Dependency Analysis of Sanskrit Compounds*

Amba Kulkarni[1], Pavankumar Satuluri[2], Sriram Krishnan[3,4]
[1] Department of Sanskrit Studies, University of Hyderabad
[2] School of Humanities and Social Sciences, IIT Roorkee
[3] Central Sanskrit University, Prayagraj
[4] IIIT, Hyderabad
ambakulkarni@uohyd.ac.in

Abstract

Sanskrit is rich in compound formation. Therefore a sentential parsing is incomplete without analysing the compounds revealing the relations between its components and also the hierarchies between the nested components.

Constituency analysis for a compound appears as a natural choice given the hierarchical structure with binary compound formation, barring the *Dvandva* (co-ordinative) and *Bahuvrīhi* (exo-centric) compounds involving more than two components. The compound analysis, in addition to the hierarchical structure, also involves identification of the type of the compound. Often a compound in isolation is ambiguous, confusion between the *Tatpuruṣa* and *Bahuvrīhi* being the most common. It is the context that helps resolve such ambiguities in certain cases. Hence access to the context, at least to the sentence, is desirable. Use of '*asamartha*' compounds is also very frequent where the expectancies of the modifier or the iic(in initio composi) of the compound are satisfied by other words in the sentence.

In this paper we propose a representation for the compound analysis as a plane tree without edge-crossings that is equivalent to the constituency structure. This representation makes the extention of the dependency sentential parser to compound analysis trivial. Further it is natural for a human being to cognise the meaning of a compound and also understand the compound structure as a hierarchical binary tree rather than a dependency tree. Therefore we also represent the dependency parse of a compound as a constituency tree thereby producing a human friendly output.

1 Introduction

The rich compound formation in Sanskrit makes sentential parsing incomplete without compound analysis. Compound analysis involves revealing the relations between their components and also the hierarchies between the nested components. The constituency structure fits very well for the analysis of compounds. Sanskrit, on the other hand, being a free word order language the constituency structure at sentence level does not make any sense. The dependency structure marking the relations between words provides an apt representation to understand the semantics of a sentence. It is possible to plug-in a constituency parser for the compound analysis into a dependency parser for the sentential analysis provided the compound analysis can be carried out independent of the sentential analysis. But the compound analysis is sometimes context dependent and is influenced by other words in the sentence. For example. the compound word '*Rāmeśvaraḥ*' is an endo-centric compound (*Tatpuruṣa*) with its paraphrase *Rāmasya īśvaraḥ* in the sentence '*Sītā rāmeśvaraṃ namaskaroti*'. (Sita salutes Ramesvara). But in the sentence '*Hanumān Rāmeśvraḥ asti*' (Hanuman is Ramesvara), the word '*Rāmeśvaraḥ*' is an instance of exo-centric compound (*Bahuvrīhiḥ*) with a paraphrase '*Rāmaḥ īśvaraḥ yasya saḥ*' (one whose God is Rama). One may argue that the constituency parser for compounds may be designed to produce all possible analyses and then the sentential parser may be used to filter out the correct parsed structure. However, the situation is not that simple. During the sentential analysis, one cannot always treat a compound word as a single unit that gets connected to other words in the sentence. But we need an access to its constituent comoponents as well. For example in the sentence '*Devadattasya gurukulaṃ prasiddhaṃ asti*' (Devadatta's teacher's school is famous), the word *Devadattasya* is connected to the constituent *guru* and not to the word *gurukulaṃ*. Such compounds termed as *asamartha*, are not rare in Sanskrit literature. The very first *śloka* from Kālidāsa's Meghadūtam has an expression *śāpena astaṅgamitamahimā* where the word *śāpena* is related to the first constituent *astaṅgamita* of the compound *astaṅgamitamahimā*.

* Initial version of this paper was presented as a keynote speech at LINGUINDIC conference at Oxford University on 12th June, 2025.

Thus the challange is how to develop a parser that can provide the constituency analysis of a compound and the dependency analysis of a sentence involving compounds where the dependency analysis may involve the compound constituents. In this paper, we discuss a solution to this challange. Next section compares the constituency and dependency structures and notes the differences between the two representation. While the hierarchy is perceivable in the constituency structure, in the dependency structure it is not perceivable. On the contrary, the dependency structure has a notion of head, and the constituency structure of a compound does not mark the head among the components. The dependency structure is more economical in terms of number of nodes and the edges. We introduce a dependency structure for a compound in the third section. We ensure that such a dependency structure is equivalent to the corresponding constituency structure, preserving the projectivity of the constituency structure. The head of the dependency structure is reinterpreted as the right component thereby making it easy to reconstruct the surface structure automatically from the dependency tree structure. In the fifth section, we present an algorithm to analyse a compound producing its dependency structure. Since for a human being it is natural to understand a compound from its constituency structure, we develop a user interface to show the constituency analysis when requested by the user.

2 Constituency and Dependency Structures

A compound is a syntactic construction that represents the information in a compact way as one word, which otherwise would have been represented as a phrase or as a clause with several words. For example, the compound *sumitrānandavardhanaḥ* (enhancer of the delight of Sumitrā) has a paraphrase *Sumitrāyāḥ ānandasya vardhanaḥ*. The constituency analysis of the compound *sumitrānandavardhanaḥ* is represented in Figure 1, where the leaf nodes represent the components and the non-terminal nodes represent the partial compounds with their compound types identified. The intermediate node is marked with the internal compound along with its type. Alternatively, one can also mark the intermediate nodes only with their types to avoid redundancy leading to a succint representation as in Figure 2. In that case the lexical content of the intermediate node can be constructed from the combination of the lexical content of its leaf nodes. The type of a compound does provide some syntactico-semantic information about the relation between the nodes.

