Computational Analysis and Graphical Representation of Navya-Nyāya Expressions - Nyāyacitradīpikā

A dissertation submitted to the University of Hyderabad

for the award of the degree of

Doctor of Philosophy

in

Sanskrit Studies

Arjuna S.R.

12HSPH01

Department of Sanskrit Studies

School of Humanities

University of Hyderabad

Hyderabad

December 2016

Computational Analysis and Graphical Representation of Navya-Nyāya Expressions - Nyāyacitradīpikā

A dissertation submitted to the University of Hyderabad

for the award of the degree of

Doctor of Philosophy

in

Sanskrit Studies

by

Arjuna S.R.

12HSPH01

under the guidance of

Prof. Amba P. Kulkarni

Department of Sanskrit Studies

School of Humanities

University of Hyderabad

Hyderabad

December 2016

Declaration

I, **Arjuna S.R.**, hereby declare that the work embodied in this dissertation entitled "**Computational Analysis and Graphical Representation of Navya-Nyāya Expressions - Nyāyaci-tradīpikā**" is carried out by me under the supervision of Prof. Amba P. Kulkarni, Professor, Department of Sanskrit Studies, University of Hyderabad, Hyderabad and has not been submitted for any degree in part or in full to this university or any other university. I hereby agree that my thesis can be deposited in Shodhganga/INFLIBNET.

A report on plagiarism statistics from the University Librarian is enclosed.

Arjuna S.R.
12HSPH01

Date:
Place: Hyderabad

Signature of the Supervisor

Department of Sanskrit Studies
University of Hyderabad, Hyderabad

Certificate

This is to certify that the dissertation entitled **Computational Analysis and Graphical Representation of Navya-Nyāya Expressions - Nyāyacitradīpikā** submitted by **Arjuna S.R.** bearing registration number **12HSPH01** in partial fulfilment of the requirements for the award of **Doctor of Philosophy** in the School of **Humanities** is a bonafide work carried out by him under my supervision and guidance.

This dissertation is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for award of any degree or diploma.

Parts of this dissertation have been:

A. published in the following publications:

1. **Natural Language Processing - ICON-2014**, ISBN: **9789383635528**, Chapter: Parsing
2. **Sanskrit and Computational Linguistics**, 2016, ISBN: **9788193231906**, Chapter: I

B. presented in the following conferences:

1. **Segmentation of Navya-Nyāya Expressions**, International Conference on Natural Language Processing (ICON) - 2014, Goa University, Goa - 2014
2. **Analysis and Graphical Representation of Navya-Nyāya Expressions**, 16^{th} World Sanskrit Conference, Bangkok, Thailand - 2015
3. **Type-identifier for Navya-Nyāya Expressions**, Philosophical Contributions of Prof. Biswambar Pahi, Jaipur, India - 2016

Further, the student has passed the following courses towards fulfilment of course-work requirement for Ph.D:

Course Code	Course Name	Credits	Pass/Fail
SK 816	Introduction to Linguistics	4	Pass
SK 812	Natural Language Processing	4	Pass
SK 826	Research Methodology	4	Pass
SK 827	Indian and Western Logical Systems	4	Pass

Prof. Amba P. Kulkarni
Supervisor
Professor
Department of Sanskrit Studies
School of Humanities

Dr. J.S.R.A Prasad
Head
Associate Professor
Department of Sanskrit Studies
School of Humanities

Dean
School of Humanities
University of Hyderabad

Acknowledgements

This dissertation would not have been possible without the guidance and the help of many individuals who extended their support in completion of this work.

First and foremost, my utmost gratitude to my supervisor *Prof. Amba P. Kulkarni* for her guidance, patience and continuous support throughout my research work. Without her, this work would not have been possible. I feel blessed for being able to work with a dedicated, supportive and caring person like her. She was there for me in all the situations, whenever I needed her. She is the reason for all my achievements in academic life. She is the inspiration for me.

I kowtow to His Holiness *Sri Vishveshateertha Swamiji* of *Pejavara Matha, Udupi* for their unconditional love, care and support from my childhood.

I express my sincere gratitude to *Prof. Gérard Huet* for his valuable guidance in my research. He taught me a lot in my academic and personal life. I am indebted for his care and love.

I am grateful to *Prof. V. N. Jha* for the encouragement and guidance in the research. He visited the department to encourage my research and I am thankful to him for his valuable suggestions.

I express my gratitude to *Prof. Shrinivasa Varakhedi* for his continuous support in profound discussions on various research problems.

I express my heartfelt thanks to *Prof. K. V. Ramakrishnamacharyulu, Prof. K. S. Prasad, Prof. K. N. Murthy, Prof. K. Subrahmanyam, Prof. Tirumala Kulakarni* for their support and en-

couragements in my research.

I express my sincere gratitude to *Dr. J. S. R. Anjaneya Prasad* for his constant support and encouragement in my research. He helped me a lot with care.

I also convey my regards to my teacher *Dr. A. Haridasa Bhatta* who gave me valuable insights on my work.

I am indebted to *Dr. Anil Kumar* for his help, care and support during my research. He is like brother to me.

I am grateful to *Dr. Pavankumar S, Dr. Monali Das, Dr. Siva P, Dr. Surendra K, Mr. Sivasenani N,*
Ms. Preeti Shukla, Mr. Devanand Shukl, Mr. Krishnamohan K, Dr. Shailaja N, Dr. Vinaya B, Dr. Vani M, Dr. Sreedevi K, Dr. Anupama R, Mr. Sanjeev P, Mr. Madhusoodan P, Mr. CG Krishnamurthi, Ms. Gowri, Ms. Kiranmayee, Ms. Sonia for thier support.

I should thank my most beloved friend *Karunakar M* for his help and support in academic and personal work.

I am also thankful to my friends *Pavankumar, Monali, Praveen Gatla, Jatin Sharma, Gaya Hadiya, Jayshree Gajjam, Gauri Sahoo, Sanal, Raghavan, Ambika Prasad Pani, Rik Ganguly, Santosh Yadav, Imran, Vijay, Venkat Rao uncle of Social Science Canteen* and my beloved group *HCU Kannada Balaga* for their love and support.

I am most grateful to my *parents*, *family* and my beautiful fiancée *S*rividya for their continuous support and encouragement.

I cannot forget the office staff of our department, who provided me all kind of infrastructural help. I thank everyone from Department of

Sanskrit Studies office.

I would like to thank one and all, who have directly or indirectly been instrumental in the completion of my research work and dissertation.

Contents

List of Figures

Dissertation related papers presented at Conferences

Arjuna S.R. and Amba Kulkarni, "Segmentation of Navya-Nyāya Expressions". International Conference on Natural Language Processing. December 18^{th}-21^{th} 2014. Goa University, Goa.

Arjuna S.R. and Amba Kulkarni, "Analysis and Graphical Representation of Navya-Nyāya Expressions". World Sanskrit Conference. June 28^{th}-July 2^{nd} 2015. Bangkok, Thailand.

Amba Kulkarni and Arjuna S.R., "Type-identifier for Navya-Nyāya Expressions". Philosophical contributions of Prof. Biswambar Pahi. March 12^{th}-14^{th} 2016. University of Rajasthan, Jaipur, Rajasthan.

Synopsis

Nyāya (Indian Logic) is one of the fundamental branches of philosophy in Sanskrit. Sage Gautama is known as the founder of Nyāya philosophy. Taking into consideration the developments in Nyāya, one can classify the Nyāya literature into two broad divisions.

1. Prācīna-Nyāya (Ancient Logic, 600 BC - 1200 AD)
2. Navya-Nyāya (Modern Logic/Neo-Logic, 1200 AD - till date)

According to the tradition the period of Prācīna-Nyāya ranges from Sage Gautama to Gaṅgeśa and post Gaṅgeśa period as Navya-Nyāya. In ancient times, debate was one of the important means to express the thoughts or ideas of one's own philosophy to the scholars and the common people as well. Indian intellectual tradition considers debate seriously, and it came up with specific rules regarding the conduct of a debate. In Nyāyasūtra, 5.2.1 - 24, Sage Gautama himself defines the nigrahasthānas. But around 9th century, Śrīharṣha, an Advaita vedāntin in his work Khaṇḍanakhaṇḍakhādya came up with many fallacies in Nyāya philosophy. After this, Udayana felt the need of a new technical language, where there is no ambiguity in expressing the issues. In the works of Udayana, Ātmatattvaviveka and Nyāyakusumāñjali, we find the earlier traces and hints towards the necessity of a technical language

and efforts towards its creation. A few decades later, Gaṅgeśa with the influence of Udayana, came up with an idea to bring unambiguity in the debate process. Thus he completely concentrated on pramāṇa part, not on the prameya as Nyāya tradition did. He developed a new technical language in his monumental work. This gave rise to a new offshoot of Nyāya, Navya-Nyāya (NN).

Development of new language for the debate made Gaṅgeśa stood apart from all other philosophers. He emphasized on the development of many technical terms that brought unambiguity in the process of debate. This technical language influenced all other branches of philosophy in a big way. Of course, Navya-Nyāya also contributed towards theoretical insights into the Nyāya philosophy.

We notice the seeds of Navya-Nyāya in the works of Nyāyakusumāñjali of Udayana. But later Gaṅgeśa (12th century) provided a strong footing through his Tattvacintāmaṇi and thus renowned as the founder of NN. NN is famous for its sophisticated and unique language to express the thoughts in an unambiguous way. This language of NN deals with verbal cognition, logic and epistemology. This language influenced almost every Indian philosophy. In recent times, Computer Scientists(4) also noticed the importance of this formal language.

Goal of research

There are two types of difficulties in understanding this language.

- Linear structure with long compounds
- Concepts associated with the conceptual terms

There are noteworthy efforts in understanding of the complexity of conceptual terms. Shukla(40) with his lucid explanations eases the

complexity of the NN technical terms. Ingalls(13) compared the NN concepts with western logic. Scholars like Matilal(27) and many others contributed to ease this difficulty.

A few scholars concentrated on the understanding of the syntax of the NN technical language. Kulkarni(20) analysed this language with computational perspective using the modified version of Conceptual Graph. Ganeri(9) provides a formal description of various primitive terms of NN. Scholars like Varakhedi(45) and a few others put their effort in this field.

NN Expressions are used to describe the cognitive structure (jñānākāraḥ) as well as the physical world around us (sambaddha-padārthaḥ). An NN Expression is a compound. A compound, in Sanskrit, is written as a single word without any gap or hyphen in between the components, with components joined together following euphonic changes. This makes the processing of Sanskrit compounds more challenging. Kumar et al.(26) describe the steps involved in processing Sanskrit compounds and also discuss the associated computational complexity. The steps are -

1. Splitting a compound into components.
   This involves undoing euphonic transformations.
2. Analysing its constituent structure.
   At this stage a compound is analysed showing how the components are grouped together.
3. Identifying relations between the components.
   Now the relation between the components thus grouped is made explicit.
4. Providing a paraphrase of the compound.

Finally a paraphrase of the compound is generated.

We illustrate these steps with two examples: an English one followed by an NN Expression.

Example 1: Consider the long compound '*lake water pollution reduction log*'.

We skip step 1, since the components here are already split.

1. Constituency analysis for this compound is
   ((((lake-water)-pollution)-reduction)-log)
2. Relations between the components are now marked.
   ((((lake-water)T7-pollution)T6-reduction)T7-log)T6
   Here T stands for Tatpuruṣa (an endo-centric) compound and the numbers 6 and 7 indicate the genitive and the locative case markers.
3. The paraphrase of this compound is generated.
   Log of the reduction in pollution of water in lake.

Example 2: Consider the following NN Expression which defines earth as a substance with smell as its characteristic property.

gandhatvāvacchinnagandhaniṣṭhādheyatānirūpitādhikaraṇatāvatī.

1. After splitting the compound into its components, we get
   gandhatva-avacchinna-gandha-niṣṭha-ādheyatā-nirūpita-adhikaraṇatā^vatī.
   Here the components are separated by hyphen and the derivational suffix '-vatī' is separated by a caret.
2. The constituency parse of this compound is
   ((((gandhatva-avacchinna)-((gandha-niṣṭha)-ādheyatā))-nirūpita)-adhikaraṇatā)^vatī
3. After identifying the relations between the components, we get

((((gandhatva-avacchinna)T3-((gandha-niṣṭha)T7-ādheyatā)K)K-nirūpita)T3-adhikaraṇatā)K^vatī

where K, T3, and T7 stand for Karmadhāraya, and Tatpuruṣa compounds with instrumental and locative case suffixes. These are all endo-centric compounds, with a requirement of nominative, instrumental and locative case suffixes during paraphrasing.

4. Finally the paraphrase of this compound is

   Sanskrit: gandhatvena avacchinnā, gandhe niṣṭhā yā ādheyatā, tannirūpitā adhikaraṇatā^vatī

   Gloss: by_smellness delimited in_smell residing which substratum-ness determined_by_that superstatum-ness possessing

   English: An object which has superstatum-ness which is determined by the substratum-ness that is residing in the smell and is delimited by the smell-ness.

In the traditional oral method of teaching, the teacher used to provide the paraphrase of such long compounds starting from the innermost compound, building in a bottom-up approach, joining one component at a time, explaining the type of the compound. This would then create a whole knowledge structure in the mind of a student. With the advancement of new technology, now it is possible to represent the same knowledge pictorially, which helps a modern student who relies more on visual aids than memory to understand such complex compounds easily.

The NNE can be represented pictorially after the constituency parse. For instance, This NNE ((((gandhatva-avacchinna)-((gandha-niṣṭha)-ādheyatā))-nirūpita)-adhikaraṇatā)^vat-pṛthivī can be represented in Conceptual Graphs as shown in Figure - 1.

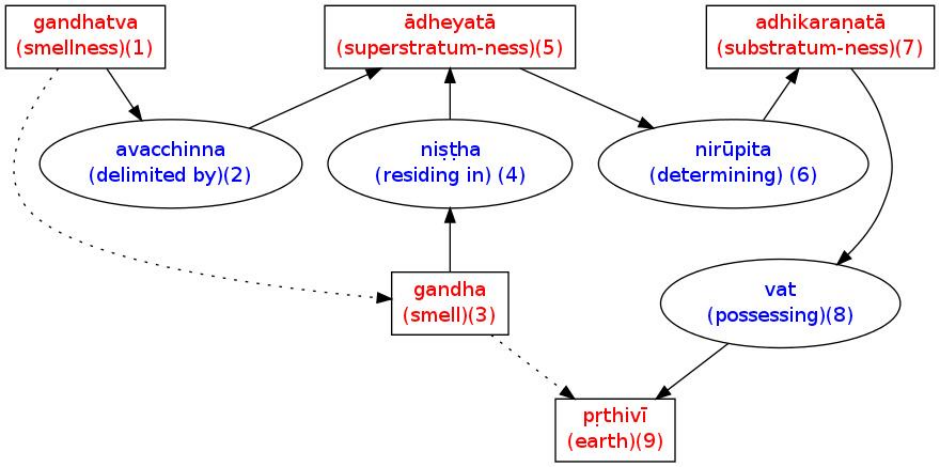

Figure 1: The NNE represented in Conceptual Graphs

'A picture is worth 1000 words' so goes an English idiom. The best way to ease this complexity is by representing the linear structure in a diagrammatical form. We see use of diagrams to express NNEs since long, as early as in 20^{th} century. Vamacaraṇabhaṭṭācārya(33) used diagrams in his teachings. Later in the 60s, Wada(46) mentions that Kitagawa started using diagrams to explain the NN theories. In 1987, V.N.Jha(16) came up with a better solution. He started representing the NNE in a unique diagrammatic form. This method simplified and helped the Nyāya as well as other school students to understand the structure of NN terminology. Later in 1994, Amba Kulkarni put her efforts to build a bridge between NN and Western logic in her M.Tech thesis(20). In this connection, she opted Conceptual Graph, a diagram-

matic representation scheme to show Navya-Nyāya's linear structure in a better way. Next Shrinivasa Varakhedi in his PhD thesis discussed about Knowledge Representation and used the diagrammatical representation method to show Navya-Nyāya structure(45). Toshihiro Wada(46) also used diagrams extensively in his works. Tirumala Kulakarni and Jaideep Joshi also used diagrams(23) to explain the complex NN terms in an easier way.

But all these efforts are manual. Our goal of research is to build a software which renders an NNE diagrammatically, probably automatically and if needed with some human inputs.

Chapter 1

Overview

1.1 Introduction

Nyāya (Indian Logic) is one of the fundamental branches of philosophy in Sanskrit. Kauṭilya in Arthaśāstra emphasizes -

"*Pradīpaḥ sarvavidyānām pradīpaḥ sarvakarmaṇām|*
Āśrayaḥ sarvadharmāṇām śaśvadanvikṣhikī matā".

Sage Gautama is known as the founder of Nyāya philosophy. Taking into consideration the developments in Nyāya, one can classify the Nyāya literature into two broad divisions.

1. Prācīna-Nyāya (Ancient Logic, 600 BC - 1200 AD)
2. Navya-Nyāya (Modern Logic/Neo-Logic, 1200 AD - till date)

According to the tradition, the period of Prācīna-Nyāya ranges from Sage Gautama to Gaṅgeśa and post Gaṅgeśa period as Navya-Nyāya. But a few historians[1] consider three divisions - Prācīna-Nyāya, Madhyama-Nyāya and Navya-Nyāya.

[1] A History of Indian Logic by Satish Chandra Vidyabhushana.

In ancient times, debate was one of the important means to express the thoughts or ideas of one's own philosophy to the scholars and the common people as well. Debate or Dialogue used to take place to remove the confusions in rituals, to highlight the importance of the philosophies and many other purposes. Traditional scholars from different philosophical backgrounds used to meet often at one place and demonstrate their views. Some opposition used to raise on it and then the debate will start between them. Indian intellectual tradition considers debate seriously and it came up with specific rules regarding the conduct of a debate. In Nyāyasūtra, 5.2.1 - 24, Sage Gautama himself defines the *nigrahasthāna*s. But around 9^{th} century, Śrīharṣha, an Advaita vedāntin in his work *Khaṇḍanakhaṇḍakhādya* came up with many fallacies in Nyāya philosophy. After this, Udayana felt the need of a new technical language, where there is no ambiguity in expressing the issues. In the works of Udayana, Ātmatattvaviveka and Nyāyakusumāñjali, we find the earlier traces and hints towards the necessity of a technical language and efforts towards its creation. A few decades later, Gaṅgeśa with the influence of Udayana, came up with an idea to bring unambiguity in the debate process. Thus he completely concentrated on *pramāṇa* part, not on the *prameya* as Nyāya tradition did. He developed a new technical language in his monumental work. This gave rise to a new offshoot of Nyāya, Navya-Nyāya (NN).

The important division between Prācīna and Navya-Nyāya philosophy is based on fundamental issues. Similar to other philosophies, Prācīna-Naiyāyikas concentrate on the salvation and they discuss the topics related to it. But Navya-Naiyāyikas did not stick to this and concentrated on pramāṇas and the development of a new technical language.

Development of new language for the debate made Gaṅgeśa stand apart from all other philosophers. He emphasized on the development of many technical terms that brought unambiguity in the process of debate. This technical language influenced all other branches of philosophy in a big way. Of course, Navya-Nyāya also contributed towards theoretical insights into the Nyāya philosophy.

1.2 Navya-Nyāya

We notice the seeds of Navya-Nyāya in the works of Nyāyakusumāñjali of Udayana. But later Gaṅgeśa (12^{th} century) provided a strong footing through his *Tattvacintāmaṇi* and thus is renowned as the founder of NN. NN is famous for its sophisticated and unique language to express the thoughts in an unambiguous way. This language deals with verbal cognition, logic and epistemology. This Navya-Nyāya Technical Language(NNTL) was so much powerful in its unambiguous expressions that it became the lingua-franca of almost all scholarly works of various branches of knowledge such as Mīmāṃsā 'exegesis' (38), Vyākaraṇa 'grammar' (7), Sāhitya 'literature' (14), Jaina philosophy(43), and even Law(19). In recent times, the importance of this formal language was also noticed by the computer scientists(4).

1.3 Influence of Navya-Nyāya Technical Language

The importance and usefulness of the technical language of NN were noticed by everybody and within no time, it spread across all branches of knowledge systems.The use of NNTL in Vyākaraṇa made it stand

apart from the old texts on Vyākaraṇa and thus resulted in a new discipline *Navya-Vyākaraṇa*.

We find two usages of NNTL - to disambiguate a text and to define the technical terms. Unambiguity being the main criterion in the knowledge systems, it became one of the important branches of essential studies for any Sanskrit scholar. Below we give a few glimpses of the pervasion of NNTL in various branches of knowledge systems, with an example for each.

- Knowledge Branch: Mīmāṃsā

  Text: Mīmāṃsākaustubha of Khanṇḍadeva.

  Context: In the Mīmāṃsā sūtra 2.1.4 on the discussion on how फल is related to धात्वर्थ,

  Example: ननु सम्भवत्यपूर्वाख्य-अवान्तरव्यापारद्वारा सोमादेः फलकरणत्वे क्रियारूपाश्रयव्याप्यत्वं विना फलकामनायां सत्यां स्वत एवाऽपूर्वोत्पत्तिद्वारा फलोत्पत्तिप्रसङ्गेन विधिवैयर्थ्यप्रसङ्गात् आश्रयाकाङ्क्षोपपत्तेः आश्रयस्यापि च गुणनिष्ठकृतिफलजनकतानिर्वाहार्थत्वेन दृष्टविधयैव ------ (pp-9, Mīmāṃsāk-oustubha, Choukambha Sanskrit Series, Benaras, 1932.)

- Knowledge Branch: Vyākaraṇa

  Text: Vaiyākaraṇabhūṣhaṇasāra of Kauṇḍabhatta.

  Context: While defining the meaning of the लृङ्,

  Example: प्रवर्तमानत्वञ्च प्रवृत्तिजनकज्ञानविषयतावच्छेदकत्वं तच्च इष्टसाधनत्वस्य अस्तीति तदेव विध्यर्थः। (pp-29, Vaiyākaraṇabhūṣhaṇasāra, The Narayana Press, Calcutta, 1984.)

- Knowledge Branch: Sāhitya

  Text: Rasagaṅgādhara by Paṇḍitarāja Jagannātha.

  Context: Explaining the concept of Upamā,

  Example: न चात्र यदि नासाग्रस्थितमौक्तिकं तस्या आननमालक्षितबुधाश्लेषं राकेन्दोः

मण्डलमिव विलसतीति तादृशराकेन्दुमण्डलनिरूपितसादृश्यप्रयोजकविलासाश्रयः तादृशमाननमिति तात्पर्यम्।। (pp-12, Rasagaṅgādhara-Part-II, Sampurnanand Sanskrit University, Varanasi, 1981.)

