Vaiśeṣika Graph Grammar (VGG) System

Rajesh TAVVA[a,1] and Navjyoti SINGH[a]

[a] *Center for Exact Humanities, International Institute of Information Technology, Hyderabad, Telangana, India*

Abstract. Graph grammars, being a natural extension of string grammars as well as tree grammars, are highly expressive as well as powerful enough to capture the generative structure of a wide variety of scenarios - both simple and complex. But, unfortunately, we do not find many instances which leverage this power graph grammars provide us. Most of the graph grammars available so far are either toy grammars or limited to addressing highly specialized problems. In this paper, we present Vaiśeṣika Graph Grammar (VGG) system, which is as wide as any graph grammar can get i.e., a grammar for reality as theorized in Vaiśeṣika - the foundational ontology found in Indian analytic tradition. This paper extends the work presented in [1]. In [1], the generative as well as the interpretation rules of this system were presented. Here, apart from briefly discussing some of these rules, we also present a parser for such a system which makes it extremely powerful in terms of its ability to classify an input graph as a valid Vaiśeṣika graph or an invalid one which, in itself, is an immensely significant breakthrough for any ontological application. Apart from that, we also present a verification mechanism to verify the interpretations of graphs generated by the generative rules. This is done, at least statistically, if not formally, to give a statistical proof of the soundness of the system (that every generated graph has at least one Vaiśeṣika interpretation, and no useless graphs are generated.)

Keywords: Vaiśeṣika, Graph Grammar, Generative Ontology, Parser

1. Introduction

The notion of a grammar in itself is quite intriguing. It is a finite number of rules which can generate a (potentially) infinite number of sentences. It is something which brings parsimony into the system. A grammar captures the repeating patterns in a system and encodes them into a finite set of rules. It is extremely simple and yet highly powerful.

Graph grammars, compared to string and tree grammars, are more powerful, expressive and intuitive. Yet, there are very few instances of uses of graph grammars in real world. In this paper, we present the parser for Vaiśeṣika Graph Grammar (VGG) system[2] whose generation rules (GRs) and interpretation rules (IRs) were already presented in [1]. This is a grammar for reality itself based on Vaiśeṣika - the foundational ontology found in Indian analytic tradition.

[1]Corresponding author, Center for Exact Humanities, IIIT-Hyderabad, Gachibowli, Hyderabad, Telangana, India; E-mail: vrktavva@research.iiit.ac.in

[2] One can find a working prototype of this system here: http://ceh.iiit.ac.in/vgg

In ontology engineering, a fairly recent discipline, one of the fundamental questions is how to construct ontologies as well as how to analyze a given scenario or state-of-affairs (SOAs) in terms of known ontological categories. Currently these problems are addressed mostly manually or, even if done automatically, involve a lot of human intervention at various stages. Hence there are various problems like interoperability issues, internal contradictions within the systems etc. One way to overcome these problems is to minimise human intervention as much as possible, and what better way can be there to do it than to construct grammars for such systems where the human intervention is minimal, and is required only at the beginning of building the grammar. Once the generation or parsing process is set in motion, the system is on its own and requires no human intervention in between.

We presented one such system, a novel idea called Generative Ontology, based on Vaiśeṣika in [1] in which the GRs and IRs of the system are presented. But they only enable one to generate graph structures as well as interpret them according to Vaiśeṣika system. There is still one crucial step left to complete the system i.e., to give it an ability to decide whether any given graph is derivable within the system or not. This eliminates the burden of generating every possible graph and matching it with the input graph to decide upon its derivability. It is similar to the role which a parser plays in compilers in programming languages. It parses a program in a particular language and identifies syntax errors in it saving a lot of resources to the system. In that sense, a parser is an immensely useful and necessary tool and it gives enormous power to the existing system.

In this paper, we present the parser for VGG as well as give a statistical proof for the soundness of VGG. Since this is an extension of the work done in [1], we will provide a brief overview of it in Section 2, and then move on to present the parser in Section 3, and then present the results and conclusion in Section 4.