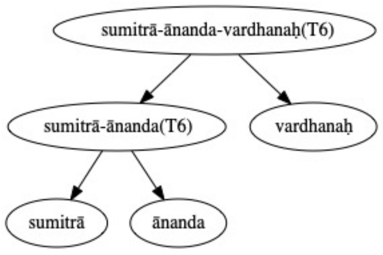

Figure 1: Elaborated Constituency structure

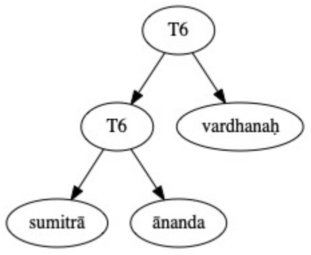

Figure 2: Succinct Constituency structure

In the case of Sanskrit compounds, since the compound formation is binary,[1] the constituency tree is a binary tree. Further, the order of the elements is the same as the order of the leaf nodes in the tree, and thus the tree structure is completely projective on the linear compound structure. A constituency tree marks two kind of information:

- the hierarchy of component groupings, and
- the intermediate nodes marking relations between the components at the leaf node.

A constituency tree with n terminal nodes has $n-1$ non-terminal nodes and $2n-2$ edges. The non-terminal nodes, except the root node, denote the partial intermediate compounds. The root node corresponds to the full compound. The $2n-2$ edges mark information about the grouping of the nodes and are unlabelled.

Let us compare the constituency analysis of a compound with the dependency analysis of its paraphrase. The paraphrase of the compound *Sumitrā-ānanda-vardhanaḥ* is *Sumitrāyāḥ ānandasya vardhanaḥ*. Its dependency analysis is shown in Figure 3. Here the relations are marked between the words, and not between the nominal stems/verbal roots. The relations are represented as directed labelled edges. They

[1]with an exception of *bahupada bahuvrīhi* (an exocentric compound involving more than two components) and *bahupada dvandva* (a co-ordinating compound involving more than two components).

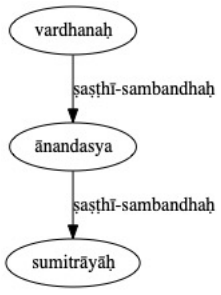

Figure 3: Dependency structure corersponding to the paraphrase

are named by the relation the word at the head node bears to that at the tail. In this example, *Sumitrāyāḥ* and *ānandasya* both have genitive case and are related to their immediately following words respectively. We do not decipher the semantics associated with the case suffix and mark the relation as *ṣaṣṭhīsambandhaḥ*. We note that the dependency tree has only three nodes each corresponding to a word in the paraphrase and two edges between them. In general a dependency tree has n nodes and $n-1$ edges if there are n words in a sentence.

Thus we see that the dependency trees are more economical. The differences between a constituency representation and a dependency representation are tabulated in Table 1.

Criterion	Constituency	Dependency
number of nodes	$2n-1$	n
number of edges	$2n-2$	$n-1$
Edges	do not mark any information	mark the relation between the nodes they connect.
Head	**not marked**	**marked**
Root Node	represents the underlying compound	corresponds to the head-word
Hierarchy	**directly perceivable**	**not perceivable**

Table 1: Constituency Versus Dependency for ‘n’ elements

3 Dependency representation for Compounds

The dependency structure being more economical, we would like to explore the possibility of representing a compound analysis as a dependency tree, so that the dependency parser can be used to analyse a sentence including the compound analysis. In the context of compounds, we foresee two problems in the dependency representation. The first one corresponds to an interpretation of the head node of each directed edge and the second one is regarding the representation of hierarchy in the dependency.

3.1 Head node interpretation

In traditional grammar the head of a compound is its semantic head. For example, in an endo-centric compound *rāja-puruśaḥ* (King’s person), *puruśa* is the head. In an *Avyayībhāva* compound such as *yathā-ucitam* (as appropriate), the initial component *yathā* is the head (see Figure 4). In the case of *Dvandva* (co-ordinative) compounds such as *rāma-lakṣamaṇau* both components are equally important and finally the exo-centric (*Bahuvrīhi*) compound such as *pītāmbaraḥ*, refers to an entity (a person wearing yellow cloth), different from the referents of its constituents *pīta* (yellow) and *ambara* (cloth). Thus it poses problems in interpreting the head to be the semantic one. We prefer a syntactic representation so that from the dependency representation, one can go back to the constituency and vice versa.

We decided not to confuse the head of the dependency edge as the semantic head of the compound. In the compounds the word order is very important. Hence we interpret the element at the head node as the right component, and the element at the tail node as the left component of the compound. Thus, for a compound ‘a-b’ of type ‘R1’, ‘b’ is the head and it is related to the tail ‘a’ by relation ‘R1’. For a compound ‘b-a’ of type ‘R2’, ‘a’ is the head, is related to the component ‘b’, by relation ‘R2’ (See Figure 5).