- Knowledge Branch: Jaina

  Text: Nyāyāvatāravivṛti of Siddharṣhi.

  Context: While discussing what kind of *vikalpa* is used here,

  Example: अङ्गुल्यग्रनिर्दिश्यमानपुरोवर्तिनीलस्वलक्षणदर्शनबलायातत्वात् नैल्यविकल्पस्य तदेवाध्यवस्यति न भूतं भावि काककुवलयादिगतं वा इति चेत्, तर्हि विकल्पः स्वलक्षणनिष्ठः प्राप्तः, नियतदेशदशावच्छिन्नार्थक्रियासमर्थार्थग्रहणात् । ... (pp-140, Nyāyāvatāravivṛti, Jaina Sahitya Vikasa Mandal, 1971)

- Knowledge Branch: Vaiśeṣika

  Text: Śaśadhara's Nyāyasiddhāntadīpa's commentary by Śivāditya.

  Context: In the commentary, while discussing about what exactly शिष्टाचारत्वं is,

  Example: अभीष्टोपायत्वमात्रेण शिष्टाचारात्मकहेतूपपत्तेः। श्रुतिबोधितत्वविशेषणावच्छेदेन साध्यव्यापकताग्राहकविपक्षबाधकतर्काभावात् विशिष्टे साध्ये हेतोः अप्रयोजकत्वात् इति भावः। (pp-11, Nyāyasiddhāntadīpa, Santiniketan Viswabharathi Library, 1903)

- Knowledge Branch: Nyāya (Law)

  Text: Dāyabhāga of Jīmūtavāhana. Context: While discussing what will be designated by the word गन्तृ in a particular context,

  Example: ननु धात्वर्थगमनानुकूलकृतिमत्येव गन्त्रादिव्यपदेशः लाघवात्, न तु धात्वर्थतावच्छेदकफलानुकूलव्यापारवति गौरवात् । (pp-16, Dāyabhāga, Sidheswara Press, Kolkata, 1893)

- Knowledge Branch: Uttaramīmāṃsā

  It is well known that Ācārya Śaṅkara, Rāmānuja, Madhva and other ācāryas, who wrote the commentary on Brahmasūtra also used this NNTL in their works.

1.4 Motivation and Goal of research

There are two difficulties in understanding NNTL viz. its linear structure with long compounds and the concepts associated with the conceptual terms. There are noteworthy efforts to understand the conceptual difficulties by many scholars. Shukla(41) eases the complexity of NN technical terms by explaining them in a simple and lucid way. Jha(16) simplifies the big chunk of a Navya Nyāya Expression (NNE) using the diagrams and explaining the concepts in a simple way. Bhatta(3), with his uncomplicated way of explanation and using the diagram elaborates the complex invariable concomitance topic. Ingalls(13), Shaw(39), Mohanty(31), Matilal(28) tried to compare the NN concepts with the concepts in the Western logic and provide logical representations for various important concepts such as *Vyāpti* etc.

The other efforts concentrated on the understanding of the syntax of NNEs. Kulkarni(20), trying to build a bridge between Navya-Nyāya and western logic, analyses the NN in a computational perspective using the modified version of Conceptual Graph. Varakhedi(45) showed the relevance of NN for the Knowledge Representation. Ganeri(9) provides the formal description of various primitive terms of NN. Patil(33) uses the graphical rendering of expressions in his commentary of popular NN text *Tarkasaṃgraha*. Kulakarni and Joshi(23) expounds the technical language of NN in a remarkable way using pictures and graphs. Almost

every scholar used graphical representation in their texts to explain the NN concepts.

We chose to concentrate only on the difficulty in the analysis due to the linear structure of NNE. A single NNE runs into pages, which is very hard for a human to comprehend[2]. In spite of a continuous stream of characters involving arbitrarily long compounds, the cognitive structure being described by such an expression helps a human mind to understand them.

1.4.1 Parsing an NN Expression

NN Expressions are used to describe the cognitive structure (jñānākāraḥ) as well as the physical world around us (sambaddha-padārthaḥ). An NN Expression is a compound. A compound, in Sanskrit, is written as a single word without any gap or hyphen in between the components, with components joined together following euphonic changes. This makes the processing of Sanskrit compounds more challenging. Kumar et al. (26) describe the steps involved in processing Sanskrit compounds and also discuss the associated computational complexity. The steps are

1. Splitting a compound into components.

   This involves undoing euphonic transformations.

2. Analysing its constituent structure.

   At this stage, a compound is analysed showing how the components are grouped together.

3. Identifying relations between the components.

[2]You may refer to Miller's article(29) for more information regarding the human capacity of understanding.

Now the relation between the components thus grouped is made explicit.

4. Providing a paraphrase of the compound.

   Finally, a paraphrase of the compound is generated.

We illustrate these steps with two examples: an English one followed by an NN Expression.

Example 1: Consider the long compound 'lake water pollution reduction log'. We skip step 1 since the components here are already split.

1. Constituency analysis for this compound is

   ((((lake-water)-pollution)-reduction)-log)

2. Relations between the components are now marked.

   ((((lake-water)**T7**-pollution)**T6**-reduction)**T7**-log)**T6**

   Here **T** stands for *Tatpuruṣa* (an endo-centric) compound and the numbers 6 and 7 indicate the genitive and the locative case markers.

3. The paraphrase of this compound is generated.

   Log of the reduction in pollution of water in the lake.

Example 2: Consider the following NN Expression which defines earth as a substance with the smell as its characteristic property.

gandhatvāvacchinnagandhaniṣṭhādheyatānirūpitādhikaraṇatāvatī. (1)

1. After splitting the compound into its components, we get

   gandhatva-avacchinna-gandha-niṣṭha-ādheyatā-nirūpita-adhikaraṇatā^vatī.

   Here the components are separated by a hyphen and the derivational suffix '-vatī' is separated by a caret.

2. The constituency parse of this compound is

((((gandhatva-avacchinna)-((gandha-niṣṭha)-ādheyatā))-nirūpita)-adhikaraṇatā)^vatī

3. After identifying the relations between the components, we get
   ((((gandhatva-avacchinna)**T3**-((gandha-niṣṭha)**T7**-ādheyatā)**K**)**K**-nirūpita)**T3**-adhikaraṇatā)**K**^vatī
   where K, $T3$ and $T7$ stand for *karmadhāraya* and *tatpuruṣa* compounds with instrumental and locative case suffixes. These are all endo-centric compounds, with a requirement of nominative, instrumental and locative case suffixes during paraphrasing.
4. Finally, the paraphrase of this compound is
   Sanskrit: *gandhatvena avacchinnā, gandhe niṣṭhā yā ādheyatā, tannirūpitā adhikaraṇatā^vatī*
   Gloss: by_smellness delimited in_smell residing which substratum-ness determined_by_that superstratum-ness possessing
   English: An object which has superstratum-ness which is determined by the substratum-ness that is residing in the smell and is delimited by the smell-ness.

In the traditional oral method of teaching, the teacher used to provide the paraphrase of such long compounds starting from the innermost compound, building in a bottom-up approach, joining one component at a time, explaining the type of the compound. This would then create a whole knowledge structure in the mind of a student. With the advancement of new technology, now it is possible to represent the same knowledge pictorially, which helps a modern student who relies more on visual aids than memory to understand such complex compounds easily.

'A picture is worth 1000 words' so goes an English idiom. The best way to ease this complexity is by representing the linear structure in a diagrammatical form. We see the use of diagrams to express NNEs since long, as early as in 20^{th} century. Vamacaraṇabhṭṭācārya(33) used diagrams in his teachings. Later in the 60s, Wada(47) mentions that Kitagawa started using diagrams to explain the NN theories. In 1987, V.N.Jha(16) came up with a better solution. He started representing the NNE in a unique diagrammatic form. This method simplified and helped the Nyāya as well as other school students to understand the structure of NN terminology. Later in 1994, Amba Kulkarni put her efforts to build a bridge between NN and Western logic in her M.Tech thesis(20). In this connection, she opted Conceptual Graph, a diagrammatic representation scheme to show Navya-Nyāya's linear structure in a better way. Next Shrinivasa Varakhedi in his PhD thesis discussed Knowledge Representation and used the diagrammatical representation method to show Navya-Nyāya structure(45). Toshihiro Wada(47) also used diagrams extensively in his works. Tirumala Kulakarni and Jaideep Joshi also used diagrams(23) to explain the complex NN terms in an easier way.

But all these efforts are manual. Our goal of the research is to build a software which renders an NNE diagrammatically, probably automatically and if needed with some human inputs.

1.5 The organisation of thesis

In Chapter 1, we see the introduction of NN and the NNTL and the usage of NNTL in other philosophies. We state the goal of this research

as well.

In Chapter 2, we introduce the first step, the segmentation of the NN Expressions. We discussed all the earlier efforts and our present effort in this part.

In Chapter 3, we introduce the Constituency Parsing of the NN Expressions. We elaborate the Context-free grammar written in a parser generator called 'Yacc' and a lexical analyser 'Lex'. How the parsing works, how it is developed and what are the salient features of NNE which helped us making this tool more automatic are elucidated in this chapter.

In Chapter 4, we introduce the Type-identifier of the NN Expressions. We analysed the compounds of NNE which helped out in improving this tool. We discuss the development of this tool in detail.

In Chapter 5, we present the history of graphical representation used in NN. Then we introduce the usage of the Conceptual Graphs for NN Expressions. We explain the Conceptual Graphs renderer for an NNE.

In Chapter 6, we demonstrate all the modules packaged together in the form of a software - *Nyāyacitradīpikā* with an example. We have put the screen-shots of each step explaining the flow.

In Chapter 7, we conclude our research work and mention the future work in this path.

1.6 Contribution of the thesis

The contribution of the thesis is the development of a computational tool to ease the difficulty in understanding the NN Expressions. This work has produced a semi-automatic tool to analyse the NNEs. This tool can segment an NNE according to Nyāya domain, then parse it to

understand the proper semantic structure of it and then render it in a graphical form. It also identifies the type of the compound in the NNE. This work will help the students and teachers of NN to study NN in a better way.

Chapter 2

Segmentation for NN Expressions

The first step in understanding an NNE is to identify the components in a compound. This process of identifying the components of a compound or continuous language string is called Segmentation. Word segmentation is important for languages like Sanskrit which is so much influenced by the oral tradition that the word boundaries undergo euphonic changes resulting into a continuous string of phonemes. The rich productive morphology resulting into the formation of long compounds aggravate the problem. There are significant efforts in this area in the past. Huet(10), Huet and Goyal(11), Hyman(12), Mittal(30), Kumar et al.(26), Natarajan and Charniak(32) have contributed efficaciously to this field.

Hyman(12) describes a Finite State Transducer (FST) for the Paninian sandhi rules. Huet(10) has discussed the segmentation in Sanskrit in detail and has built an efficient Finite State Automata (FSA) based

segmenter. Mittal(30) describes two approaches; one using FST and the other one based on Optimality Theory, by defining the posterior probability function to choose among the valid splits. Kumar et al.(26) used different posterior probability function and obtained better results. Natarajan and Charniak(32) proposed sandhi splitting based on the Dirichlet process.

The NN school of Indian tradition sees the culmination of productive compound formation in the form of compounds running into pages. The components of such compounds are typically formed with more than one *taddhita* (secondary derivational) suffixes. Such compounds also use the technical terms of NN.

Here is an example of linguistic expression in Navya-Nyāya (NNE) involving a compound with nine components:

samavāyasambandha-avacchinna-gandhatva-avacchinna-gandha-niṣṭha-ādheyatā-nirūpita-adhikaraṇatāvatī.

समवायसम्बन्ध-अवच्छिन्न-गन्धत्व-अवच्छिन्न-गन्ध-निष्ठ-आधेयता-निरूपित-अधिकरणतावती

For the sake of readability we show the components split by '-', but in the printed texts this is written as a single word with underlying phonological changes as

samavāyasambandhāvacchinnagandhatvāvacchinnagandhaniṣṭhādheyatā nirūpitādhikaraṇatāvatī.

समवायसम्बन्धावच्छिन्नगन्धत्वावच्छिन्नगन्धनिष्ठाधेयतानिरूपिताधिकरणतावती

All the efforts related to segmentation described earlier had focused on general Sanskrit texts. But for much more complex and domain-specific inputs like NNE, which is known for long compounds, use of technical vocabulary and productive use of secondary derivational suffixes (*taddhita*) a specially trained segmenter is needed.

We report below on our efforts in building a segmenter for NNE, in two stages. First, we report our initial efforts using Heritage engine, followed by building a special morphological analyser and its use for segmentation in Sanskrit Computational Linguistics Platform (SCL/संसाधनी of the University of Hyderabad.

2.1 Preparation of Gold data

The important part of the process is to collect the data and analyse it manually. As a first step, we collected NNEs manually from *Āloka*(44) commentary on *Tarkasaṅgraha* and *Pañcalakṣaṇīsarvasvam*(36). Total 49 expressions were collected from *Āloka* commentary and 352 expressions from *Pañcalakṣaṇīsarvasvam* of *Mathurānātha*. We selected these two only because the first one is a commentary on common and famous text in Navya-Nyāya philosophy. This commentary is small and full of NNEs, which is useful for the understanding the structure of NNEs. The latter one is a bit bigger and it deals with, five definition of invariable concomitance(Pañcalakṣaṇī), is also most discussed in Navya-Nyāya philosophy. The 49 NNEs obtained from the *Āloka* commentary were used for the development purpose and we set aside the 352 NNEs for testing purpose. All the collected NNEs were further analysed for their

components. These two are our gold-data for testing these tools. Gold data is a well-annotated dataset which is reviewed manually by experts related to the particular area which the dataset belongs to. Once the dataset is considered as a gold dataset, any one can use that data which is clean and authentic. These NNEs from gold-data were manually segmented for testing our segmenter.

We extracted the possible combination of secondary derivational suffixes that are found in the selected texts. Figure 2.1 shows these possible combinations, in the form of Finite State Automaton. The numbered nodes indicate the possible final states and the edge labels indicate the taddhita suffixes that can follow a *prātipadikam*.

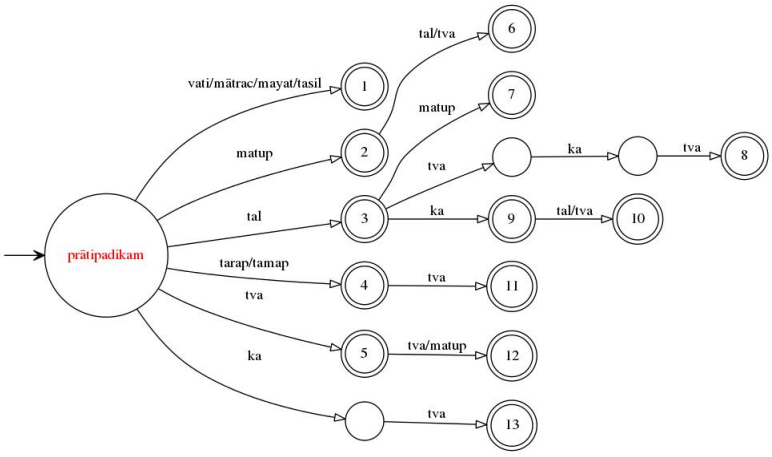

Figure 2.1: FSA showing possible *taddhita* suffixes in NNE

2.2 Sanskrit Heritage Reader for NNEs

Arjuna and Huet(1) summarise the difficulties in handling NNEs as follows.

1. Long compounds,
2. Technical vocabulary,

3. Productive use of *taddhita* suffixes and
4. Semi-formal compound structure.

After understanding the difficulties in segmenting an NNE, the Heritage segmenter was enhanced. The salient features of the enhancement are -

1. New data-banks were added for the inflected forms of the *taddhita* suffixes viz. -*tal* (Fem), -*tva* (Neu) and -*matup* (in all three genders),
2. Technical vocabulary of Navya-Nyāya was acquired in the lexicon,
3. Segmenter transitions were added to accommodate *taddhita* productivity,
4. Word mode for single *pada* was used rather than sentence in order to curb over-generation and
5. Lean interface described in Huet and Goyal(11) was used in order to share the huge solution space.

The recall of the segmenter after this enhancement was 91%.

Problems in Heritage segmenter

There are three problems with this segmenter for a Naiyāyika. The first problem is with the number of solutions. For a typical NNE, this segmenter results with thousands and sometimes even millions of solutions. We can see the problem in Figure-2.2. Typically the top-most row gives the most probable choice and thus for a compound with n components, n choices by the user results in the proper split of the compound. Thus even if there are thousands of solutions, the

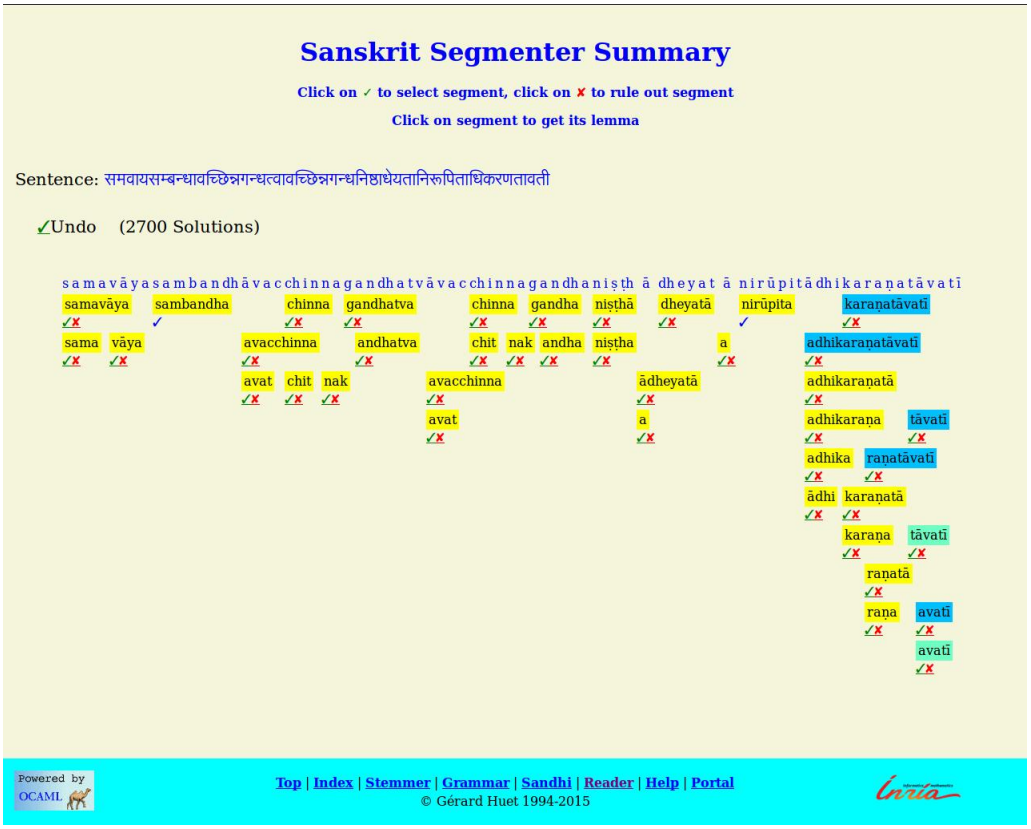

Figure 2.2: First problem in Heritage segmenter

user has to look for only a handful of choices. Hence this problem is not that serious.

The second problem with this segmenter is with the granularity. The Sanskrit Heritage segmenter is enhanced with the technical vocabulary of Navya-Nyāya. But still, it splits many technical words into components. For example, *nirūpita*, *avacchinna*, *samānādhikaraṇa* etc. are split as *ni-rūpita*, *ava-chinna* and *samāna-adhi-karaṇa* respectively. We can see an instance in below Figure-2.3.

In order to understand the NNEs that use their own specialised

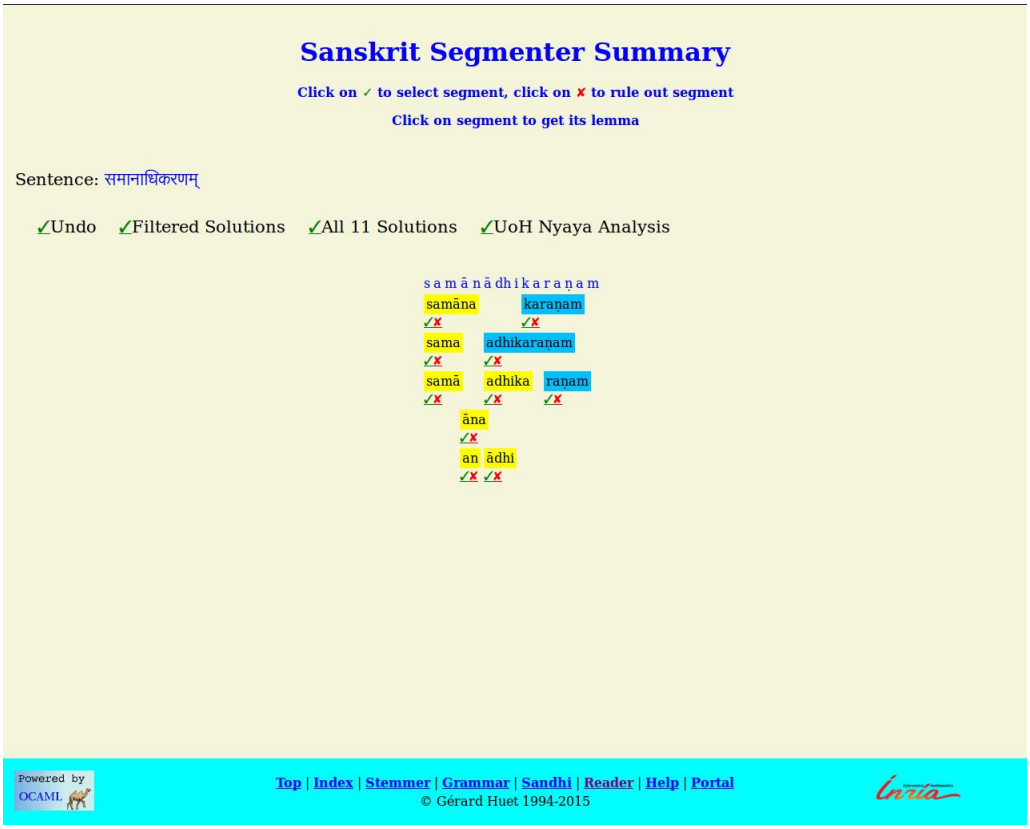

Figure 2.3: Second problem in Heritage segmenter

technical vocabulary with well-defined meanings, to get a broader picture of an NNE, a Naiyāyika(Indian Logician) prefers to hide the derivation of these technical terms and would like to see these words as single units without any splits.