2. Overview

2.1. Vaiśeṣika System

Since a significant portion of the system (Vaiśeṣika categories, Generative Rules and Interpretation Rules) are formally presented in [1] already, we give a brief overview of them here in a more informal manner for the comprehension of the rest of the paper.

Vaiśeṣika system is a foundational ontology which classifies all the entities in reality into 6 categories[3]: (1) Substance (e.g: material entities like ball, car as well as non-material entities like soul, space and time) (2) Quality (e.g: color, size) (3) Action (e.g: rising up, falling down, motion) (4) Universal (e.g: car-ness, ball-ness, red-ness) (5) Ultimate Differentiator (located in each ultimate substance (explained below) and differentiates one from the other) and (6) Inherence (explained below).

Before comprehending any of the Vaiśeṣika categories, the first category we need to understand is ‘inherence’. Inherence is one of the primary relations of Vaiśeṣika

[3] It is actually a six-plus-one category system. The seventh category i.e., absence is considered to be added later to the list of fundamental categories. The complete list of Vaiśeṣika categories and subcategories can be found in [13].

system. It is the second most prevalent relation in reality after self-linking relation[4]. Some instances of inhering can be color inhering in a rose, treeness inhering in all the trees, a whole inhering in its parts etc., (inherence needs to be understood as something located in something, but in an aspatial manner).

Substance, philosophically speaking, is that *which stands below* i.e., all other categories ride on it or are dependent on it while this doesn't ride on anything else. Similarly, an ultimate substance, in Vaiśeṣika, is that category which *stands below* everything else - even other substances. It needs to be understood in terms of inherence - as something inhering in something. In Vaiśeṣika, qualities and actions inhere in substances; universals inhere in qualities, actions and substances; while some substances (wholes) inhere in other substances (parts). These latter substances in which other things can inhere, but they don't inhere anywhere else, are called ultimate substances (USs).

Among USs, there are two types: ubiquitous ultimate substances (UUSs) and mobile ultimate substance (MUSs). UUSs are those which are either in contact or in disjunct with all other substances. MUSs are those which are in contact with some while in disjunct with other substances. These are formal definitions of UUS and MUS provided in [1]. Not only these, but all the fundamental categories as well as some subcategories of Vaiśeṣika are defined formally, in terms of inherence, in [1]. For instance, *universal (U)* is something which inheres in 2 or more entities, but nothing inheres in it. In this way, many categories and subcategories of Vaiśeṣika are defined formally, purely using the idea of inherence, conjunct (contact) and disjunct - i.e., how many entities inhere in a given entity, what is the configuration of contacts, disjuncts etc.[5]

The Interpretation Rules (IRs) of VGG can be considered to be a visualization of the formal definitions of Vaiśeṣika categories and subcategories. So a quick look at them can enable us to understand these definitions easily. They are presented in Section 3.2 of [1].

Like in [1], this paper's focus is not on defending Vaiśeṣika description of reality or the rationale behind its categorial system. One can refer to [5] and [14] for that. The idea of this paper is to show the possibility of an ontological system in which the SOAs can be generated, interpreted and also parsed.

2.2. Generation and Interpretation Rules of Vaisesika

As mentioned in Section 2.1, many Vaiśeṣika categories and subcategories can be defined formally in terms of inherence, contact and disjunct. Actually these also form the fundamental relations in the system apart from the self-linking relation. Their formal definitions can be found in Section 2 of [1].

A punctuator is a boundary which separates as well as brings two entities into some relationship. Its structure is like <x|y,R> where x and y are the entities being punctuated, and *R* is the relational context in which they are punctuated. Here *R*

[4] Self-linking relation is not a Vaiśeṣika category since it is not an entity. In other words, it requires no other relational entity to bring together the two relata. It is both the relata themselves put together. For the structure of self-linking relation, see [1].

[5] Refer to Section 2 of [1] for all the formal definitions of Vaiśeṣika categories and subcategories.

constitutes the entire rest of the universe which itself is a set of all the chains of entities and punctuators between x and y. So a punctuator has a recursive structure.