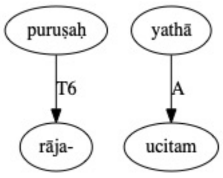

Figure 4: Dependency of a *Tatpuruṣa* and *Avyayībhāvaḥ* with *pradhāna* as compound-head

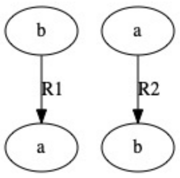

Figure 5: Direction of arrow matters

3.2 Representing Hierarchy

With the notion of head as the right component, now let us see what will be the dependency representation of a nested compound, and how to retrieve the hierarchy from that representation. Figures 6 and 7 show the equivalent dependency trees for the two possible constituency trees with three nodes. We notice here that while the dependency tree in Figure 6 can be unambiguously transformed into the corresponding constituency tree, in the case of dependency tree in Figure 7, the nesting is not obvious from the Dependency tree. As per our convention the 'head' or the parent node in the dependency tree is to the right of the child node. In Figure 7, the node 'c' is to the right of its children 'a' and 'b'. But the order between these two child nodes is not marked. Hence we cannot decide the order between the nodes 'a' and 'b'. Therefore we decided to decorate the nodes of the dependency tree with their indexing order. The modified figures are shown in Figures 8 and 9. Here from the indices associated with the nodes, it is clear that, in Figure 9 the distance between 'c' and 'a' is 2 while that between 'c' and 'b' is 1, and thus 'b' is nearer to 'c' forming an inner compound 'b-c' which is further compounded with 'a'.

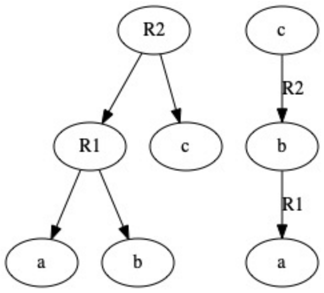

Figure 6: Equivalent Trees - type 1

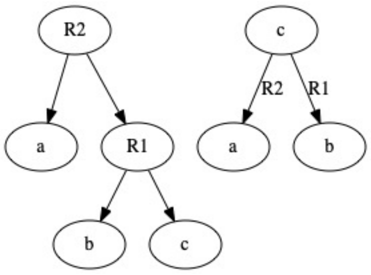

Figure 7: Equivalent Trees - type 2

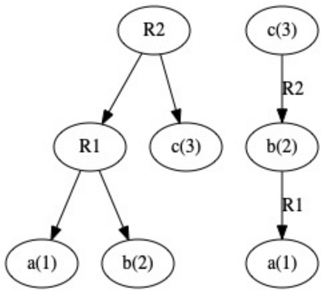

Figure 8: Type-1 with word-index

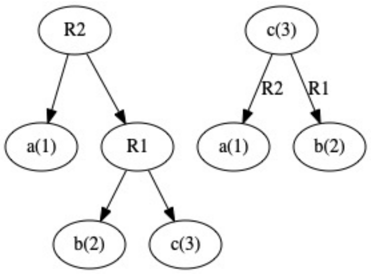

Figure 9: Type-2 with word-index

3.3 Ensuring projectivity

Now we need to ensure that this dependency representation is also projective honoring the word order of the components in a compound. That is we need to ensure that if there is an edge between any two components say a_i and a_{i+j}, then all the components from a_{i+1} to a_{i+j-1} should connect only to one of the components between a_i and a_{i+j}. Therefore we impose a condition for the dependency representation of a compound analysis that the projectivity condition be honored. With this condition, these dependency trees are plane trees, i.e. a connected acyclic graph, without edge-crossings.

Thus, we have proposed a dependency representation for the constituency structure of a compound where

- a dependency tree is a plane tree without edge-crossings,
- each vertex of the plane tree corresponds to a component of a compound,
- for every edge, the top node denotes the right component, and the bottom node represents the left component,
- the root node represents the right most component of a compound.

4 Equivalence between various representations

Compound analysis can be represented in three different ways.

- as a parenthesized expression,
- as a binary tree, and
- as a plane tree without edge crossings.

4.1 Parenthesized expression

The compound formation is binary and is also non-associative. That is, $(a - (b - c))$ is not the same as $((a - b) - c)$. Hence compound parsing may be viewed as parenthesizing a string. The number of ways in which a string of length $n + 1$ can be parenthesized is a Catalan number, C_n. For example, a compound a-b-c-d can be parenthesized in $C_3 = 5$ different ways as

1. $(((a - b) - c) - d)$
2. $((a - (b - c) - d)$
3. $(a - ((b - c) - d))$
4. $(a - (b - (c - d))$
5. $((a - b) - (c - d))$

4.2 Constituency structure: Binary Tree

Compound formation being binary, it may also be represented as a plane binary tree. A compound with $n + 1$ components is essentially a nested compound involving n binary compounds. Therefore the possible analyses can be represented as plane binary trees with $2n + 1$ vertices where $n + 1$ vertices are the terminal nodes representing the constituents and n non-terminal vertices represent the intermediate compounds. For example, the possible plane binary trees with 7 vertices where 4 of them are terminal nodes are shown in Figure 10.

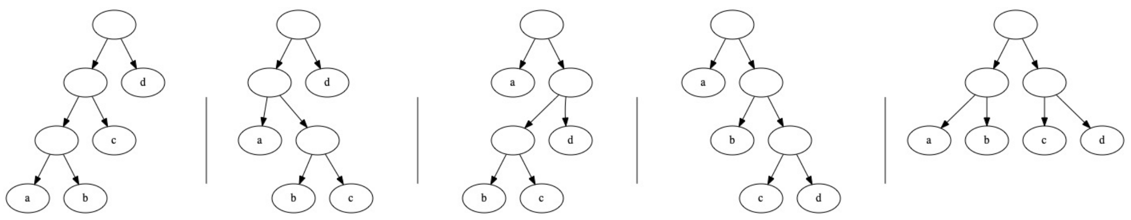

Figure 10: Binary trees with 4 constituents

It is obvious from the representation that the parenthesized expression and the constituency structures in §4.1 are just variants of the same representation in single and two dimensions respectively.