Thus while admitting the fact that the term *samānādhikaraṇa* is compositionally equal to *samāna-adhi-karaṇa* or *vyadhikaraṇa* being compositionally equal to *vi-adhi-karaṇa*, these being technical terms, a *Naiyāyika* would like to look at them as a packaged entry with all the analysis hidden. Treating such technical words as a single unit

would also result in lesser choices for user selection.

Finally, while the user interface has its own advantages, one would like to reduce the user interaction as far as possible, pushing the correct solution to the top or preferably the first position for further automatic processing of such compounds.
We started working on solving these three major problems to get better output so as to ease the process of connection with NN-Parser.[1]
In order to improve the current Sanskrit Heritage Reader to handle domain specific NNEs, we enhanced the tool, as described below.

- **Added some technical terms of Navya-Nyāya to the lexicon.**

  A few technical terms were not recognised by Reader before, viz. *vāraka, anumāpaka* etc. So we lexicalised such words.

- **Displaying words with their prefix as a single word.**

  One of the main problems with this tool was, it was splitting the prefix and words. For instance, *nirūpita* was split as *ni* and *rūpita* which is not desired by a user. Now this problem has been solved. It shows *nirūpita* as a single entry now. When we click on that word for Morph analysis, it displays *ni-rūpita* pointing out that `*ni*' is a prefix of word *rūpita* separated with `hyphen'(-).

- **Segments inchoative compound(cvi).**

  In the previous version of Sanskrit Heritage Reader, inchoative

[1]I acknowledge here the Raman-Charpak Scholarship awarded by the CEFIPRA for the duration March'15-June'15 that enabled me to work closely with Prof. Gérard Huet at Inria, Paris.

compound (cvi compound) were not segmented. For instance, *adhikaraṇībhūtābhāvaḥ* was not split as *adhikaraṇībhūta-abhāvaḥ*. Now we cleared that incompleteness and enhanced the system to split this also.

Programs were modified further to get better result resolving minor bugs as well. The recall of the system went up to 97% with these changes.

2.3 Saṁsādhanī for NNEs

Arjuna and Kulkarni(2) points out these three major problems in Sanskrit Heritage Reader in segmenting the NNEs and while the enhancement of Heritage Reader was on, we also started exploring the improvement of the SCL segmenter simultaneously. As a first step, we used the same domain specific corpus that was collected by Arjuna and Huet(1) and enhanced the morphological analyser of SCL to handle the *taddhita* suffixes reported in Figure 2.1. The 49 NNEs obtained from the *Āloka* commentary were used for the development purpose and we set aside the 352 NNEs for testing purpose. All the collected NNEs were further analysed for their components. The statistics showed that there are a few nominal stems, which are not typical of NN, but occur frequently as a component in the NNEs. These stems are *artha*, *ātmaka*, *pūrvaka*, *vidha*, *kara* etc. which occur as a final component of a compound (*in fine compositi* or *samāsa-uttarapada*). Figure 2.4 shows the sequence of *taddhita* suffixes after which these stems occur. We extended our morphological analyser to handle the derivational morphology -- both the secondary derivations as well as frequent compounds

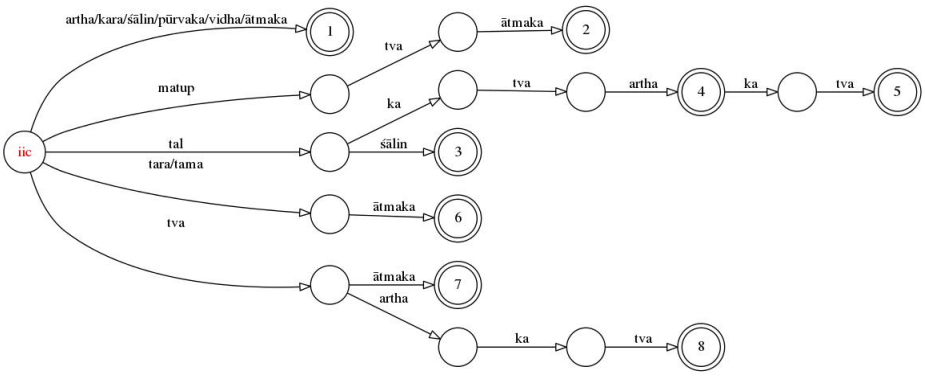

Figure 2.4: ifcs(*in fine compositi* or *samāsa-uttarapada*) found in NNEs

shown in Figures 2.1 and 2.4 respectively. We also extended our lexicon with the technical terms in Navya-Nyāya.

The treatment of some *taddhita* suffixes such as `*ka*' and the treatment of *kṛt* suffixes (primary derivatives) in compound formation need special mention. The *taddhita* suffix `*ka*' results in a *bahuvrīhi* (exo-centric) compound. Such compounds, since having their head external to the compound, they are more like adjectives and thus can decline in all the three genders. For example, if `the one which has the smoke as the cause' refers to a masculine referent, it will have the form *dhūmahetuka*, and if the referent is a feminine, it will have the form *dhūmahetukā*. But such words when occuring as a component of a compound as iics (*in initio compositi* or *samāsa-pūrvapada*), they will always undergo an operation of **puṁvadbhāva**[2] resulting into a word in the masculine gender. Thus we get *dhūmahetuka* as an iic

[2]The condition for *puṁvadbhāva* is given in Paninian *sūtra* 6.3.42. It says - In *Karmadhāraya* compound and in those cases where the second component of a compound ends in a *jātīya* or *deśīya* suffix, the word in the feminine gender will assume the *bhāṣitapuṃska* 'expressed as a masculine' form.

form, whatever be the object it refers to.

Similarly, the adjectives formed by the non-finite suffixes ($kṛt$ suffixes) such as $ṇvul$ (in the sense of an agent), or kta (in the sense of an object) also take the base form[3] when they occur in the compounds as iics. For example, a compound `a lady cook' will be $pācakastrī$ and not $pācikāstrī$.

The Paninian $sūtra$ that governs the formation of such compounds is $puṃvatkarmadhārayajātīyadeśīyeṣu$ 6.3.42. We have enhanced our morphological analyser to take into account this phenomenon.

We were able to split all 49 examples from the development data with this enhancement. The possible splits in each case were in thousands and in 3 cases even in tens of thousands. This was mainly because, though the technical terms were available in the lexicon, machine in addition to this lexical term, also showed all possible splits of such words. For example, for $sambandhāvacchinna$ is split as $sambandha + avacchinna$ and also as $sambandha + ava + chinna$ and so on. The second one should be pruned out since in the context of Navya-Nyāya, $avacchinna$ being a technical word need not be split further. So we needed a splitter that discards splits of morphologically analysable long words. We describe below the algorithm of this splitter followed by its performance on the development data as well as the test data.

[3]The technical term for such base forms in Sanskrit is the one with $puṁvadbhāva$.

Algorithm of SCL-NN segmenter

The main aim of this algorithm is to reduce the over-generation ensuring that the imposed conditions do not under-generate and at the same time, push the most preferred solution to the top of the possible solutions. The over-generation and the under-generation are measured only with respect to the NN vocabulary. Thus a split which is an over-generation from the NN point of view may be a genuine split in the classical Sanskrit. The salient features of the algorithm are stated below.

1. The sandhi rules are of the form $u \rightarrow v + w; f$, where f indicates the frequency of the rule which was observed in the Sanskrit Consortium Corpus[4]. Even if u is just a concatenation of v and w, without any underlying phonetic change, then also we treat it as a sandhi rule, in order to use the frequency information.

   The splitter scans the string from the left and looks for the longest match each time. At each juncture, typically more than one sandhi rules are available. In case there are more than one applicable rules, the one with longer u is preferred over the smaller ones and in the case of two rules with matching u of equal length, the one with the higher frequency is chosen.

   For example, consider a string *adhikaraṇatānirūpaka*. There are two possible splits for this viz. *adhikaraṇatā* + *nirūpaka*, and

[4]This corpus developed by Sanskrit Consortium, which is manually tagged of around 150K words and has around 30K examples of compound words. Refer - "Statistical Constituency Parser for Sanskrit Compounds" of Amba Kulkarni and Anil Kumar, ICON-2011.

$adhikaraṇatā + anirūpaka$. The first split corresponds to a split rule $\bar{a}n \rightarrow \bar{a} + n$, which involves a window of two phonemes. The second split corresponds to the split rule $\bar{a} \rightarrow \bar{a} + a$ which involves a context of only one phoneme. The preference for two phoneme rule produces the split $adhikaraṇatā + nirūpaka$ before other split $adhikaraṇatā + anirūpaka$. Thus we ensure that the most likely output appears before the other solutions.

There are four ways in which $\bar{a}$ can be split, viz. a+a, a+$\bar{a}$, $\bar{a}$+a and $\bar{a}$+$\bar{a}$, with the frequency of occurrence in the Sanskrit Consortium corpus as 3413, 2072, 350 and 233 respectively. Machine uses these rules in the decreasing order of frequency to ensure that the most probable one is reported first.

2. Preference is given to the NN vocabulary over others. The expression $avacchinnakāryatā$ is wrongly split as $avacchinnaka$+$āryatā$ as the more preferred one rather than the correct split $avacchinna$+$kāryatā$, because greedy match prefers the longest component in the beginning. As we notice, the phoneme sequence `ka' can be potentially a $taddhita$ suffix of the first component as well as an initial sequence of an NN technical vocabulary. We resolve such conflicts in favour of the NN technical vocabulary.
3. The splitting is done recursively following the depth first search. The boundaries at which the string is split and the split rule used are remembered. The string is not split twice at the same place with the same split rule. This is to avoid the further splitting of bigger components and thereby increasing

the precision. For example, a string *pratiyogitānirūpaka* is split as *pratiyogitā+nirūpaka* with a rule $\bar{a}n \rightarrow \bar{a}+n$ and as *pratiyogitā+anirūpaka* with a rule $\bar{a} \rightarrow a+a$. But the string *pratiyogitā* is not split further as *prati+yogitā*, nor is *nirūpaka* as *ni+rūpaka*.

4. The treatment of *puṁvadbhāva* in the derivational morphology of compounds help in pruning out the wrong splits such as *niṣṭhā+ādheyatā* for *niṣṭhādheyatā*. *Puṁvadbhāva* ensures that we get only the valid split *niṣṭha+ādheyatā*.
5. A split is considered to be an over-generation if it does not contain any NN technical term.

Analysis of the Result

Our aim was to improve the precision and also get the correct solution to the top of the list. We first discuss the precision and recall.

Precision and Recall

We tested 49 NNEs collected from *Āloka* commentary of *Tarkasaṅgraha* on both the Sanskrit Heritage splitter as well as the SCL-NN splitter. The number of possible splits produced by both these splitters is reported in Table-2.1 and Table-2.2.

No of Solutions	No of Cases
0-100	10
101-1,000	15
1,001-100,000	18
100,000	5
Time-out	1
Total	49

Table 2.1: Number of solutions of Sanskrit Heritage Splitter

As is obvious from the tables, the number of solutions is reduced

No of Solutions	No of Cases
0-5	14
6-10	11
11-100	18
101-1000	5
1000	1
Total	49

Table 2.2: Number of solutions of SCL-NN Splitter

drastically, increasing the precision.

The result of the test data of 352 examples (see Table-2.3) from *Pañcalakṣaṇīsarvasvam* also confirms that the new algorithm prunes out all irrelevant splits. The recall is around 91%, which is as good as the recall of Sanskrit Heritage splitter and at the same time, the number of solutions is reduced substantially, increasing the precision almost 100 times.

No of Solutions	No of Cases	Percentage
0-5	196	55.7
6-10	56	15.9
11-100	72	20.4
101-1000	13	3.6
1000	3	1
No Split	12	3.4
Total	352	100

Table 2.3: Number of solutions of SCL-NN Splitter

Correct Solution

We compared all the generated solutions with the manually tagged Gold data. The Table-2.4 shows the number of cases corresponding

to the position of the correct solution among the ones produced. In 42 cases, the first solution produced by the machine was the correct one. Later we tested examples from *Pañcalakṣaṇīsarvasvam*. The results

Position	No. of Cases	Percentage
1	42	86
2	2	4
3	4	8
7	1	2
Total	49	100

Table 2.4: Position of the correct solution in the Development data

are shown in the Table-2.5.

Position	No. of cases	Percentage
1	264	75
2-5	42	11.9
6-10	6	1.7
11-100	7	2.0
101	2	0.6
No Split	12	3.4
No-correct solution	19	5.4
Total	352	100

Table 2.5: Position of the correct solution in the test data

2.4 Saṁsādhanī-NN Segmenter with controlled lexicon

The above two approaches, in stages, improved the precision as well as recall of the NNE segmenter. But still, the segmenter was not usable by a Naiyāyika. The Naiyāyika always wondered if a human being can split the compound in a unique way, why does the machine find it difficult? Can we reduce further the `ambiguities' the

machine encounters? The answer is yes. We looked at the multiple splits produced by the segmenter and identified the impossible splits and provided the reasons for pruning them out. The main reason was that every expression had some word in it which was not found in the lexicon and hence the expression was split in a wrong way. This prompted us to build a special morphological analyser with the vocabulary from the Nyāya texts. The lexicon for the morphological analyser was built from the high-frequency words found in the Nyāya texts. This change resulted in a drastic improvement in the performance.

No of solutions	Cases	Percentage
1	340	96.59
2	12	3.40
Total	352	100

Table 2.6: Performance of Saṁsādhanī-NN splitter on test data

The performance of this segmenter over the 49 examples from the āloka commentary of Tarkasaṅgraha was 100%. And also 352 examples from *Pañcalakṣaṇīsarvasvam* are shown in Table-2.6. These results confirm that the newer algorithm prunes out all irrelevant splits. The recall is 100%, against the recall of 97% of Sanskrit Heritage enhanced splitter for NN and the previous version of Saṁsādhanī segmenter. At the same time, the total number of splits is reduced substantially, increasing the precision to 97%. Another remarkable point is, in all the examples the correct split was always found at the first place.

With both the segmenters - Heritage and SCL, performing well, we moved further to our next task of building a constituency parser for

a segmented NNE. The Heritage segmenter needed a human interaction in choosing the correct parse, while the SCL segmenter was fully automatic. The user is given a choice to select the segmenter of his choice. In the next chapter, we describe the constituency parser for NNE.

Chapter 3

Constituency Parser for NNE

Constituency parser takes a segmented compound as an input and produces a binary tree showing the syntactic composition of the compound. The segmented expression needs further analysis to get the underlying constituency structure. For example, a compound with three components *a-b-c* may be analysed in two different ways viz. *(a-(b-c))* and *((a-b)-c)*. As the number of components increase, the number of possible analyses grows fast, and is represented by a Catalan number (10). It is the meaning compatibility (*sāmarthya*), that triggers the correct analysis. Kulkarni and Kumar(21) proposed a statistical constituency parser that uses the statistical properties of a tagged corpus to model the *sāmarthya*. Due to unavailability of the tagged corpus for NN, it was not possible to follow this approach for parsing. The well-defined syntax of NNEs discussed by Ganeri(9) motivated us to look at the constituency parsing of NNE afresh from the computational linguistics point of view.

3.1 Syntax of NN Expressions

The NNEs are compounds. According to Matilal(27), ``The NN Expression involves a small number of technical terms together with a non-logical vocabulary''. In order to develop a parser for NNEs, we need to understand the syntax of an NNE, in terms of the categories of the constituents involved and the arrangement of the terms belonging to various categories. Ganeri(9) in the informal description of the NN classifies the constituents of an NNEs into 6 categories.

1. **Primitive Terms**

   These are the nouns such as *ghaṭa* `pot', *bhūtala* `ground', *gandha* `smell', etc..

2. **Abstract Functor**

   This is a derivational suffix `tva' or `tā' (-ness or -hood), that maps a noun to an abstract noun. For example, the smell is mapped to smell-ness, pot to pot-ness.

3. **Relational Abstract Expressions**

   The relational abstract expressions are derived from relation-denoting terms by adding a `tva' or `tā' (-ness or -hood) suffix. For example, *pitṛ* `father' is a relation-denoting term. By adding `tva' suffix, it changes to *pitṛtva* `father-hood', a relational abstract expression. Some other relational abstract expressions are *putratva* `son-hood', *ādheyatā* `superstratum-ness' and *adhikaraṇatā* `substratum-ness'.

4. **Conditioning Operator**

   The conditioning operator *nirūpita* `determining' operates on the relational abstract expressions to form a term. For example,

X-nirūpita-pitṛtva `father-hood determined by X'.

5. **Sentence-forming Operator**

   The terms such as *niṣṭha* `resident in' and *avacchinna* `delimited by' combine a relational term with another term to form an NNE.

6. **Negation Functor**

   abhāvaḥ `Negation/absence'.

These 6 categories help in understanding both the syntax as well as the semantics of the NN Expression. Our ultimate goal is to represent an NNE as a Conceptual graph. A Conceptual graph distinguishes between a concept and a relation. With this at the back of your mind, we re-looked at this classification. Before we look at the classification, we need to understand the role of the derivational suffix *tva*.

In NN technical language, the tva suffix occurs in two different contexts. One is tva as in `*ghaṭa-niṣṭha-ghaṭatvam*' and another is tva as in `*ghaṭa-bhedatvam*'. In the first example, `*ghaṭatvam*' represents an intrinsic property of *ghaṭa* which resides in *ghaṭa*. So the *anuyogin* and *pratiyogin* of `*niṣṭha*' relation are `*ghaṭa*' and `*ghaṭatvam*' respectively. Thus here the tva suffix is attached to the second component of the compound. In the second example, suffix tva is attached to the compound *ghaṭa-bheda*. Here *tva* is not the property of *bheda* but of *ghaṭa-bheda*. This difference can be captured by the parse. The first one will be represented as ((ghaṭa-niṣṭha)-ghaṭa^tvam) and the second will be represented as (ghaṭa-bheda)^tvam. In the first case, suffix tva is part of a component of the compound. But in the second case, *tva* operates on a compound. So it is necessary to disambiguate between these two usages. The matup suffix as well has the same ambiguity. For

example, consider `*ghaṭapaṭatvavat*' and `*sādhyābhāvavat*'. If we look at the parsed output of both, the first one is (ghaṭa-paṭatva^vat) and the second one is (sādhya-abhāva)^vat. So at the segmentation level, we split these secondary suffixes *vat* and *tva* and separate them with a caret `^' sign.

Among the six categories of Ganeri, the *primitive terms* and the *relational abstract expressions* represent the conceptual terms. *Conditioning operators* and *sentence-forming operators* represent the conceptual relations. The *abstract functor* `tva' suffix is a morpheme which denotes a derivational suffix that maps a noun to an abstract noun. In addition to the *abstract functor* -- the `tva' suffix, we also need a derivational suffix `vat' (possessing) which maps an abstract term to a noun. We treat *tva* and *vat* as relations. The reason for considering them to be relations and not concept term will become more clear in the next section where we discuss some salient features of NNE. Table - 3.1 shows the contrast between our classification with Ganeri's classification.

Ganeri's classification	Our classification
Primitive term Relational Abstract Expression Negation functor Abstract functor	Conceptual term
Conditioning Operator Sentence-forming Operator	Conceptual Relation

Table 3.1: Difference of classification

3.2 Some salient features of NNEs

- Further observations of these NNEs reveal that the concepts and the relations as described above, alternate in an expression. For example, consider *gandhatva-avacchinna-gandha-niṣṭha-ādheyatā*. Here the components *gandhatva*, *gandha* and *ādheyatā* denote the concepts and the components *avacchinna* and *niṣṭha* denote the relations.
- Every relation is binary. The two relata are called *anuyogin* and *pratiyogin*. ``A relation in NN is always a relation of something(*pratiyogin*) in something(*anuyogin*)".(Kulkarni(20)) If `R' is a relation which connects two concepts `a' and `b' resulting in an expression `a-R-b', then the term `a' is called a *pratiyogin* and the term `b' is called an *anuyogin*. For example, in the expression *gandha-niṣṭha-ādheyatā*, the term *gandha* is the *pratiyogin* and *ādheyatā* is the *anuyogin* of the relation *niṣṭha*. Such a compound thus always will be parsed as ((a-R)-b) and never as (a-(R-b)), i.e. as *((gandha-niṣṭha)-ādheyatā)* and not as *(gandha-(niṣṭha-ādheyatā))*. *Pratiyogin* always is to the immediate left of the relation. Thus this constraint rules out almost half of the possible parses. The *Anuyogin*, on the other hand, need not be to its immediate right and this results in ambiguity. While with three components, then, the NN Expression `a-R-b' is not ambiguous, with five components `a-R-b-S-c' where `a', `b' and `c' are the concept denoting terms and `R' and `S' are the relation denoting terms, there is an ambiguity with the *anuyogin* of `R'. The two possible parses being, ((a-R)-((b-S)-c)) and ((((a-R)-b)-S)-c).

In the first case the *anuyogin* of `R' is `c', while in the second, it is `b'. It is the context that tells us which parse is correct.

For example, in *samavāyasambandha-avacchinna-gandha-niṣṭha-ādheyatā*, the *anuyogin* of *avacchinna* is *ādheyatā*, while in *dravyatva-avacchinna-gandha-niṣṭha-ādheyatā*, the *anuyogin* of *avacchinna* is *gandha*. So, if there are `n' concept nodes after a relation node `R', the *anuyogin* of `R' potentially can be any of these `n' concept nodes. It is the context that decides which is the correct *anuyogin*.

- Another important feature of the NNE is the well-nested constraint. The resulting constituency structure should be well-bracketed, without any crossings. In other words, if the *anuyogin* of a relation at k^{th} position is at `j', then the *anuyogin* of any relation lying between `k' and `j' can not be beyond `j'.
- Further, in an NNE, the relation term *'nirūpita'* always connects mutually related relational abstract expressions. For instance, ādheyatā and adhikaraṇatā are mutually related relational abstract expressions, similarly pitṛtva and putratva are mutually related relational abstracts. Thus the *pratiyogin* of a *nirūpita* restricts the *anuyogin* to be the mutually related abstract expression.

These 4 conditions viz.
(i) Concepts and Relations alternate, (ii) *pratiyogin* of a relation is always to the left, (iii) the constituency structure is well-nested and (iv) the *pratiyogin* and *anuyogin* of *nirūpita* are mutually related relational abstracts reduce the possible parses to a considerable degree.

3.3 Building a constituency Parser

We have seen above that it is not possible to decide the *anuyogin* of a relation automatically. It is the context which plays an important role in the disambiguation. Therefore, we decided to build an user-interface that will assist a user to provide the inputs interactively in order to parse the given segmented compound expression.

The process of human assisted parsing is described below.

