only at designing an algorithm to convert Treebanks from one type of representation to the other, but also to judge the adequacy of the enriched constituency structure from parsing point of view. This paper has been organized as follows. Section 2 discusses the history and origin of this work. Section 3 describes the background of the constituency and dependency structures, utilized for Sanskrit language. Section 4 discusses the algorithm we used for converting constituency structure into dependency structure. Section 5 describes the results obtained by our approach, with some examples. Section 6 concludes this paper with the directions for future work.

2 Origin of the work

The dataset we are using in this work has its origin in the remarkable treatise on Sanskrit Syntax by Apte (Apte, 1885) which is the most authentic book on Sanskrit Syntax even after 125 years. The work was initiated in 1986 by Brendan Gillon, then a senior fellow at the American Institute of Indian Studies, at Deccan College, put all the prose exercise sentences from Apte's Student Guide onto 5×7 cards, assigning a syntactic parse to each sentence, giving each sentence an English translation and annotating each sentence for miscellaneous syntactic and semantic facts. On the basis of these sentences, Brendan Gillon published the grammar underlying his syntactic parse of these sentences (Gillon, 1996).

In 1991, Brendan Gillon transferred the material from a paper format to an electronic format, making revisions. An example sentence in this dataset is given below:

```
Example{3}
Source{1.1.3 (P) <U 4.5.3>} % Apte{7,3}

Parse
[S [INJ haa ] [ADV katham ]
   [NP1s [NP6 (mahaaraaja<Dasharathasya) ] (dharma<daaraa.h) ]
   [VP 0 [NP1 (priya<sakhii) [NP6 me ] [NP1 Kaushalyaa ] ] ] ]

Gloss{Oh, how is it that the legal wife of King Dasharatha is my dear
friend Kaushalyaa}

Comment{copula: covert: predicational: NP1s VP }
```

Each example is given a serial number, its source - the corresponding reference in Apte's book. Then, its constituency parse is provided in a tree structure. The Sanskrit text is transliterated into Roman using the Velthuis notation[1]. Finally, the gloss (translation) of the prose is provided along with some observations regarding syntax in the field 'comment'. The proper nouns are transliterated following the English convention of capitalisation. The constituency structure is enriched reflecting the morphological information such as the case marker. The underlying constituency structure of the compounds is also shown clearly marking the head of the compound. The requirement that constituency tree be a binary is also done away with, resulting into a more flat structure than the normal hierarchical phrase structure.

In 2004, Gérard Huet re-engineered the document in order to parse it mechanically, and he verified its correct syntactic structure after typographical corrections. He devised an abstract syntax to formalize this constituency structure. In the abstract syntax, the above constituency structure is represented as below:

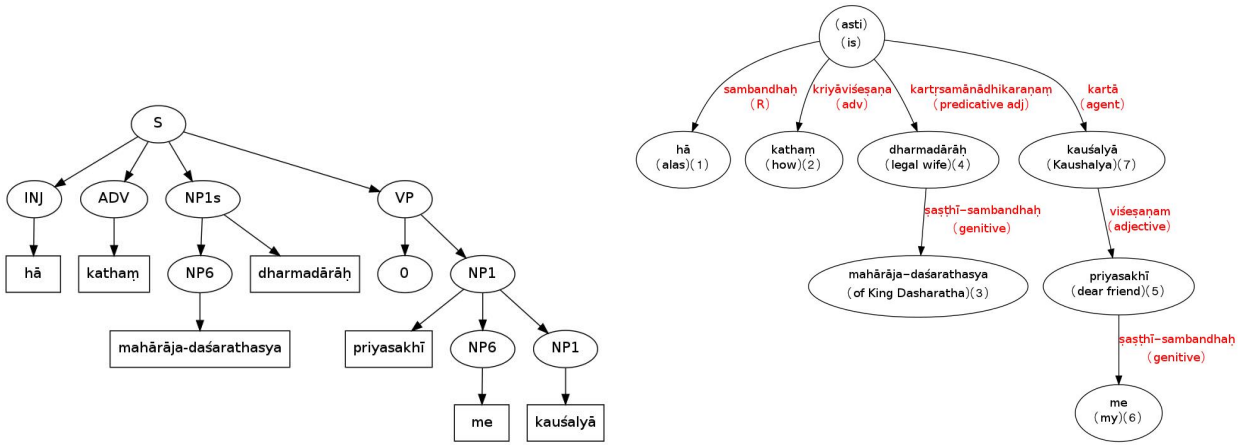
```
list Tag_tree.syntax =
[S
```

[1]Originally developed in 1991 by Frans Velthuis for use with his devnag Devanagari font, designed for the TeX typesetting system.

```
[INJ ("haa", 1); ADV [("katham", 2)];
 NP
  ([Case 1; Role Subject],
  [NP ([Case 6], [N (Compound (Stem <mahaaraaja>, Stem <Dasharathasya>),
   3)]);
   N (Compound (Stem <dharma>, Stem <daaraa.h>), 4)]);
 VP0
  [NP
    ([Case 1],
    [N (Compound (Stem <priya>, Stem <sakhii>), 5);
     NP ([Case 6], [N (Stem <me>, 6)]);
     NP ([Case 1], [N (Stem <Kaushalyaa>, 7)])]);
   NIL 8]]]
```

Each stem is given a unique index. The syntax, while preserving the original structure of the text, gives additional structuring with the word numbers, explicit case markers and stems for the compounds. While these constituent trees preserve much of the tagging related information, they still do not have the gender and number information for the substantives, for instance. This information can enhance the constituency representation further.

The same set of sentences were also parsed manually by Sheetal Pokar, a research scholar at the University of Hyderabad, showing the dependency structure. Sheetal followed the annotation guidelines developed by the Consortium of Institutes working on the Development of Sanskrit Computational Tools[2]. This tagset has a little above 40 tags marking various relations. The dependency tree for the example 3, discussed above, is shown in Figure 1. It is a directed tree with nodes corresponding to the words in the sentence and edges corresponding to the relation between the head and the modifier. Each node has a number indicating the word index. A generic relation *sambandhaḥ* (R) is used if the relation does not fall under any of the given tags. As one may notice, both the constituency as well as the dependency structures posit a NULL verb 'to be' *asti*. Among the Indian schools dealing with verbal cognition, not all schools accept the insertion of missing copula. We follow the grammarian school who accept this insertion.



(a) (b)

Figure 1: Example 3: (a). Constituency Parse and (b). Dependency Parse

While the two structures in Figure 1 mark different kind of information, we notice that the dominance relation in the constituency structure under each phrase corresponds to the the modifier-modified relation in the dependency tree. This was the main motivation to develop the converter to convert a phrase structure into an unlabelled dependency structure.

[2] http://sanskrit.uohyd.ernet.in/scl

3 Dependency and Constituency Structures

Verbal understanding of any utterance requires the knowledge of how words in that utterance are related to each other. There are two major representational frameworks for representing this knowledge as a parse tree viz. constituency and dependency parse trees. Constituency trees show how the individual units in a sentence are grouped together leading to semantically richer phrases in the constituency structures. The dependency structure, on the other hand, shows how each word is related to other words in the sentence either directly or indirectly.

The constituency structure derives from the subject-predicate division of Latin and Greek grammars that is based on term logic. Basic clause structure is understood in terms of a binary division of the clause into subject (noun phrase NP) and predicate (verb phrase VP). These ideas originated with Leonard Bloomfield (Bloomfield, 1962) and were further developed by a number of American structuralist linguists, including Harris (Harris, 1955) and Wells (Wells, 1947). Though these grammars were initially conceived as applying only to phrases, it was shown that such rules could be used for analyzing compounds as well as derivational morphology for Sanskrit. Gillon showed that the same extension works for classical Sanskrit as well (Gillon, 1995).

The dependency analysis dates back to Pāṇini who uses the syntactico-semantic relations called *kāraka* relations for the linguistic analysis of a sentence. In modern times, the seminal work of Tesnière (Tesnière, 1959) became the basis for the work on dependency grammar. Meaning-Text Theory (Melĉuk, 1988), Word grammar (Hudson, 1984), Functional Generative Description (Segall et al., 1986) are some of the flavours of the dependency grammar. A dependency parse is generally modelled as a directed tree with nodes representing the words and edges representing the possible relations between them. A typed dependency parse also labels the relations. For every element (word or morph) in a sentence, there is just one node in the syntactic structure.