4.3 Dependency Structure: Plane trees without edge-crossings

The third representation of the compound structure is the proposed dependency representation with a plane tree drawn without edge crossings. It has been shown in Stanley (1999, p176) that the total number of plane trees with $n+1$ vertices is also a Catalan number, C_n. For example, all possible plane trees with four vertices are shown in Figure 11. We append the node label with the position of the component, since we need it to decide the closest component while transforming it to a binary constituency tree.

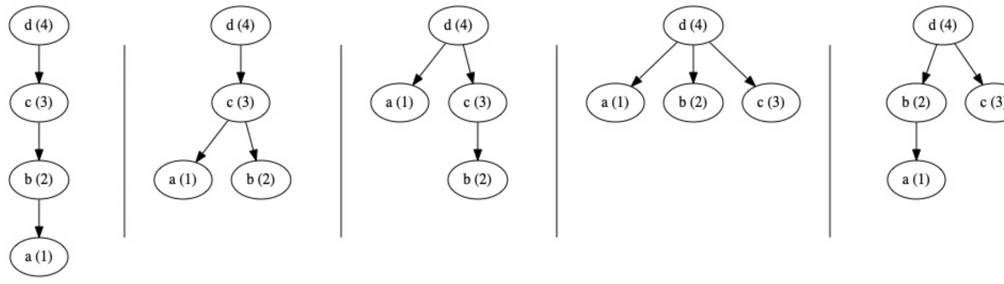

Figure 11: Plane trees with 4 constituents

The proposed dependency representation, which is a plane tree without edge-crossings, is also a variant of the binary constituency structure. Following transformation converts a binary constituency tree to its corresponding plane tree. See Figure 12.

- Label the head node of a binary tree with its right node, and merge the two nodes into one.

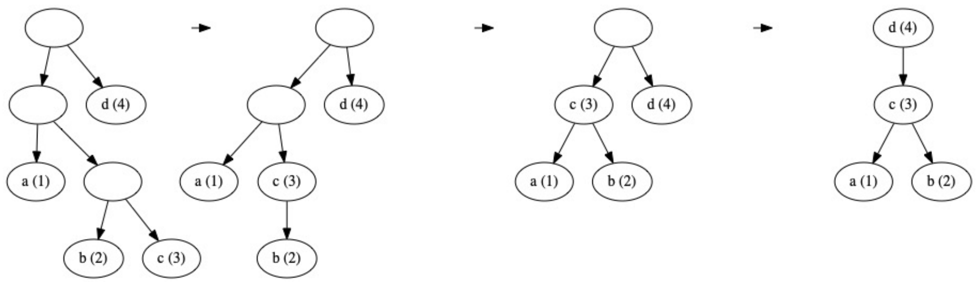

Figure 12: Constituency to Dependency: An illustration

To get the binary constituency tree corresponding to a plane tree without edge-crossings

- Start form the bottom node.
- If there are more than one children, start with the closest child node. Introduce an empty head node connecting all other branches to it, except the closest child branch. Join the head node with the closest child to this empty node as a right branch. Repeat this till all every node is binary branching.
- For each non-empty head node, introduce a right node with the same label and join it with the head node. Remove the label of the head node.

Figure 13 illustrates the steps.

Thus we see that the constituency representation and the dependency representation faithfully represent the compound analysis.

5 Extending Dependency Parsing to Compounds

The Saṁsādhanī platform provides a dependeny parser which is based on the principles of Indian theories of śābdabodha (Kulkarni, 2019; Kulkarni, 2021). In this parser a compound word was treated as a single unit. But now we split the compound into its constituents and establish relations between them. We first discuss the sources of various factors necessary to get the *śābdabodha* of a compound followed by the algorithm.

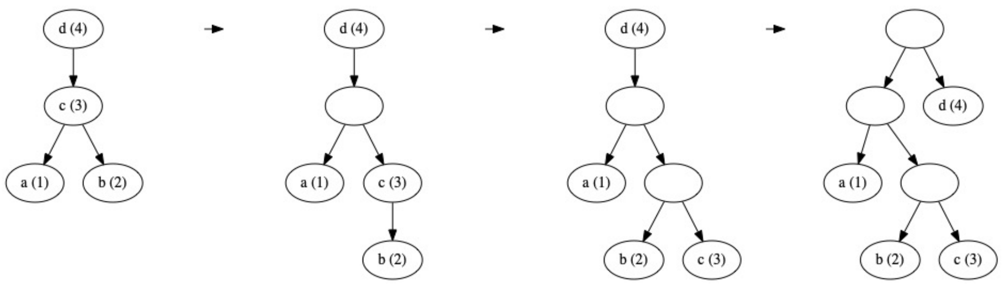

Figure 13: Dependency to Constituency: An illustration

5.1 Factors governing *śābdabodha* of compounds

The same three factors viz. *ākāṅkṣā*, *yogyatā* and *sannidhi* which are useful to establish the relations between the words in a sentence, are also useful for establishing relations between the components of a compound. The compound formation, according to Pāṇini, is governed by the meta rule *samarthaḥ padavidhiḥ* (P 2.2.1), that is 'an operation involving finished words conveys the same meaning as that of its paraphrase'.[2] Alternately, *samartha* is also interpreted as having a meaning which is connected (*sambaddhārtha*) or fused (*saṁsṛṣṭārtha*).