1. A segmented compound is an input to the parser.
2. Machine identifies the category of each component as either a concept or a relation.
3. For each relation, the component to the immediate left is marked as its *pratiyogin*.
4. Taking into consideration the constraints viz. well-nested-ness and the related relational abstracts in the case of nirūpita, we list all possible *anuyogin*s for each relation.
5. This data is then presented to the user through an interactive interface as shown in Figure - 3.1.

   The input for this is the segmented string

   samavāyasambandha-avacchinna-gandhatva-avacchinna-gandha-niṣṭha-ādheyatā-nirūpita-adhikaraṇatāvat-vastu

   In this figure, the first row of the table contains the components of the given compound, the second row gives the index of each component and the third row lists all the possible indices of the *anuyogin*s for each relation term. You can also notice that the relation term *nirūpita* has *ādheyatā* as its *pratiyogin* and hence following the constraint described above, it automatically chose

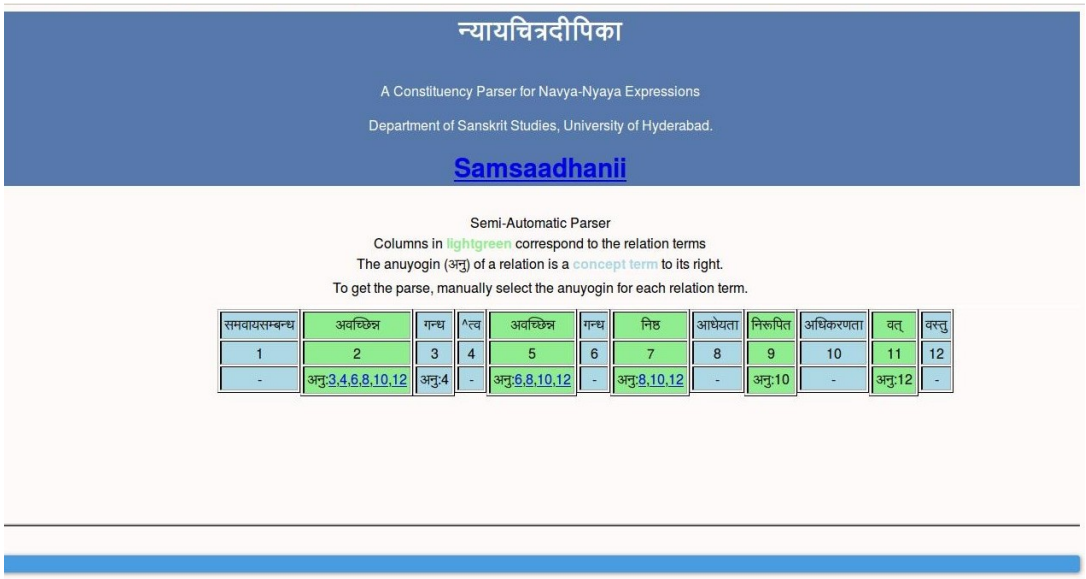

Figure 3.1: A screen-shot of the interface

adhikaraṇatā as its *anuyogin*.

When the user selects an Anuyogin, then the nested parenthesis constraint removes all incompatible solutions reducing the possibilities at each selection. For example, after the user selects *8* below the 2^{n}d *avacchinna*, the possible *anuyogin*s for the relations at 5^{t}h and 7^{t}h position also reduce following the proper nested constraint stated earlier. Figure - 3.2 shows the possible *anuyogin*s after this selection.

The *anuyogin* below the concept term *gandha* needs some clarification. *gandha* and *tva* both are concept terms. An implicit unspecified relation is imagined between these two concepts, the *pratiyogin* of which is *gandha* and *anuyogin tva*. We show this *anuyogin* in the 3^{r}d row under the concept *gandha*. We extend this to other cases as well when the NN Expressions do not

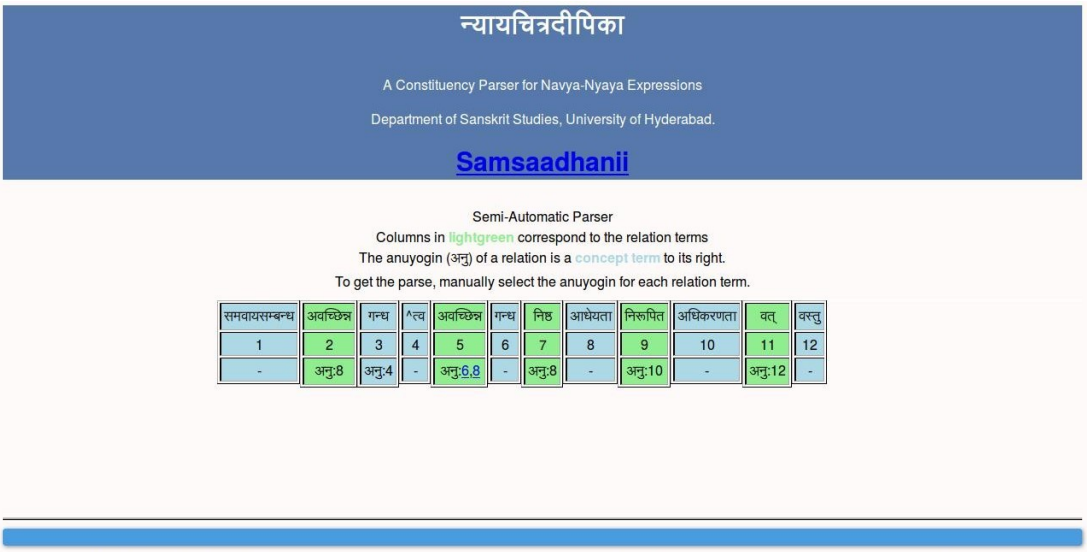

Figure 3.2: A screen-shot of the interface after user-selection

specify the relations between the concepts explicitly. For example, the expression *ghaṭa-abhāva-vat-avṛttitvam* has two concepts *ghaṭa* and *abhāva* as consecutive nodes. In such cases, we treat them as a compound with unspecified relation and produce a parse: (((*ghaṭa-abhāva*)-*vat*)-*avṛttitvam*). In this case, an implicit relation is imagined between *ghaṭa* and *abhāva*. The *pratiyogin* of this relation is *ghaṭa* and the *anuyogin* is *abhāva*. This *anuyogin* showed below the *ghaṭa*.

6. The user repeats step 5 until all ambiguities are removed and each relation has one and only one *anuyogin*.
7. This table is then converted into a nested parenthesis expression by introducing an open parenthesis before *pratiyogin* of each relation term and a closed parenthesis after *anuyogin* of each relation term.

Chapter 4

Type-Identifier

4.1 Earlier efforts

Semantically Pāṇini classifies Sanskrit compounds into four major types: a) Avyayībhāva, b) Tatpuruṣa, c) Bahuvrīhi, d) Dvandva. This classification is not sufficient for the complete understanding of a compound. For example, the paraphrase of a compound *gandhatvāvacchinna* is *gandhatvena avacchinna*, while the paraphrase of *ghaṭābhāvaḥ* is *ghaṭasya abhāvaḥ* and both of them belong to the same class of *Tatpuruṣa*. In the given instances, the paraphrases are different due to the semantic differences and it happens in all the types of compound. Based on their semantic differences, these compounds are further sub-classified into 59 sub-types. The types and sub-types of the compound are based on standards evolved by the project entitled *"Development of Sanskrit Computational Tools and Sanskrit-Hindi Machine Translation System"* sponsored by Ministry of Information Technology, Government of India, New-Delhi.(See Appendix - A)
There are very few efforts in the automatic detection of a compound

type and sub-type. Kumar et al.(26) discuss the steps involved in the analysis of Sanskrit compounds and show how the corpus statistics helps in building a type identifier. Later in 2013, Kulkarni and Kumar(25) used clues from Pāṇini's Aṣṭhādhyāyī in identifying the types of certain compounds. They have critically gone through the Pāṇinian sūtras related to compound formation providing semantic clues. They used a proper combination of both statistical and rule-based methods to decide the compound type semi-automatically. Kulkarni et al.(22) extend this work further for modern Indian languages. They discuss how semantic classification of Pāṇini can be applied to modern Indian languages, with a focus on Hindi and Marathi. According to them, access to the semantic content of the components and also wider context is needed to decide the type of a compound.

The NNEs have the specific vocabulary and the compound type is predictable in many cases. We decided to take advantage of this fact and build a domain-specific type-identifier for NNEs.

4.2 Analysis of NNE compounds

Compounding of words is always between two components at a time, except in *Dvandva*, *Bahupada-Bahuvrīhi* and *Bahupada-Tatpuruṣha* compounds. Regarding the components of an NN compound, Matilal(27) observes: `The NN Expression involves a small number of technical terms together with a non-logical vocabulary'. Thus, the components in an NN compound are of two types: 1) a relational term and 2) a non-relational term. Examples of relational terms are *avacchinna*, *nirūpita*, *niṣṭha*. Relational and non-relational

terms both can appear either as a first (*pūrvapada*) component or as a second (*uttarapada*) component. For example, in a compound *ghaṭatva-avacchinna*, the term *avacchinna* is the second term. Now consider a compound with three components *ghaṭatva-avacchinna-ādheyatā*, which is parsed as *((ghaṭatva-avacchinna)-ādheyatā)*. Here, *avacchinna* is the second component of a compound *ghaṭatva-avacchinna* and is the semantic head of the compound. This compound is further combined with *ādheyatā* to form a complex compound[1]. Similarly, we can find examples where the words *niṣṭha*, *nirūpita*, etc. also can appear either as the second term or as the head of the first term of a component.

Semantics of compounds with *avacchinna* as one component

The term *avacchinna*, as Ganeri(9) describes it, is a sentence forming operator. It joins two terms say `X' and `Y' into a sentence `X-avacchinna-Y'. Now `X-avacchinna-Y' is always to be parsed as *((X-avacchinna)-Y)*. And this will be then paraphrased as `Y which is delimited by X'. For example, the three component compound *((ghaṭatva-avacchinna)-ādheyatā)* will be paraphrased as *ghaṭatvena-avacchinnā yā sā ādheyatā* `The super-stratum-ness, which is delimited by pot-ness'. Generalising this, the paraphrase of *((X-avacchinna)-Y)* will be `X{3} avacchinna yat tat Y', where {3} indicates the instrumental case suffix. Now, following the annotation scheme developed by the Sanskrit consortium, we represent this as `((X-avacchinna)T3-Y)K1', where `T3' stands for an endocentric compound with a relation expressed by the instrumental

[1] samastapada-garbhita-samāsa

case suffix (*Tṛtīyā-Tatpuruṣa*) and `K1' stands for a copulative compound with an adjective as the first component (*Viśeṣaṇa-pūrvapada-Karmadhāraya*).

Semantics of compounds with *niṣṭha* as one component

The term *niṣṭha* is another sentence forming operator as described by Ganeri. Similar to *avacchinna*, `X-niṣṭha-Y' is always to be parsed as *((X-niṣṭha)-Y)*. And this will be then paraphrased as `Y (which is) residing in X'. For example, the three component compound *((ghaṭa-niṣṭha)-ghaṭatvam)* will be paraphrased as *ghaṭe-niṣṭham yat tat ghaṭatvam*. If we generalise, the paraphrase of *((X-niṣṭha)-Y)* will be `X{7} niṣṭha yat tat Y', where {7} indicates the locative case suffix. *avacchinna*, *yat* and *tat* will assume the correct form following the gender of Y. Now, following the same annotation scheme, we represent this as `((X-niṣṭha)T7-Y)K1', where `T7' stands for an endocentric compound with a relation expressed by the locative case suffix (*Saptamī Tatpuruṣa*).

Semantics of compounds with *nirūpita* as one component

The term *nirūpita* is termed as a conditioning operator by Ganeri. It also takes two arguments `X' and `Y' to generate a new term `X-nirūpita-Y'. This expression is always parsed as *((X-nirūpita)-Y)*. Its paraphrase is `Y (which is) described by X'. For example, the three component compound *((ādheyatā-nirūpita)-adhikaraṇatā)* will be paraphrased as *ādheyatayā-nirūpitā yā sā adhikaraṇatā*. If we generalise, the paraphrase of *((X-nirūpita)-Y)* will be `X{3} nirūpitā yā sā Y', where {3} indicates the instrumental case suffix. Now, following the same annotation scheme, we represent this as `((X-niūpita)T3-

Y)K1', where `T3' stands for an endo-centric compound with a relation expressed by the instrumental case suffix (*Tṛtīyā-Tatpuruṣa*).
Thus we see that the relation term as the (head of) second component or the (head of) the first component decides the type of a compound. *Appendix B* gives the list of relation terms along with the possible type of the compound when they are the (head of the) first or the (head of the) second components. If either 1^{st} or the 2^{nd} component of a compound is not a relational term then the usual rules of classical Sanskrit are applicable. Now we look at the context-free grammar of this tool.

4.3 Context-free grammar

CFG is a particular type of formal grammar discussed in formal language theory. These formal grammars are divided into four classes. This classification is due to Noam Chomsky((5),(6)) and it is known as *'Chomsky hierarchy'*.

- Type-0 or Unrestricted grammar.
- Type-1 or Context-Sensitive grammar.
- Type-2 or Context-Free grammar.
- Type-3 or Regular grammar.

NNEs can be generated with Type-2/Context-Free grammar.
If a compound has only two components, then the extraction of both the component is trivial. However, when a compound has more than two components, we need to have a parse of the compound showing how the components are joined one at a time. With a compound with `n' components, there will be `n-1' compounds. Each compound will have two components, which may themselves be compound.

Thus given such a parsed structure, expressed linearly, now we need to scan this parsed structure and identify the first and second components at each level. This problem is similar to the evaluation of a mathematical expression involving nested parenthesized expressions. The context-free grammar to analyse such an expression is given in Table-4.1. For each nested compound, the head is synthesized from its components and finally, the compound type is decided with a function taking two arguments viz. the 1^{st} and 2^{nd} component. The function then computes the type following the rule specified in *Appendix-B*.

NNE	:	compound
	;	
compound	:	Ppada '-' Upada
	;	
Ppada	:	'(' **pada**
	;	
Upada	:	**pada** ')'
	;	
pada	:	compound
	|	concept
	;	

Table 4.1: Context-free Grammar to identify the compound-types

The production rules with the attributes of the CFG is shown in Table-4.2.

In this grammar, `Ppada' stands for a *pūrvapada* `the first component' and `Upada' stands for an *uttarapada* `the second component'. The input for this grammar is a parsed compound. Note that Ppada and Upada consume the open and close parenthesis along with the component. The type of the compounds depends upon the heads of

NNE	:	compound
	;	
compound	:	Ppada '-' Upada
		↑.type = f(Ppada.head, Upada.head)
	;	
Ppada	:	'(' **pada**
		↑.head = pada.head
	;	
Upada	:	**pada** ')'
		↑.head = pada.head
	;	
pada	:	compound
		↑.head = ↓.head
	|	concept
		↑.head = ↓.head
	;	

Table 4.2: Production rules with attributes

Ppada and Upada. For instance, let the input be *((gandha-niṣṭha)-ādheyatā)*. In this input, `*(gandha*' is the *pūrvapada* and `*niṣṭha)*' is the *uttarapada* for the first level of compounding. Then at next level, `*((gandha-niṣṭha)*' is the *pūrvapada* and `*ādheyatā)*' is the *uttarapada*. The word or pada after the symbols (,) is the key to identifying the types of the compound in any NNE. In the above instance, *gandha* and *ādheyatā* are non-relational terms and *niṣṭha* is a relational term. Hence, as discussed above, when *niṣṭha* is *uttarapada*, then the type of that compound is T7 - *Saptamī-Tatpuruṣha*. If it is the head of the *pūrvapada*, then the type of compound will be K1 - *Viśeṣaṇa-pūrvapada-Karmadhāraya*. Similarly, when *avacchinna* is a *uttarapada*, the compound type will be T3 - *Tṛtīyā-Tatpuruṣha* and if it is head of *pūrvapada*, then it will be K1 - *Viśeṣaṇa-pūrvapada-Karmadhāraya*.

The constituency parse of the expression *((gandha-niṣṭha)-ādheyatā)* following this grammar is shown in Figure 4.1. We will see how the

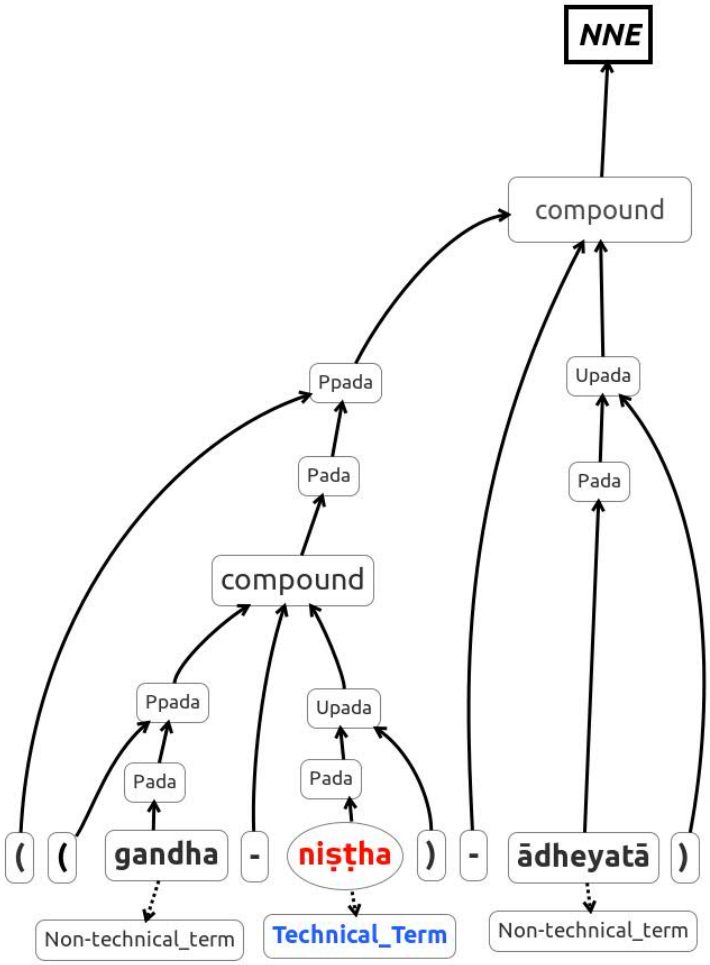

Figure 4.1: Constituency parse corresponding to the grammar

head gets computed in Fig 4.2 and Fig 4.3 with the same example. From Figure - 4.1, we see that the head of the compound(*gandha-niṣṭha*) is *niṣṭha* and hence the compound type is T3. Later, from Figure - 4.3 we notice that the head of the compound(*((gandha-niṣṭha)-ādheyatā)*) is *ādheyatā* and the head of the *pūrvapada* is *niṣṭha*.

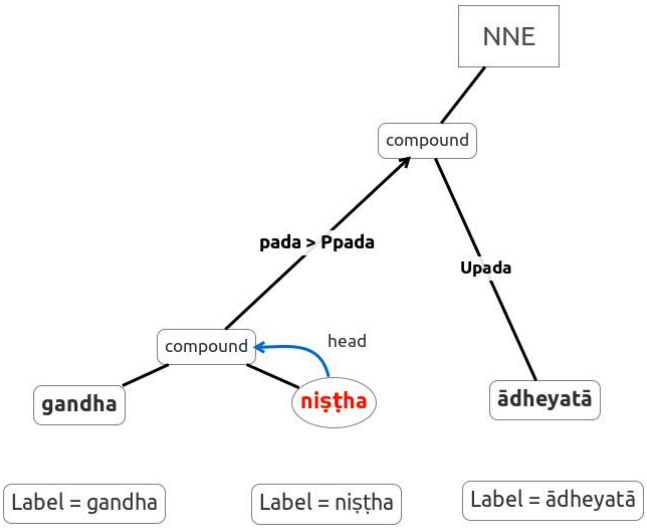

Figure 4.2: Head-info computed according to the grammar - step 1

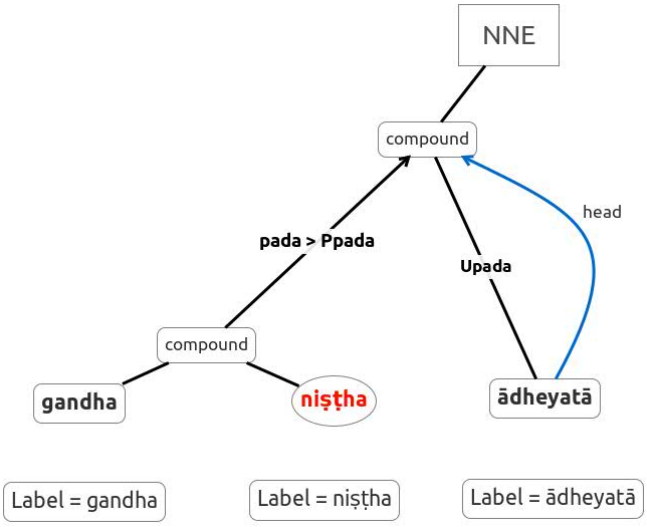

Figure 4.3: Head-info computed according to the grammar - step 2

Therefore the compound type is K1. Thus the whole compound is (*((gandha-niṣṭha)T7-ādheyatā)*) K1.

4.4 Analysis of the result

We tested this type identifier on 352 NNEs collected from Māthurīpañcalakṣhaṇīsarvasvam. Each NNE has more than two components. The minimum number of components were 3 and the longest NNE has 40 components. The 352 NNEs together had 2329 binary compounds. Of these 1340 compounds could not be identified, since none of the components of these compounds was from the NN technical vocabulary. Remaining 989 compounds are recognised correctly. Among the 59 fine-grain compound types, only 8 types of compound are used with NN technical vocabulary. These types are - K1 (*Viśeṣaṇa-pūrvapada-Karmadhāraya*), T3 (*Tṛtīyā-Tatpuruṣha*), T6 (*Śaṣhṭhī-Tatpuruṣha*), Bs6 (*Śaṣhṭhyartha-Bahuvrīhi*), T7 (*Saptamī-Tatpuruṣa*), K6 (*Sambhāvanā-pūrvapada-Karmadhāraya*), T5 (*Pañcamī-Tatpuruṣha*), Tds (*Samāhāra-Dvigu*). Table-4.3 gives the classification of these compounds along with their frequency of occurrence. Our grammar could recognise all of them correctly.

4.5 Conclusion

This is an effort to ease the process of understanding NNEs with the assistance of a computational device. Once the compound types are identified, it is possible to generate the paraphrase also automatically. In order to identify the compound types of the compounds which do not involve NN technical terms, we need the lexical semantics. Anil Kumar(24) and Pavankumar(37) have listed various semantic

Tag	No. of cases
K1	283
Bs6	203
T6	186
T3	178
T7	74
K6	45
T5	13
Tds	7
Total	989

Table 4.3: Frequency distribution of identified compound types

features that are needed for the analysis as well as the generation of the compounds. Further, the advances in machine learning will also help in identifying suitable parameters for correct identification of the compound types.