Xia and Palmer (Xia and Palmer, 2001) discuss three different algorithms to convert dependency structures to phrase structures for English. They also attempt to clarify the differences in representational coverage of the two approaches. In the first stage, they identify the head of each constituent of the sentence, which is further modified to retrieve the semantic head. In the second phase, they label each of the dependency extracted with a grammatical relation using patterns defined over the phrase structure tree. (Palmer et al., 2009; Xia et al., 2009; Bhatt et al., 2009) discuss a multi-layered representation framework for Hindi and Urdu, where the information from syntactic as well as dependency parse is presented together. They first construct a dependency parse and then convert it into the constituency parse tree using conversion rules. A conversion rule is a (DS pattern, PS pattern) pair, where DS and PS correspond to dependency and phrase structure respectively.

The constituency parse we are dealing with being enriched with linguistic information pertaining to the morphology of the simple as well as compound words, it was much simpler to convert this structure into a dependency structure. In the next section, we will discuss our algorithm for converting a constituency structure to a dependency structure.

4 Conversion from Constituency to Dependency Structure in Sanskrit

The notion of 'head' is very important for both the constituency and dependency structures. In the constituency structure, the head determines the main properties of the phrase. Head may have several levels of projection. In the dependency structure, on the other hand, the head is linked to its dependents. The core of the algorithm is to identify the head of each phrase in the constituency tree and establish its relation with the head of its parent node. And also to establish the relation between the head with its dependents. The head for each XP is the node X within that XP. Thus the head for an NP is the noun, head for a VP is the verb, and so on. The head for the S is the head of a VP, in case it is a simple sentence, and head of the VP of the main clause in case it is a complex sentence. In case of complex sentences, we identified the main clause taking clues from the connectives. Each relation is named after the XP of the modifier. Our algorithm for finding the head node is implemented on the abstract syntax discussed in section 2 before. A rough outline of the algorithm is:

1)The head of VP is the ROOT node in the dependency tree.

a) In the case of sentences with sub-ordinate clauses, identify the main clause taking clues from the connectives.
b) Head of a clause is an auxiliary, if present, otherwise the main verb is the head.
c) In the case of sentences with quotative markers, the verb of the main clause is the head.
(The later rule is stronger than the previous.)
2) All the XPs within VP are dependent on the ROOT.
3) If S is the parent of VP, then all the XPs which are children of S are also dependent on this ROOT.

Finding the head for each node was not trivial though, as many of the parses involve dislocated phrases, which were not fully marked. We had to enrich the constituency trees by incorporating the dislocation information, which was provided in comments and was missing from the tree notations. We used '!' and '$' to indicate the dislocation. '!' indicates the position from where a component is dislocated, while '$' indicates the dislocated component. An example of a constituency parse, enriched with dislocation information is given below.

```
Example{2}
Source{1.1.2 (P) <V 3.28; V 3.6.3>} % Apte{7,2}

Parse
[S [ADV sarvatra ] [NP6 audarikasya $1]
   [VP 0 [NP1 abhyavahaaryam [PRT eva ] ] ]
   [NP1s !1 vi.saya.h ] ]

Gloss{In every case, a glutton's object is only food.}

Comment{copula: covert: predicational: VP NP1s
"eva" in predicate NP
left extraposition from NP1s of NP6 within MC, modulo adverbial ADV.}
```

This constituency tree involves one dislocated phrases. This information is marked with '!1'for the place from which it is dislocated and with '$1' for the phrase that has been dislocated. This dislocation information is used by our algorithm to find the right relata for the dislocated words. In case of more than one dislocated phrases, they are numbered sequentially.

5 Results and Discussions

We implemented our algorithm on a dataset of 232 sentences and matched the output of our algorithm with the Gold dataset, the dependency graph constructed manually for the sentences by Sheetal. Figure 2 shows the dependency structure for Example 3, produced by our conversion algorithm.

