TOWARDS AUTOMATING THE GENERATION OF DERIVATIVE NOUNS IN SANSKRIT BY SIMULATING PĀṆINI

Thesis submitted to
Indian Institute of Technology Kharagpur
for the partial fulfillment of the requirements
for the award of the degree of

Master of Technology
In
Computer Science & Engineering
by

Amrith Krishna
13CS60R12
M.Tech CSE

Under the guidance of
Prof Pawan Goyal

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

TOWARDS AUTOMATING THE GENERATION OF DERIVATIVE NOUNS IN SANSKRIT BY SIMULATING PAṆINI

Thesis submitted by

Amrith Krishna
13CS60R12
M.Tech CSE

Approved by

Prof Pawan Goyal
(Supervisor)

Prof Rajib Mall
(Head of the Department)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

Declaration

I, Amrith Krishna (13CS60R12) hereby declare that the thesis on "TOWARDS AUTOMATING THE GENERATION OF DERIVATIVE NOUNS IN SANSKRIT BY SIMULATING PĀṆINI" is a presentation of my original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgement of collaborative research and discussions. The work was done under the guidance of Dr. Pawan Goyal, at Indian institute of Technology, Kharagpur.

Amrith Krishna

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

Certificate

This is to certify that the project entitled "TOWARDS AUTOMATING THE GENERATION OF DERIVATIVE NOUNS IN SANSKRIT BY SIMULATING PĀṆINI" is a bonafide record of the work carried out by Mr. Amrith Krishna (Roll No.13CS60R12) under my supervision and guidance for the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science & Engineering during the academic session 2013-2015 in the Department of Computer Science & Engineering, Indian Institute of Technology, Kharagpur.

The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree.

Prof. Pawan Goyal

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Kharagpur, India 721302

Acknowledgment

I express my sincere gratitude to Dr. Pawan Goyal, Assistant Professor, Computer Science & Engineering Department, Indian Institute of Technology, Kharagpur for his supervision and guidance during the project. His valuable advices, encouragement, suggestions and friendly attitude towards me, during this work helped a long way in bringing this project report to this stage. I am also thankful to him for extending all kind of help and for providing the necessary facilities required during this work.

I am very much thankful to Prof. Rajib Mall, Head, Department of Computer Science & Engineering Department, IIT Kharagpur for providing necessary facilities during the research work.

I am thankful to Faculty Advisor Prof. Sudeshna Sarkar and all the faculty members of the department for their valuable suggestions, which helped me improve on this research work.

Amrith Krishna

Abstract

About 1115 rules in Aṣṭādhyāyī from A.4.1.76 to A.5.4.160 deal with generation of derivative nouns, making it one of the largest topical sections in Aṣṭādhyāyī, called as the Taddhita section owing to the head rule A.4.1.76. This section is a systematic arrangement of rules that enumerates various affixes that are used in the derivation under specific semantic relations. We propose a system that automates the process of generation of derivative nouns as per the rules in Aṣṭādhyāyī. The proposed system follows a completely object oriented approach, that models each rule as a class of its own and then groups them as rule groups. The rule groups are decided on the basis of selective grouping of rules by virtue of anuvṛtti. The grouping of rules results in an inheritance network of rules which is a directed acyclic graph. Every rule group has a head rule and the head rule notifies all the direct member rules of the group about the environment which contains all the details about data entities, participating in the derivation process. The system implements this mechanism using multilevel inheritance and observer design patterns. The system focuses not only on generation of the desired final form, but also on the correctness of sequence of rules applied to make sure that the derivation has taken place in strict adherence to Aṣṭādhyāyī. The proposed system's design allows to incorporate various conflict resolution methods mentioned in authentic texts and hence the effectiveness of those rules can be validated with the results from the system. We also present cases where we have checked the applicability of the system with the rules which are not specifically applicable to derivation of derivative nouns, in order to see the effectiveness of the proposed schema as a generic system for modeling Aṣṭādhyāyī.

Keywords: – Sanskrit Computational Linguistics, Paninian Studies, Astadhyayi.

Contents

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Objective . . . 3
- 1.3 Implementation Model . . . 4
- 1.4 Organization of the thesis . . . 4

2 Related Work 6
- 2.1 Introduction . . . 6
- 2.2 Linguistic Aspects of Aṣṭādhyāyi . . . 6
  - 2.2.1 Taddhita . . . 7
- 2.3 Automation of Aṣṭādhyāyi . . . 7
- 2.4 Conclusion . . . 8

3 Linguistic and Structural aspects of the Taddhita Section 10
- 3.1 Linguistic Phenomena in Derivational Morphology . . . 10
- 3.2 Organization of Taddhita rules . . . 12
- 3.3 Techniques of rule arrangement in Taddhita and Aṣṭādhyāyi . . . 14
  - 3.3.1 Anuvritti . . . 14
  - 3.3.2 Adhikāra . . . 15

4 System Implementation and the Derivation Process 16
- 4.1 Overview of the Implementation System . . . 16
  - 4.1.1 Tools and Techniques Used . . . 18
  - 4.1.2 Rule Triggering and Propagation . . . 19
  - 4.1.3 Śabdarūpa Representation . . . 23
- 4.2 Derivational Process along with the storage and propagation of linguistic features. 24
- 4.3 Derivation of a Derivative Noun . . . 25
- 4.4 Rule Selection and Conflict Resolution . . . 26

5 Evaluation Results 30
5.1 Evaluation Framework 30
5.2 Analysis of wrong cases and other special cases. 31

6 Discussion and Conclusion 34
6.1 The Schema as a General Schema for Modelling Aṣṭādhyāyī 34

List of Figures

1.1 Relation between a Taddhita nominal and its base nominal 2

3.1 An instance of inheritance hierarchy in Taddhita section 12
3.2 Anuvritti of 'it' section in Aṣṭādhyāyī 14

4.1 Overview of Implementation System 17
4.2 UML diagram for observer design pattern 18
4.3 UML diagram for multilevel inheritance 19
4.4 Triggering schema. 20
4.5 Affixation under patronymic relation for चटका 22
4.6 Representation of Environment object 24

5.1 Snapshot of the evaluation framework 31

List of Tables

3.1 Instances of affix polysemy, homonymy and synonymy in Sanskrit and English . 11

4.1 Derivation Process for औपगव . Here each horizontal line partition represents one step in the derivation, where one of the operations among insertion, elision or substitution takes place in column 2. Column 1 shows assignment of some technical term, and column 3 shows subsequent operation after gaining the technical term as a property. Column 2 shows the effect due to the operation. In column 2 we can see the effect of the rules on the environment, where an insertion, elision or substitution takes place. 29

5.1 My caption 32

Chapter 1

Introduction

Aṣṭādhyāyī, the central part of Pāṇini's grammar, is a classic and seminal work on descriptive linguistics. Aṣṭādhyāyī provided a complete description of the Sanskrit language spoken at that time period, and is also often praised for the computational principles and programming concepts used in it. Pāṇini's grammar for Sanskrit has Aṣṭādhyāyī as its central component, a collection of about 4000 rules along with other supplementary materials like Gaṇapata and Dhathupata which are lexical lists, Unadi sutras etc. Some of these materials are taken from other works on Sanskrit grammar that prevailed before his work. Aṣṭādhyāyī literally translates to 8 chapters, and as the translation goes Aṣṭādhyāyī has 8 chapters, each of which is divided into 4 quarters or pāda. Linguists across the world prize the brevity achieved by Aṣṭādhyāyī in its organization of rules, owing largely to tradition of passing information orally that prevailed in India at that point of time. But brevity has not come at the cost of completeness and is often praised for computational insights it provides, which amuses not only linguists but mathematicians and computer scientists across the globe.

Approximately one fourth (about 1115) of rules in Aṣṭādhyāyī deal with generation of derivative nouns (and adjectives), which are derived by affixation from other nouns (or adjectives). The affixes that are used for derivation of derivative nouns are enumerated in the Taddhita section of Aṣṭādhyāyī and hence the affixes that come under this section are called Taddhita. Taddhita section starts with rule A.4.1.76 **तद्धिताः** of Aṣṭādhyāyī which is an adhikāra rule and its influence is till the end of fifth chapter i.e. A.5.4.160 **निष्प्रवाणिः च**. Though Pāṇini did not provide any semantic definition for Taddhita, it is based on rule A.5.1.5 **तस्मै हितम्** which means "beneficial to that" (Bhate, 1989), where "that"(**तस्मै**) is intended for the base nominal from which the derivation will take place. Aṣṭādhyāyī considers both nouns and adjectives as a single category called as Prātipadika (**प्रातिपदिक**), and hence the affixes used in Taddhita are not category changing affixes (Deo, 2007).

1.1 Motivation

Aṣṭādhyāyī has received praises from domain experts in the fields of linguistics, mathematics and computer science, for the deep computational insights it carries. Rules of Aṣṭādhyāyī are often compared to a computer program for its rigor and coverage (Goyal et al., 2009). In fact, Hyman has shown that the individual rules of Pāṇini may be reframed such that a finite state transducer can be compiled, owing to the fact that the language that it generates will be a regular language (Hyman, 2009). Kiparsky, proved that the Pāṇinian formalism when taken as it is, is at least as powerful as a context sensitive language (Penn and Kiparsky, 2012). All these research substantiates the plausibility of automation of Aṣṭādhyāyī. In fact there have been various efforts in the form of formal frameworks, design schemas and implementations from the researchers in computational Sanskrit. Some of those works will be discussed in Chapter 2. But there have been no implementation efforts on automation of Taddhita section of Aṣṭādhyāyī. Generation of derivative nouns from any given valid nominal gives a person the power to expand the vocabulary of the language which helps him/her to express in the most concise way, what he/she has intended. With an automated generator of derivative nouns, the programmer can always keep a finite list of base nouns (both common and proper) along with the finite rules as stated in Aṣṭādhyāyī and then generate new nouns. By this the user is never limited by a finite set of vocabulary.

A derivative noun generator helps us to preserve etymological information of the nouns so generated. Given a generated nominal, we can find the base noun from which it has been derived and also the relation it holds with base noun also can be inferred. This helps to maintain relations between various words within the same language, and will be a valuable addition in the form of supplementary information in Sanskrit dictionaries and other lexical databases like Wordnet or IndoWordnet (Bhattacharyya, 2010) to be more specific. For e.g.: If someone needs to express a statement “Son of Upagu”, a new nominal can be derived as shown in Figure 1.1. Here the nominal **औपगव** is derived from the base nominal ‘**उपगु**’,and with the base it shares the relation `**अपत्यं**', which is the patronymic relation.