Chapter 5

Graphical Representation

Graphical Representation is a visual display of a data using the graph. This schema is one of the effective way to express the complex things in a easier way. If a complex NNE can be represented faithfully through a graph, it will be helpful to students and the others as well to understand it in a better and easier way.

5.1 Earlier efforts

Representing of NNEs in graph structure is not a latest phenomenon. Traditional scholars have also used this method in teaching the NN to their disciples in a potential way. Vāmacaraṇa Bhaṭṭācārya (1880-1931) used the graphs in his teaching(33). Figure - 5.1 shows one such example.

There are many noteworthy efforts in recent times by many scholars to simplify the complexity of the NN using graphs as we mention earlier. Jha(16) is well-known for his way of teaching using the diagrams. He came up with his own diagrammatical representation to

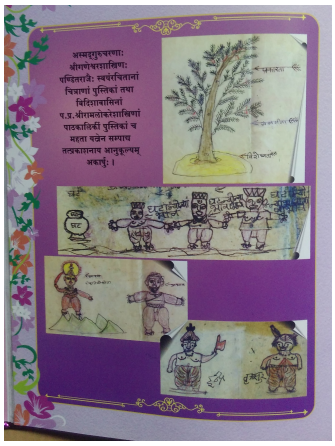

Figure 5.1: The graphs used by traditional scholars

explain the NN theories. He mentions that he learnt it from one of his Japanese student long back to understand the NN using diagrams. He improved the notations as well. In 1994, Kulkarni(20) put an useful effort in understanding the NN structure. She took the five definition of invariable concomitance (Jāgadīśī Pañcalakṣaṇī) and tried to explain using the graph. The graph used by Kulkarni is modified version of the Conceptual Graphs. This effort is to build a connection between Indian and western logicians. Wada(46) also used the graphs in his works. He also analysed the NN language and its technicality and explained it in a simpler way. Kulakarni and Joshi(23) used the graphs in simplifying the language of NN in a unique way. This is an attempt to show the relevance of the NN in today's age to the traditional scholars and to help the people from technical backgrounds like computer science and western logic who are interested to know the contribution of Indian logic and its historical role. Patil(33), being a traditional scholar, used graphs in his work for the convenience of explaining the NN theories. Though this work is in Sanskrit, he clar-

ifies many of the core technical queries with his simplicity in explanation.

These were all manual and inspired by these great efforts, we wanted a tool that renders the graph automatically. So we build a graph renderer for NNEs, following the structure of Conceptual Graphs(CGs). We chose CGs as a representation, since CGs are well studied as well as linked to the First Order Logical analysis, inference engines etc. Let us see the details of Conceptual Graph.

5.2 What is Conceptual Graph?

The Conceptual Graphs(CGs) was originally designed as a semantic representation for natural language. It is developed by John F. Sowa(42). As mentioned in the official web-page of CG - `CGs are a system of logic based on the existential graphs of Charles Sanders Pierce and the semantic networks of Artificial Intelligence'. The importance of this CG is -

- It is easily readable and formal for computational purpose.
- It represents both the linguistic structure as well as the knowledge structure.
- CG is general. Parse trees, Petri nets etc. are special cases of this representation.

For instance, ``A cat is on a mat'' is represented in CG as in Figure - 5.2- Here `cat' and `mat' are the concepts and are represented using

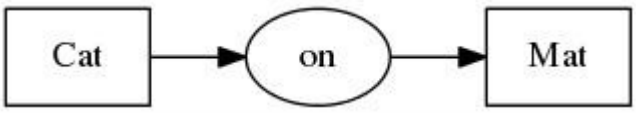

Figure 5.2: An example of CG

boxes and the relation `on' is represented using an oval.

As mentioned by Sowa in Frank Van Harmelen and Porter(8), graph-based semantic representations were famous in theoretical and computational linguistics. Margaret Masterman (1961) introduced the *semantic networks*, which is a graph-based notation. *Correlational nets* introduced by Silvio Ceccato are based on 56 types of different relations. *Dependency Graph* presented by David Hays, formalised the notation developed by the linguist Lucien Tesniére in 1959. These efforts succeeded in representing the relational structures underlying natural language semantics, but failed to convey the full First Order Logic(FOL).

In the late 70s, number of graph notations were designed to represent the FOL. CG is also one of the effort in the same way as an intermediate language for mapping natural language questions and assertions to a relational database. CG is basically extended version of Charles Sanders Pierce's Existential Graphs. So once we are successful in representing the NNEs in CGs, then it will be easier for researchers trained in western logic to understand the structure of NNEs.

Another important reason we chose the CG is because there is a similarity in structure of both NNE and CG. As discussed by Polovina(34), CG has a general form as shown in Figure - 5.3

Figure 5.3: General form of CG

This may be read as - ``The Relation of Concept1 is a Concept2''. The arrows shows the direction the diagram should be read. This general form is similar in NNEs as well. In an NNE also we saw in the previous chapter, Concept1 is followed by a Relation, which is followed by a Concept2. Figure - 5.4 shows the structure for an NNE *gandha-niṣṭha-gandhatvam*.

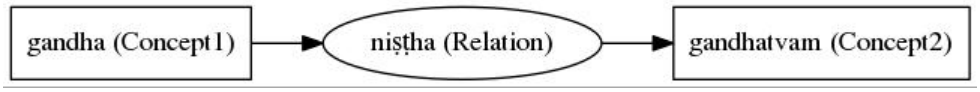

Figure 5.4: General form of an NNE

This particular similarity also made us to chose CG over other graphical representation schemes.
Another point to be mentioned is the display form of the CG in our research work were produced using the *Graphviz* software[1]. Now we look at how CG is used for NN Expressions.

5.3 Conceptual Graphs for NN Expression

The canonical form in NN is `X has Y'. For example, the canonical form corresponding to the sentence

Father of Rama is Dasharatha. (1)

is

[1] www.graphviz.org

Dasharatha has father-hood of Rama. (2)

The preposition `of' being ambiguous, this is further disambiguated as

Sanskrit: *Rāma-nirūpita-pitṛtva^vān Daśarathaḥ* (3)
Gloss: Rama-determining-fatherhood-possessing Dasharatha
English: Dasharatha has father-hood determined by Rama

or, more precisely as

Sanskrit: *Rāma-niṣṭha-putratva-nirūpita-pitṛtva^vān Daśarathaḥ* (4)
English: Dasharatha has father-hood determined by the son-hood resident in Rama.
NN Expression in (3) is an abbreviated version of the NN Expression in (4). In these expressions the terms father-hood (*pitṛtva*) and son-hood (*putratva*) are the abstract terms derived from the relation denoting terms `father', `son' etc. and hence these are termed as relational abstracts. The terms *niṣṭha (residing in), nirūpita* (described by) and *vān* (possessing) denote the relations between the conceptual terms. The term *nirūpita* `determined by' is a relational term which conditions the relational abstract properties. The NN Expression in (4) is represented in conceptual graph as in Figure 5.5. If we read this CG in Figure 2 along the directions of the arrow, we get the NN Expression in (4). When a conceptual node has more than one incoming arrows, there are multiple ways of producing the NN expression. To have a one-one correspondence between the NN Expression and

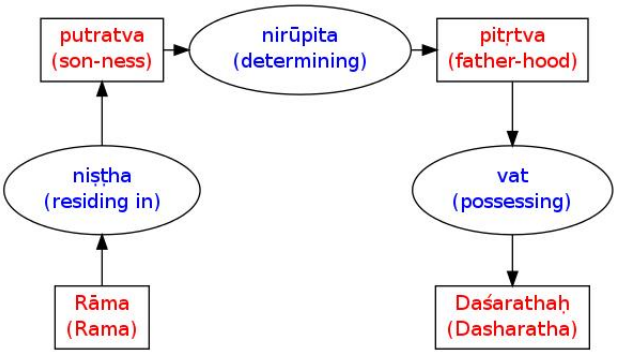

Figure 5.5: Conceptual Graph for (4)

the CG, we mark the position of the component in parenthesis. The modified CG for (4) is shown in Figure 5.6.

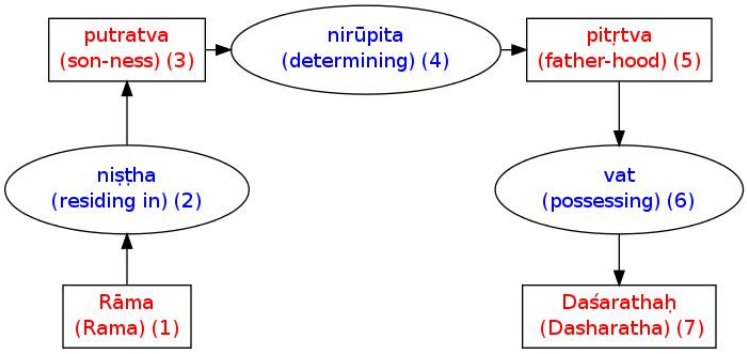

Figure 5.6: Conceptual Graph with position information for (4)

Consider now another sentence.

Sanskrit: *Rāmaḥ hastena brāhmaṇāya dhanam dadāti.* (5)

Gloss: Rama{nom.} hand{instr.} Brahmin{dat.} money{acc.} give{pres., active, 3sg}.

English: Rama gives money to a Brahmin with (his) hands.

The verbal cognition for this sentence according to the grammarian school is

Sanskrit: *rāma-niṣṭha-kartṛtva-nirūpaka-hasta-niṣṭha-karaṇatva-nirūpaka-brāhmaṇa-niṣṭha-sampradānatva-nirūpaka-dhana-niṣṭha-karmatva-nirūpaka-dānakriyā.* (6)

English: An activity of giving characterised by the agent-hood in Rama, the instrument-ness in the hand, the recipient-ness in a Brahmin and the object-hood in the money.

Figure 5.7 shows the rendering of this expression as a conceptual graph. The Nyāya school differs from the grammarian's school in

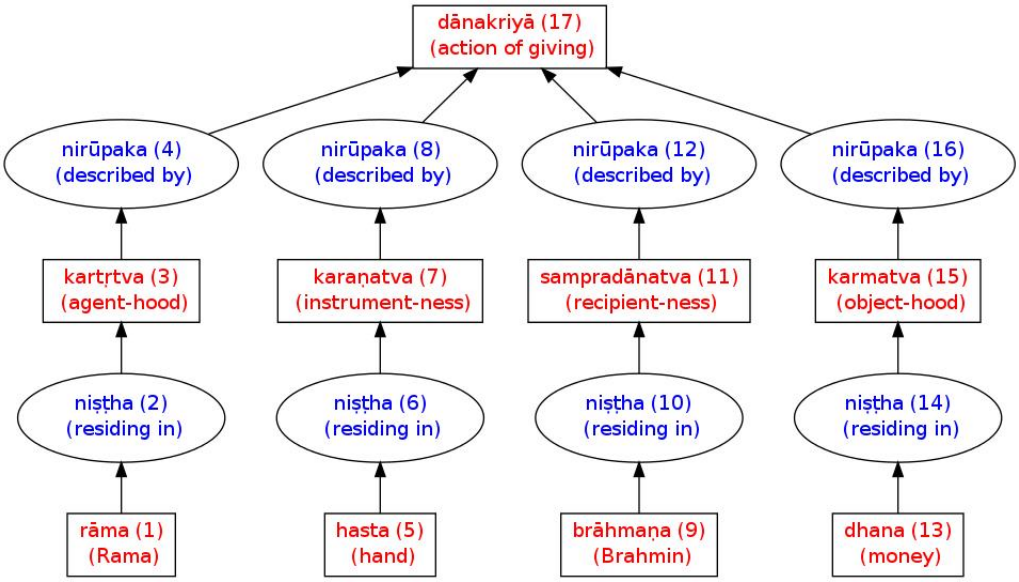

Figure 5.7: Conceptual Graph with position information for (6)

terms of the chief qualificand of this cognition. While for a grammarian the activity is the chief qualificand, for a logician the chief qualificand is the term in nominative case.

So the verbal cognition according to the Nyāya school is

Sanskrit: *hasta-niṣṭha-karaṇatva-nirūpaka-brāhmaṇa-niṣṭha-sampradānatva-nirūpaka-dhana-niṣṭha-karmatva-nirūpaka-dānakriyā-anukūla-kṛti-vat-rāmaḥ.* (7)

English: Rama has the agent-hood of an activity described by the instrument-ness in the hand, the recipient-ness in a Brahmin and the object-hood in the money.

Figure 5.8 shows the rendering of this expression as a conceptual graph.

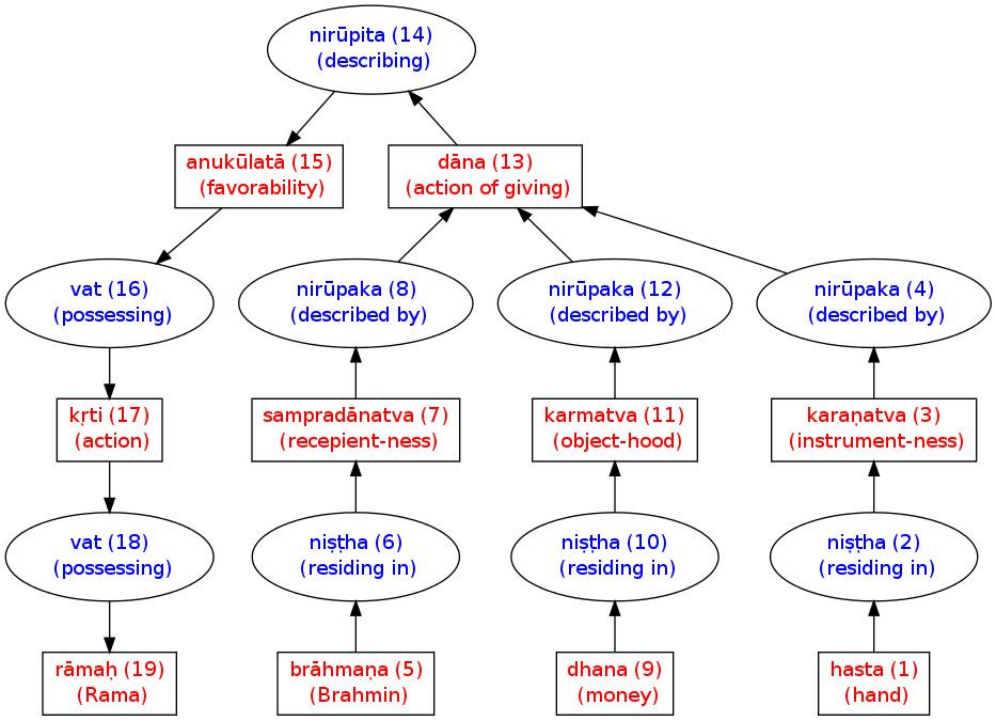

Figure 5.8: Conceptual Graph with position information for (7)

An NNE: *samavāyasambandha-avacchinna-gandhatva-avacchinna-gandha-niṣṭha-ādheyatā-nirūpita-adhikaraṇatā^vat-pṛthivī.* (8)

Now let us look at the expression (8) where NN Expression is used

to define the `Earth'. `Earth', according to the Indian school of ontology, is an object which has a characteristic property of having smell which differentiates it from the other objects. The NN Expression (8) expresses this precisely. In this definition the components *avacchinna* `delimited by', *niṣṭha* `resident in', *vat* `possessing' and *nirūpita* `determining' denote conceptual relations while the components *gandhatva* `smell-ness', *gandha* `smell', *adhikaraṇatā* `locus-hood' and *ādheyatā* `superstratum-ness' denote the concepts. The conceptual graph corresponding to this structure is shown in Figure - 5.9. The dotted lines show the ontological reality viz. that smell-ness is the inherent property of the smell and that the earth has smell as its characteristic property. The thick lines show the connection between the concepts through the conceptual relations expressed in the NN Expression (1). Note here the positions of various concepts in the diagram. Just like a *Naiyāyika*, who is a realist, we also tried our best to represent the hierarchy of concepts following the ontological status of them. This aspect of the diagram is very well prominent in the diagrams of Jha(16). We want to preserve this aspect. The NN Expressions are used to describe the situations or events as well, in addition to the cognitive structures. For example, the fact `a pot is on the ground' is described as

Sanskrit: *ghaṭa-niṣṭha-ādheyatā-nirūpita-adhikaraṇatā^vat bhūtalam.* (9)

English: The ground which has substratum-ness determined by the superstratum-ness in a pot.

where one cognises the situation with ground as the chief qualifi-

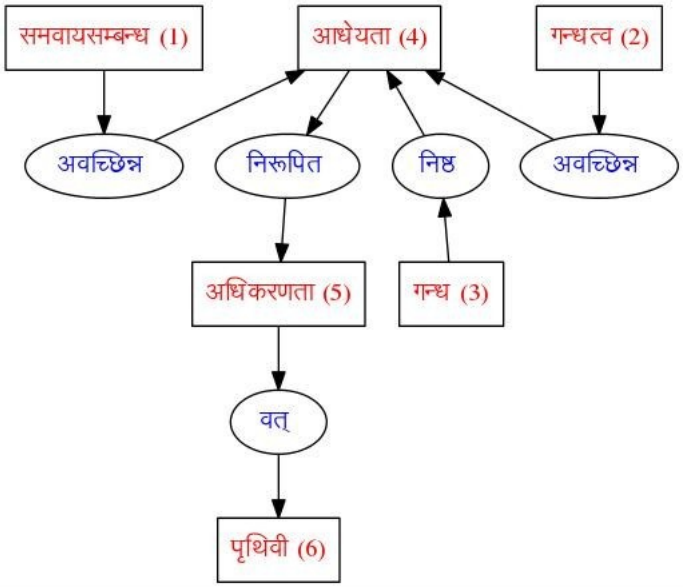

Figure 5.9: Conceptual Graph for (1)

cand in the cognition[2]. This is represented diagrammatically in Figure 5.10. On the other hand if one cognises it with the pot as the chief

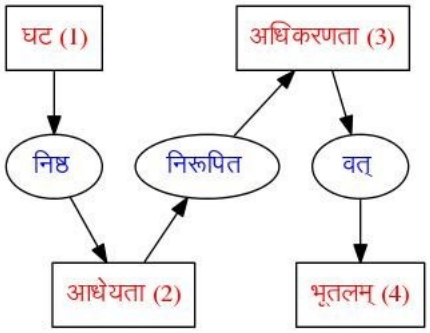

Figure 5.10: Conceptual graph corresponding to (9)

qualificand[3], then the cognition is described as

Sanskrit: *bhūtala-niṣṭha-adhikaraṇatā-nirūpita-ādheyatā^vān ghaṭaḥ.* (10)

[2] *ghaṭavadbhūtalam* 'pot-possessing-ground'.
[3] bhūtale ghaṭaḥ 'pot on the ground'

English: The pot which has superstratum-ness delimited by the substratum-ness resident in the ground.

Figure 5.11 gives the rendering of this expression as a Conceptual Graph. Note that the relations *vān* `possessing' and *niṣṭha* `resident in' are inverse of each other.

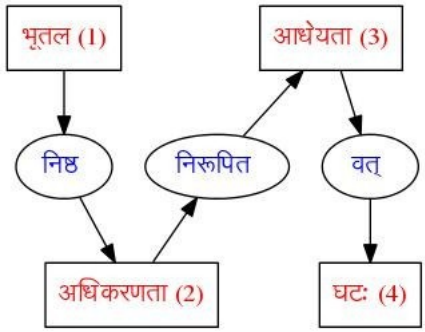

Figure 5.11: Conceptual graph corresponding to (10)

5.4 Compressed CGs

The CGs can also be drawn with relations represented by edges rather than the oval nodes. This way the graphs would be compact. The edges then will be labelled.

For instance, `Cat (is) on Mat' is represented as in Figure 5.12

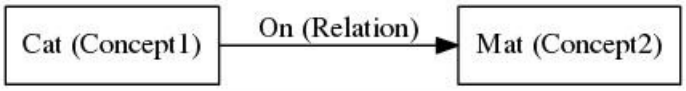

Figure 5.12: An instance of SCL graph

Figure 5.13 gives the rendering of the expression (10) as a compressed graph.

These graphs are very close to the representation of Jha(16) and Kulkarni(20). The only major difference being, while Jha and Kulka-

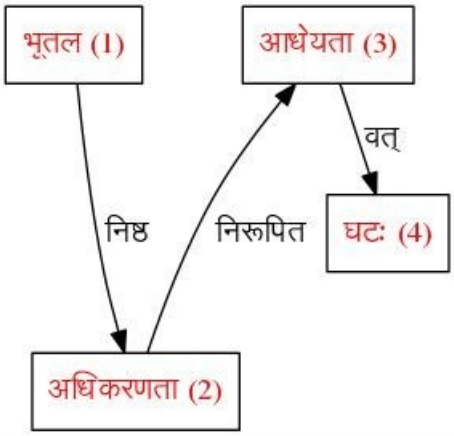

Figure 5.13: SCL graph corresponding to (10)

rni used edges with different heads to represent different types of relations, here we label the edges.

In what follows we now explain the Context Free Grammar that transforms a Constituency-Parsed NNEs into a CGs and Compressed CGs.

5.5 Grammar of NN Expressions

The constituency parsed NNE being a well-formed structure following the Context Free Grammar(CFG), we decided to write a CFG to parse this structure and then transform it into a CG. Now we define the grammar G for an NN Expression below.

G = (N,T,P,NNE), where

N: Set of non-terminal symbols viz. compound_concept, compound_relation, concept_term and relation_term.

T: Set of terminal symbols viz. relation and concept.

NNE: The start symbol.

P: Production rules described in Table-5.1. The concepts are the

NNE	:	compound_concept
	;	
compound_concept	:	'(' compound_relation '−' concept_term ')'
	;	
compound_relation	|	'(' concept_term '−' relation_term ')'
	;	
concept_term	:	concept
	|	NNE
	;	
relation_term	:	relation
	;	

Table 5.1: Production rules

nouns, relational abstract expressions and the negation functor and the terms derived with *tva* suffix from nouns. Relations are the sentence forming operators *niṣṭha* and *avacchinna* and the conditioning operator *nirūpita* along with their inverse relations viz. *vṛtti* / *āśraya*, *avacchedaka* and *nirūpaka*, respectively.