Given the recursive structure of punctuator and the complex nature of its relational context as a web of entities and punctuators, it is best imagined as a graph (made up of nodes and edges) which can also be understood as a state-of-affair (SOA). It is shown in [1] that the three simple punctuators - (1) self-linking (2) inherence and (3) conjunct-disjunct punctuators - constitute the building blocks for all the complex punctuators (SOAs) in Vaiśeṣika. It is also shown how these complex punctuators can be generated from the simple punctuators (alphabet) using generative rules (GRs), and then interpreted with Vaiśeṣika categories using interpretation rules (IRs).

GRs can be considered as the syntactic portion of reality whereas IRs can be considered the semantic portion of it. GRs generate pure structures with no inherent meaning whereas IRs give meaning to them by labeling them with Vaiśeṣika categories. In [1], it is shown that all the graphs generated by GRs could be interpreted using IRs with Vaiśeṣika categories, but the validity of those interpretations is provided here, in this paper, in Section 4. VGG uses Single Pushout (SPO) approach [8] for graph transformations.

A screenshot of an interpreted graph in VGG system is provided in Figure 1.

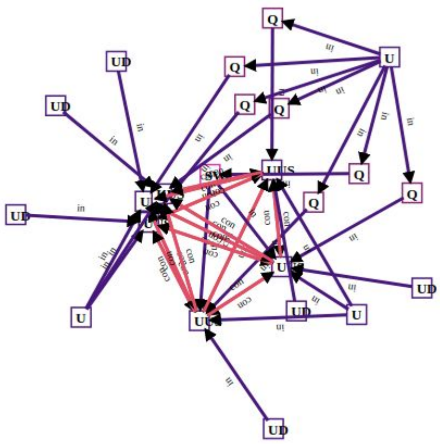


Figure 1. A graph generated in VGG followed by interpretation with Vaiśeṣika categories. Here the following categories are depicted - UD (Ultimate Differentiator), Q (Quality), SW (Substantial Whole), UUS (Ubiquitous Ultimate Substance), U (Universal). And the edges 'in' and 'con' stand for inherence and contact respectively.

In this paper, we present the parser for VGG which completes the system and gives it real power in terms of recognizing an input graph to be derivable or non-derivable within the system using the GRs of the system.

3. Parser

While building a system like VGG, one needs to distinguish two questions here:

- Whether a graph is derivable within the system or not?
- Whether all the graphs derivable within the system are Vaiśeṣika graphs or not?

Both the questions are independent of each other and hence require different approaches to address them. The first one is purely a structural or a syntactic question whereas the second one is a semantic one. The first one is a question of decidability whereas the second one is a question of soundness. The first one is answered by constructing a parser which parses the structure of a given graph to decide if it belongs to the language (the set of graphs generated) of the system. The second one is answered by verifying whether all the generated graphs (using the generative rules of VGG) are interpretable (using the interpretation rules of VGG) i.e., is there a correlation between the generation rules and the interpretation rules. This can be done by verifying if there is a consistency in the generative labels (intermediate labels used in generative grammars) and the interpretation labels (the Vaiśeṣika labels used to categorize nodes after interpretation). If there is a consistency, then the insight used while generating the graphs matches with the insight while interpreting them though both are starting from completely different directions. We show that, at least statistically, the system has soundness since there is a consistent mapping between the generative labels and the interpretation labels.

The parser, like generative as well as interpretation grammars, is a grammar, and hence consists of a set of graph transformation rules which apply on the input graph and stop after all the rules are exhausted. The parser, as mentioned above, addresses a decidability problem and hence is supposed to recognize a graph as belonging to the system or not based purely on its structure and not its labels. The rules of the parser are arrived at, broadly, by reversing the generative rules of VGG as well as the LHS and RHS of each rule, though not mechanically. One has to add/delete some NACs[6] in the generative grammar to formulate the parser rules. An input graph to a parser is said to belong to the generative grammar of VGG if the process of parsing ends at the start graph of the generative grammar of VGG. If not, the input graph to the parser is a non-derivable graph i.e., such a structure will never be generated by the generative grammar of VGG.