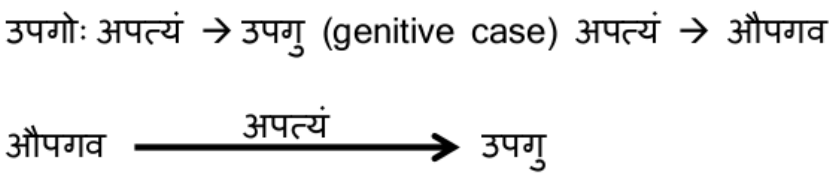

Figure 1.1: Relation between a Taddhita nominal and its base nominal

1.2 Objective

Though Aṣṭādhyāyī was intended for human understanding and usage, the work attempts to automate the process of deriving Taddhitas in complete adherence to Aṣṭādhyāyī. The work aims to generate the correct form of affixation for any given nominal by following the sequence of rules applied as per Aṣṭādhyāyī in the derivation process. In this thesis I propose an implementation model that helps in organising the rules of Aṣṭādhyāyī, such that the organisation leads to automated triggering of rules for proper affixation of nominals. Also in the process, Pāṇini uses a rich set of linguistic features varying from phonological features to semantic features for the derivation process. The system tries to keep record of all those features and store them as object attributes such that, on later implementations the feature infomration is not lost, and hence can be used for further derivations and analyses. This approach doubles as a pedagogy tool for the learners and enthusiasts of Sanskrit language and its grammar. Understanding of prakriya or the derivation process is very crucial in understanding of Sanskrit grammar. As the generator will be not only show the final form but also the sequence of rule triggered, a new learner can understand the various transformations that occur in the process for the derivation from base to final form.

In ancient India, teaching has been primarily done orally, where the disciples learn from their teacher through reciting of the sutras or rules. Aṣṭādhyāyi is no exception to it, but due to lack of a well-documented written form, some of its information that was assumed that a learner should possess as a prerequisite has been lost as time passed by from generations to generations. The unavailability of those information especially related to conflict resolution and blocking among rules of Aṣṭādhyāyi, led to academic debates among scholars, which has not yet converged to a consensus amongst them. The system proposed here is open to various conflict resolution techniques that have gained some ground over the period. We will be applying various conflict resolution methods and more importantly the schema of design allows adding many more conflict resolution techniques as per the desire of the programmer. Once a conflict resolution method is implemented, then the method's effectiveness may be validated through the results it produces and then its merits and demerits be compared with results so obtained from other conflict resolution methods as well.

The objectives of this thesis can be enumerated into the following three tasks:

- Automation of Taddhita section of Aṣṭādhyāyī, so as to simulate the affixation process of derivative nouns and to evaluate how the rule organisation helps to handle affix polysemy,homonymy and synonymy.
- Implementation of various methods for rule selection, conflict resoultion and blocking of rules and to check its effects in Taddhita section and in other general cases.

- Implementation of a model for storage of linguistic features that are obtained by the entities participating in derivation and how the same can be used for later derivations and analyses.

Hence the work requires to automate rules in Aṣṭādhyāyī that deal with Taddhita section as well as other associated rules, which help in the derivation process.

1.3 Implementation Model

The proposed system adopts a completely object oriented approach in modeling Aṣṭādhyāyī. The rules of Aṣṭādhyāyī are modeled as classes and so is the environment that contains the entities for derivation. The rules are then grouped based on the notion of topicality by virtue of anuvṛtti. In our proposed system the rule group formation is achieved through formation of inheritance network i.e. multilevel inheritance formed between individual rule classes, which has been inspired from the inheritance network that Pāṇini used in Aṣṭādhyāyī (Deo, 2007). Pāṇini uses anuvṛtti to carry forward the inherited components to child rules, though it needs to be noted that signifying of the inheritance is one of the aspects of anuvṛtti, and hence our proposed model does not form the inheritance network over all the usages of anuvṛtti, but rather a subset of it. The principles discussed in Aṣṭādhyāyī that enable rule selection and rule application like A.1.4.2 **विप्रतिषेधे परं कार्यम्**, siddha and asiddha are built right into the core architecture of proposed system. In addition to these principles, the system provides a functionality to adopt different conflict resolution methods that are discussed outside of Aṣṭādhyāyī. The system does not restrict itself by adopting any particular conflict resolution method, instead it facilitates to try out various methods that have been mentioned in various scholarly works. This will thus help to evaluate various methods and report on their accuracy as no single set of conflict resolution methods has gained consensus among the scholars.

1.4 Organization of the thesis

In Section 2, we will be discussing about various attempts towards formalizing rules of Aṣṭādhyāyi, and modeling Aṣṭādhyāyi in part or full to a automated system. In Section ??, we will look into the linguistic and structural features of Taddhita section. We will be discussing about various linguistic characteristics that the affixes in the domain possess. We will also be looking into the arrangement of rules in Taddhita section and how the arrangement forms an inheritance hierarchy. In Section ?? we will be describing the working of the proposed system, tools and techniques used for the implementation and also modeling of data environment and rule classes. Section 6.1 will show the applicability of the proposed schema as a general framework to model entire Aṣṭādhyāyi by considering how the proposed system handles rules that are not specific to the derivation of a derivative noun. The section will also talk about the various conflict resolution methods for rule selection which we have employed and its effect on some of the well known instances in conflict resolution (Cardona, 1965). Section ?? shows and

discusses about the results from the expert evaluation we have conducted. Finally, Section ?? discusses the salient features and application domains of the system along with directions for future research.

Chapter 2

Related Work

2.1 Introduction

Aṣṭādhyāyi has received much attention from computational linguists from the latter half of 20th century. Aṣṭādhyāyi was much lauded for its brevity, completeness and computational insights it provides. There have been works from eminent scholars about the formalism of Pāṇini's Grammar, its expressive power, and the derivation process (prakriyā) it follows. In Section 2.2 I will be discussing about the works on linguistic aspects and computational insights that Aṣṭādhyāyi provides. In the second Section 2.3 I will be looking into the various attempts in automating Aṣṭādhyāyi. The scetion will be concluded with Section 2.4 discussing about the how the work described in thesis differs from other works, and yet how these works have influenced the line of thought for the thesis.

2.2 Linguistic Aspects of Aṣṭādhyāyi

Seminal works from Cardona (1965) and Staal (1965) on formalizing rules of the grammar with stress on the meta-rules that state about the context sensitive aspects was a starting point with further enhancements from Cardona (1969) where he applies his formalization for more rules that is related to phonetic changes.

Cardona (2009) highlights the relevance of affixation and how it is well integrated with syntax as a continuum in Pāṇini's derivational system. He points out the contrast that Pāṇini's system bears with the system that western grammarians follow, where morphology and syntax are treated as independent components in derivation. Penn and Kiparsky (2012) focus on the expressive power of Aṣṭādhyāyi rules and also the expressive power of formalism that Pāṇini used to design Aṣṭādhyāyi. They demonstrate that the formalism of the grammar has far more

expressive power than that of regular languages and Context Free Languages. Their work emphasizes on the power of formalism that has built-in capacity for disambiguation at syntactic level.

2.2.1 Taddhita

Bhate (1989) talks about the organization of Taddhita rules specifically and goes into detail about the domain of influence of default affixes and then the sub-domain of semantic senses inside the affix domain. The work also discusses in detail about how the Taddhita section differs from other kind of noun derivations like the kṛdanta section which is, nouns derived from verbs and also from the samāsa which is, compound words formation. Deo (2007) in her work coins the approach constrained separatism approach, which she claims what Pāṇini has followed inTaddhita. She also states idea of networks of rules that helps in the derivation process.

2.3 Automation of Aṣṭādhyāyi

Hyman (2009) developed a finite state transducer after re-framing individual rules in Aṣṭādhyāyi, resulting in generation of strings that belong to regular languages and performs sandhi (सन्धि) at word boundaries for any two given word combinations. Hyman had introduced an XML vocabulary for encoding rules in Aṣṭādhyāyi that helped him in implementing the finite state transducer. Scharf (2009) discusses about Scharf's and Hyman's combined efforts in developing XML formalization that not only deals with sandhi but also with nominal declensions and verb conjugations.

There have been various attempts to automate Aṣṭādhyāyi in parts as well as modeling it entirely. Goyal et al. (2009) implemented an inflectional morphology generator that takes as input a noun from the user and then generates all 21 forms of noun declensions, known as vibhakti system in Sanskrit grammar. The authors talk about the programming perspectives that need to be considered when encoding rules in Aṣṭādhyāyi, and various computational aspects that Aṣṭādhyāyi possesses. They also talk about the need of conflict resolution methods for competing rules that can be applied in the same context. Jha et al. (2009) has developed a system that is an inflectional morphology analyser. They have developed independent systems for verb and noun forms and their corresponding inflections. Though their work was not related to simulating Aṣṭādhyāyi, but they claim that they take into account the Pāṇinian way of analysis. Satuluri and Kulkarni (2014) takes on generation of Sanskrit compounds called as samāsa that deals with about 400 rules of Aṣṭādhyāyi which helps them to form compound words from independent words in Sanskrit. Their work talks about various kinds of semantic features that act as the constraints governing compound formation. The Language Technologies Research Centre (LTRC) of IIIT Hyderabad has developed Dependency parsers for many Indian languages including Hindi based on the Pāṇini's Karaka system[14][15]. They

have used a modified version of Karaka system of Pāṇini to capture the free-word order property that some of the Indo-European languages possess (Bharati et al., 2008).

Subbanna and Varakhedi (2009) have talked in length about the Computational Structure of the Aṣṭādhyāyi, and introduced the concept of rules that continuously observe the environment or the subject to which modifications are to be made. They have also talked about Siddha,Assidhavat and Asiddha principles used in Aṣṭādhyāyi.There has been an in-depth study on siddha and asiddha principles by johshi and Roodbergen (1987), where they talk about the order in which rules need to be applied. Subbanna and Varakhedi (2009) mentioned about the grouping of rules based on the general-exception relation between rules and formation of rules as a tree structure, but commented that the feasibility of automation needs to be checked. They also talks about various conflict resolution methods that are mentioned in the sūtras as well as in other vārttikas. Subbanna and Varakhedi (2010) presented a computational model based on the principle of asiddhatva, an improvised model over the one discussed in Subbanna and Varakhedi (2009). Mishra (2009) talks of the nature of grammar which performs the analysis of constituent elements and then its reconstitution using various set of operational rules as mentioned in Aṣṭādhyāyi. Mishra (2010) in his work discusses about vedāṅga principles and extends his work by considering the common methodological approach of ancillary disciplines for rule application. His work provides a good walk-through for the entire derivation process that begins with introduction of atomic elements to coming up with the desired final form. Kulkarni (2009) establishes the issue with phonological over-generation that can occur, if one is to strictly adhere to the rules defined by Pāṇini. Jha and Mishra (2009) sheds some light in formalizing semantic categorization rules when he deals with kāraka systems. These kind of issues have been a matter of debate among linguists for quite long. Many principles that are not stated in Aṣṭādhyāyī but in other texts written at various points of time have surfaced to deal with such issues, mainly those which concern about conflict resolution in rule selection. Cardona (1997) discusses about various principles like utsarga-apāvada, antaraṅga-bahiraṅga, nitya-anitya etc. in detail.