5.6 NN Expressions to Conceptual Graphs

In a CG, the concepts are represented by boxes, relations by the ovals, and the relations connect the concepts. In NN, except the *paryāpti* relation, which is used in connection with numbers, all other relations are binary. Thus every relation node needs two relata. Thus, in order to draw a CG corresponding to an NN relation, we need

1. Node labels,
2. Node types and
3. the two relata for every relation.

The attribute grammar defining the synthesized attributes for drawing the CG is defined in Table-5.2.

An attribute grammar is a context-free grammar extended with attributes, semantic rules and conditions. Synthesized attribute is an attribute that gets its values from the values of attributes of the children. It helps us in understanding the CFG and the attributes connected with that in a better way.

NNE	:	compound_concept ↑.head = ↓.head
	;	
compound_concept	:	'(' compound_relation '−' concept_term ')' ↑.head = concept_term.head establish a link between the head of the compound_relation to the head of the concept_term
	;	
compound_relation	|	'(' concept_term '−' relation_term ')' ↑.head = relation_term.head establish a relation between the head of the concept_term to the head of the relation_term
	;	
concept_term	:	concept ↑.head = ↓.position draw a concept node
	|	NNE ↑.head = ↓.head
	;	
relation_term	:	relation ↑.head = ↓.position draw a relation node
	;	

Table 5.2: Production rules with attributes

The node labels and the node types correspond to the **intrinsic** attributes of the terminal nodes *concept* and *relation*, which are avail-

able from the lexer. The two rules in the grammar above corresponding to *compound_relation* and *compound_concept* provide the links between a relation and a concept term.

5.6.1 An illustration

We work out an example illustrating the working of this grammar. Consider the fragment of the NNE

<<gandha-niṣṭha>-ādheyatā>

The constituency parse of this following the grammar in Table 5.2 is shown in Figure 5.14. Since no significant semantic action is associated with the concept_term node and at the nodes returned by the lexer, we collapse these nodes as in Figure 5.15 to make the graph more compact.

In order to generate a graph from this parse tree, we associate a `concept-structure' with each concept and a `relation-structure' with each relation. Figure 5.16 explains this.

Various stages in Parsing are shown in the Figure 5.17, 5.18, 5.19 and 5.20 below -

In the level of Fig - 5.17 with the given NNE shown through this parse tree, all the necessary information like *position*, *label* and where the relation term *niṣṭha* is connected to its left. But it is still not known where it connected to its right.

In the level of Fig - 5.18 shown through the parse tree, concept node acquires the `head' position from its child as shown in the highlighted part.

In the level of Fig - 5.19 shown through the parse tree, relation node inherits the `head' position as shown in the highlighted part.

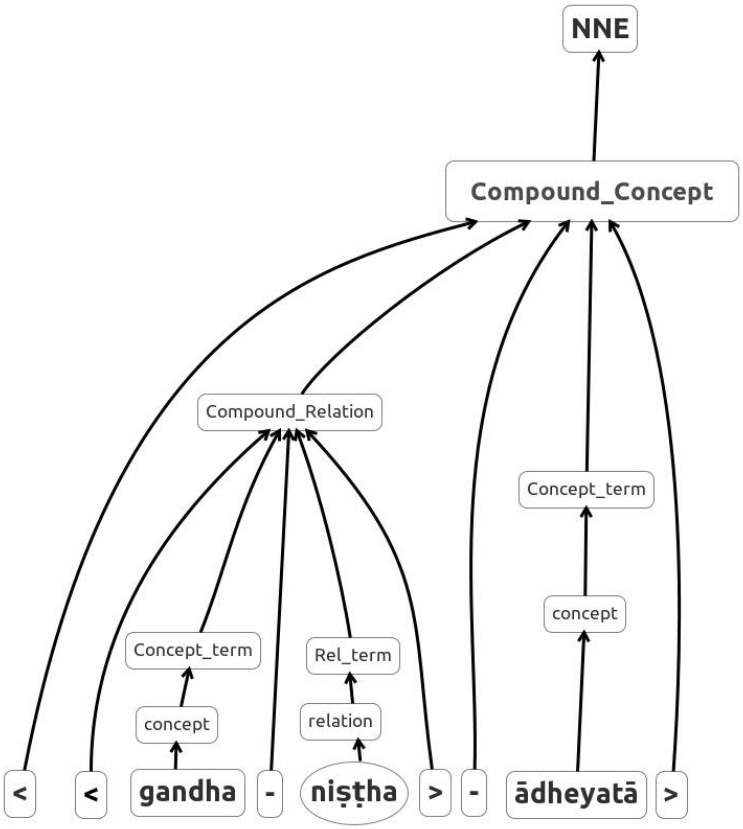

Figure 5.14: Constituency parse corresponding to the grammar

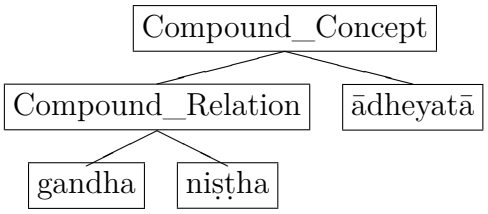

Figure 5.15: Compact parse - 1

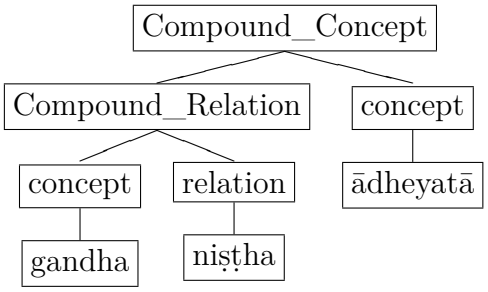

Figure 5.16: Compact parse with position information

$$\begin{bmatrix} \text{Position} = 1 \\ \text{Label} = \text{gandha} \end{bmatrix} \begin{bmatrix} \text{Position} = 2 \\ \text{Label} = \text{niṣṭha} \\ \text{Left} = 1 \\ \text{Right} = ? \end{bmatrix} \begin{bmatrix} \text{Position} = 3 \\ \text{Label} = \text{ādheyatā} \end{bmatrix}$$

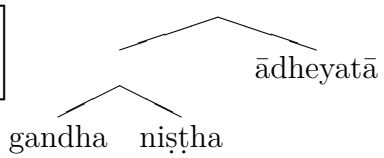

Figure 5.17: Compact parse

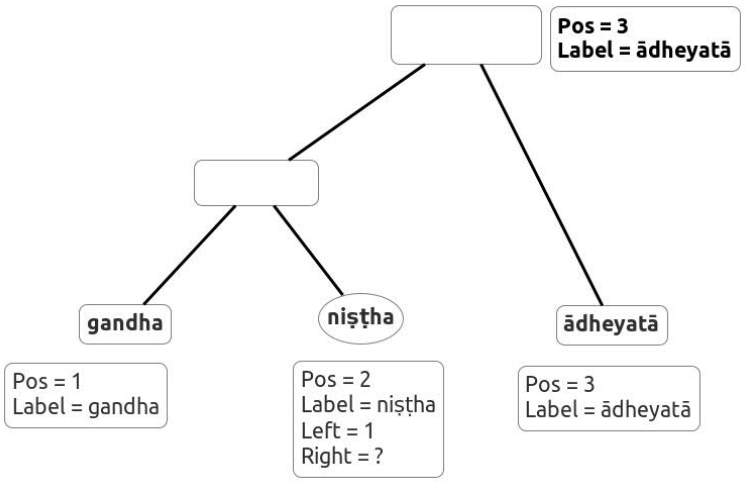

Figure 5.18: concept node acquires the 'head' position from child

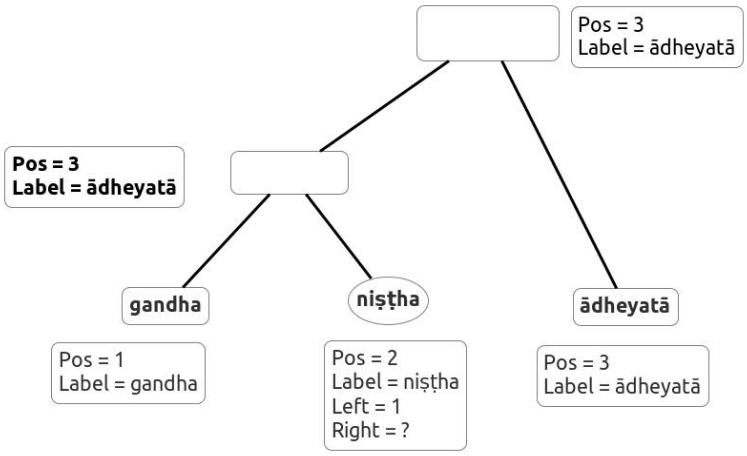

Figure 5.19: relation term inherits the 'head' position

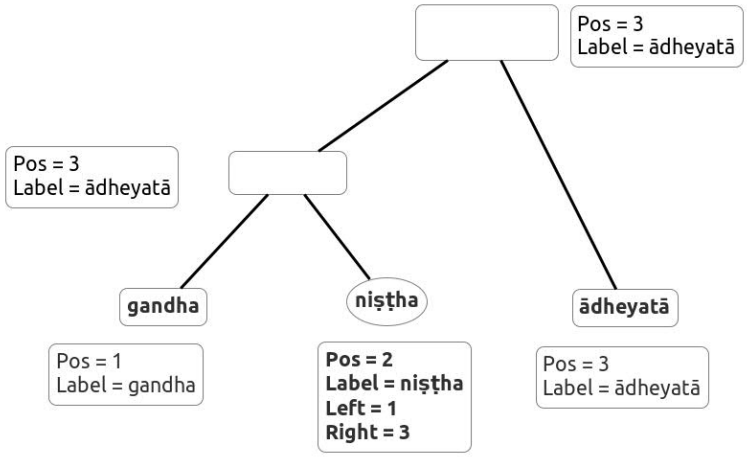

Figure 5.20: relation node inherits the position of 2^{nd} relata

In the level of Fig - 5.20 shown through the parse tree, the relation node inherits the position of 2^{nd} relata. It is now known that the relation term *niṣṭha* is connected to the concept term ādheyatā which is there to its right.

Once every relation node gets its right node position filled in, we draw the CG.

We noticed that NN has a canonical structure for expressing the cognitive structure as well as for describing the physical reality. This structure is used only when there is an ambiguity. Hence in a cognitive structure, only that part of the expression which is ambiguous is expanded. Thus typically the NN Expressions are heterogeneous mixtures of the expressions with unspecified relations and the NN Expressions. So there is a need to handle suchheterogeneous structures as well. For example, the expression *sādhyābhāvādhikaraṇanirūpita*vastu contains an NN term *nirūpita* and the remaining part of the expression is NN expression with unspecified relations with 3 components *sādhya*, *abhāva* and *adhikaraṇa*. The grammar in Table - 5.2, is extended further to handle these cases as well. In such cases, we establish a relation between the concept nodes, but leave the relation unspecified.

```
NNE          : compoundC
             ;
compoundC    : '<' compoundR '-' concept_term '>'
             | '<' concept_term '-' concept_term '>'
             ;
compoundR    : '<' concept_term '-' rel_term '>'
```

```
                ;
concept_term    : NNE
                | concept
                ;
rel_term        : relation
                ;
```

Now the graphical representation for the previous sentence following the extended grammar will be as shown in Figure - 5.21.

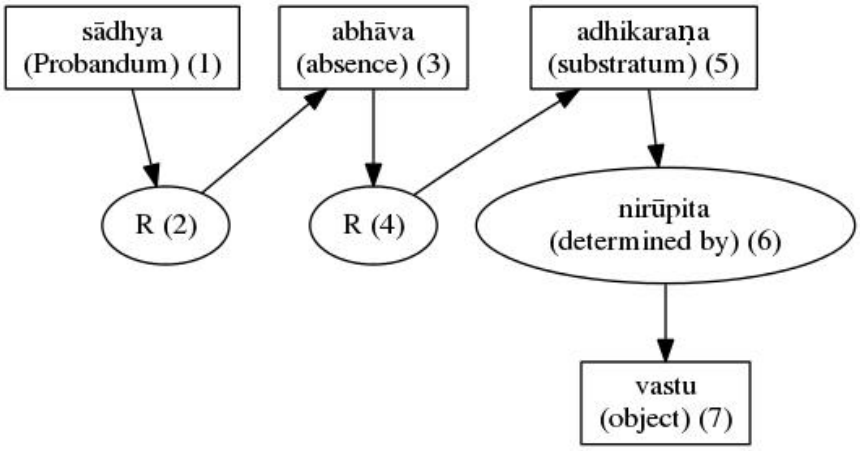

Figure 5.21: CG generated by modified grammar

Chapter 6

Nyāyacitradīpikā

All the modules discussed earlier have been packaged together in the form of a software *Nyāyacitradīpikā*. The system is available online at http://sanskrit.uohyd.ac.in/scl --> tools --> *Nyāyacitradīpikā*. The segmenter, constituency parser and the graph renderer are at the back end. The front end user interface has two options - one with pre-tested examples and the other one that allows any random NNE text. The interface first calls the segmenter and shows the segmented output. Next the user can choose to parse this segmented string. The interactive user interface facilitates user to select the correct *anuyogin* in the given context. Once all the *anuyogin*s have been selected, one can generate the CG or a compressed CG.
We illustrate this process with one example.

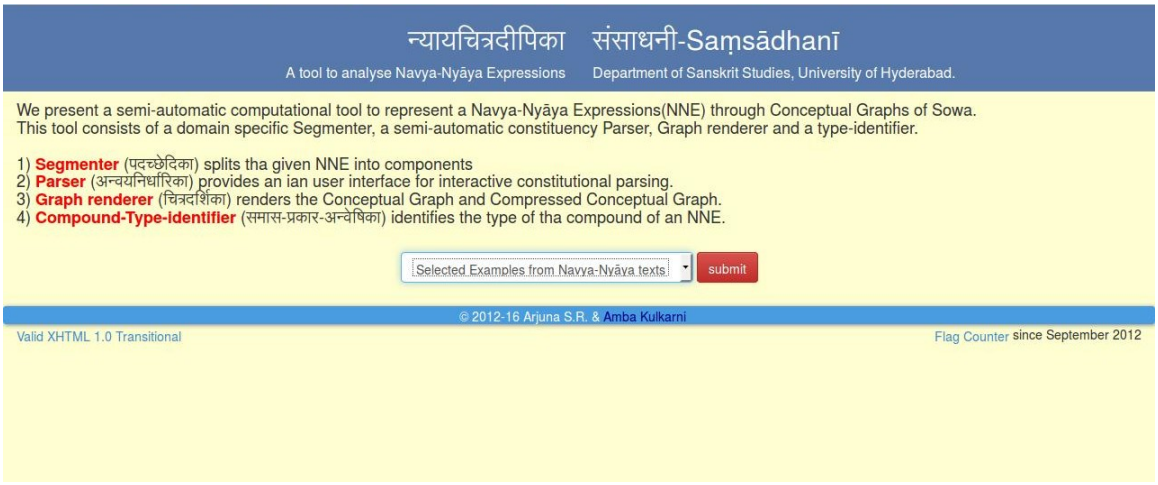

Figure 6.1: Homepage of *Nyāyacitradīpikā* with two modes

This is the interface of the *Nyāyacitradīpikā*. We can select either option from `selected examples from NN texts' and `my own example' and proceed further. We must choose an example in pre-tested examples and select a segmenter - SCL segmenter or Sanskrit Heritage Reader for segmentation.

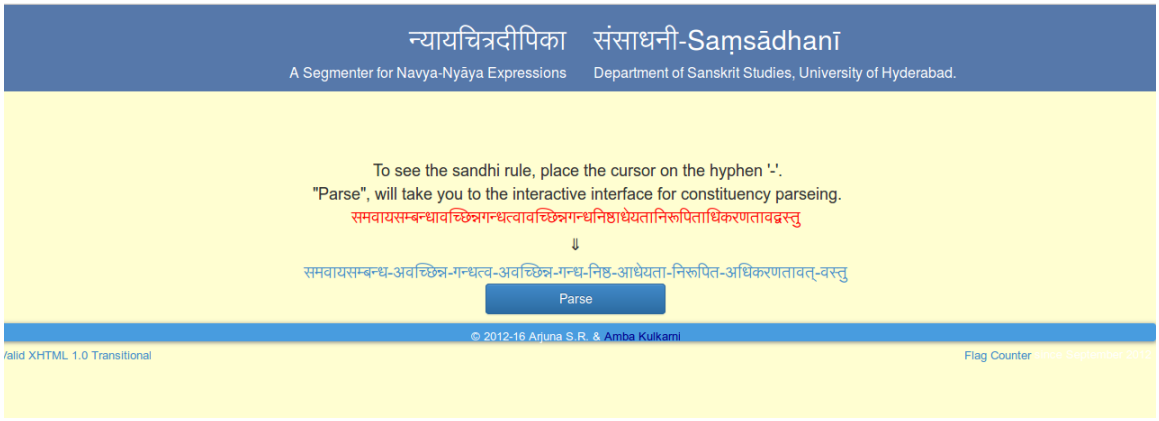

Figure 6.2: Segmented output from SCL segmenter

This interface shows the segmented output of *pṛthivī lakṣhaṇa* from SCL segmenter. We can proceed for the constituency parsing by clicking the `Parse' button.

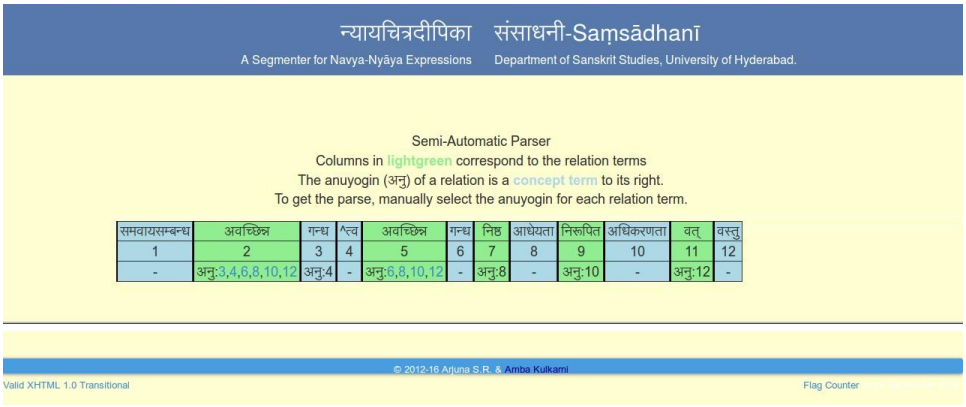

Figure 6.3: user interface to select *anuyogin*

This interface is to select the *anuyogin* of the relations. The interactive user interface facilitates user to select the correct *anuyogin* in the given context.

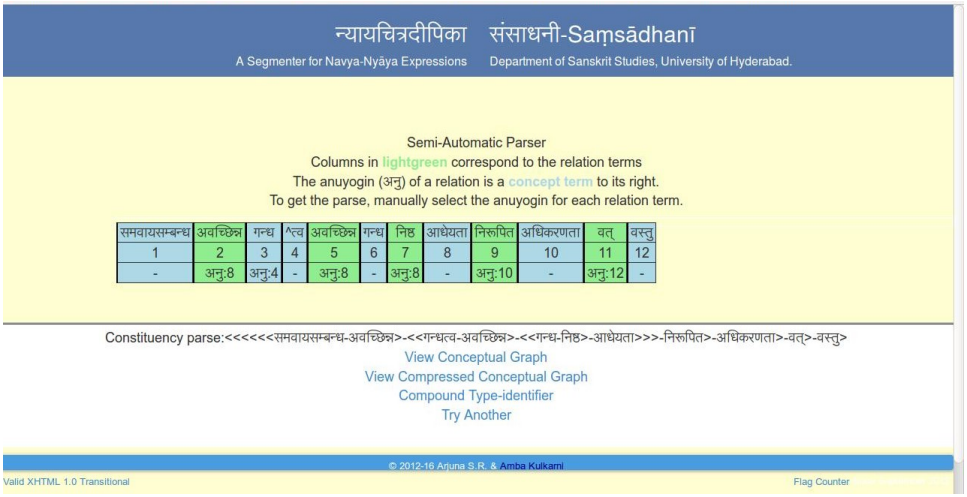

Figure 6.4: Completely disambiguated NNE

This interface is after choosing the *anuyogin*s of all the relations. We can proceed for Conceptual Graph by clicking on `View Conceptual Graph' and for Compressed Conceptual Graph by clicking `View Compressed Conceptual Graph'. We can proceed for Type-identifier also by clicking on `Compound Type-indentifier'.

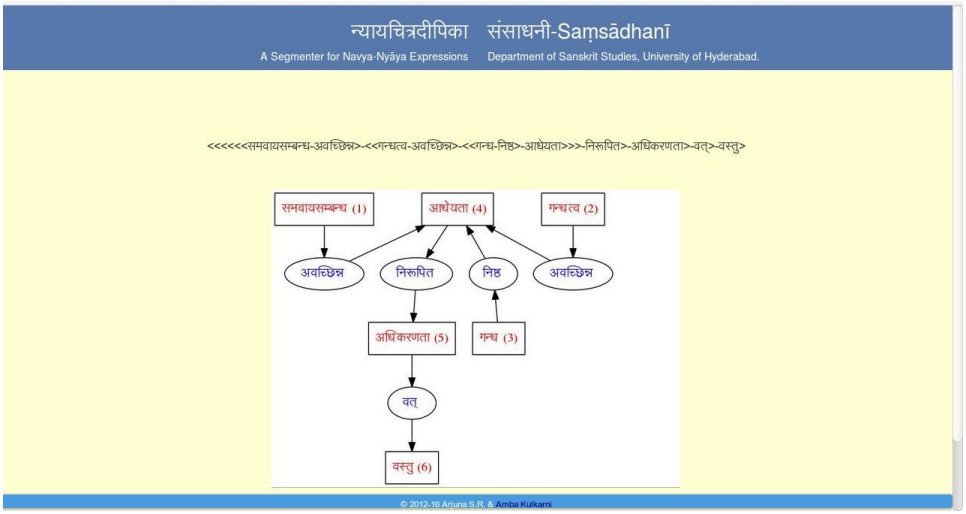

Figure 6.5: Conceptual Graph (CG) of the selected NNE

This interface shows the Conceptual Graph. We have to follow the number along with its relation to read the graph.