The clues for *ākāṅkṣā* and *yogyatā* for the analysis of compounds can be found through various *sūtras* that act as the governing *sūtras* (*samāsa-vidhāyaka-sūtras*) for compound formation. For example in the *sūtra dvitīyāṣritātītapatitagatātyastaprāptāpannaiḥ* (P 2.1.24)[3] Pāṇini provides a list of words that can be compounded with another word which is in the second case termination. This list serves as a clue that acts as a source of the expectancy for another word. Further the *sūtras* such as *prathamā nirdiṣṭaṁ samāsa upasarjanam (P1.2.43)* and *upasarjanaṁ pūrvam (P2.2.30)* provide us a clue regarding the position of these words in a compound. Thus we know which words have an expectancy and which words have a potential to satisfy these expectancies. Pāṇini uses various linguistic and extra-linguistic features such as morphological, lexical, ontological, taxonomical and semantic to state the conditions under which a compound can be formed. These have been listed in detail in Kulkarni and Kumar (2013), Satuluri (2016) and Krishnan et al. (2025).

Apart from these some of the indeclinables also provide a clue as to with which component they get connnected to. For example, a negative particle such as *a* or *an* in the case of *nañ Tatpuruṣa* always gets connected with the following component, and not the previous component. Thus the compound *śubhāśubhau* can be split only as *(śubha-(a-śubhau))* and not as *((śubha-a-)śubhau)*. Similarly the word *yathā* gets connected only with the following words whereas *iva* gets connected to the previous one.

Sannidhi puts another constraint in the compound formation. This relates to the proper nesting. For example, one cannot have a compound a-b-c-d with inner compounds a-c and b-d. Thus as in the case of sentential parsing, in compound-parsing as well, the three factors *ākāṅkṣā*, *yogyatā* and *sannidhi* play an important role.

5.2 Algorithm

The four steps involved in compound analysis are segmentation, constituency analysis, type identification and paraphrasing. In what follows we assume that the input is segmented and the compound components are seperated with '-' so as to distinguish them from the segmented words with space in between.

The advantage of this algorithm is that we carry out both the constituency analysis and the type identification together. Constituency analysis corresponds to extracting a tree from the graph, and the type identification corresponds to establishing labelled edges between the components. For deciding the constituency we use the constraint of *sannidhi* and for deciding the type of a compound, Pāṇini's governing rules for compound analysis are used, which provide the clues for *ākāṅkṣā* and *yogyatā*.

Pāṇini classifies the compounds into four broad classes. We further sub-divide them into around fifty-four types (see Appendix 6) as discussed in Anilkumar (2012)[4] so that their correct paraphrase can be generated. Some of these types express associated semantics, while others are more morpho-syntactic

[2]The translations are taken from Joshi (1968).

[3]'a word in the second case termination (is preferably compounded) with (the case inflected words) *ṣrita* etc. conveying the same meaning as the formally corresponding word group and is called *tatpuruṣa*'. Joshi (1968)

[4]guidelines for compound tagging are available at `https://sanskrit.uohyd.ac.in/scl/GOLD_DATA/Tagging_guidelines//samaasa_tagging16mar12-modified.pdf`

in nature. For example, a compound of type *viśeṣaṇa-pūrvapada-karmadhāraya* provides a semantic information that the first component is a modifier of the second one. The type *dvitīyā-tatpuruṣa* provides us a hint that the missing case suffix of the iic is accusative. But what role the accusative case refers to with respect to the two words involved is not made explicit.

The algorithm for processing the compounds is similar to the one for sentential parser (Kulkarni, 2019; Kulkarni, 2021).

1. Defining a node
   We define one node each corresponding to each morphological analysis for every component in a compound.

2. Establishing directed edges
   We use clues from all the governing rules for compound formation from the Aṣṭādhyāyī to establish directed edges between the components. Head of all these directed edges is assumed to be to the right.

   In the absence of any such clues, the initial graph will have all possible edges between all the morphological analyses of all the components of a compound with a restriction that the head is always to the right. Thus, with n components $n(n-1)$ directed edges are possible. Further they can be labelled in as many possible ways as there are compound types. With the constraint that an edge between two nodes is established only if there is an expectancy and semantic compatibility, the number of edges reduces drastically.

3. Defining constraints
   We have already used one constraint of right headed-ness while establishing the edges. Other constraint is that of *sannidhi*, which states that the words which have expectancies should be close to each other. Or in other words there cannot be an intervention of unrelated components between related components. This constraint, as is discussed in **?**; Kulkarni (2021) reduces to a well-nestedness criterion.

4. Solving the constraints
   We extract all possible trees from this graph that satisfy the constraints, and produce all possible solutions.

5. Prioritizing the solutions
   Weights are associated with the various compound types that prioritise the solutions in case of multiple possible solutions. The same formula as defined for sentential parser is used here for prioritisation.

Thus we see that the steps involved in the processing of compounds are the same as that for a sentence. Only one exception is there. The word order in a sentence is flexible as against in compounds. Further, the *utthāpya ākāṅkṣā* in the case of a sentence permits a violation of well-nestedness constraint. However, in the case of compounds, we do not entertain the violation of well-nestedness.

5.3 Integrating compound and sentence analysis

Since these steps are exactly in line of those in the dependency parsing, compound analysis becomes an integral part of a sentence analyser. The compound words relate to other words in a sentence through their final components.

We list two major advantages of integrating the compound analysis with the sentential analysis.

- Handling *Tatpuruṣa* and *Bahuvrīhi* ambiguity
  An isolated compound such as *pītāmbaram* is ambiguous between a *Bahuvrīhiḥ* and *Tatpuruṣaḥ*. But when a proper context is provided then the integrated parser can analyse the compound correctly as shown in Figure 14 which corresponds to the sentence *pīta-ambaram viṣṇuṁ paṣya.* But in the absence of the word *viṣṇuṁ*, the sentence would be ambiguous between two possible analyses of the compound as shown in Figures 15 and 16.