We claim that the parser identifies only the right graphs with the right combination of labels as valid graphs generated by the generative rules of VGG. It would reject both (1) graphs with wrong structures as well as (2) graphs with right structures but wrong combination of labels - as non-derivable graphs.

In our grammar, the node-labels are Ω_V = {g, C, D, h, i, p, q, r, s, u, v, e} and edge-labels are Ω_E = {in, con, dis}. Each of the node-labels stands for the following: g – start node (this is the only node in the start graph of generative grammar of VGG, but

[6] An NAC is a Negative Application Condition which is also a graph. If an NAC matches with the graph in question along with its LHS, then the rule cannot be applied. So one should check for this condition as well before applying a rule.

this should become the end graph in the parser if the parsing is done successfully), C – conjunct entity, D – disjunct entity, h, i, p, q, r, s, u, v – are all various node labels used in the process of parsing. These are taken from generation grammar itself and their corresponding Vaiśeṣika categories which they are intended to stand for, is also kept intact. The node labels used during generation (or pre-interpretation) stage are supposed to map to the following Vaiśeṣika categories after interpretation (post-interpretation) stage, as shown in Table 1.

Table 1. Mapping between pre-interpretation (syntactic) and post-interpretation (semantic) labels

Pre-interpretation label (purely syntactic)	**Post-interpretation label (semantic)**	**Corresponding Vaiśeṣika category**
g (start node)	not applicable	not applicable
h	not applicable	not applicable
i	not applicable	not applicable
p	MUS	Mobile Ultimate Substance
q	SW	Substantial Whole
r	UUS	Ubiquitous Ultimate Substance
s	U	Universal
u	Q	Quality
v	UD	Ultimate Differentiator

At the end of the generation all the (pre-interpretation) labels were replaced by a common label, e, to show that the nodes they were labeling can be interpreted later based purely on their structures and not on their labels. Now we will use the same ‘e’ to start the process of parsing! And the edge labels stand for the following: in – inherence relation, con – conjunct relation, dis – disjunct relation. But in the rules below we have differentiated edges based on their arrows instead of their labels for aesthetic purposes. An inherence relation has one arrow (since it is an asymmetric relation), conjunct relation has two arrows and a thick line whereas disjunct relation has two arrows and a dashed line (both are symmetric relations).

No two entities have more than one edge (of any type) between them. That is a default NAC for every rule and hence not being specified with each rule.

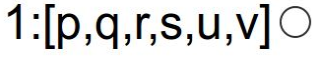
Since we are interested in only the structure of the input graph, we can assume all the nodes of the input graph are either anonymous or uniformly labeled (except those of C and D since they, anyway, are revealed by their corresponding edges and cannot be otherwise). We will assume all the nodes (except that of C and D) are labeled uniformly, with the label ‘e’ (the same label with which we homogenized all the nodes at the end of generation. From here, we will try to backtrack the generation process

with the hope of reaching the start graph of generation grammar of VGG. For this, we will apply the generation rules in reverse, though, with some changes in their NACs. So the first layer of parser rules will be the last layer of generative rules and so on. We will look at the parser rules (PRs) now.

The concept of layers is such that once we reach layer 'n', we cannot apply rules of layers 1 to n-1. A layer can have one or more rules. The rules in a given layer can be applied in any order.

The first layer consists of six rules corresponding to the six labels that the nodes can take during the generation mechanism in VGG. These six can be applied in any order to give us all possible graphs with all possible combinations of node-labels. Here we combine all these six rules into a single rule for optimizing the space. The square bracket in the below rule indicates 'or' i.e., the label 'e' in LHS can be replaced with any one of the labels listed between the square brackets in the RHS.

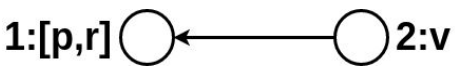
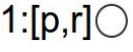
Table 2. PRs Layer 1, Rules 1 to 6

LHS	RHS
	

An important point to note here is that the parser, like any graph transformational system, can take multiple paths in the process of transformation. Even if one of these paths ends at the start graph of generative grammar of VGG, the parsed graph is said to be derivable in our system i.e., it would be generated by the generative rules of VGG.