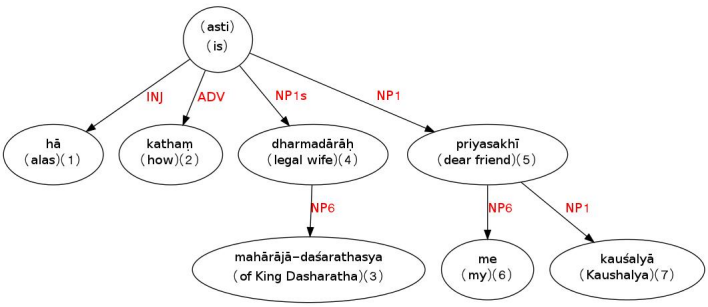


Figure 2: Dependency graph constructed by conversion from constituency structure

The conversion algorithm captures the relations between various constituents and produces a dependency graph. Labels of the dependency graph correspond to the intermediate nodes in the phrase structure. On comparing the graph in Figure 2 with the graph in Figure 1, we find that most of the original connections were captured. However, there was a mismatch in the relation between words *kauśalyā*, *priyasakhī* and *me*. This discrepancy is because of the non-agreement between the annotators as to which one is the head. In case of sentences with apposition, in Sanskrit, the two annotators have difference of opinion as to which among the two is the head. This resulted in the mismatch between the two graphs. The relations in this graph are labelled by the dominating XP.

Pāṇini's grammar provides rules for assigning case markers given the syntactico-semantic relation between the relata. Inverting these rules it should be possible to get the relation labels. These relation labels, however, will not be obtained deterministically. The non-determinism will lead to multiple labelled dependency structures. Hence we could not assign the labels from the tagset, and resorted to the names of the phrases which the word belongs to.

Below we give an example where we found an exact match between the manual dependency graph and the dependency graph, produced by our conversion algorithm, using the constituency parse of the sentence.

```
Example{29}
[S [VP [NP7 tatra ] [CNJ ca ]
       [NP5 [NP6 [AP6 (((nikhila<(dhara.nii<tala))<parya.tana)<khinnasya) ]
                  (nija<balasya) ]
            (vizraama<heto.h) ]
       [NP2 [AP2 katipayaan ] divasaan ]
       ati.s.that ] ]
Gloss{And he remained there for a few days in order to rest his army
exhausted from roaming the entire surface of the earth.}
```

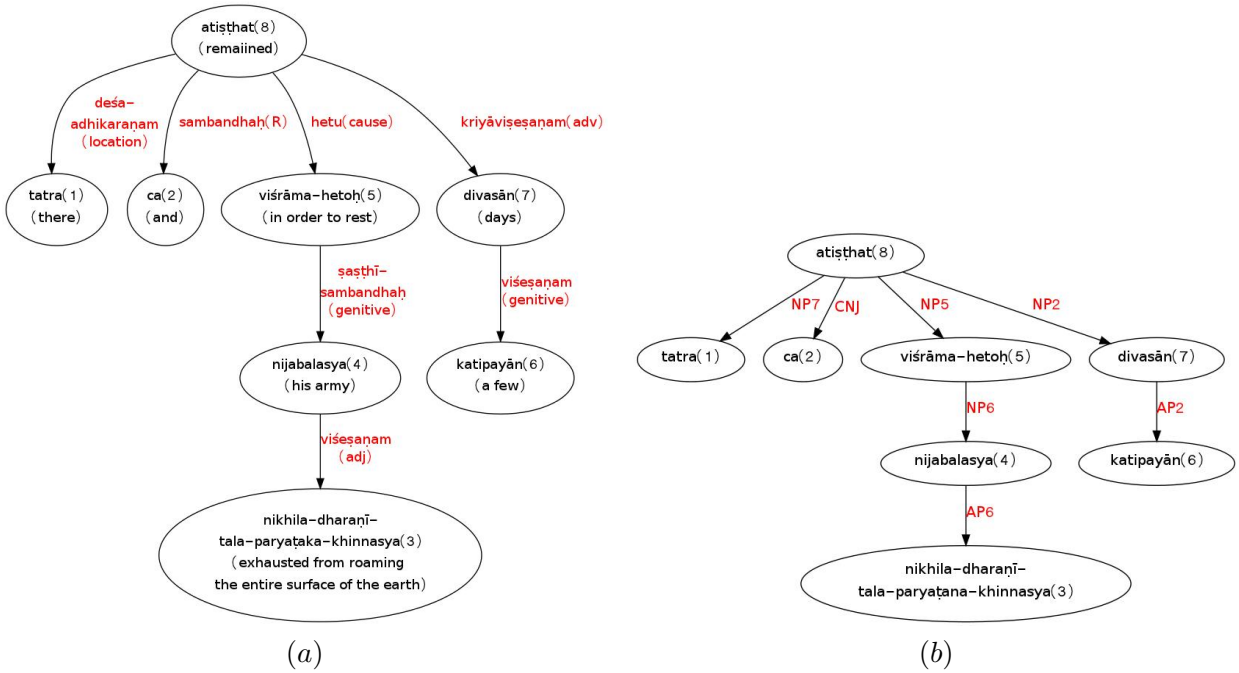


Figure 3: Example 29: (a). Dependency graph constructed manually and (b). Dependency graph constructed by conversion from constituency structure