2.4 Conclusion

To the best of authors' knowledge, the system which we are going to propose is the first of its kind, that is focused specifically on generation of derivative nouns. The proposed system embraces a unique approach of forming rule groups where similar rules are grouped together to form a Directed Acyclic Graph (DAG). The similarity of rules is based on the notion of topicality present among the rules by virtue of usage of anuvṛtti. The approach can be treated as analogous to the model, what is proposed in Subbanna and Varakhedi (2009), but they propose the formation of similar topic DAG through utsarga-apāvada relations. One cannot comment on the similarity of the approaches without a proper comparison of DAGs formed from both them.

Moreover Subbanna and Varakhedi (2009) mentioned that they had not checked the feasibility of automating their notion of rule group formation. Another important feature that our system uses is that it automatically notifies the relevant rule classes whenever the data environment state changes. This eliminates the overhead of linear searching over each rule, or each rule polling the environment to find its application. Instead the mechanism allows the rules to be updated about environment state changes whenever there is one and yet the rule classes can refrain from state dependency issues with the environment (Szallies, 1997).

Chapter 3

Linguistic and Structural aspects of the Taddhita Section

In the Taddhita section, Pāṇini identifies about 300 semantic relations under which Prātipadikas can be generated with Taddhita affixes. It is mentioned as a sub-section to pratyayādhikāra that deals with all kinds of affixes. The rules in Taddhita section deal with three entities namely semantic relations, affixes and stems or collection of stems from gaṇapāṭha. The rules are defined in such a way so as to facilitate affixation of the proper Taddhita affix with respect to semantic relation intended. The rules often deal with properties of the entities involved at various levels from phonological, morphological, syntactic and semantic levels. There are two types of derivations possible that involve Taddhita affixes (Sharma, 2002).

Prātipadika + Taddhita-affix
Prātipadika + Taddhita-affix + strī-affix

3.1 Linguistic Phenomena in Derivational Morphology

Derivational morphology in Sanskrit, like in many other languages poses some of the well known facts about many to many correspondences between forms and affixes. Taddhita affixes exhibit affix polysemy, homonymy, synonymy and non-compositionality. . Considering a single affix and multiple senses we can discuss about affix polysemy and homonymy and with multiple affix and same semantic relation we can talk about affix synonymy. In fact, in Sanskrit non-compositionality of the taddhita affixes are also well discussed . In affix polysemy, the same affix is used to denote related senses like in the case of patronymic and provenance relation. In affix homonymy the same affix is used in distinct and unrelated semantic contexts like in the case of personal nouns or abstract nouns. Affix synonymy deals with the same semantic sense but

uses different affixes. For example, for the patronymic relation **अपत्यम्**, multiple affixes can be used, like **अ(ण्), अयन(फक्), इ(ञ्)** depending on the stems used (Deo, 2007). Table 3.1 shows some instances of each of these phenomena, with analogy to english language.

Affix Homonymy			
Sanskrit (affix –)		English (affix -ian)	
Base	Base	Base	Base
Library Derived	Library Derived	Library Derived	Library Derived
Librarian	Librarian	Librarian	Librarian
Affix Homonymy			
Affix Homonymy			

Table 3.1: Instances of affix polysemy, homonymy and synonymy in Sanskrit and English

3.2 Organization of Taddhita rules

Taddhita section is primarily subdivided into five pratyayādhikāras or domain of control of five pratyayas.

- The five rules are:

  I A.4.1.83 **प्राक् दीव्यतः अण्** - **अण्** suffix is the default affix to be used for affixation for all the rules till A.4.4.1. The influence of A.4.1.83 is till the term **दीव्यति** is found (or till another pratyayādhikāra is found) and **दीव्यति** appears in rule A.4.4.2 **तेन दीव्यति खनति जयति जितम्**.

  II A.4.4.1 **प्राक् वहतेः ठक् ।** - **ठक्** suffix is the default affix to be used for affixation for all the rules till A.4.4.76. **वहति** appears in rule A.4.4.76 **तत् वहति रथयुगप्रासङ्गम्**.

  III 4.4.75 **प्राक् हितात् यत्** - **यत्** is the default affix to be used for affixation for all the rules till A.5.1.5.**हितम्** appears in rule 5.1.5 **तस्मै हितम्**.

  IV 5.1.1 **प्राक् क्रीतात् छः** - **छः** is the default affix to be used for affixation for all the rules till A.5.1.37 **क्रीतम्** appears in rule 5.1.37 **तेन क्रीतम्**.

  V 5.1.18 **प्राग् वतेः ठञ्** - **ठञ्** is the default affix to be used for affixation for all the rules till A.5.1.115. **वतिः** appears in rule 5.1.115 **तेन तुल्यं क्रिया चेत् वतिः** .

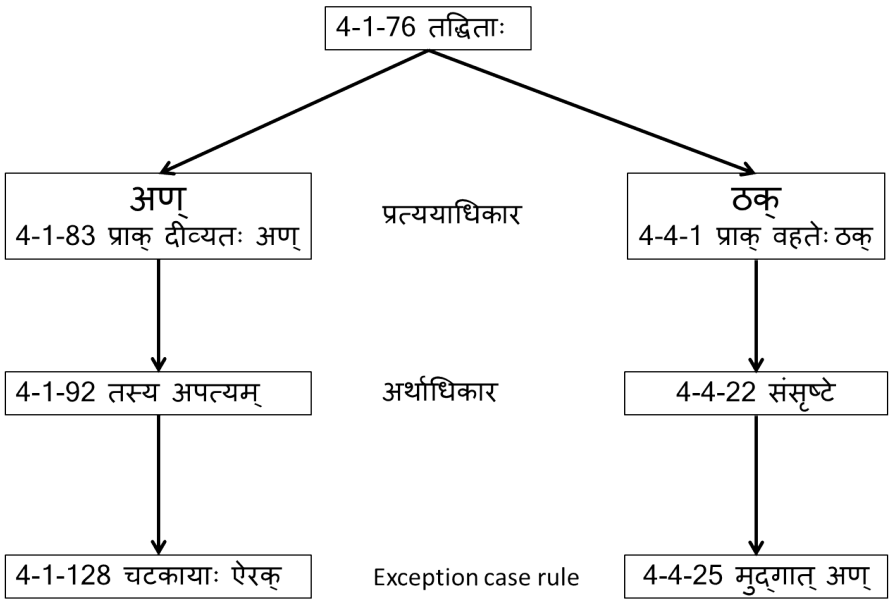

Figure 3.1: An instance of inheritance hierarchy in Taddhita section

Now each of the above given rules, which can be called as the pratyayādhikāra rules, specifies what is the most prominent affix or the default case affix that can be applied to all the

rules which are under its domain. For the set of rules under a single pratyayādhikāra, they are further categorized on the basis of arthādhikāra rules. Each arthādhikāra heads a set of rules which are a proper subset of rules that come under the pratyayādhikāra. The arthādhikāra rules state the semantic conditions or the semantic rules under which the default affix can be attached. Arthādhikāra rules serve the purpose of a topic head that groups a set of rules as a rule group based on a topic, which is a semantic relation here and it also acts as an operational rule (**विधि**), as it carries the semantic sense under which the affix can be attached. Rules A.4.1.92 **तस्य अपत्यम्**, A.4.2.1 **तेन रक्तं रागात्**, A.4.2.69 **तस्य निवासः** etc. denote the semantic senses such as patronymic, coloured by means of that and provenance respectively. This kind of arrangement handles affix polysemy and affix homonymy. But to handle affix synonymy, there are other operational rules, that come under the domain of arthādhikāra rules. They can be seen as exception rules or rules to handle special cases. These rules limit the application of default affix specified in pratyayādhikāra and mention what other affixes can be used instead, under special conditions. In this way, affix synonymy and non-compositionality is taken care of. Deo (2007) calls this form of arrangement as constrained seperationism, where the rules form a multilevel single inheritance network. The template for the multilevel inheritance network will be of the type, "Default Affix rule" -"Semantic Sense rule" - "Special case rules". Figure 3.1 shows an instance of how the hierarchy in Taddhita works. As already discussed, A.4.1.83 states about the the default affix **अण्** for rules under its domain, A.4.1.92 states about the patronymic relation under which a nominal will get the default affix. Now rule A.4.1.128 states about a special case when the stem **चटका** is used with the patronymic relation. In such a case the suffix **ऐरक** needs to be attached. This rule overrides the default case rule. Now consider rule A.4.4.25. This rule talks about usage of **अण्** as an exception or special case when a stem **मुद्ग** is used. The rule has A.4.4.1 as its default affix rule and the default affix for the domain is **ठक्**. The rule's semantic sense is **संसृष्टे**, which means 'properly mixed with'. Here the rule A.4.4.25 accounts for the non-compositionality of the affix **अण्** and is not specified in its domain but as an exception rule in another default affix domain. In this way Pāṇini is able to achieve constrained separationism (Deo, 2007) which is an alternative to both the separationist and non-separationist viewpoints in the case of derivational morphology. Here Pāṇini has employed a multilevel but single inheritance hierarchy through pratyayadhikara and arhtadhikara, where arthadhikara inherits from only one pratyaya, and affix synonymy that may occur is being handled through exception handling rules.

It is to be noted that there are some more rules in the Taddhita section which do not come under any of the five pratyayadhikāra rules. Those rules have arthādhikāras of their own but has no default affix to be attached. This group of rules is treated as extraneous to otherwise systematic network of the section (Bhate, 1989). As we will be discussing about the implementation schema of proposed generation system in the subsequent section, it will become evident that this anomaly in no way is going to affect the system.

3.3 Techniques of rule arrangement in Taddhita and Aṣṭādhyāyi

Aṣṭādhyāyi is a linear arrangement rules across eight chapters as mentioned. Aṣṭādhyāyi was supposed to be passed on in oral form. So Pāṇini had to take care of two aspects. He needed to somehow define the thematic domain and the boundary of the domain for the rules, and also he needs to achieve brevity as he needed to keep the entire Aṣṭādhyāyi as short as possible for ease of recitation. He has used special kind of rules called Adhikara rules for specifying the thematic domain and further subdomains for a given set of rules. He has efficiently used the concept of ellipsis in the form of anuvritti to achieve brevity

3.3.1 Anuvritti

1-3-2 उपदेशेअच् अनुनासिकः इत्
1-3-3 हल् अन्त्यम्
1-3-4 न विभक्तौ तुस्माः
1-3-5 आदिः ञिटुडवः
1-3-6 षः प्रत्ययस्य आदिः
1-3-7 चुटू
1-3-8 लशकु अतद्धिते

Figure 3.2: Anuvritti of 'it' section in Aṣṭādhyāyī

In anuvritti or what we can call as recurrence, in order to keep the individual sutras short, some information is omitted from those sutras which can be obtained from immediately nearby rules. So a rule when read alone doesn't provide the complete information and hence needs to be read with the entire anuvritti. For eg the rule number A.4.1.123 is **शुभ्रादिभ्यः च** which means "for stems in Subra and co. as well", and doesn't makes any sense unless read with the anuvritti. But the same reule when read with anuvritti is **शुभ्रादिभ्यः च प्रत्ययः परः च आद्युदात्तः च तद्धिताः समर्थानां प्रथमात् वा प्राक् दीव्यतः अण् स्त्रीपुंसाभ्यां नञ्स्नञौ भवनात् तस्य अपत्यम् ढक्** . Here the rule translates to "for stems in śubhra and co. as well, that come under adhikara of pratyaya, under the sub domain of taddhita, and pratyayādhikāra of aṇ where the semantic relation is apatyam and if the stem is in genitive, apply the pratyaya ḍhak. " Anuvritti may be passing down of entire rule as in A.4.1.82 or a

partial component inheritance as in case of A.4.1.120. The components being passed down as anuvritti can be a semantic relation, a condition to be satisified, a context in which it needs to be applied, an adhikara or a particular operation. Anuvritti or recurrence can be seen as block of control as well. To visualize consider the snippet given below in Figure 3.2. Here anuvritti is denoted via indentation and those which are inherited is made bigger in size.