  You will notice here that the compound analysis is shown as a constituency analysis, though it is analysed using the dependency structure. The interactive user interface allows one to hide the constituency structure and open it only when necessary.

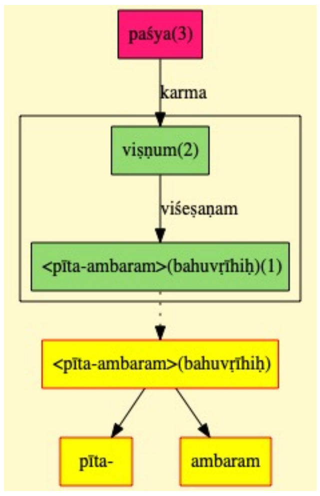

Figure 14: Ambiguity resolved

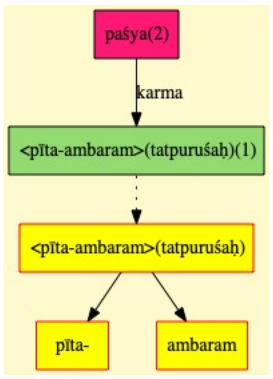

Figure 15: Tatpuruṣaḥ

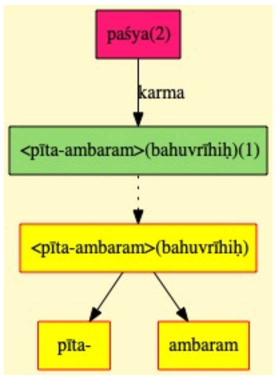

Figure 16: Bahuvrīhiḥ

- Handling *asamartha* compounds
  The integration also provides us with a chance to handle the *asamartha* compounds where the iic or the sub-ordinate component of a compound is related to some other word in a sentence, as in *Devadattasya gurukulam gaccāmi* (see Figure 17). Here the word *Devadattasya* is related to *guru* and not *gurukulam*. It is possible to handle such compounds by allowing the first component of a compound to connect with any other word in a sentence provided this component is *sāpekṣa* (with an expectancy) such as the relational terms (*guru*, *pitā*, *putra*, etc.) or a *kṛdanta* (non-finite verbal form) as in *śāpena astaṅgamitamahimā* (deprived greatness due to the curse), where *śāpena* (due to the curse) is the *hetuḥ* (cause/reason) for *astaṅgamita* (deprived greatness). Figure 18 shows the analysis of the first verse from the Meghadūtam.

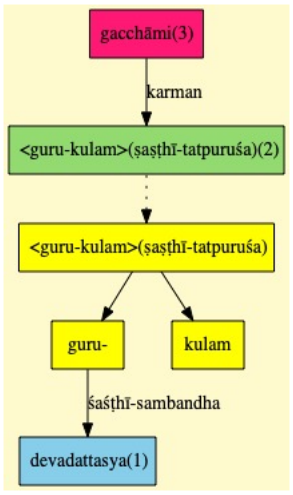
gacchāmi(3)
karman
<guru-kulam>(ṣaṣṭhī-tatpuruśa)(2)
<guru-kulam>(ṣaṣṭhī-tatpuruśa)
guru-
kulam
śaṣṭhī-sambandha
devadattasya(1)

Figure 17: Asamartha - example 1

5.4 Tasks ahead

The major task is to model *yogyatā* by understanding the various linguistic and extra-linguistic features that help in establishing relations between various words at the level of compounding as well as at a sentential level. Another important task is to interlink the Heritage segmenter[5] with the parser in such a way that without any user-intervention machine takes the unsegmented verse / sentence and produces all possible parsed structures as an output, prioritised on some scale.

[5]`https://sanskrit.inria.fr`

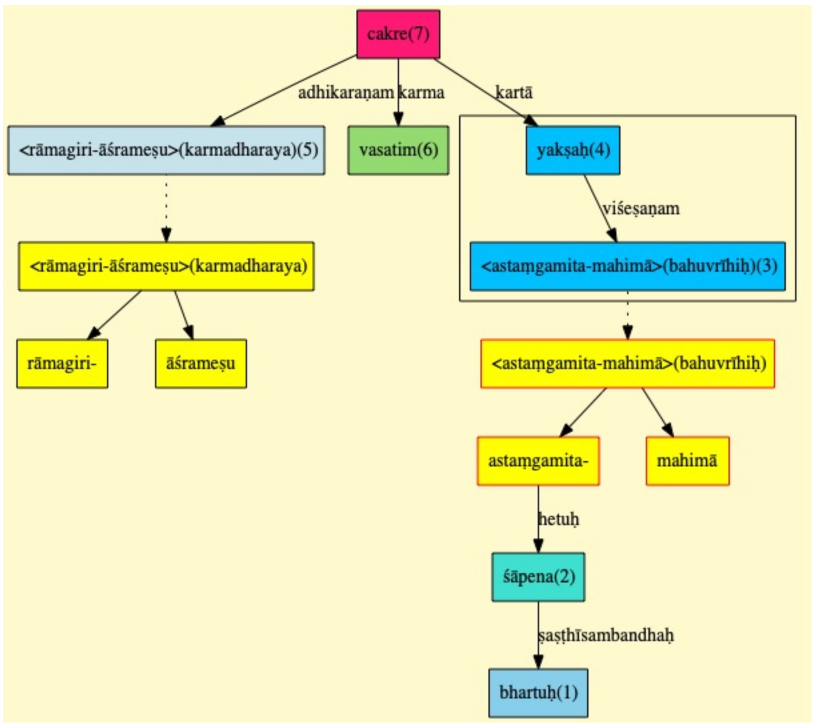
cakre(7)
adhikaraṇam
karma
kartā
<rāmagiri-āśrameṣu>(karmadharaya)(5)
vasatim(6)
yakṣaḥ(4)
viśeṣaṇam
<astaṃgamita-mahimā>(bahuvrīhiḥ)(3)
<rāmagiri-āśrameṣu>(karmadharaya)
rāmagiri-
āśrameṣu
<astaṃgamita-mahimā>(bahuvrīhiḥ)
astaṃgamita-
mahimā
hetuḥ
śāpena(2)
ṣaṣṭhīsambandhaḥ
bhartuḥ(1)