Out of 232 cases, we found 97 such cases with exact match. For the rest of the cases,
1) In 40 cases, number of words in dependency and phrase-converted graph are different. For example, the words such as *kadācit* 'probably', *yadyapi* 'even if', *tathāpi* 'even then', *athavā* 'or' etc. were treated in one structure as a single word while in the other as two words. These words at morphological level consist of two morphemes which have independent existence. So it was natural to treat these as two words, in the constituency trees. However, at the semantic level, these two words indicate a single meaning which at times is non-compositional. The annotator of dependency graph has treated these as a single word.
2) In 95 cases, one or more relations do not match. These were due to various reasons such as

1. differences in the treatment and identification of adjectives,
2. disagreement in the attachment, and
3. cases of ellipsis, null head, and cases where the treatment of conjunct '*ca*' (and) differs.

These differences are very much important from linguistic analysis point of view. However due to space constraint we illustrate here only two cases where the treatment of the two annotators differ.

```
Example{72}
[S [VOC sakhi [VOC Vaasanti ] ]
   [VP 0 [NP4 du.hkhaaya [PRT eva ] [NP6 su-h.rdaam ] ] ]
   [NP1s [ADV idaaniim ] [NP6 Raamasya ] darshanam ] ]

Gloss{Oh my friend Vaasanti, seeing Raama now leads only to
the unhappiness of his friends.}
```

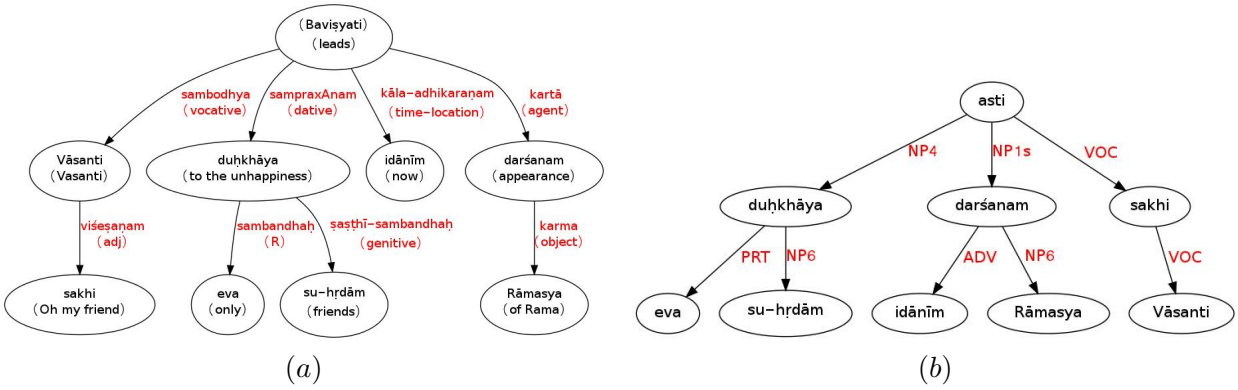


Figure 4: Example 72: (a). Dependency graph constructed manually and (b). Dependency graph constructed by conversion from constituency structure

In this example, there are two places where the annotators disagree.
a) The attachment of the word *idānīm* (now).
b) The decision of head in case of vocative with a modifier.
The null copula (VP 0) in the constituency structure is replaced by the Sanskrit verbal form *asti* uniformly. The annotator of dependency graph has provided a more appropriate verb *bhaviṣyati*. However, we ignore this difference.