3.3.2 Adhikāra

It is a form of inheritance where classes of rules belonging to same thematic domain inherit the thematic head so as to be grouped together (Deo, 2007). The pratyayadhikara A.4.1.83 is one such adhikara rule. The rule infact is a sub-domain inside A.4.1.76, which is a subdomain of A.3.3.1. Similarly rule A.4.4.1 share common ancestor adhikara rules as of that for A.4.1.83. Hence we claim that the Aṣṭādhyāyī is a multilevel inheritance hierarchy. The device of adhikara indicates the homogeneity of topic (?). It classifies the rules that come in its domain with rules that come in a different domain. A rule can never be under the influence of two Adhikaras where one adhikara is not a subset of the other in terms of domain reach. A rule can have multiple adhikara as shown in that for A.4.1.92, it has adhikara rules starting from A.4.1.76. But no two adhikaras are at equal level, each adhikara rule is a subset of other rule. Hence there can be multilevel inheritance but no multiple inheritance in Aṣṭādhyāyī.

Chapter 4

System Implementation and the Derivation Process

4.1 Overview of the Implementation System

For automating the derivation process, we suggest the following method which is based on object oriented concepts. The derivation process happens in a central object called as environment, which essentially has methods and attributes that takes care of conversion of user input to suitable data structures, triggering of rules and book-keeping of rules applied. Each rule forms a class and each instance of the rule class (henceforth to be referenced as rule itself) is registered with the environment, such that whenever there is a change in the environment, the rules are notified. Each rule checks for the possibility of it being applied on the environment and for a rule, if all its conditions are satisfied, ideally the rule can be applied on the environment. However, in a general scenario multiple rules may claim their competency for application on the environment. To handle such scenarios we keep those competing rules in a list called candidate list. Then a conflict resolution method is employed which decides the winner rule. The winner rule gets to apply on the environment and other rules are removed from list. By removal of rules other than the winner rule, we mean that the removed ones are not applied on the present instance of environment, although they are notified when the environment change happens again. Figure 4.1 shows the schema of the implementation system.

As it is evident from discussion about Taddhita section, Pāṇini among various applications of anuvṛtti, uses it for carrying the topicality between different rules as well. It is to be noted that adhikāra rules are rules whose sole purpose is mentioning the topicality of the rules under its domain of influence. But even for adhikāra rules, anuvṛtti itself is used for carrying the domain's influence to other rules. Apart from the adhikāra rules, there are other rules as well

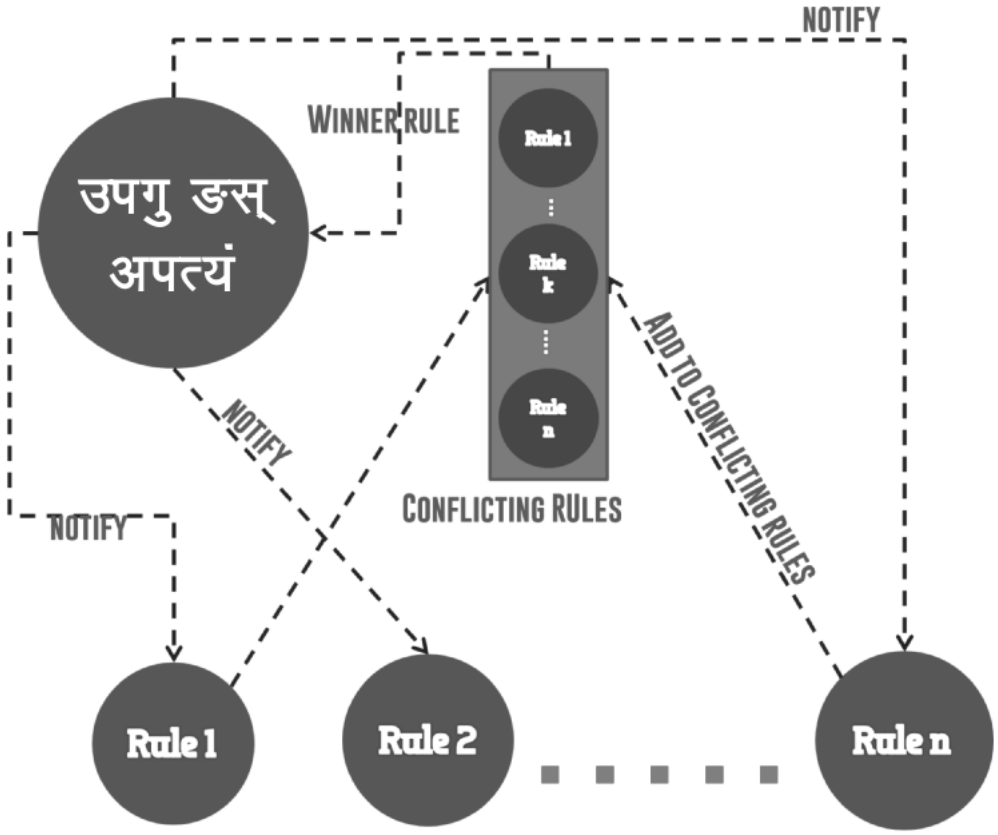

Figure 4.1: Overview of Implementation System

in which anuvṛtti is used to carry the topicality. For example consider the rules concerning **इत्**, i.e. rules from 1.3.2 to 1.3.8. Here **उपदेशे** and **इत्** are being carried forward to other rules as well. **उपदेशे**, which means 'when an upadeśa is encountered', perform some action. This condition leads to a common topicality for all the rules under the anuvṛtti. This is similar to the notion of arthādhikāra rules in Taddhita where the notion of semantic sense is being carried forward to subsequent rules through anuvṛtti. Both the techniques, anuvṛtti and adhikāra are employed in the entire Aṣṭādhyāyī and are not unique to the Taddhita section. We can infer that a subset of the anuvṛtti rules, mostly the ones which carry the notion of topicality can be used to design an inheritance hierarchy of classes. For the implementation we will be using this notion of topicality via anuvṛtti, in grouping of rules to form rule groups which is an inheritance hierarchy among rules and the rule in which the portion of anuvṛtti appears, becomes the head rule.

4.1.1 Tools and Techniques Used

We are following a completely object oriented approach for the implementation of the system. The following tools and techniques will be employed to achieve the principles discussed in section 4.1

Observer Design Patern

Observer design pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically (?). The object to which the state changes occur is called subject and it maintains a list of its dependents, called observers. As shown in Figure 4.2, each observer object which is the sūtra or rule in our case, is registered to an object called as "Subject" class which represents the environment in our case. Subject class has a method to register the observers. Whenever a change in value of some attribute of subject occurs, it calls the method 'notify' of observer abstract class, which is implemented for each object of the Observer.

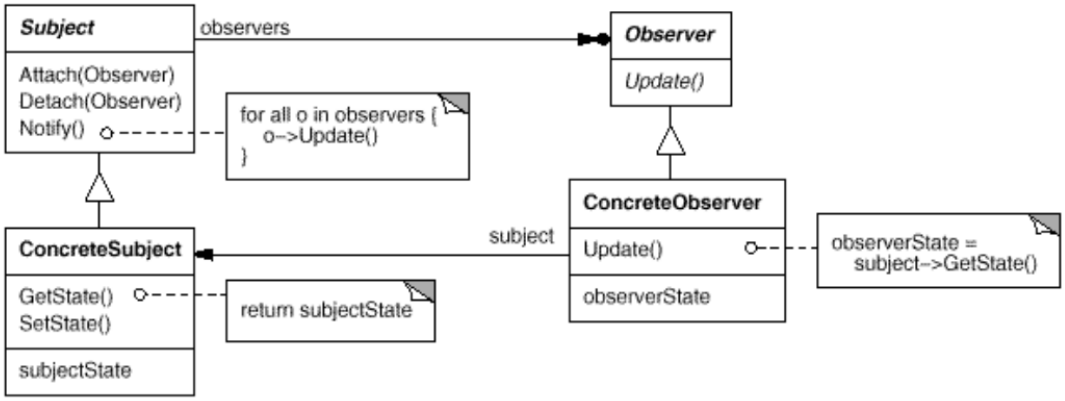

Figure 4.2: UML diagram for observer design pattern

Multilevel Inheritance

To form rule groups i.e. inheritance network among rules we use multilevel inheritance between the rules. Though multilevel inheritance is allowed, multiple inheritance is not allowed in the system and hence no single rule will inherit from two distinct rules directly. Figure 4.3 shows the inheritance achieved for each class in Aṣṭādhyāyī. All classes inherit from a base class called "sūtra", which is a generic class that defines all possible features that a rule or sūtra can possess. All other rules inherit from it. The adhikāra rules, anuvṛtti of interpretive rules, default condition rules etc. form super class for other operational rule classes.

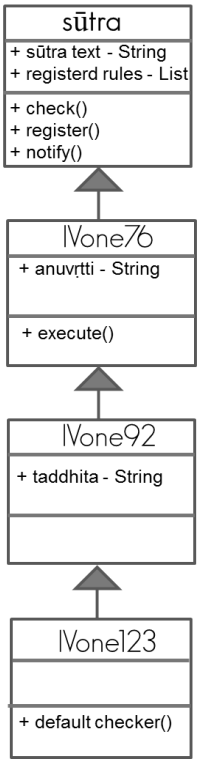

Figure 4.3: UML diagram for multilevel inheritance

4.1.2 Rule Triggering and Propagation

The environment forms the subject class for the observer design pattern. However, not all rules are observing the environment but only those rules that do not come under the domain of any adhikāra rules or a controlling head of a rule group through anuvṛtti (which are mostly rules that assigns the technical terms to assignments). Those are represented by classes R1, R2 and so on in Figure 4.4. Then there are rule groups as represented by RG1, RG2 and so on. These are collections of rules with same head. The rule classes that come under the same head in the group, inherit from the head rule class. The head rule object, as per observer design pattern registers the inherited classes' objects as observers. Now the head class notifies the observers whenever the environment satisfies the head rule's conditions. The conditions checked are those conditions that need to be satisfied by all the rules registered under the head. Here, by 'head', we mean either an adhikāra or a component passed on by anuvṛtti. Now top level rules are those rules which observe the environment directly and get notified whenever the environment changes. For a rule, if the environment satisfies its conditions, it will be added to the candidate list. But if the environment satisfies the conditions that is applicable to an entire rule group, then the environment object is passed on to the next level and this continues till an exception or specific rule is encountered or else returns back to where the default rule resides. By this

model, we can employ conflict resolution at each level, and resolve some of them locally and only rules that have no common head at any level come to the candidate list at top-most level, from where the winner rule will be selected.