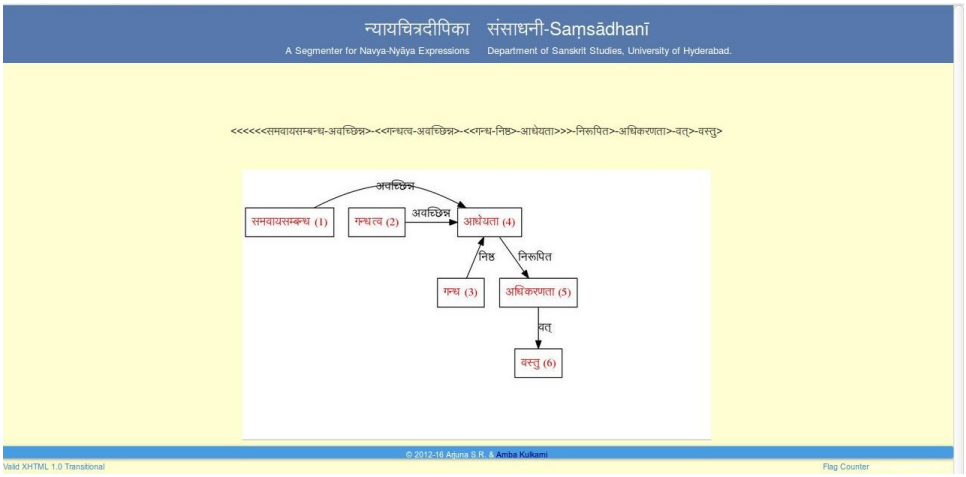

Figure 6.6: Compressed CG of the selected NNE

This interface shows the Compressed Conceptual Graph. Here also we have to follow the number to read the graph.

Figure 6.7: Identified compound types of the selected NNE

This interface shows the identified compound types of the selected NNE.

Chapter 7

Conclusion

This automatic conversion of NN Expressions into a Conceptual Graph is the first step towards understanding the complex and long NN Expressions. This rendering into a CG should ease the process of understanding the semantics associated with these expressions.

The contributions of this tool from the Sanskrit Studies perspective are,

- This will be a pedagogical tool for Sanskrit Studies.

  In the traditional learning the students are trained to conceptualise the NN expressions, when teachers explain the structure of the NNE. But the students not trained in gurukula system, since do not have good exposure to the traditional method of learning, find it difficult to visualise the underlying structure of NNEs. For such students this will be an aid to understand NNE effectively. Till now eminent teachers of NN such as Prof. Vasishtha Narayana Jha, Prof. Tirumala Kulakarni, to name a few, have been using the diagrams on board, to explain

the NNEs. But now with the availbility of these tools, students and teachers both will be benefited.

- For non-Sanskrit modern scholars, this will be a bridge to understand NN.

  The non-Sanskrit modern scholars from Language Science, Mathematics and Computer Science can easily understand the NN through this tool.

- Easier to understand Predicate Logic, because CGs can be converted into Predicate Logic.

  As mentioned in Chapter 5.2, a CG can be converted into Predicate Logic. This tool can convert an NNE to CG automatically from which one can convert it into Predicate Logic for further semantic processing.

Future directions

- Nyāyacitradīpikā has to be enhanced further for paraphrase generation.
- We can look forward to understand the semantics of the technical terms in order to build an inference engine.
- Constituency parsing is semi-automatic. We can look for more cues to make it fully automatic.

Appendices

A - Table of Semantic classifications

Semantic classifications of Sanskrit compounds					
अव्ययीभाव			तत्पुरुष		
1	अव्यय-पूर्वपद-अव्ययीभाव	A1	1	प्रथमातत्पुरुष	T1
2	अव्यय-उत्तरपद-अव्ययीभाव	A2	2	द्वितीयातत्पुरुष	T2
3	तिष्ठद्गुप्रभृति-अव्ययीभाव	A3	3	तृतीयातत्पुरुष	T3
4	संख्यापूर्वपद-नद्युत्तरपद-अव्ययीभाव	A4	4	चतुर्थीतत्पुरुष	T4
5	नद्युत्तरपद-अन्यपदार्थसंज्ञायाम्	A5	5	पञ्चमीतत्पुरुष	T5
6	संख्यापूर्वपद-वंश्योत्तरपद-अव्ययीभाव	A6	6	षष्ठीतत्पुरुष	T6
7	पारे-मध्ये-पूर्वपदषष्ठ्युत्तरपद-अव्ययीभाव	A7	7	सप्तमीतत्पुरुष	T7
बहुव्रीहि			8	नञ्-तत्पुरुष	Tn
1	द्वितीयार्थबहुव्रीहि	Bs2	9	प्रादि-तत्पुरुष	Tp
2	तृतीयार्थबहुव्रीहि	Bs3	10	कु-तत्पुरुष	Tk
3	चतुर्थ्यर्थबहुव्रीहि	Bs4	11	गति-तत्पुरुष	Tg
4	पञ्चम्यर्थबहुव्रीहि	Bs5	12	तद्धितार्थद्विगु	Tdt
5	षष्ठ्यर्थबहुव्रीहि	Bs6	13	उत्तरपदद्विगु	Tdu
6	सप्तम्यर्थबहुव्रीहि	Bs7	14	समाहारद्विगु	Tds
7	दिग्वाचक-बहुव्रीहि	Bsd	15	उपपद	U
8	संख्योभयपद-बहुव्रीहि	Bss	16	द्वितीयोपपद-तत्पुरुष	U2
9	उपमानपूर्वपद-बहुव्रीहि	Bsu	17	तृतीयोपपद-तत्पुरुष	U3
10	प्रहरणविषयक-बहुव्रीहि	Bsp	18	चतुर्थ्योपपद-तत्पुरुष	U4
11	ग्रहणविषयक-बहुव्रीहि	Bsg	19	पञ्चम्योपपद-तत्पुरुष	U5
12	सङ्ख्योत्तरपद-व्यधिकरण-बहुव्रीहि	Bvs	20	सप्तम्योपपद-तत्पुरुष	U7
13	सहपूर्वपद-व्यधिकरण-बहुव्रीहि	BvS	21	मयूरव्यंस्कादि	Tm
14	प्रादि-व्यधिकरण-बहुव्रीहि	Bvp	22	बहुपद-तत्पुरुष	Tb

15	उपमानपूर्वपद-व्यधिकरण-बहुव्रीहि	BvU	कर्मधारय		
16	नञ्-बहुव्रीहि	Bsmn	1	विशेषण-पूर्वपद-कर्मधारय	K1
17	बहुपद-बहुव्रीहि	Bb	2	विशेषण-उत्तरपद-कर्मधारय	K2
द्वन्द्व			3	विशेषण-उभयपद-कर्मधारय	K3
1	इतरेतर-द्वन्द्व	Di	4	उपमान-पूर्वपद-कर्मधारय	K4
2	समाहार-द्वन्द्व	Ds	5	उपमान-उत्तरपद-कर्मधारय	K5
3	एकशेष-द्वन्द्व	E	6	अवधारणापूर्वपद-कर्मधारय	K6
अन्य (others)			7	सम्भावनापूर्वपद-कर्मधारय	K7
1	द्विरुक्ति	d	8	मध्यमपदलोपिकर्मधारय	Km
2	केवल-समास	S	-		

B - Relation terms along with the possible type of the compound

Head of the first component	Compound type
निष्ठ	K1
वृत्ति	K1
निरूपित	K1
निरूपक	K1
अवच्छिन्न	K1
अवच्छेदक	K1
रहित	T6
एक	Tds
द्वि	Tds
त्रि	Tds
चतुर्	Tds
पञ्च	Tds
षट्	Tds
सप्त	Tds
अष्ट	Tds
नव	Tds
दश	Tds
शत	Tds
सहस्र	Tds
विध	K1
आत्मक	K1

उक्त	K1
जन्य	T5
अनुकूल	K1
आधारक	K1
अधिकरणक	K1
कार्यक	K1
कारणक	K1
विशेष्यक	K1
विशेषणक	K1
प्रकारक	K1
संसर्गक	K1
संसर्गिक	K1
विषयक	K1
विषयिक	K1
लक्ष्यक	K1
लक्षणक	K1
वृत्तिक	K1
सामान्यिक	K1
अभावीय	K1
अधिकरणीय	K1
प्रतियोगिताक	K1
अनुयोगिताक	K1
आधेयताक	K1
आधारताक	K1
अधिकरणताक	K1

कार्यताक	K1
कारणताक	K1
विशेष्यताक	K1
विशेषणताक	K1
प्रकारताक	K1
संसर्गताक	K1
संसर्गिताक	K1
विषयताक	K1
विषयिताक	K1
लक्ष्यताक	K1
लक्षणताक	K1
वृत्तिताक	K1
Head of the second component	Compound type
निष्ठ	T7
वृत्ति	T7
निरूपित	T3
निरूपक	T6
अवच्छिन्न	T3
अवच्छेदक	T6
अभाव	T6
भेद	T6
भिन्न	T5
रहित	T3
सामानाधिकरण्यम्	T6
आत्मक	Bs6

जन्य	T5
उक्त	T7
आदि	Bs6
सम्बन्ध	K4
स्वरूप	T6
पक्षक	Bs6
साध्यक	Bs6
हेतुक	Bs6
प्रतियोगिक	Bs6
अनुयोगिक	Bs6
आधेयक	Bs6
आधारक	Bs6
अधिकरणक	Bs6
कार्यक	Bs6
कारणक	Bs6
विशेष्यक	Bs6
विशेषणक	Bs6
प्रकारक	Bs6
संसर्गक	Bs6
संसर्गिक	Bs6
विषयक	Bs6
विषयिक	Bs6
लक्ष्यक	Bs6
लक्षणक	Bs6
वृत्तिक	Bs6

प्रतियोगिताक	Bs6
अनुयोगिताक	Bs6
आधेयताक	Bs6
आधारताक	Bs6
अधिकरणताक	Bs6
कार्यताक	Bs6
कारणताक	Bs6
विशेष्यताक	Bs6
विशेषणताक	Bs6
प्रकारताक	Bs6
संसर्गताक	Bs6
संसर्गिताक	Bs6
विषयताक	Bs6
विषयिताक	Bs6
लक्ष्यताक	Bs6
लक्षणताक	Bs6
वृत्तिताक	Bs6

Table 2: Relation terms along with the possible type of the compound

C - Programs of all the tools

We have enclosed all the programs here.

Segmenter program

The program of segmenter is written in Perl language.

```
#!/usr/bin/perl

my $myPATH="/home/arjun/scl";
use GDBM_File;
tie(%LEX1,GDBM_File,"$myPATH/NN/segmenter/S1.dbm",
GDBM_READER,0644) || die "can't open S1.dbm ";
tie(%LEX2,GDBM_File,"$myPATH/NN/segmenter/S2.dbm",
GDBM_READER,0644) || die "can't open S2.dbm ";
tie(%LEX3,GDBM_File,"$myPATH/NN/segmenter/S3.dbm",
GDBM_READER,0644) || die "can't open S3.dbm ";
tie(%LEX4,GDBM_File,"$myPATH/NN/segmenter/S4.dbm",
GDBM_READER,0644) || die "can't open S4.dbm ";

require "$myPATH/NN/segmenter/nyAya_words.pl";

$Max_Word_Size=25;

while($in = <STDIN>){
chomp($in);
$found = "";
%SPLIT = ();
```

```
%SPLIT_CHECKED = ();

($ans, $found)  = split("#", &split_recursive_sandhi($in,0,""));
$ans = &prioritise($ans);
@ans = split("/",$ans);
$ans_found = 0;
foreach $a (@ans) {
 print $a,"\n";
 $ans_found++;
}
if(!$ans_found) { print "No answer found \n";}
elsif($ans_found == 1) { print "One answer found \n";}
else { print "$ans_found answers found\n";}
}

sub split_recursive_sandhi{
 my($in,$absolute_position,$found) = @_;

my($wrd1,$wrd2,$i,$k,$final_ans,$ans,@t,$t,$len,
$pUrva,$str2Bmatched,$uttara,$fld1,$fld2,$fld3,
@ans,$local_found,$m,$position,
$local_ans,$lex,$tmp,$wrd2_position,$l);
$len = length($in);
  $final_ans = "";
    if($len < $Max_Word_Size) { $i = $len;}
    else {$i = $Max_Word_Size;}
```

```
 $max_window_size = 4;
 while($i>0) {

 $local_found = $found;

for($k=$max_window_size; $k>=1; $k--){
    if($i + $k < $len) {
 $local_ans = "";
 if($debug) {print "k = $k\n";}
 $in =~ /^(.{$i})(.{$k})(.*)$/;
 $pUrva = $1;
 $str2Bmatched = $2;
 $uttara = $3;
 $lex = "LEX".$k;

 if(${$lex}{$str2Bmatched}) {
    @ans = split(/#/,${$lex}{$str2Bmatched});
    for ($m = 0; $m <= $#ans; $m++ ){
       if($debug) { print "sandhi Rule: $ans[$m]\n";}
       ($fld1,$fld2,$fld3) = split(/,/,$ans[$m]);
        $tmp=$in;
    $tmp =~ s/^$pUrva$str2Bmatched$uttara$/$pUrva$fld1 $fld2$uttara/;
        ($wrd1, $wrd2) = split(/ /,$tmp);
    $position = $absolute_position+length($pUrva).",".$fld3;
        $wrd2_position = $absolute_position+length
        ($pUrva)+length($str2Bmatched)-length($fld2)+1;
```

```
   if((!&split_boundary_pos($position,$found)
   && !&from_non_pUrvapaxa_list($wrd2)
   && !&from_non_pUrvapaxa_list($wrd1))){
   if((length($wrd1) < $Max_Word_Size)
   && !$MO_CHECKED{$wrd1}){
    if($debug) {print "Calling morph for $wrd1\n";}
      $MO{$wrd1} = &get_morph_ana($wrd1);
      if($debug) {print "$MO{$wrd1}\n";}
      $MO_CHECKED{$wrd1} = 1;
   }
  if((length($wrd2) < $Max_Word_Size)
  && !$MO_CHECKED{$wrd2}){
    if($debug) {print "Calling morph for $wrd2\n";}
      $MO{$wrd2} = &get_morph_ana($wrd2);
      if($debug) {print "$MO{$wrd2}\n";}
      $MO_CHECKED{$wrd2} = 1;
  }
  if($MO{$wrd1} && $MO{$wrd2}) {

   $local_ans .= "/".$wrd1."{$fld3}".$wrd2;
   if($position ne "") {
$local_found = &add_position($local_found,$position);
   }
  }
  elsif($MO{$wrd1}) {

    if(!$SPLIT_CHECKED{$wrd2}) {
```

```
     ($ans,$sub_found) = split(/#/,
&split_recursive_sandhi($wrd2,$wrd2_position-1,$found));

 } elsif($SPLIT{$wrd2})
 { $ans = $SPLIT{$wrd2};}
 else {$ans = "";}

 if($ans ne "") {
  if($position ne "") {
     $local_found =
     &add_position($local_found,$position);
  }
  @sub_found = split(/#/,$sub_found);
  foreach $l (@sub_found) {
    if($l ne "") {
       $local_found =
       &add_position($local_found,$l);
    }
  }
   $ans =~ s/\/\//\//g;
   $ans =~ s/^\///g;
   @t = split("/",$ans);
   foreach $t (@t) {

      $local_ans .= "/".$wrd1."{$fld3}".$t;
   }
 }
```

```
          }
        }

      }
    }
   if($local_ans) {
      $final_ans .= "/".$local_ans;
   }
  }
  }

  $found = $local_found;
  $found =~ s/^#//g;
  $i--;

$SPLIT{$in} = $final_ans;
$SPLIT_CHECKED{$in} = 1;
$final_ans =~ s/\/\//\//g;
$final_ans =~ s/^\///g;

$final_ans."#".$found;
}
1;
sub get_morph_ana{
 my($word1) = @_;
 my($ans);
 system("$myPATH/NN/segmenter/client_splitter.sh
```

```
 $word1 | grep . | grep -v '\*'> /tmp/SKT_TEMP/tt");
 if(-s "/tmp/SKT_TEMP/tt") { $ans = 1;} else { $ans = 0;}
 system("rm /tmp/SKT_TEMP/tt");
return $ans;
}
1;
sub split_boundary_pos{
my($start,$found) = @_;
my(@found,$f);
@found = split(/#/,$found);
foreach $f (@found) {
 if($f eq $start) {
    return 1;}
}
return 0;
}
1;
sub add_position {
my($str,$pos) = @_;
     $str =~ s/\-/:/g;
     $pos =~ s/\-/:/g;
     $str =~ s/\+/;/g;
     $pos =~ s/\+/;/g;
     if(($str !~ /^$pos#/) &&
        ($str !~ /^$pos$/) &&
        ($str !~ /#$pos#/) &&
        ($str !~ /#$pos$/)){
```

```
          $str .= "#".$pos;
     }
          $str =~ s/:/-/g;
          $str =~ s/;/+/g;
$str;
}
1;
sub from_non_pUrvapaxa_list {
my($w) = @_;

if($w eq "ava-") { return 1;}
if($w eq "avacCinnA-") { return 1;}
if($w eq "anu-") { return 1;}
if($w eq "Cexa-") { return 1;}
if($w eq "Cexaka-") { return 1;}
if($w eq "CexakawA-") { return 1;}
if($w eq "Cexakawva-") { return 1;}
if($w eq "Cexya-") { return 1;}
if($w eq "CinnA-") { return 1;}
if($w eq "nirUpiwA-") { return 1;}
if($w eq "niRTA-") { return 1;}
if($w eq "nis-") { return 1;}
if($w eq "paryApwA-") { return 1;}
if($w eq "pra-") { return 1;}
if($w eq "prawiyogikA-") { return 1;}
if($w eq "rUpiwa-") { return 1;}
if($w eq "SAlI-") { return 1;}
```

```
if($w eq "Taka-") { return 1;}
if($w eq "TakA-") { return 1;}
# The viSeRaNa (BARiwa puMlinga, undergoes puMswva
in bahuvrIhi and karmaXAraya compounds
if($w eq "viSeRaNA-") { return 1;}
if($w =~ /^akawA/) { return 1;}
if($w =~ /^araN/) { return 1;}
if($w =~ /^avaw/) { return 1;}
if($w =~ /^AraN/) { return 1;}
if($w =~ /^Cinna/) { return 1;}
if($w =~ /^Cexaka/) { return 1;}
if($w =~ /^CexakI/) { return 1;}
if($w =~ /^iRyakawA/) { return 1;}
if($w =~ /^kawA/) { return 1;}
if($w =~ /^raN/) { return 1;}
return 0;
}

sub prioritise{
my($ans) = @_;
my($low,$high,@ans,$oneans,$word,@words,$count,$max_count,$i,@ANS);

@ans = split("/",$ans);
$max_count = 0;
foreach $oneans (@ans) {
 $curr_ans = $oneans;
 $oneans =~ s/[ ][a-zA-Z]+[ ]*;.*//;
```

```
 @words = split(/ /,$oneans);
 $count = 0;
 foreach $word (@words) {

   if($NYAYA_words{$word}) { $count++;}
 }
 $ANS[$count] .= "/". $curr_ans;

 if($max_count < $count) { $max_count = $count;}
}
$ans = "";
foreach ($i=$max_count; $i >= 0 ; $i--){
 if($ANS[$i] ne "") { $ans .= "/". $ANS[$i];}
}
$ans =~ s/\/\/+/\//g;
$ans =~ s/^\///;
$ans;
}
1;
```

Constituency Parser program

We have written the programs in Lex(Lexer generator) and Yacc(yet Another Compiler Compiler).

Lex program

```
%{
#include "nneparse.tab.h"
%}
%option noinput
%option nounput
%%
niRTa|vqwwi|avacCinna|nirUpiwa|nirUpaka|avacCexaka|
ASraya|SAl[iI]|vaw|vawI|vawyaH|vAn|ka{strcpy
(yylval.nodeinfo.token,yytext);return sambanXaH;}
[^-\n]+ {strcpy(yylval.nodeinfo.token,yytext);return concept;}
\n { return '\n';}
\-{ return '-';}
%%
```

Yacc program

```
%union {
struct node{
char token[1000];
} nodeinfo;
}
%{
#include <stdlib.h>
```

```
#include <stdio.h>
#include <string.h>

struct termstruct{
char word[1000];
char type[10];
char pratiyogin[10];
char anuyogin[100];
} terminfo[100];

int counter,i,relation_found;
char tmp[100];
extern int debug;

int yylex();
int yyerror();
%}

%token <nodeinfo> sambanXaH
%token <nodeinfo> concept

%type <nodeinfo> terms
%type <nodeinfo> TC
%type <nodeinfo> TR
%%
examples: example
        | examples example
```

```
        ;
example: terms '\n' {
        for(i=1;i<counter;i++){
                        printf("@%s\t",terminfo[i].type);
                         printf("%s\t",terminfo[i].word);
                                printf("%d\t",i);
               //if(!strcmp(terminfo[i].type,"relation")){
                  if(strcmp(terminfo[i].pratiyogin,"")){
                         printf("%s\t",terminfo[i].pratiyogin);
                                } else { printf("-\t");}
               //if(!strcmp(terminfo[i].type,"relation")){
                  if(strcmp(terminfo[i].anuyogin,"")){
                        printf("%s\n",terminfo[i].anuyogin+1);
                         // first char is ',', to ignore it,
                         we start printing from +1 position
          } else { printf("-\n");}
                                }
                   }
       ;
terms: terms '-' TR
     | terms '-' TC
     | TC
     ;
TC: concept { strcpy(terminfo[counter].word,$1.token);
strcpy(terminfo[counter].type,"concept");
sprintf(tmp,"%d",counter);
for(i=1;i<counter;i++){
```

```
    if(!strcmp(terminfo[i].type,"relation")){
        strcat(terminfo[i].anuyogin,",");
        strcat(terminfo[i].anuyogin,tmp);
    }
}
           if(!strcmp(terminfo[counter-1].type,"concept")) {
    sprintf(tmp,"%d",counter-1);
    strcat(terminfo[counter-1].pratiyogin,tmp);
               }
counter++;
     }
   ;
TR: sambanXaH {
    strcpy(terminfo[counter].word,$1.token);
    strcpy(terminfo[counter].type,"relation");
    sprintf(tmp,"%d",counter-1);
    strcat(terminfo[counter].pratiyogin,tmp);
    counter++;
    }
   ;
%%
#include <stdio.h>
#include <stdlib.h>

int debug;
int yyerror(char *s) {
        fprintf(stderr,"%s\n",s);
```

```
        return (0);
}

int main(int argc, char *argv[]){
  counter = 1;
  for(i=1;i<100;i++){
    strcpy(terminfo[i].anuyogin,"");
    strcpy(terminfo[i].pratiyogin,"");
  }
  debug = 0;
  if(argc > 1) {
     if(index(argv[1],'D')) debug=1;
  }
  yyparse();
  return 1;
}
```

Graph renderer program

This program is also written in Lex and Yacc.