Let us look at another example, where the ambiguity in morphological analysis has led to the ambiguity in the readings resulting in the disagreement in the parse.

```
Example{46}
[S [NP1s aarya.h ]
   [VP daapayatu [NP2 [NP6 me ]
                      [NP4 (Vaisha.mpaayana<aanayanaaya) ]
                      (gamana<abhyanuj~naam) ]
                 [NP3 taatena ] ] ]

Gloss{May you, sir, make my father give me permission to go to bring
Vaisha.mpaayana.}
```

In this example, the pronominal form *me* is ambiguous between two readings. It can be either a genitive or a dative of the first person pronoun *asmad* 'I'. The sub-ordinate clause is analysed by Gillon as 'the permission for going to bring Vaiśampāyan by me'. In Sanskrit the first person pronoun in such cases takes genitive case marker. Sheetal on the other hand has analysed it as 'the permission to me for going to bring Vaiśampāyan', where *me* is analysed with dative case. It is clear that 'to bring Vaiśampāyan' is the purpose for going. But since 'permission for going' is a compound in Sanskrit [3], and the 'permission' being the head of this compound, Sheetal avoided linking 'to bring Vaiśampāyana' with 'permission to go' as it results into an '*asamartha samāsa* (incompatible compound formation, where the external modifier connects the modifier component of a compound and not the head). This has resulted into the differences in annotation. Such compounds are not rare. Thus, in order to provide a correct parse, in case of dependency structure, it is necessary to show the internal structure of the compounds as well,

[3]In Sanskrit a compound is always written as a single word without any space in between.

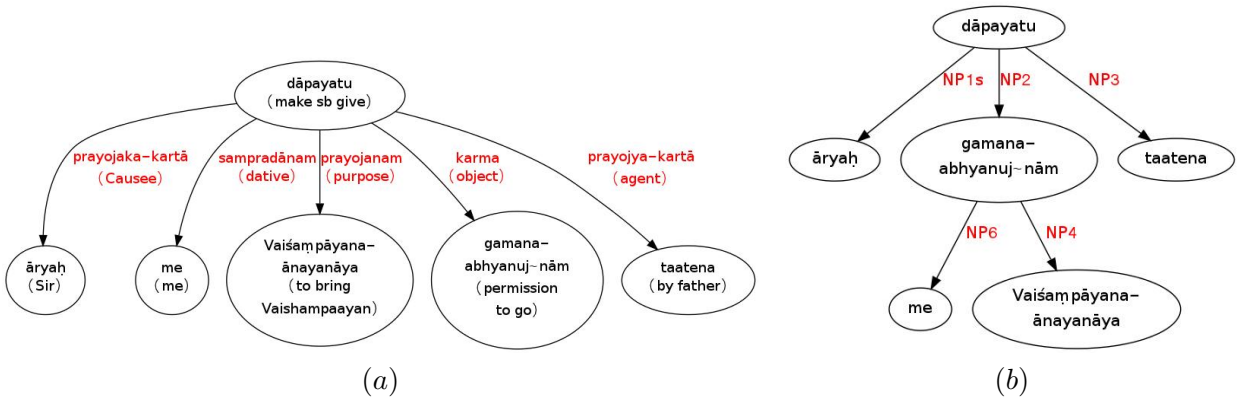


Figure 5: Example 46: (a). Dependency graph constructed manually and (b). Dependency graph constructed by conversion from constituency structure

rather than treating a compound unanalysed. This will allow one to connect the elements to the part of a compound other than the head. Similar treatment is necessary in the phrase structure annotation as well.

We end with example 2 from section 4, where the dislocation information in the constituency tree helps in retrieving the correct dependency structure. In this example, even though the word *audarikasya* has been displaced, the displacement information in the parse tree positions it at the correct place in the dependency tree constructed from the constituency structure.