Once a winner rule is selected, the rule's intended action is executed first and then its parent object is called which performs its relevant portion in execution, if any. This continues until all the rules in hierarchy are called. It is to be noted that many a times certain rules like the adhikāra rules do not have anything to execute of their own, in such cases the rule object just passes on the environment to its parent object. By this design, redundancy of hard coding the same rules again and again per rule object is saved, just like the way anuvṛtti helps a person to avoid repeating the rules when reciting Aṣṭādhyāyī.

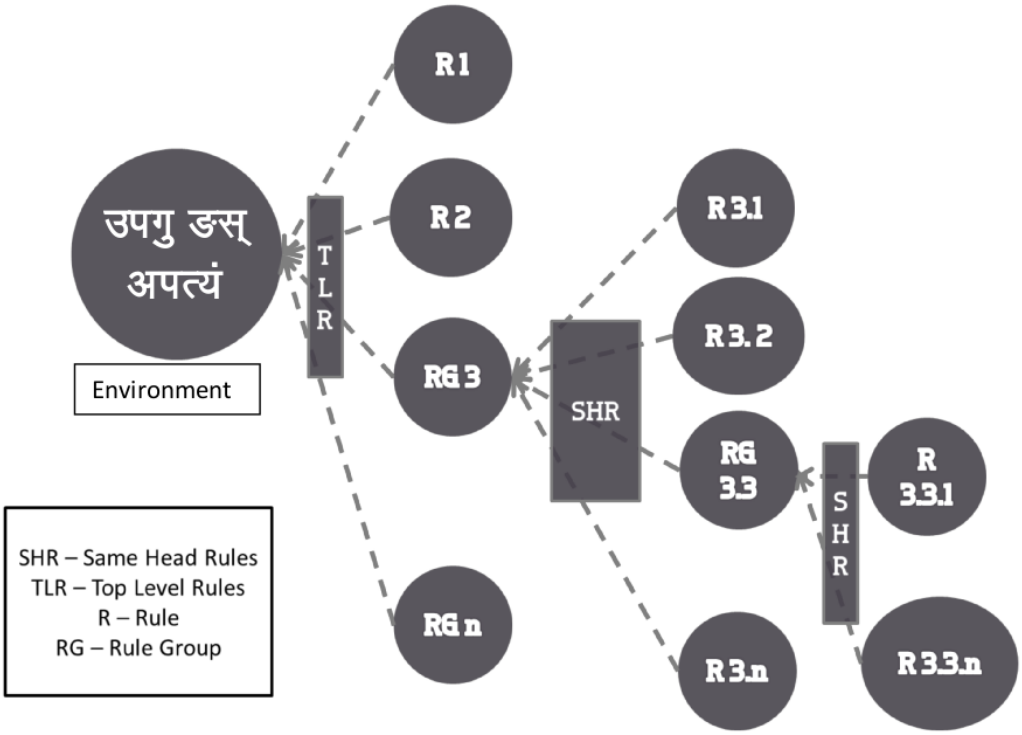

Figure 4.4: Triggering schema.

Working

Let us consider the working of the system with reference to affixation for semantic relation अपत्यम्, i.e. the patronymic relation. Figure 4.5 shows the trace for the affixation process of Taddhita affix when the semantic relation is patronymic and the stem is चटका. In Figure 4.5, the solid lines show inheritance hierarchy between the rule classes. The dashed line on the left of each

block shows traversing of the hierarchy by triggering of rules from head to specific rules. The traversal checks for eligible rule to apply on environment within the rule group, and this can be called as the checking phase. The dotted lines on the right of each block show the trace of rules that get executed. The starting node, i.e. the node that heads the dotted edge with label 1 is the exact rule that gets executed and other nodes in the path show those rules that have been executed due to anuvrtti. This can be called as the execution phase. As an example let us consider the case where the environment is initialized with **चटकायाः अपत्यम्**. The triggering starts from head rule at the top level i.e. from A.3.1.1 to rule A.4.1.128 as shown in Figure 4.2. Here in rule A.3.1.1, it does not have any condition to check, so it directly notifies all of its direct descendants. Now among the direct descendants, rule A.4.1.1 checks for presence of any of the two affixes **ङिः, आप्** or if the environment has a prātipadika in it, i.e. it checks for conditions mentioned directly in the rule. As it is a prātipadika, the condition will evaluate to 'true',and all its descendants are notified. In due course, A.4.1.76, A.4.1.83 are also notified. These rules as well do not have any extra checks as they are adhikāra rules and hence all its direct descendants are notified. When A.4.1.92 is notified, it checks for semantic condition and the checking turns out to be true for A.4.1.92, while the checking will evaluate to 'false' for all of its sister nodes, i.e. other direct descendants of A.4.1.83. The special case rules registered under A.4.1.92 are notified, of which A.4.1.128 satisfies the remaining conditions. As it does not have any rules registered to it, it becomes the terminal node and hence it is added to the candidate list. Since for this case no other rule is contesting, the rule emerges winner and starts its execution from A.4.1.128. The rule adds the affix **ऐरक्** to environment, and passes the environment to its parent class i.e. A.4.1.92. A.4.1.92 does not have anything to execute of its own, so it simply passes environment to A.4.1.83, which checks if any affix is already attached, as the affix '**ऐरक्**' is attached in this case, no action is performed. The environment gets passed to parent node of each rule finally this terminates at top level rule A.3.1.1. Now consider the derivation of **उपगोः अपत्यं**. As with the case of **चटकायाः अपत्यं**, the path of the rules checked for eligibility of rule application remains the same till rule A.4.1.92 is reached. Once A.4.1.92 is reached it will notify all of its descendants as well. But since no rule will find its application, A.4.1.92 will become the final node of the path here. It is added to candidate list, which while executing will simply call the parent rule A.4.1.83. A.4.1.83 checks for presence of any new Taddhita affix in the environment. If the check evaluates to false, i.e. if no new Taddhita affix is found, A.4.1.83 treats this as default case scenario and attaches **अण्**, the default case affix to environment. After each execution of a rule, the system checks for presence of any new assignment of 'technical term' or else all rules in the top level are notified as discussed in Section 4.2.

The arthādhikāra rules inherit from its corresponding pratyayādhikāra rules, apart from the ones already mentioned in Section 3.2. When it comes to affixation, during the checking for eligibility of a rule to apply i.e at the checking phase, we need to traverse the pratyayādhikāra rule, before an arthādhikāra rule is reached. Though a pratyayādhikāra rule is visited during checking, no action is taken there. The pratyayādhikāra rule simply passes the environment

to all arthādhikāra rules which are its direct descendants. In fact for a single affixation, all the pratyayādhikāra rules get notified from its parent rule, and those rules in turn notify all their direct descendants as there is not enough information to select a single pratyayādhikāra at during the checking phase. So in effect, the process of affixation for taddhita starts by checking for the right semantic condition i.e at the arthādhikāra rules as it is in the case of traditional system of derivation. Before that, the other rules either simply pass on the environment to their descendants or check for conditions that is necessary for the process to qualify as a case for affixation under taddhita. The effect of pratyayādhikāra rule comes during the execution phase i.e at the applying of affix phase and not on the checking phase. During the execution phase the pratyayādhikāra which is parent to the winner arthādhikāra rule acts as the final gate that makes sure that the environment has the valid taddhita affix added to the environment before it reaches its parent. The rule checks if any taddhita affix is introduced by virtue of special case rules, and if no such execution has taken place, then the default affix is added to the environment. The environment is then passed on to higher level rules that takes care of other generic aspects about the environment.