Figure 18: Asamartha - example 1

6 Conclusion

In this paper we have argued that compounds can be analysed following the dependency framework effectively. Equivalent dependency representation for the constituency analysis is proposed where the word order between the components in constituency structure is reflected into the headed-ness. This is followed by the extension of the current sentential parser to handle compound analysis. Finally we illustrate its effectiveness in resolving the ambiguities in compound type using the contextual words in the sentence, and also demonstrate how this integration can handle *asamartha* compounds. Though a compound is analysed using the dependency framework, it being natural to understand a compound structure as a constituency tree, we display it as a constituency tree. This is possible due to the equivalence between the two structures. We also develop a user interface that facilitates user to hide the constituency tree if not needed.

References

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

V. V. Bhandare. 1995. Structural and semantic aspects of the dvandva compound. *Annals of the Bhandarkar Oriental Research Institute*, 76(1/4):89–96.

Brendan S. Gillon. 1993. Bhartṛhari's solution to the problem of asamartha compounds. *Études Asiatiques/Asiatiche Studien*, 47(1):117–133.

Brendan S. Gillon. 2007. Exocentric (bahuvrīhi) compounds in classical Sanskrit. In Gérard Huet and Amba Kulkarni, editors, *Proceedings, First International Symposium on Sanskrit Computational Linguistics*, pages 1–12.

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.

H. H. Hock, editor. 1991. *Studies in Sanskrit Syntax*. Motilal Banarsidass, Delhi.

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.

S D Joshi. 1968. *Patañjali's Vyākaraṇa Mahābhāṣya Samarthāhnika (P 2.1.1) Edited with Translation and Explanatory Notes.* Center of Advanced Study in Sanskrit, Pune.

Amrith Krishna, Pavankumar Satuluri, Shubham Sharma, Apurv Kumar, and Pawan Goyal. 2016. Compound type identification in Sanskrit: What roles do the corpus and grammar play? In Dekai Wu and Pushpak Bhattacharyya, editors, *Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WSSANLP2016)*, pages 1–10, Osaka, Japan, December. The COLING 2016 Organizing Committee.

Amrith Krishna, Ashim Gupta, Deepak Garasangi, Jivnesh Sandhan, Pavankumar Satuluri, and Pawan Goyal. 2020. Neural approaches for data driven dependency parsing in Sanskrit. In *Proceedings of the Computational Sanskrit & Digital Humanities: Selected papers presented at the 18th World Sanskrit Conference.* Association for Computational Linguistics.

Sriram Krishnan, Pavankumar Satuluri, Amruta Barbadikar, Prasanna Venkatesh, and Amba Kulkarni. 2025. Compound type identification in sanskrit. In Amba Kulkarni and Oliver Hellwig, editors, *Proceedings of 'Computational Sanskrit and Digital Humanities' section of 19th World Sanskrit Conference.* ACL Anthology.

Amba Kulkarni and Anil Kumar. 2011. Statistical constituency parser for Sanskrit compounds. In *Proceedings of ICON 2011.* Macmillan Advanced Research Series, Macmillan Publishers India Ltd.

Amba Kulkarni and Anil Kumar. 2013. Clues from Aṣṭādhyāyī for compound type identification. In Malhar Kulkarni, editor, *Proceedings of the 5th International Sanskrit Computational Linguistics Symposium.* D. K. Printworld.

Amba Kulkarni, Sheetal Pokar, and Devanand Shukl. 2010. Designing a Constraint Based Parser for Sanskrit. In G N Jha, editor, *Fourth International Sanskrit Computational Linguistics Symposium*, pages 70–90. Springer-Verlag, LNAI 6465.

Amba P Kulkarni, Preeti Shukla, Pavankumar Satuluri, and Devanand Shukl. 2015. How free is the 'free' word order in Sanskrit. In Peter Scharf, editor, *Sanskrit syntax*, pages 269–304. Sanskrit Library.

Amba Kulkarni, Sanal Vikram, and K Sriram. 2019. Dependency parser for sanskrit verses. In *Proceedings of the 6th International Sanskrit Computational Linguistics Symposium*, pages 14–27.

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.

Amba Kulkarni. 2019. *Sanskrit Parsing based on the theories of Śābdabodha.* IIAS, Shimla and D K Printworld.

Amba Kulkarni. 2021. Sanskrit Parsing Following Indian Theories of Verbal Cognition. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 20(2), apr.

Anil Kumar, V Sheebasudheer, and Amba Kulkarni. 2009. Sanskrit compound paraphrase generator. *Proceedings of ICON.*

Anil Kumar, Vipul Mittal, and Amba Kulkarni. 2010. Sanskrit compound processor. In Girish Nath Jha, editor, *Sanskrit Computational Linguistics*, pages 57–69, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bhagyalata Pataskar. 1996. Some observations about the compound structure of *Aṣṭādhyāyī*. *Annals of the Bhandarkar Oriental Research Institute*, 77(1/4):121–131.