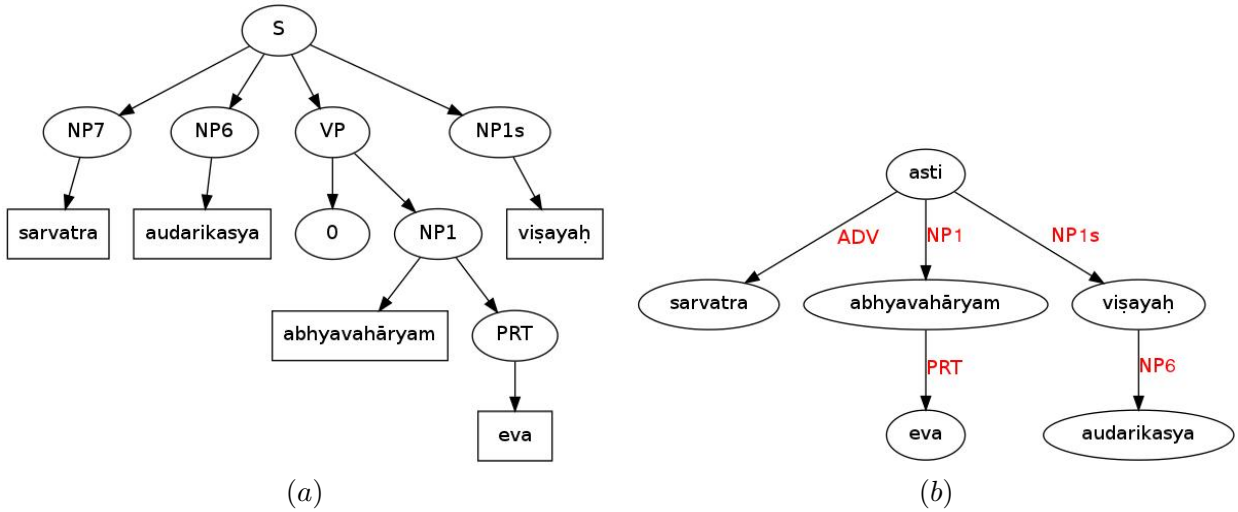


Figure 6: Example 2: (a). Constituency structure and (b). Dependency graph constructed by conversion from constituency structure

6 Conclusions and Future Work

This work focussed mainly on conversion from constituency to dependency structure. The sentences in our dataset are chosen from (Apte, 1885), which is an authentic book for higher learning of Sanskrit, covering a wide range of grammatical constructions. The tool was tested on a dataset of 232 sentences and the initial results were encouraging. Specifically, most of the cases of mismatch were linguistic issues and need further discussion. The phrase labels indicating the case labels is an important extension of the constituency trees to accommodate morphologically rich languages. The enriched constituency structure has an advantage of recording the word order, and at the same time marking the dislocation information. In this work, we have shown that such enriched constituency tree can help construct the unlabelled dependency tree automatically. Further, one may also try inferring the dependency relation names, and use statistical parsing to resolve non-determinism favouring popular usages.

Another interesting aspect would be to try the other way conversion, that is, from dependency to phrase structure. The main challenge for this conversion is to find out the projection table corresponding to each lexical item. This work will also be a first step towards an abstract syntax, which can inherit the properties of both the constituency and dependency structures. If so, this would be an alternative formalism for tagging the Sanskrit corpus.

Acknowledgements

The authors would like to acknowledge the discussions with Gérard Huet, INRIA Paris Rocquencourt, towards enriching the abstract syntax. The work was also supported by Emilie Aussant and Sylvie Archaimbault, laboratoire HTL, Université Paris Diderot.

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.

Rajesh Bhatt, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, and Fei Xia. 2009. A multi-representational and multi-layered treebank for hindi/urdu. In *Proceedings of the Third Linguistic Annotation Workshop*, pages 186–189. Association for Computational Linguistics.

Leonard Bloomfield. 1962. Language. *New York: Holt*.

A. Culotta and J. Sorensen. 2004. Dependency tree kernels for relation extraction. In *42nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 423–429, Barcelona, Spain.

Monali Das and Amba Kulkarni. 2013. Discourse level tagger for mahābhāṣya - a Sanskrit commentary on pāṇini's grammar. In *Proceedings of the 10th International Conference on NLP, Delhi, India*.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Brendan S. Gillon. 1995. Autonomy of word formation: evidence from Classical Sanskrit. *Indian Linguistics, 56 (1-4)*, pages 15–52.

Brendan S Gillon. 1996. Word order in classical sanskrit. *Indian Linguistics*, 57(1-4):1–35.

Brendan S. Gillon. 2009. Tagging classical Sanskrit compounds. In Amba Kulkarni and Gérard Huet, editors, *Sanskrit Computational Linguistics 3*, pages 98–105. Springer-Verlag LNAI 5406.

Pawan Goyal, Vipul Arora, and Laxmidhar Behera. 2009. Analysis of Sanskrit text: Parsing and semantic relations. In Gérard Huet, Amba Kulkarni, and Peter Scharf, editors, *Sanskrit Computational Linguistics 1 & 2*, pages 200–218. Springer-Verlag LNAI 5402.

Pawan Goyal, Gérard Huet, Amba Kulkarni, Peter Scharf, and Ralph Bunker. 2012. A distributed platform for sanskrit processing. In *Proceedings of 24th COLING, Mumbai, India*.

A.D. Haghighi, A.Y. Ng, and C.D. Manning. 2005. Robust textual inference via graph matching. In *Human Language Technology Conference (HLT) and Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 387–394, Vancouver, Canada.