Lex program

```
%{
#include "nne2diagram.tab.h"
int type;
%}
%option noinput
%option nounput
%x LEVEL
%%
niRTa {strcpy(yylval.nodeinfo.token,yytext);return sambanXaH;}
vqwwi {strcpy(yylval.nodeinfo.token,yytext);return sambanXaH;}
avacCinna {strcpy(yylval.nodeinfo.token,yytext);return sambanXaH;}
nirUpiwa {strcpy(yylval.nodeinfo.token,yytext);return sambanXaH;}
nirUpaka {strcpy(yylval.nodeinfo.token,yytext);return sambanXaH;}
avacCexaka {strcpy(yylval.nodeinfo.token,yytext);return sambanXaH;}
ASraya {strcpy(yylval.nodeinfo.token,yytext);return sambanXaH;}
SAl[iI]|vaw|vawI|vawyaH|vAn
{strcpy(yylval.nodeinfo.token,"vaw");return sambanXaH;}
ka {strcpy(yylval.nodeinfo.token,"ka");return sambanXaH;}
aBexa {strcpy(yylval.nodeinfo.token,"aBexa");return sambanXaH;}
[a-zA-Z]+wva
{strcpy(yylval.nodeinfo.token,yytext);yylval.nodeinfo.level=3;
return concept;}
[a-zA-Z]+wA
```

```
{strcpy(yylval.nodeinfo.token,yytext);yylval.nodeinfo.level=3;
return concept;}
[a-zA-Z]+
{strcpy(yylval.nodeinfo.token,yytext);yylval.nodeinfo.level=1;
return concept;}

niRTa:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,yytext); type=sambanXaH ; BEGIN LEVEL;}
vqwwi:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,yytext); type=sambanXaH ;BEGIN LEVEL;}
avacCinna:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,yytext); type=sambanXaH ;BEGIN LEVEL;}
nirUpiwa:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,yytext); type=sambanXaH ;BEGIN LEVEL;}
nirUpaka:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,yytext); type=sambanXaH ;BEGIN LEVEL;}
avacCexaka:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,yytext); type=sambanXaH ;BEGIN LEVEL;}
ASraya:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,yytext); type=sambanXaH;BEGIN LEVEL;}
```

```
SAl[iI]:|vaw:|vawI:|vawyaH:|vAn:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,"vaw"); type=sambanXaH ;BEGIN LEVEL;}
ka:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,"ka"); type=sambanXaH ;BEGIN LEVEL;}
aBexa:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,"aBexa"); type=sambanXaH ;BEGIN LEVEL;}

[a-zA-Z]+wva:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,yytext); type=concept; BEGIN LEVEL;}
[a-zA-Z]+wA:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,yytext); type=concept; BEGIN LEVEL;}
[a-zA-Z]+:/[0-9]
{yytext[yyleng-1] = '\0';
strcpy(yylval.nodeinfo.token,yytext); type=concept; BEGIN LEVEL;}
<LEVEL>[0-9]+
{yylval.nodeinfo.level=atoi(yytext); BEGIN INITIAL; return type;}
\n { return '\n';}
\<{ return '<';}
\>{ return '>';}
\-{ return '-';}
%%
```

Yacc program

```
%union {
struct node{
char token[1000];
int level;
int next_level;
int head;
} nodeinfo;
}
%{
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

struct termstruct{
char word[1000];
char type[10];
int level;
} terminfo[100];

struct relstruct{
 char relname[100];
 int from;
 int to;
 int level;
} relinfo[200];
```

```
int T_counter;
int rel_counter;
int i,j,k,found,min,max;
extern int debug, cg_or_real;
extern int getmin(), getmax();
extern char level[10][100];
int yylex();
int yyerror();
%}
%token <nodeinfo> sambanXaH
%token <nodeinfo> concept

%type <nodeinfo> NNE
%type <nodeinfo> compoundC
%type <nodeinfo> compoundR
%type <nodeinfo> concept_term
%type <nodeinfo> rel_term
%%
examples: example
        | examples example
        ;
example: NNE '\n' {
                                printf("@digraph @G\n{\n");
                            printf("@labelfloat=@true;\n");
                                max = 1;
       for(i=1;i<=T_counter;i++){
```

```
                                j = terminfo[i].level;
                                if (j > max) max = j;
                                sprintf(level[j],"%s
                                @Node%d",level[j],i);
                              }
    for(i=1;i<T_counter;i++){
                             printf("@Node%d\t[@label=\"%s
                            (%d)\" ",i,terminfo[i].word,i);
                               printf("@fontcolor=\"@red\"
                                  @shape = \"@box\" ]\n");
                              }
    for(i=1;i<rel_counter;i++){
if(cg_or_real == 2) {
                                 printf("@Node%d -> @Node%d
                           [@label=%s]\n",relinfo[i].from,
                         relinfo[i].to,relinfo[i].relname);
}
if(cg_or_real == 1) {
                            printf("@Node%da\t[@label=\"%s
                               \" ",i,relinfo[i].relname);
                              printf("@fontcolor=\"@blue\"
                                 @shape = \"@oval\" ]\n");
                           printf("@Node%d -> @Node%da \n",
                                       relinfo[i].from,i);
                           printf("@Node%da -> @Node%d \n",
                                       i,relinfo[i].to);
                                    j = relinfo[i].level;
```

```
                                        if (j > max) max = j;
                                    sprintf(level[j],"%s @Node%da",
                                              level[j],i);
    }
                                    }

        for(j=1;j<=max;j++){
                                printf("%d [@style=@invis]\n",j);
                                    }
                                    printf("{ ");
        for(j=1;j<max;j++){
                                    printf("%d ->",j);
                                    }
                                    printf("%d",j);
                                 printf(" [@style=@invis]\n}\n");
        for(j=1;j<=max;j++){
                                printf("{@rank=@same %d %s}\n",j,
                                       level[j]);
                                    }

           printf("@rankdir=@TB}\n");
        T_counter=1;
        rel_counter=1;

                                  }
        ;
NNE: compoundC {
```

```
                        $$.head = $1.head;
                        $$.level = $1.level;
}
   ;
compoundC: '<' compoundR '-' concept_term '>' {
  relinfo[rel_counter].from = $2.head;
  relinfo[rel_counter].to = $4.head;
strcpy(relinfo[rel_counter].relname,$2.token);
relinfo[rel_counter].level=$2.level;
  rel_counter++;

$$.head = $4.head;
$$.level = $4.level;
}
 | '<' concept_term '-' concept_term '>' {
  relinfo[rel_counter].from = $2.head;
  relinfo[rel_counter].to = $4.head;
                     strcpy(relinfo[rel_counter].relname,"@R");
  rel_counter++;

$$.head = $4.head;
$$.level = $4.level;
}
        ;
compoundR: '<' concept_term '-' rel_term '>'{
$$.head = $2.head;
strcpy($$.token,$4.token);
```

```
        $$.level = $4.level;
        }
        ;
concept_term: NNE {
 $$.head = $1.head;
 if(debug) { printf("NNE\n");
 fflush(stdout);}
}
    | concept {
 strcpy(terminfo[T_counter].word,$1.token);
 strcpy(terminfo[T_counter].type,"concept");
 terminfo[T_counter].level=$1.level;
         $$.level = terminfo[T_counter].level;
 $1.head = T_counter;
 T_counter++;

 $$.head = $1.head;

 if(debug) {printf("concept_term = %s
 pos = %d type = concept\n",
 $1.token,T_counter);fflush(stdout);}
}
    ;
rel_term: sambanXaH {
 strcpy($$.token,$1.token);
 $$.head = $1.head;
 $$.level = $1.level;
```

```
 if(debug) {printf("relation = %s
 pos = %d type = relation\n",
 $1.token,T_counter);fflush(stdout);}
}
    ;
%%
#include <stdio.h>
#include <stdlib.h>

int debug;
int cg_or_real;
char level[10][100];
int yyerror(char *s) {
        fprintf(stderr,"%s\n",s);
        return (0);
}
int main(int argc, char *argv[]){
  T_counter = 1;
  rel_counter = 1;
  min = 1;
  max = 3;
  debug = 0;
  cg_or_real = 2;
 /* By default it produces a graph that is close to reality.
   cg(C) = 1; real(R) = 2 */
 /* Usage: nne2diagram.out [DCR] */
```

```
  if(argc > 1) {
     if(index(argv[1],'D')) debug=1;
     if(index(argv[1],'C')) cg_or_real=1;
     if(index(argv[1],'R')) cg_or_real=2;
  }
  yyparse();
  return 1;
}
int getmin(int a,int b){
if(b > 0) {
  if(a > b) { a = b;}
}
return a;
}
int getmax(int a,int b){
if(b > 0) {
  if(a < b) { a = b;}
        }
return a;
}
```

Type-identifier programs

This program is also written in Lex and Yacc.

Lex program

```
%{
#include "typeidentifier.tab.h"
%}
%option nounput
%option noinput
%%
[a-zA-Z]+ {strcpy(yylval.padainfo.word,yytext);
return concept;}


\n { return '\n';}
\<{ return '<';}
\>{ return '>';}
\-{ return '-';}
%%
```

Yacc program

```
%union {
struct node{
char word[1000];
char head[100];
} padainfo;
}
```

```
%{
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char type[10];

int yylex();
int yyerror();
%}

%token <padainfo> niRTa
%token <padainfo> vqwwi
%token <padainfo> nirUpiwa
%token <padainfo> nirUpaka
%token <padainfo> avacCinna
%token <padainfo> avacCexaka
%token <padainfo> aBAva
%token <padainfo> concept

%type <padainfo> compound
%type <padainfo> Ppada
%type <padainfo> Upada
%type <padainfo> pada
%type <padainfo> example
%%
examples: example
        | examples example
```

```
        ;

example: compound '\n' { printf("%s \n",$1.word); }
       ;

compound : Ppada '-' Upada {
                                          strcpy(type,"@R");
 if(!strcmp($1.head,"niRTa")) strcpy(type,"@K1");
 if(!strcmp($1.head,"vqwwi")) strcpy(type,"@K1");
 if(!strcmp($1.head,"nirUpiwa")) strcpy(type,"@K1");
 if(!strcmp($1.head,"nirUpaka")) strcpy(type,"@K1");
 if(!strcmp($1.head,"avacCinna")) strcpy(type,"@K1");
 if(!strcmp($1.head,"avacCexaka")) strcpy(type,"@K1");
 if(!strcmp($1.head,"rahiwa")) strcpy(type,"@T6");
 if(!strcmp($1.head,"eka")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"xvi")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"wri")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"cawur")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"paFca")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"Rat")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"sapwa")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"aRta")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"nava")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"xaSa")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"Sawa")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"sahasra")) strcpy(type,"@Tds");
 if(!strcmp($1.head,"viXa")) strcpy(type,"@K1");
```

```
if(!strcmp($1.head,"Awmaka")) strcpy(type,"@K1");
if(!strcmp($1.head,"ukwa")) strcpy(type,"@K1");
if(!strcmp($1.head,"janya")) strcpy(type,"@T5");
if(!strcmp($1.head,"anukUla")) strcpy(type,"@K1");
if(!strcmp($1.head,"AXAraka")) strcpy(type,"@K1");
if(!strcmp($1.head,"aXikaraNaka")) strcpy(type,"@K1");
if(!strcmp($1.head,"kAryaka")) strcpy(type,"@K1");
if(!strcmp($1.head,"kAraNaka")) strcpy(type,"@K1");
if(!strcmp($1.head,"viSeRyaka")) strcpy(type,"@K1");
if(!strcmp($1.head,"viSeRaNaka")) strcpy(type,"@K1");
if(!strcmp($1.head,"prakAraka")) strcpy(type,"@K1");
if(!strcmp($1.head,"saMsargaka")) strcpy(type,"@K1");
if(!strcmp($1.head,"saMsargika")) strcpy(type,"@K1");
if(!strcmp($1.head,"viRayaka")) strcpy(type,"@K1");
if(!strcmp($1.head,"viRayika")) strcpy(type,"@K1");
if(!strcmp($1.head,"lakRyaka")) strcpy(type,"@K1");
if(!strcmp($1.head,"lakRaNaka")) strcpy(type,"@K1");
if(!strcmp($1.head,"vqwwika")) strcpy(type,"@K1");
if(!strcmp($1.head,"sAmAnyIya")) strcpy(type,"@K1");
if(!strcmp($1.head,"aBAvIya")) strcpy(type,"@K1");
if(!strcmp($1.head,"aXikaraNIya")) strcpy(type,"@K1");
if(!strcmp($1.head,"prawiyogiwAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"anuyogiwAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"AXeyawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"AXArawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"aXikaraNawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"kAryawAka")) strcpy(type,"@K1");
```

```
if(!strcmp($1.head,"kAraNawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"viSeRyawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"viSeRaNawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"prakArawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"saMsargawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"saMsargiwAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"viRayawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"viRayiwAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"lakRyawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"lakRaNawAka")) strcpy(type,"@K1");
if(!strcmp($1.head,"vqwwiwAka")) strcpy(type,"@K1");
```

```
if(!strcmp($3.head,"niRTa")) strcpy(type,"@T7");
if(!strcmp($3.head,"vqwwi")) strcpy(type,"@T7");
if(!strcmp($3.head,"nirUpiwa")) strcpy(type,"@T3");
if(!strcmp($3.head,"nirUpaka")) strcpy(type,"@T6");
if(!strcmp($3.head,"avacCinna")) strcpy(type,"@T3");
if(!strcmp($3.head,"avacCexaka")) strcpy(type,"@T6");
if(!strcmp($3.head,"aBAva")) strcpy(type,"@T6");
if(!strcmp($3.head,"Bexa")) strcpy(type,"@T6");
if(!strcmp($3.head,"Binna")) strcpy(type,"@T5");
if(!strcmp($3.head,"rahiwa")) strcpy(type,"@T3");
if(!strcmp($3.head,"sAmAnAXikaraNyam")) strcpy(type,"@T6");
if(!strcmp($3.head,"Awmaka")) strcpy(type,"@Bs6");
```

```
if(!strcmp($3.head,"janya")) strcpy(type,"@T5");
if(!strcmp($3.head,"ukwa")) strcpy(type,"@T7");
if(!strcmp($3.head,"Axi")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"sambanXa")) strcpy(type,"@K4");
if(!strcmp($3.head,"svarUpa")) strcpy(type,"@T6");
if(!strcmp($3.head,"pakRaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"sAXyaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"hewuka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"prawiyogika")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"anuyogika")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"AXeyaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"AXAraka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"aXikaraNaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"kAryaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"kAraNaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"viSeRyaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"viSeRaNaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"prakAraka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"saMsargaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"saMsargika")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"viRayaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"viRayika")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"lakRyaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"lakRaNaka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"vqwwika")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"prawiyogiwAka")) strcpy(type,"@Bs6");
if(!strcmp($3.head,"anuyogiwAka")) strcpy(type,"@Bs6");
```

```
 if(!strcmp($3.head,"AXeyawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"AXArawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"aXikaraNawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"kAryawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"kAraNawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"viSeRyawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"viSeRaNawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"prakArawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"saMsargawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"saMsargiwAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"viRayawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"viRayiwAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"lakRyawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"lakRaNawAka")) strcpy(type,"@Bs6");
 if(!strcmp($3.head,"vqwwiwAka")) strcpy(type,"@Bs6");

 sprintf($$.word,"%s-%s%s",$1.word,$3.word,type);
                                    strcpy($$.head,$3.head);
                                          }
        ;

Ppada : '<' pada  {sprintf($$.word,"<%s",$2.word);
strcpy($$.head,$2.head);}
;
Upada : pada '>'{sprintf($$.word,"%s>",$1.word);
strcpy($$.head,$1.head);}
;
```

```
pada : compound { strcpy($$.word,$1.word);
strcpy($$.head,$1.head);}
| concept { strcpy($$.word,$1.word);
strcpy($$.head,$1.word);}
;

%%
#include <stdio.h>
#include <stdlib.h>

int yyerror(char *s) {
        fprintf(stderr,"%s\n",s);
        return (0);
}

int main(int argc, char *argv[]){
    yyparse();
    return 1;
}
```

Bibliography

[1] S. R. Arjuna and G. Huet. Semi-automatic analysis of Navya-Nyāya compounds, SALA-30, Hyderabad, 2014.

[2] S. R. Arjuna and A. Kulkarni. Segmentation of Navya-Nyāya Expressions. In ICON-2014, 2014.

[3] A. H. Bhat. Pañcalakṣaṇī. Poornaprajna Samshodhana Mandiram, Bengaluru, 2004.

[4] R. Briggs. Knowledge Representation in Sanskrit and Artificial Intelligence. AI Magazine, 6(1):32--39, 1985.

[5] N. Chomsky. Three models for the description of Language. IRE Transactions on Information Theory, 1956.

[6] N. Chomsky. On Certain Formal Properties of Grammars. In Information and Control, pages 137--167, 1959.

[7] C. P. Dwivedi. Vyākaraṇabhūṣaṇasāraḥ of Kaunḍabhaṭṭa. Chowkambha Sanskrit Prathishthan, New Delhi, 2005.

[8] V. L. Frank Van Harmelen and B. Porter. Handbook of Knowledge Representation. Elsevier Science, 2007.

[9] J. Ganeri. Towards a formal regimentation of the Navya-Nyāya technical language-I. Logic, Navya-Nyāya & Applications, pages 109--124, 2008.

[10] G. Huet. Themes and Tasks in Old and Middle Indo-Aryan Linguistics, pages 307--325. Motilal Banarsidass, Delhi, 2006.

[11] G. Huet and P. Goyal. Design of a lean interface for Sanskrit corpus annotation. In D. M. Sharma, R. Sanghal, K. Kr.Arora, and B.K.Murthy, editors, Proceedings of ICON-2013, pages 177--186, 2013.

[12] M. D. Hyman. From Paninian Sandhi to Finite State Calculus. In Sanskrit Computational Linguistics, pages 253--265, 2008.

[13] D. H. H. Ingalls. Materials for the Study of Navya-Nyāya Logic. Harvard University Press, 1951.

[14] B. Jha and M. Jha. Rasagaṅgādhara of Panḍitarāja Jagannatha. Chaukambha Vidyabhavan, Varanasi, 1993.

[15] U. Jha, editor. A Primer of Navya-Nyāya Language and Methodology. Asiatic Society, Kolkata, 2004.

[16] V. N. Jha. Viṣayatāvāda of Harirāma Tarkālaṇkāra. University of Pune, Pune, 1987.

[17] V. N. Jha, editor. The Philosophy of Relations. Satguru Publications, 1990.

[18] V. N. Jha, editor. Dictionary of Nyāya Terms. University of Pune, Pune, 2001.

[19] Jīmūtavāhana. Dāyabhāgaḥ. Siddheśvar Press, Kolkata, 1893.

[20] A. Kulkarni. Navya-Nyāya for Scientists and Technologists: A first step, (mtech). Master's thesis, Indian Institute of Technology, Kanpur, 1994.

[21] A. Kulkarni and A. Kumar. Statistical constituency parser for Sanskrit compounds. In Proceedings of ICON 2011. Macmillan Advanced Research Series, Macmillan Publishers India Ltd., 2011.

[22] A. Kulkarni, S. Paul, M. Kulkarni, A. Kumar, and N. Surtani. Semantic Processing of Compounds in Indian Languages. In COLING-2012 Proceedings, pages 1489--1502, 2012.

[23] T. Kulkarni and J. Joshi. The language of Logic - Navyanyāya Perspectives. Manipal university Press, Manipal, India., 2013.

[24] A. Kumar. An Automatic Compound Processing. PhD thesis, Department of Sanskrit Studies, University of Hyderabad, 2012.

[25] A. Kumar and A. Kulkarni. Clues from Aṣṭādhyāyī for compound type. In M. Kulkarni and C. Dangarikar, editors, Recent Researches in Sanskrit Computational Linguistics Fifth International Symposium Proceedings, pages 62--83. D. K Printworld (P) Ltd, New Delhi, 2013.

[26] A. Kumar, V. Mittal, and A. Kulkarni. Sanskrit compound processor. In Proceedings of the 4th International Sanskrit Computational Linguistics Symposium, 2010.

[27] B. K. Matilal. The Navya-Nyaya Doctrine of Negation. Harvard University Press, Cambridge, Massachusetts, 1968.

[28] B. K. Matilal. Epistemology, Logic and Grammar in Indian Philosophical Analysis Ed. Jonardon Ganeri. Oxford University Press, 2005.

[29] G. A. Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. The Psychological review, 63:81--97, 1956.

[30] V. Mittal. Automatic Sanskrit Segmentizer Using Finite State Transducers. In ACL (Student Research Workshop), pages 85--90, 2010.

[31] J. N. Mohanty. Classical Indian Philosophy. Rowman and Littlefield Publishers, Inc., 2000.

[32] A. Natarajan and E. Charniak. S^3 - Statistical Sandhi Splitting. In IJCNLP, pages 301--308, 2011.

[33] D. Patil. Vidyādharī. Samarth media center, Pune, 2014.

[34] S. Polovina. An Introduction to Conceptual Graphs. In ICCS-2007, pages 1--14, 2007.

[35] D. Rao, editor. Tarkasaṃgraha with Śābdabodha and Nyāya-bodhinī. Poornapragna Vidyapeetha, 2015.

[36] S. Sastry. Pañcalakṣaṇīsarvasvam. Bharatiya Vidya Sansthan, Varanasi, 2005.

[37] P. Satuluri. Sanskrit Compound Generation: With Focus on the Order of Operations. PhD thesis, Department of Sanskrit Studies, University of Hyderabad, 2016.

[38] C. Shastri. Mīmāṃsākoustubhaḥ of Khanḍadeva. Chowkambha Sanskrit Series Office, Varanasi, 1991.

[39] J. L. Shaw. The Nyāya on Cognition and Negation. Journal of Indian Philosophy, Vol-8, Num-3, pages 279--302, 1980.

[40] B. Shukla. Māthurī Pañcalakṣaṇī. Rajasthan Hindi Granth Academy, Jaipur, 1984.

[41] B. Shukla. Navya-nyāya ke pāribhāṣika padārth. Paramarsha Pratishtan, Pune, 1998.

[42] J. F. Sowa. Conceptual Structures. Addison-Wesley, 1985.

[43] A. Upadhye. Nyāyāvatāra with Vivṛti commentary. Jaina Sahitya Vikasa Mandala, Bombay, 1971.

[44] Varadacharya. Tarkasaṅgraha with Āloka commentary. Arya Grantha Prakashan, Mysore, 2007.

[45] S. Varakhedi. Knowledge Representation schemes of Navya-Nyāya and other Western systems. PhD thesis, Poornaprajna Samshodhana Mandiram, Bengaluru, 2004.

[46] T. Wada. The Analytical Method of Navya-Nyāya. Journal of Indian Philosophy, 29:519--530, 2001.

[47] T. Wada. The Analytical Method of Navya-Nyāya. Egbert Forsten, Groningen, 2007.