The second layer is about ultimate differentiator (v). Wherever it is found, we will just delete it. This layer has two rules which are applied on the TGs[7] of first layer. Again, representing both these rules together for parsimony purposes.

Table 3. PRs Layer 2, Rules 1 and 2

LHS	RHS
	

The above rule states whenever a 'v' is found in a 'p' (or 'r'), just delete it. Its edges get deleted with it automatically since the grammar does not allow the possibility of dangling edges.

There are 3 NACs associated with this rule:

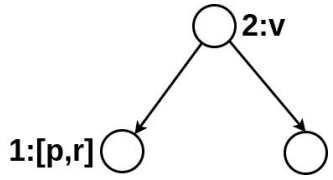
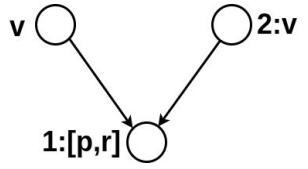
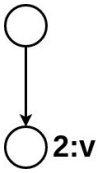
1. v should not inhere in any other node (when we say 'v should not inhere', we mean 'v like structure should not inhere'. The parser makes sure of that!)

[7] TGs (Terminal Graphs) of a given layer are those graphs on which the rules of that particular layer are no more applicable. NTGs (Non-Terminal Graphs) of a given layer are those on which the rules of that particular layer can still apply.

2. No other v should inhere in this p (or r) in which this v is inhering.
3. Nothing should inhere in this v.

The anonymous nodes in NAC_1 and NAC_3 stand for any node, like in interpretation rules of VGG.

Table 4. PRs Layer 2, NACs of Rules 1 and 2

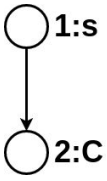
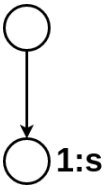
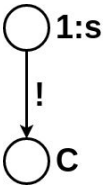
NAC_1	NAC_2	NAC_3
		

The next few layers (3 to 8) are about universals (s). We will look for the nodes corresponding to universals and remove them.

Layer 3 has only one rule. It checks if 's' inheres in all Cs, and if yes, deletes 's' directly. This rule has two NACs.

1. Nothing should inhere in this s.
2. It should inhere in every C i.e., there should be no C in which it is not inhering.

Table 5. PRs Layer 3, Rule 1 with NACs

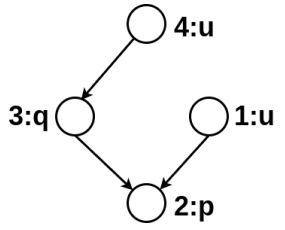
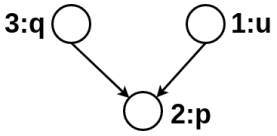
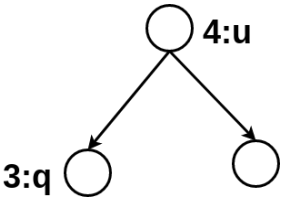
LHS	RHS	NAC_1	NAC_2
			

The exclamation mark (!) in NAC_2 indicates 'not' operator. So NAC_2 reads as '1:s not inhering in C'. So if NAC_2 is true i.e., if there is a C in which 1:s does not inhere, then the rule will not be applicable.

The layers 4 to 8 are quite similar to layer 3, except that 'C' is replaced by 'D' (for disjunct), 'p' (for MUS), 'q' (for SW), 'r' (for UUS), 'u' (for Q) respectively in each layer. So not listing those layers separately here.

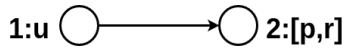
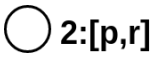
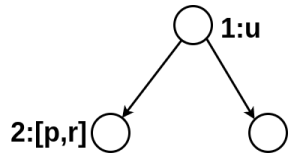
Layer 9 deals with the deletion of quality (u) from a substantial whole (q). It has only one rule which says that while a quality (u) is inhering in a MUS (p), and another quality (u) inhering in a substantial whole (q), remove the quality from SW under the condition that this quality (u) does not inhere anywhere else (this condition constitutes the unique NAC of this rule).