Zellig S Harris. 1955. From phoneme to morpheme. *Language*, 31(2):190–222.

Oliver Hellwig. 2009. Extracting dependency trees from Sanskrit texts. In Amba Kulkarni and Gérard Huet, editors, *Sanskrit Computational Linguistics 3*, pages 106–115. Springer-Verlag LNAI 5406.

R. Hudson. 1984. *Word Grammar*. Basil Blackwell, Oxford.

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

Gérard Huet. 2009. Formal structure of Sanskrit text: Requirements analysis for a mechanical Sanskrit processor. In Gérard Huet, Amba Kulkarni, and Peter Scharf, editors, *Sanskrit Computational Linguistics 1 & 2*. Springer-Verlag LNAI 5402.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *Proceedings of the 16th Nordic Conference on Computational Linguistics (NODALIDA)*, pages 105–112.

Amba Kulkarni and Monali Das. 2012. Discourse analysis of sanskrit texts. In *Proceedings of the workshop on Advances in Discourse Analysis and its Computational Aspects, 24th COLING, Mumbai, India*.

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

Amba Kulkarni and K. V. Ramakrishnamacharyulu. 2013. Parsing Sanskrit texts: Some relation specific issues. In Malhar Kulkarni, editor, *Proceedings of the 5th International Sanskrit Computational Linguistics Symposium*. D. K. Printworld(P) Ltd.

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

Malhar Kulkarni, Chaitali Dangarikar, Irawati Kulkarni, Abhishek Nanda, and Pushpak Bhattacharyya. 2010. Introducing sanskrit wordnet. In Christiane Felbaum Pushpak Bhattacharyya and Piek Vossen, editors, *Principles, Construction and Application of Multilingual Wordnets, Proceedings of the Global Wordnet Conference, 2010*. Narosa Publishing House, New Delhi.

Amba Kulkarni. 2013. A deterministic dependency parser with dynamic programming for Sanskrit. In *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pages 157–166, Prague, Czech Republic, August. Charles University in Prague, Matfyzpress, Prague, Czech Republic.

Anil Kumar. 2012. *An automatic Sanskrit Compound Processing*. Ph.D. thesis, University of Hyderabad, Hyderabad.

Dekang Lin. 1998. Dependency-based evaluation of minipar. In *Workshop on the evaluation of Parsing Systems, Granada, Spain.*

David Mitchell Magerman. 1994. *Natural Language Parsing As Statistical Pattern Recognition*. Ph.D. thesis, Stanford, CA, USA. UMI Order No. GAX94-22102.

Marie-Catherine De Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC-06*.

I. Melĉuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany.

Sivaja Nair. 2011. *The Knowledge Structure in Amarakośa*. Ph.D. thesis, University of Hyderabad, Hyderabad.

Joakim Nivre. 2006. *Inductive dependency parsing*. Springer.

Martha Palmer, Rajesh Bhatt, Bhuvana Narasimhan, Owen Rambow, Dipti Misra Sharma, and Fei Xia. 2009. Hindi syntax: Annotating dependency, lexical predicate-argument structure, and phrase structure. In *The 7th International Conference on Natural Language Processing*, pages 14–17.

C. Quirk, A. Menezes, and C. Cherry. 2005. Dependency treelet translation: Syntactically informed phrasal smt. In *43rd Annual Meeting of the Association for, Computational Linguistics (ACL)*, pages 271–279, Ann Arbor, USA.

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

P. Segall, E. Hajiov, and J. Panevov. 1986. *The Meaning of the Sentence in its Semantic and Pragmatic Aspects*. Springer, Heidelberg.

L Tesnière. 1959. *Éléments de syntaxe structurale*. Klinksieck, Paris.

Rulon S Wells. 1947. Immediate constituents. *Language*, 23(2):81–117.

Fei Xia and Martha Palmer. 2001. Converting dependency structures to phrase structures. In *Proceedings of the first international conference on Human language technology research*, pages 1–5. Association for Computational Linguistics.

Fei Xia, Owen Rambow, Rajesh Bhatt, Martha Palmer, and Dipti Misra Sharma. 2009. Towards a multi-representational treebank. In *The 7th International Workshop on Treebanks and Linguistic Theories. Groningen, Netherlands*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. *Proceedings of IWPT*, 3.