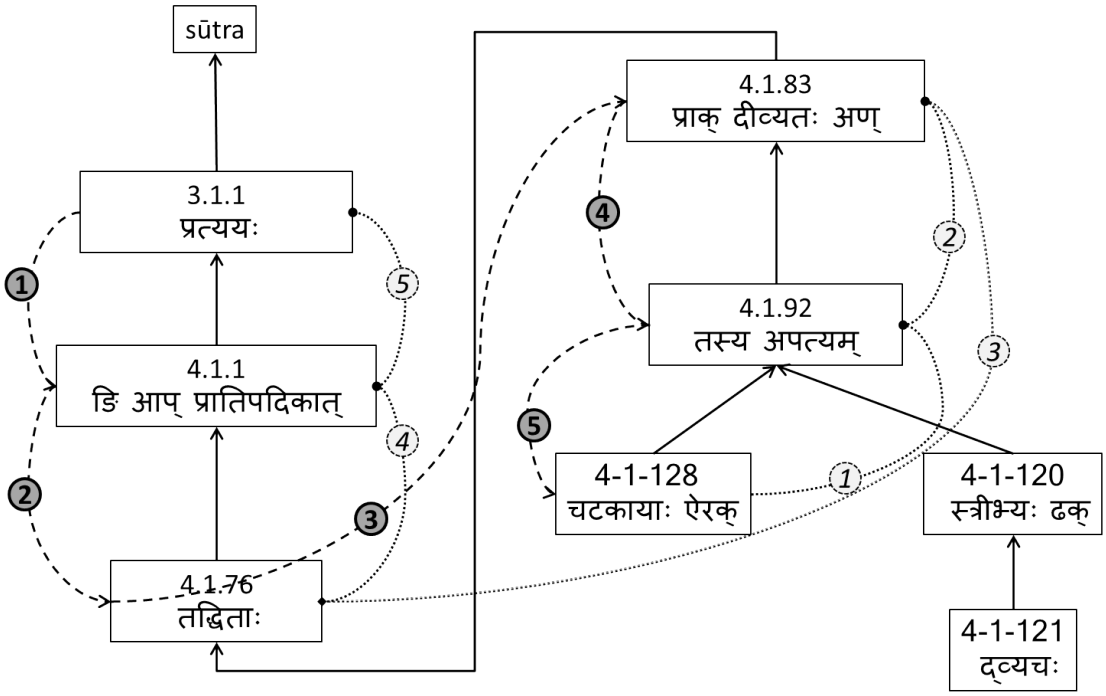

Figure 4.5: Affixation under patronymic relation for चटका

4.1.3 Śabdarūpa Representation

The environment is an object which contains methods for navigating through top level rules, list of applied rules and the śabdarūpa object as its attribute. Śabdarūpa object holds the data entities which take part in the derivation process. Entities can be stems, affixes, augments, characters or any of their properties. The most basic and atomic entity in the śabdarūpa object is another object called 'śabda'. The environment also contains various instances of the class 'śabda collection' , which contains a sequence of references to śabda objects along with a set of attributes that belongs to the collection. Instances of 'śabda collection' are used to represent various technical terms that may be attributed to environment or a part of it. Figure 4.6 shows how the śabdarūpa object is set up. Figure 4.6 shows the śabdarūpa object after the affixation of Taddhita affix **अण्** . As already seen in Section 4.1.2, the Taddhita affixation happened due to presence of technical term prātipadika. Now the technical term prātipadika is modeled as an attribute of the śabdarūpa object, and it is an object itself of the class 'śabda collection'. It contains references to sequence of all the śabda objects, which is collectively eligible for the technical term prātipadika. Similarly inside 'pratyaya' object, it has an attribute 'it marker' which also is an object of 'śabda collection' class called 'it'. If you notice though the it (**इत्**) marker is **ण्** in **अण्**, it is not the śabda's property that it is an it marker. It is the property of pratyaya that gives the śabda **ण्**, the property of it marker. This notion is captured very well in the system. It is essential that we store them as attributes separately for further reference and usage in the derivation process. For example, consider the derivation process for āśvalāyana. The term āśvalāyana is formed from aśvala by affixing 'phak' pratyaya. Here the 'it' marker is 'k', which will be stored in 'it marker' object, and later it will be elided, and hence be removed from the 'text value' of pratyaya object by application of rules A.1.3.3 and A.1.3.9 . In due course of derivation the pratyaya object will get a substitution of 'āyana' for the remaining 'pha' by rule A.7.1.2. Now in order to complete the derivation process, rule A.7.2.118 should stand valid in one of the subsequent steps. RuleA.7.2.118 requires a Taddhita affix with k as 'it' marker. If we had not stored this information as a separate attribute earlier, we would have lost this information and derivation would not have completed. Apart from the attributes, the śabdarūpa object has methods that performs the five operations of placement, augmentation, substitution, deletion and modification as categorised by Sharma (2002) for operational rules.

I Placement of a new entity - A new entity like affixation where a new independent entity is introduced to the śabdarūpa object. 4-1-128 **चटकायाः ऐरक्** does a placement operation.

II Augment an existing entity - Adding a new augment to any of the existing entities. A.4-1-97 **सुधातुः अकङ् च** is one such rule that performs augmentation of **अकङ्** followed by a placement operation.

III Substitution - Replace contents of an entity or a portion of it with new substituent contents. A.6-1-77**इको यण् अचि** does a replacement operation.

IV Deletion - Remove an entity or a portion of it. 4-1-133 **ढकि लोपः** is a rule that performs the operation.

Modification is performed as combination of any of the four operation after executing the modifying action like the vṛddhi operation followed by a substitution
In case of asiddhavat rules as discussed in Subbanna and Varakhedi (2009), the śabdarūpa object makes a complete copy of itself, one object is used for checking the conditions while in the other object, all the operations are applied. Once the system returns back to siddha section, the copy used for checking the conditions is discarded. It is also to be noted that in the representation the space between entities are also śabda objects, representing the virāma (**विराम**) as per the rule A.1.4.110 **विरामः अवसानम्**.

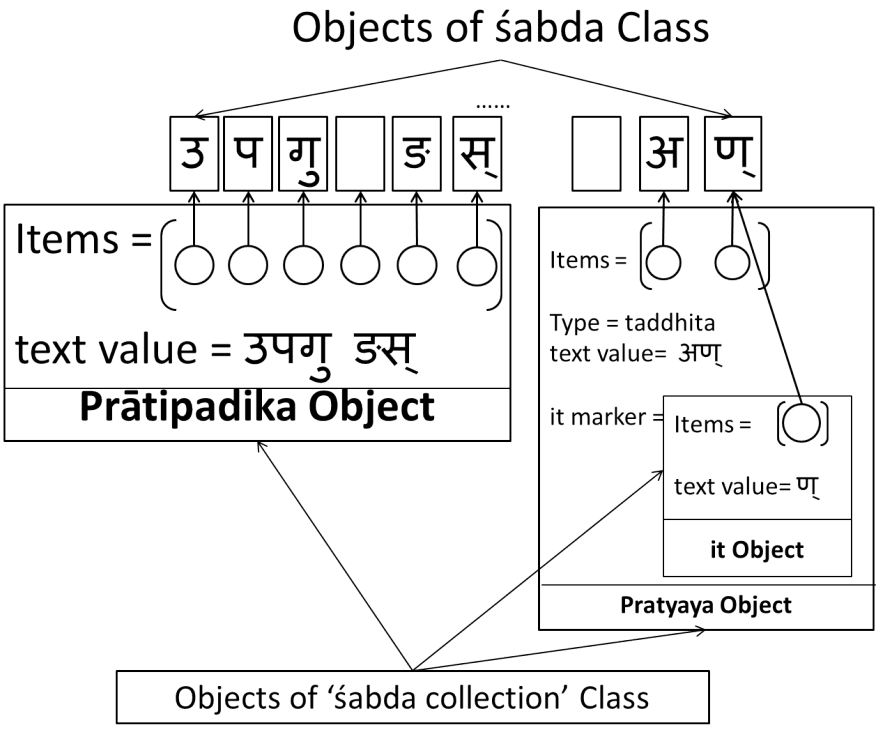

Figure 4.6: Representation of Environment object

4.2 Derivational Process along with the storage and propagation of linguistic features.

In order to discuss about the derivation process or (**प्रक्रिया**) of a derivative noun, let us take the derivation of derivative noun **औपगव** from the nominal **उपगु**, which can be considered as the "Hello

world" in Taddhita section. The essential steps in the derivation process are shown in Table 4.1. औपगव which means son of upagu, comes from the semantic sense अपत्यं by the rule 4.1.92 तस्य अपत्यम्, and the correct Taddhita affix अण् is introduced to the environment. Once the Taddhita affix is introduced, what remains is a series of operations on the environment that results in a final form. The rules so triggered may be seen as a continuous iteration of two sub-processes. One is identification and assignment of technical terms (संज्ञा) to the environment string or to a relevant substring of it. By this, the environment does not get modified but gains the technical term as an attribute. The interpretive rules or rules assign the technical terms to the environment and those rules are shown in the first column of Table 4.1. Now, by virtue of these attributes, the environment gets modified through operational rules that come within the domain of attributes as shown in the third column of Table 4.1. The effect of the rule on the environment can be seen in the second column. Triggering of operational rules is the second sub-process in the iteration and the derivation process stops when no more attributes can be attributed to the environment. Ideally, by that time the desired form must be derived.

4.3 Derivation of a Derivative Noun

In this section we will show how the nominal stem औपगव is derived in the system. In section 4.1.2, the affixation is already shown. However, some of the finer details regarding post processing after execution of the rule are not discussed, which we will be doing in this step by step walk-through of the derivation. Please refer to Table 4.1 for state of environment after each rule is applied.

- Affixation as shown in section 4.1.2. Here the user input "उपगु ङस् अपत्यं", has led to affixation of desired affix उपगु ङस् अण्. As the affix got added to the environment in the form of 'pratyaya' object as shown in Figure 4.6, two more attributes were also added to the 'pratyaya' object. The two attributes are 'upadeśa' and 'Taddhita'. Contrary to as discussed in section 4.1.2, the attributes are not objects by themselves. This is because the pratyaya object bears reference to the exact sequence of śabda objects as these two attributes are and hence separate object instantiation is not required. The prātipadika object which is an attribute of the environment, but signifying the base nominal उपगु ङस्, i.e. the stem upagu in genitive case, goes to "processed" state.

- Since there are two new attributes that are not yet in processed state in one of the objects of environment, instead of notifying the top level rules, system takes in the attribute upadeśa which is a technical term and is assigned to pratyaya object, and triggers corresponding rule group (in this case the rule group headed by A 1.3.2). This leads to instantiation of 'it marker' object as shown in Figure 4.6 and subsequently to rule 1.3.9 that leads to elision of the it marker. Though the marker is elided, neither the the object reference nor the śabda object, the marker is referring to, is removed from the environment. In fact the śabda object keeps the marker information (णित्) as a separate attribute.

- Now since no more rules can be triggered automatically, the system notifies all the top level rules of which, top level rule A.1.2.46 finds its eligibility due to presence of attribute 'Taddhita'. Though A.1.2.46 gets Prātipadikam प्रातिपदिकम् from A.1.2.45 as anuvṛtti, still it is not modeled as a descendant of A.1.2.45, as it does not represent topicality or common condition. The effect of the anuvṛtti passed on here is that of assigning of the term, which is an effect on the rule, not a cause or condition on the rule. Environment gets a new attribute Prātipadika.

- No specific rule group can be invoked by Prātipadika attribute. All top level rules are notified, of which only A.2.4.71 finds its eligibility, which also happens to be a top level rule. This results in removal of ङस्

- Similarly, the system will get the object 'aṅga' (for the technical term aṅga) as an attribute to environment, after the rule A.1.4.13 find its eligibility, and the system will directly notify the rule group headed by A.6.4.1 अङ्गस्य. A.7.2.117 will find its eligibility to apply. The exact same steps are executed for subsequent operations, where the technical terms भ and संहिता are assigned due to the the rules A.1.4.18 and A.1.4.109 respectively and by virtue of those attributes to environment, rules A.6.4.146 and A.6.1.78 respectively are executed resulting in the desired final form औपगव.

4.4 Rule Selection and Conflict Resolution

There are instances in Aṣṭādhyāyi where multiple rules find its suitability to be applied on the derivation environment. In case of such conflicts, there must be some mechanisms that helps to resolve such conflicts. Aṣṭādhyāyi doesn't have much information towards conflict resolution, though later commentaries mention about the same. It is argued that Panini has assumed those principles which were prevelant in his time as a prerequisite to understand his treatise on grammar. But this as led to unresolved debates amongst the scholars which still persists. In the wake of such a scenario, we have decided to implement the conflcit resolution module as a separate pluggable entity in our system, so that we can try out different mechanisms which the scholars in general have come to consensus. Our system internalizes the concept which is discussed in Aṣṭādhyāyi itself, which is the rule A.1.4.2. The details of how this rule works is discussed in Section . Outside of Aṣṭādhyāyi, what is described in 'paribhāṣenḍuśekhara' of 'Nāgeśa' as ' pūrvaparanityāntaraṅgāpavaḍānāmuttarottaram balīyaḥ' is generally accepted. This gives the following linear order.