Table 6. PRs Layer 9, Rule 1 with NAC

LHS	RHS	NAC_1
		

Layer 10 deals with the deletion of quality (u) from MUS (p) and UUS (r). Actually both these are separate rules, but combined into one for parsimony purpose. The combined rule is presented below.

Table 7. PRs Layer 10, Rules 1 and 2 with NAC

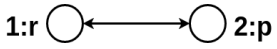
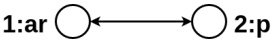
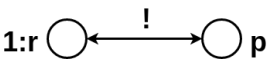
LHS	RHS	NAC_1
		

So the rule in layer 10 states that remove quality (u) from MUS (p) or UUS (r) under the condition that it does not inhere anywhere else (the condition that constitutes the unique NAC of this rule).

The next layer i.e., layer 11 has two rules which deal with deleting the contacts among MUSs (p) and UUSs (r). Since UUSs (r) should be in contact (or in disjunct, which will be handled in the next layer) with ALL the MUSs (p) we need to check if that condition is satisfied or not. But since we are not using logical quantifiers like universal quantifier or existential quantifier, we need to rely on some other technique to verify that condition. One simple method we would use is to rename 'r' to 'ar' if it is in contact with all the 'p's. Then we would delete the contacts among 'ar's and 'p's one by one. At the end, we would rename 'ar' back to 'r' to accommodate the rest of the rules based on 'r' (this rule would be part of layer 13).

The first rule of this layer renames 'r' to 'ar' if 'r' is in contact with all the 'p's. One way to do this is to make sure that there is no 'p' with which it is not in contact (this condition constitutes the unique NAC of this rule).

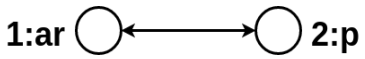
Table 8. PRs Layer 11, Rule 1 with NAC

LHS	RHS	NAC_1
		

Please note that the 'p' in RHS is mapped to the 'p' in LHS (with the number '2') whereas the 'p' in NAC_1 is not mapped to the 'p' in LHS (with any number). This is to indicate that the 'r' in LHS should not be in non-contact with *any* 'p', not just the particular 'p' which is under consideration in the LHS. Hence it is not restricted by mapping it to the 'p' in LHS with a number.

While the first rule of this layer is mostly a verification condition, the second rule is where the actual action happens. This rule states that the contact between 'ar' and 'p' should be deleted one by one. This rule is run till the end, until there are no more contacts between any pair of 'ar' and 'p'. It has no NACs associated with it.

Table 9. PRs Layer 11, Rule 2

LHS	RHS
	

Layer 12 is very similar to layer 11 except that contact should be replaced by disjunct here everywhere in the rules. The way contacts between 'r's and 'p's are removed in layer 11, in the same way disjuncts will be removed between 'r's and 'p's in layer 12. Other than that, everything is same in both the layers. So not listing layer 12 and its rules separately, and moving on to layer 13 directly.

Layer 13 has only one rule - to rename all 'ar's back to 'r's to continue with the process of parsing and enabling the application of rules based on the label 'r'. This will be applicable only on the TGs of layer 12. It has no NACs.

Table 10. PRs Layer 13, Rule 1

LHS	RHS
	

The remaining rules are not being presented here due to space constraints. For the full list of PRs, one can refer to [15].

4. Results and Conclusion

Using this parser, we tested around 100k graphs (98,634 to be precise) generated by the generative rules of VGG and found that every single one of them is a derivable graph (because the parser could parse each one of them and found at least one path where it landed up at the start graph). We then manually gave some graphs as input to the parser which, we know, are not generated by the GRs of VGG. As expected, they are classified as non-derivable graphs by this parser. This shows that the parser is powerful enough to differentiate the graphs generated by the GRs of VGG from those not generated by it.

But our goal is something bigger. We need to also ensure that these graphs generated by GRs are actually Vaiśeṣika graphs and not some arbitrary graphs. In other words, are they interpretable in terms of Vaiśeṣika categories, and if yes, how to verify that.