B Premjith, Chandni Chandran, Shriganesh Bhat, Soman Kp, and P Prabaharan. 2019. A machine learning approach for identifying compound words from a sanskrit text. In *Proceedings of the 6th International Sanskrit Computational Linguistics Symposium*, pages 45–51. Association for Computational Linguistics.

Jivnesh Sandhan, Amrith Krishna, Pawan Goyal, and Laxmidhar Behera. 2019. Revisiting the role of feature engineering for compound type identification in Sanskrit. In Pawan Goyal, editor, *Proceedings of the 6th International Sanskrit Computational Linguistics Symposium*, pages 28–44, IIT Kharagpur, India, October. Association for Computational Linguistics.

Jivnesh Sandhan, Amrith Krishna, Ashim Gupta, Laxmidhar Behera, and Pawan Goyal. 2021. A little pretraining goes a long way: A case study on dependency parsing task for low-resource morphologically rich languages. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop.* Association for Computational Linguistics.

Jivnesh Sandhan, Ashish Gupta, Hrishikesh Terdalkar, Tushar Sandhan, Suvendu Samanta, Laxmidhar Behera, and Pawan Goyal. 2022. A novel multi-task learning approach for context-sensitive compound type identification in Sanskrit. In Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na, editors, *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4071–4083, Gyeongju, Republic of Korea, October. International Committee on Computational Linguistics.

Jivnesh Sandhan, Yaswanth Narsupalli, Sreevatsa Muppirala, Sriram Krishnan, Pavankumar Satuluri, Amba Kulkarni, and Pawan Goyal. 2023. DepNeCTI: Dependency-based nested compound type identification for Sanskrit. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13679–13692, Singapore, December. Association for Computational Linguistics.

Pavankumar Satuluri and Amba Kulkarni. 2013. Generation of Sanskrit compounds. *Proceedings of ICON*, pages 77–86.

Pavankumar Satuluri. 2016. *Sanskrit Compound Generation: With a Focus on the Order of Operations.* Ph.D. thesis, University of Hyderabad Hyderabad.

Richard Stanley. 1999. *Enumerative Combinatorics, volume 2.* Cambridge University Press.

A Compound Types: Fine-grain Classification

compound type:	**avyayī- -bhāvaḥ**	**compound type:**	**karma- -dhārayaḥ**
avyaya-pūrvapadaḥ	A1	viśeṣaṇa-pūrvapada-karmadhārayaḥ	K1
avyaya-uttarapadaḥ	A2	viśeṣaṇa-uttarapada-karmadhārayaḥ	K2
tiṣṭhadguprabhṛti	A3	viśeṣaṇa- ubhayapada-karmadhārayaḥ	K3
saṃkhyāpūrvapada-nadyuttarapadaḥ	A4	upamāna-pūrvapada-karmadhārayaḥ	K4
nadyuttarapada-anyapadārthe saṃjñāyām	A5	upamāna-uttarapada-karmadhārayaḥ	K5
saṃkhyāpūrvapada-vaṃśyottarapadaḥ	A6	avadhāraṇāpūrvapada-karmadhārayaḥ	K6
pāre-madhye-pūrvapada ṣaṣṭhyuttarapadaḥ	A7	sambhāvanāpūrvapada-karmadhārayaḥ	K7
		madhyamapadalopī-karmadhārayaḥ	Km

compound type:	**tatpuruṣaḥ**	**compound type:**	**bahuvrīhiḥ**
prathamātatpuruṣaḥ	T1		
dvitīyātatpuruṣaḥ	T2	dvitīyārtha-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bs2
tṛtīyātatpuruṣaḥ	T3	tṛtīyārtha-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bs3
caturthītatpuruṣaḥ	T4	caturthyartha-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bs4
pañcamītatpuruṣaḥ	T5	pañcamyartha-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bs5
ṣaṣṭhītatpuruṣaḥ	T6	ṣaṣṭhyartha-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bs6
saptamītatpuruṣaḥ	T7	saptamyartha-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bs7
nañtatpuruṣaḥ	Tn	digvācaka-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bsd
samāhāra-dviguḥ	Tds	praharaṇaviṣayaka-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bsp
taddhitārthadviguḥ	Tdt	grahaṇaviṣayaka-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bsg
uttarapadadviguḥ	Tdu	astyartha-madhyamapadalopī(nañ)-bahuvrīhiḥ	Bsmn
gatisamāsaḥ	Tg	prādi-bahuvrīhiḥ	Bvp
kusamāsaḥ	Tk	saṃkhyobhayapada-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bss
prādisamāsaḥ	Tp	upamānapūrvapada-bahuvrīhiḥ(samānādhikaraṇaḥ)	Bsu
mayūravyaṃsakādiḥ	Tm	vyadhikaraṇa-bahuvrīhiḥ	Bv
tatpuruṣaḥ bahupadaḥ	Tb	saṃkhyottarapadaḥ vyadhikaraṇa-bahuvrīhiḥ	Bvs
tatpuruṣaḥ upapadaḥ	U	sahapūrvapada-vyadhikaraṇa-bahuvrīhiḥ	BvS
		upamānapūrvapada-vyadhikaraṇa-bahuvrīhiḥ	BvU
		bahupada-bahuvrīhiḥ	Bb

compound type:	**dvandvaḥ**
itaretarayoga-dvandvaḥ	Di
samāhāra-dvandvaḥ	Ds
ekaśeṣaḥ	E

compound type:	**kevala**
kevalasamāsaḥ	S

compound type:	**dviruktiḥ**
dviruktiḥ	d