> prior (purva) < subsequent (para) < obligatory (nitya) < internally conditioned (antaranga) < exception (apavada)

- apavāda:- The rules that are exceptions to a particular general rule are called apavāda to that rule, and the apavāda rule blocks a general rule, in case of a conflict. this is called

utsarga-apavada combination.A.4.1.128 is an exception to A.4.1.92. Hence A.4.1.128 will tkae precedednce over A.4.1.92

- antaranga (and bahiranga) :- Internally conditioned and externally conditioned operations. When there is a conflict between two types, the operations that apply within the same syntactic boundary blocks the one in which the operation needs to be taken across different entities. This can be seen as a kind of bracketing, where the internal brackets are considered first and then the external ones are considered. In case of 'vasati atra', rules A.3.4.86 and A.6.1.77 are applicable. But A.6.1.77 requires conditions from two different entities to be satisfied, 'i' of 'vasati' and 'a' of 'atra'. But A.3.4.86 requires only 'i' of 'vasati'. So A.3.4.86 will be applied.

- nitya (and anitya):- When two rules R1 ad R2 are in conflict, and after the application of R1, R2 still can be applied but if its not possible for R1 to be applied, after R2 is applied then R2 is called the nitya or the obligatory rule. R2 will have precedence over R1 in such a case. For kṛ-atus, rules A.6.1.8 and A.6.1.77 are applicable. But A.6.1.77 can still be applied even if A.6.1.8 is applied, but it doenst happen the other way round. So A.6.1.77 becomes the nitya rule here.

- para and pūrva - If none of the above conflict resolution methods are not applicable, then the rule which is stated later in Aṣṭādhyāyi gets precedence.

As it is evident from our discussions, that we have centered our system design based on the notion of topicality, and the multilevel inheritance network is formed on the basis of topical head rules and the child nodes. But when it comes to rules which doesn't fall under the similar topical heads, there are some instances in Taddhita section itself which doesn't result in proper rule selection. To resolve such a scenario we have applied the specificity hierarchy as mentioned in ?). Specificity hierarchy deals with a priority wise ordering of rules based on the linguistic features present in the rule from the most concrete to most abstract features. The linear ordering is as follows:

Phonetics < Phonology < Morphological < Semantic

The specificity of a rule is determined by the specificity aspects that are present rule as conditions to be checked. It is also to be noted that within the entities with same specificity there can be further refinement or entities that can be of higher degree in specificity than the others. In the apawya relation itself. the rules that states for the presence of semantic condition of apawyam, gotra and yuvām, all fall under the specificity hierarchy of 'semantic'. But amongst them gotra is more specific than apatyam, and so is yuvām to gotra. Hence when two rules find their application in an environment, one with apatyam and the other with gotra, the one with gotra specification will emerge as the winner rule even if latter comes before the former in the Astadhyayi ordering. Just like in case of topicality, there are no explicit markings

from avialable in Astadhyayi to identify the specificty. We need to have the prior information and encode the same in our rule classes just like the way we have doen with topicality in our implementation. We have implemented specificty hierarchy for the system in apatya section and that has improved the results substantailly which will be discussed in Section ??.

We will be discussing the effect of applying these conflict resolution methods in Taddhita section and in other specific instances which are outside the scope of Taddhita in Section ??, to establish the system's effectiveness as a automated system for simulation of entire Aṣṭādhyāyi.

Interpretational terms to be assigned	Derivation environment	Operation rule to be applied
	उपगोः अपत्यं	
	उपगु ङस् अपत्यं	
तद्धित (4.1.76 तद्धिताः)	उपगु ङस् अण्	4.1.92 तस्य अपत्यम्
इत् (1.3.3 हलन्त्यम्)	उपगु ङस् अ	1.3.9 तस्य लोपः
प्रातिपदिकम् (1.2.46 कृत्तद्धितसमासाः च)	उपगु अ	2.4.71 सुपः धातुप्रातिपदिकयोः
अङ्गम् (1.4.13 यस्मात् प्रत्ययविधिः तदादि प्रत्यये अङ्गम्)	औपगु अ	7.2.117 तद्धितेषु अचाम्
भ (1.4.18 यचि भम्)	औपगो अ	6.4.146 ओः गुणः
संहिता (1.4.109 परः सन्निकर्षः संहिता)	औपगव् अ	6.1.78 एचः अयवायावः
	औपगव	

Table 4.1: Derivation Process for **औपगव** . Here each horizontal line partition represents one step in the derivation, where one of the operations among insertion, elision or substitution takes place in column 2. Column 1 shows assignment of some technical term, and column 3 shows subsequent operation after gaining the technical term as a property. Column 2 shows the effect due to the operation. In column 2 we can see the effect of the rules on the environment, where an insertion, elision or substitution takes place.

Chapter 5

Evaluation Results

5.1 Evaluation Framework

For the evaluation, we have implemented the entire apatya section of the Aṣṭādhyāyi. By apatya section we mean Rules from A.4.1.92 to A.4.176 which deals with affixation rules for stems that needs to be used along with semantic sense of apatyam (with its subtypes gotra,yuvām) or 'the descendant of' semantic relation. For the proper execution of these rules, we were required to implement other rules that comes in the multilevel inheritance hierarchy like the pratyayādhikāra rule A.4.1.83 and its exceptions as well as those outside the Taddhita section like vṛddhi, guṇa and other associated rules. We have selected about 60 input cases that covers the whole span of 'apatya' section and obtained the outputs after affixation from the system.

A web based survey interface was used for the human judgement.[1]. Each expert was supposed to evaluate of 20 of the input cases. A total of five experts from the linguistics ad sanskrit computational lingusitics fields participated in the evaluation. For a given input case, an expert was shown the input string, along with other conditions required for correct derivation like the intention of the speaker. The experts were also provided with the sutra of the winner rule which was applied, along with other conflicting rules and finally the output. The experts were to do a binary evaluation of the correctness of output, based on the input and other constraints provided. Figure 5.1 shows a snapshot of the evaluation framework. In case of difference of opinion among the experts we took the opinion of of majority as truth value after weighing in the remarks provided by the experts.

[1]The evaluation URLs are :- Set 1 - http://goo.gl/forms/Lj5z3UzUr9, Set 2 - http://goo.gl/forms/2O31D6gF83, Set 3 - http://goo.gl/forms/goWgwMnYct

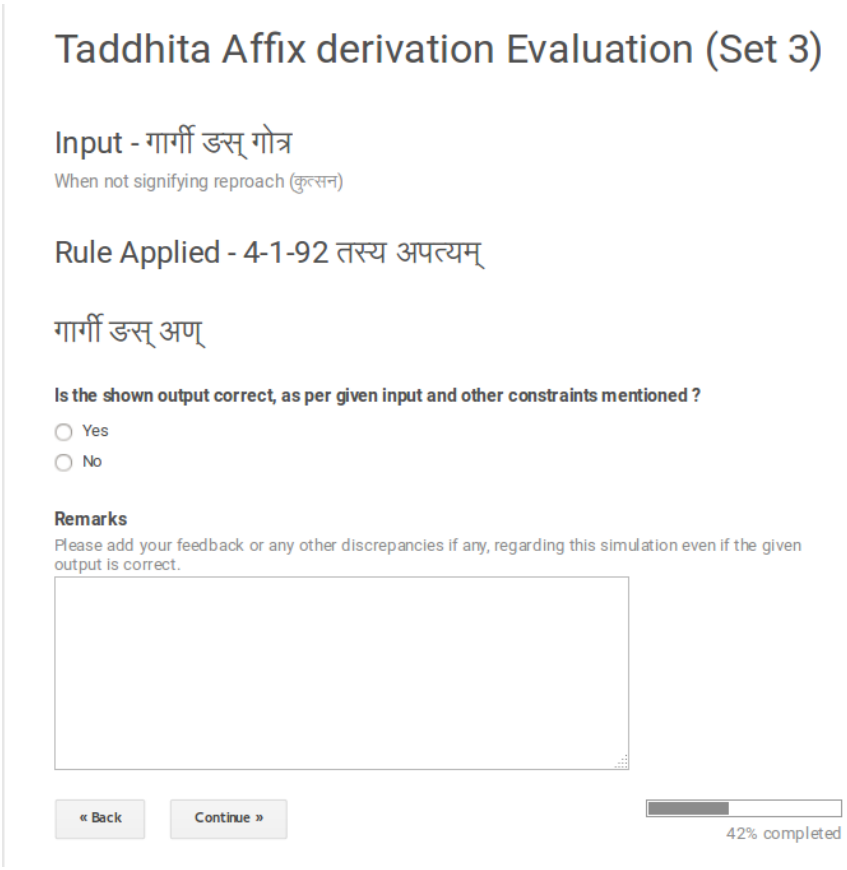

Figure 5.1: Snapshot of the evaluation framework

The results of the evaluation are shown in Table 5.1. From among the 60 input cases, a total of nine cases came out to be wrong. As already discussed in Section 6.1, the system has internalized vipratiṣedha as its default conflict resolution mechanism. The outputs were obtained with no other conflict resolution mechanism being employed. Ten cases resulted in wrong output. We later implemented the specificity hierarchy and then obtained the outputs for the same set of input cases. This time, the numebr of wrong ouput to the input cases were reduced to four cases. We will be discussing some of the wrong output cases in Section 5.2.

5.2 Analysis of wrong cases and other special cases.

- For inputs **उत्स ङस् अपत्यम्, दिति ङस् अपत्यम्**, the rules that got applied were A.4.1.95 and A.4.1.122 and those two rules essentially looks for phonological properties (ending sound, presence of vowel etc.). The correct rules to be applied for the instances are A.4.1.86 and

Number of Input cases	60
Evaluators Participated	5
Inputs per evaluator	20
Numebr of Correct cases	51
Number of Wrong output (with no external conflict resolution)	10
Number of wrong outputs (with specificity hierarchy for conflict resolution)	4

Evaluation	Accuracy	Error
With no external conflict resolution	83.33	16.67
With specificity hierarchy	93.33	6.67

Table 5.1: My caption

A.4.1.85 respectively. Though those rules have same topical heads and are specified before the current winner rules, they are less specific than the winner rules. The correct rules have specificty at morphological level where specific stems are being mentioned. please note that the mentioned rules carries entities that bear semantic specificity. But, since all the rules have the notion the effect is nullified here. In addition to that, those rules also have the above said specificity properties.

- For inputs **गर्ग ङस् गोत्र, कपि ङस् गोत्र**, the rules that got applied are A.4.1.151 and A.4.1.122 instead of rules A.4.1.105 and A.4.1.107 respectively. Any rule in apatya section, or for that matter any rule under the 'arthādhikāra' has a semantic specificity by default, which is the semantic sense. But in this case, the rules A.4.1.105 and 107 are being mentioned in a granular level of semantic specificity that of gotra, while the current applied rules deals the general case of apatya. Please note that gotra is a specific sub-type of apatya. Hence those rules should be applied.