We interpreted each of these 100k graphs using our IRs and found that every single node of it being labeled with one of the Vaiśeṣika categories. Around 4 million nodes (3,886,374 nodes, to be precise) are labeled. To validate this labeling, we checked for the correspondence between the generative labels (like p,q,r etc.) and the interpretation labels (like UD, MUS, UUS etc.) and found that there is a one-to-one correspondence between the labels presented in the following pairs: (p, MUS), (q, SW), (r, UUS), (s, U), (u, Q), (v, UD). This is a resounding (statistical) proof of the soundness of the system - that every graph generated by the GRs of VGG is a valid Vaiśeṣika graph!

Using both the above results - (1) Given an input graph, with its structure alone, the parser can classify if it is derivable i.e., generated by the GRs of the system or not, and (2) All the graphs generated by the system are valid Vaiśeṣika graphs - one can actually verify whether an input graph is a valid Vaiśeṣika graph or not. This, in itself, is a significant result from an ontology engineering perspective since distinction of valid state-of-affairs from invalid ones is immensely useful for many rich applications like semantic search, machine translation, object recognition, text summarization etc.

But we have a long way to go before we build such applications. The work presented here formalizes a very small, but foundational, portion of reality, and we need to formalize much larger portions of reality to make such complex applications possible. We are working in that direction, and hope this work also encourages others to look at the possibility of generative ontology more seriously.

References

[1] Rajesh Tavva and Navjyoti Singh, *Generative Ontology of Vaiśeṣika,* JOWO, 24th International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina (2015)

[2] Kaṇāda, *The Vaiśeṣika sutras of Kaṇāda*, with the commentary of Śamkara Miśra and extracts from the gloss of Jayanārāyaṇa. Translated in English by Nandalal Sinha. Allahabad (1911); 2nd edn. Revised and enlarged, Allahabad (1923); Reprinted New York (1974), Delhi (1986)

[3] Praśastapāda, *Padārthadharmasamgraha*, with Nyāyakandali of Srīdhara. English Translation by Ganganatha Jha. Chowkhamba, Varanasi (Reprint 1982)

[4] Udayana, *Lakshaṇāvali*, In Musashi Tachikawa, The Structure of the World in Udayana's Realism: A study of the Lakshaṇāvali and Kiraṇāvali. Springer, Heidelberg (1982)
[5] Singh, N., *Comprehensive Schema of Entities Vaisesika Category System*, Science Philosophy Interface 5(2), 1–54 (2001)
[6] Singh, N., *Formal Theory of Categories through the Logic of Punctuator* (2002) (unpublished)
[7] Singh, N., *Theory of Experiential Contiguum*, Philosophy and Science Exploratory Approach to Consciousness, pp. 111–159, Ramakrishna Mission Institute of Culture, Kolkata (2003)
[8] Rozenberg, G., et al. (eds.), *Foundations. Handbook of Graph Grammars and Computing by Graph Transformation*, vol. 1. World Scientific, Singapore (1997)
[9] https://sites.google.com/site/vrktavya/resources/Generative_Ontology_of_Vaisesika_extended_version.pdf
[10] Gottfried Wilhelm Leibniz, *The Labyrinth of the Continuum: Writings on the Continuum Problem*, 1672-1686. Translated from Latin and French to English by Richard T.W. Arthur. (2001)
[11] Franz Brentano, *The Theory of Categories*, translated from German by Roderick M. Chisholm and Norbert Guterman, The Hague, Boston, London: Martinus Nijhoff. (1981)
[12] Roberto Poli et al. (eds.), *Theory and Applications of Ontology: Computer Applications*, Springer (2010)
[13] https://sites.google.com/site/vrktavya/resources/Vaisesika_Categories_table.pdf
[14] Mukhopadhyay, P.K., *Indian Realism: A Rigorous Descriptive Metaphysics*, K P Bagchi & Company, Calcutta (1984).
[15] Vaiśeṣika Graph Grammar (VGG) System (extended version), https://sites.google.com/site/vrktavya/resources/Vaisesika%20Graph%20Grammar%20%28VGG%29%20System_extended_version.pdf?attredirects=0&d=1