- For the input **पितृष्वसृ ङस् अपत्यम्**, Ideally two output cases should appear, **पितृष्वसृ ङस् छण्** and **पितृष्वस् ङस् ढक्**. It is not rare in Taddhita derivation to see the applicability of multiple affixes for a single input case. But here the second output is the result of the rule A.4.1.133 **ढकि लोपः**. The rule states nothing but, the**ऋ** in **पितृष्वसृ** will be elided. No rule mentions affixation of **ढक्** for the given input case. Now while implementing rule A.4.1.133, the an assumption is made that the affix **ढक्** should be introduced and then the elision operation should be made. The basis of this assumption comes from the argument of many of the linguists that Pāṇini by specifying about the condition in A.4.1.133 implies that the affix **ढक्** should be introduced. For the sake of brevity he explicitly didn't mention the rule, but if the affixation is not implied then there is no purpose for the rule. Our implementation has followed this assumption.

- In case of rule A.4.1.115 **मातुः उत् सङ्घासम्भद्रपूर्वायाः**, here the stem mātṛ, will get its ṛ repalced with u provided if it satisfies the conditions given in the rule, by virtue of rule A.1.1.115.

Now when it comes to automation of such a derivation, other than explicitly specifying each of the rule to trigger inside the object definitions for each such special cases, it requires some special mechanism for triggering such rules. An elegant solution for such special cases are yet needed to be identified. Please note that triggering rules for identifying 'it' markers which is normally the next step or searching for a 'samules are already implemented, but such specific rule triggering creates a hurdle in automation.

- In case of rule A.4.1.122, the rule is applicable specifically for nominals that already has 'ifix atached to them but, unless and until one has prior knowledge about the affix. There are other affixes as well that provide 'i' to a stem just like 'ies. For such a purpose when automating, we should either keep a list of words beforehand, or else there should be some mechanism to store the extra information. Our system currently stores a list of some stems to tackle the issue as of now. But what is more significant is that our 'śabdarūpa' object provides the functionality of all the extra information like the 'ich in reality will be elided. THis is later discussed in detail in Section ??

Chapter 6

Discussion and Conclusion

6.1 The Schema as a General Schema for Modelling Aṣṭādhyāyī

In section 4.4, we have seen the working of the system. In this section we will be considering rules that are outside of Taddhita section. ?) discusses five cases in which conflict of application occurs between two competing rules. In his paper, Scharf talks about conflict between rules A.6.1.87 and A.6.1.88 where the domain of application of A.6.1.88 is properly contained within the domain of application of rule A.6.1.87. Here A.6.1.88 should emerge as the winner rule, or else 6.1.88 will never be applied to any context. In our system A.6.1.87 is a direct descendant of A.6.1.84 which is a direct descendant of A.6.1.77. The only aspect that is passed from A.6.1.77 as part of inheritance is **अचि**, which is nothing but checking for a vowel as a right context. One might argue that A.6.1.84 does not have any relevance for this checking, but A.6.1.84 is an adhikāra rule and hence the execution will never stop at the rule, instead it will surely traverse down to one of the descendant rules. Also, its domain of influence is completely within the set of rules which is under the influence of rule A.6.1.77. Now A.6.1.87 is one such rule. It is also evident from this instance as to how the rule is inheriting a set of mutually exclusive features and conditions by multilevel inheritance, avoiding the need for multiple inheritance. A.6.1.88 is also one such rule, but it needs an additional checking of condition which is checked also at A.6.1.87. So in our system A.6.1.88 inherits from A.6.1.87 and becomes descendant of 6.1.87 by virtue of the anuvṛtti of **आद्**. So whenever environment object is passed on to A.6.1.87, it first checks presence of **अ** or **आ** as the left context and then notifies all its registered rules including A.6.1.88. If the rule is applicable to A.6.1.88, i.e., if the environment satisfies all the conditions as demanded by 6.1.88 then it gets activated. So in addition to check for left context as per A.6.1.87, A.6.1.88 also checks for **एच्** as right context, which is applicable only to itself. This check is further restricting the scope of application to what occurred at rule A.6.1.77 where the checking was for **अच्**. This if evaluates to 'true' will

block the direct application of A.6.1.87 i.e. guṇa will not happen there, but instead vṛddhi will take place. If no rules turn out to be eligible for application, then only A.6.1.87 will perform the application of guṇa over the environment. Please note that the checking of other conditions that are applicable to these rules are not discussed here as those conditions are obtained by virtue of anuvṛtti and are being checked in the parent rules.

Now let us consider a case where partial blocking occurs when the rules A.6.1.77 and A.6.1.101 are in conflict. Our system deals with this in the same manner as we dealt with simple blocking. Rule A.6.1.101 inherits **अचि**, i.e. element of ac pratyāhāra, as right context. One thing to notice here is that the rule A.6.1.77 also checks for **इकः**, i.e., element of ik pratyāhāra in the context. But this is not carried forward as anuvṛtti. Now this rule is modeled as follows. The environment is passed onto A.6.1.77 and then it first checks for **अचि**. Since only this is carried as anuvṛtti, once this condition evaluates to 'true', the rule notifies all rules registered with it. It waits for any of its descendants to claim eligibility. If no rule claims eligibility then the rule checks for additional condition of **इकः** and if the condition evaluates to 'true', it claims the eligibility. In case of conflict between A.6.1.77 and A.6.1.101, the desired rule is A.6.1.101, but A.6.1.101 is a descendant of A.6.1.77, so it will claim its eligibility and hence A.6.1.77 will be blocked from claiming its eligibility as per our system. From these examples it is evident that the system, without even using any extra conflict resolution techniques, can resolve the conflicts here. These examples demonstrate that the system internalizes rule A.1.4.2 **विप्रतिषेधे परं कार्यम्**. Now consider the case when there is a conflict between the rules A.7.3.111 and A.7.1.73. Here both the rules belong to domain of aṅga, i.e., rule A.6.4.1. Neither of the rules inherit from the other, and hence both of them are at sibling level and have a common parent at rule A.6.4.1. In such a case where both the rules claim eligibility, the conflict which is at the level of rule A.6.4.1, needs to be resolved. The desired rule in such a case is A.7.1.73. The important point to be noted here is that though the system internalizes the concept of A.1.4.2, it still does not claim A.7.3.111 as the winner rule. Instead the system will pass both the rules to the conflict resolution method coded as a separate function in the system, once it reaches the object for the rule A.6.4.1.

As discussed in Section , we first implemented the four principles stated in 'Nāgeśa's' ' pūrvaparanityāntaraṅgāpavaḍānāmuttarottaram balīyaḥ'. As expected for the cases which is discussed in the section like A.6.1.77 vs.A.3.4.86 and A.6.1.8 vs. A.6.1.77 and some other cases, we were able to resolve the conflicts and apply the desired rule. But for the case of conflict between rules A.7.1.73 and A.7.3.111 our system was still applying the rule 7.3.111 as per the priority for para, the rule which is stated later. But in this case, it is not the desired rule.

Other conflict cases discussed in ?) are not discussed here as those fall into one of the scenarios already discussed.

References

References

[Bharati et al.2008] Akshar Bharati, Samar Husain, and Dipti Misra Sharma andd Rajeev Sangal. 2008. A two-stage constraint based dependency parser for free word order languages. In Proceedings of the COLIPS International Conference on Asian Language Processing 2008 (IALP).

[Bhate1989] Saroja Bhate. 1989. Panini's Taddhita rules. University of Poona, Pune.

[Bhattacharyya2010] Pushpak Bhattacharyya. 2010. Indowordnet. In proceedings of LREC-10. Citeseer.

[Cardona1965] George Cardona. 1965. On translating and formalizing pāṇinian rules. In Journal of the Oriental Institute of Baroda, volume 14, pages 306–14.

[Cardona1969] George Cardona. 1969. Studies in indian grammarians i: the method of description reflected in the śivasūtras. In Transactions of the American Philosophical Society, pages 3–48. JSTOR.

[Cardona1997] George Cardona. 1997. Panini: His work and its traditions vol 1. In Background and Introduction. 2nd ed. Motilal Banarsidass.

[Cardona2009] George Cardona. 2009. On the structure of pāṇini's system. In Sanskrit Computational Linguistics, First and Second International Symposia, Rocquencourt, France, pages 1–32. Springer.

[Deo2007] Ashwini Deo. 2007. Derivational morphology in inheritance-based lexica: Insights from pāṇini. In Lingua, volume 117.1, pages 175–201. Elsevier.

[Goyal et al.2009] Pawan Goyal, Amba Kulkarni, and Laxmidhar Behera. 2009. Computer simulation of aṣṭādhyāyī: Some insights. In Sanskrit Computational Linguistics, First and Second International Symposia, Rocquencourt, France, pages 139–161. Springer.

[Hyman2009] Malcolm D. Hyman. 2009. From pāṇinian sandhi to finite state calculus. In Sanskrit Computational Linguistics, First and Second International Symposia, Rocquencourt, France, pages 253–265. Springer.

[Jha and Mishra2009] Girish Nath Jha and Sudhir K. Mishra. 2009. Semantic processing in pāṇini's kāraka system. In Sanskrit Computational Linguistics, First and Second International Symposia, Rocquencourt, France, pages 239–252. Springer.

[Jha et al.2009] Girish Nath Jha, Muktanand Agrawal, Subash, Sudhir K. Mishra, Diwakar Mani, Diwakar Mishra, Manji Bhadra, and Surjit K. Singh. 2009. Inflectional morphology analyzer for sanskrit. In Sanskrit Computational Linguistics, First and Second International Symposia, Rocquencourt, France, pages 219–238. Springer.

[johshi and Roodbergen1987] S. D. johshi and J. A. F. Roodbergen. 1987. On siddha, asiddha and sthĀnivat. In Annals of the Bhandarkar Oriental Research Institute, volume 68, pages 541–549. Bhandarkar Oriental Research Institute.

[Kulkarni2009] Malhar Kulkarni. 2009. Phonological overgeneration in pāṇinian system. In Sanskrit Computational Linguistics, First and Second International Symposia, Rocquencourt, France, pages 306–319. Springer.

[Mishra2009] Anand Mishra. 2009. Simulating the pāṇinian system of sanskrit grammar. In Sanskrit Computational Linguistics, First and Second International Symposia, Rocquencourt, France, pages 127–138. Springer.

[Mishra2010] Anand Mishra. 2010. Modelling aṣṭādhyāyī: An approach based on the methodology of ancillary disciplines (vedanga). In Sanskrit Computational Linguistics, Fourth International Symposium, Delhi, India, pages 239–258. Springer.

[Penn and Kiparsky2012] Gerald Penn and Paul Kiparsky. 2012. On pāṇini and the generative capacity of contextualized replacement systems. In Proceedings of COLING 2012: Posters, pages 943–950.

[Satuluri and Kulkarni2014] Pawankumar Satuluri and Amba Kulkarni. 2014. Extra linguistic information needed for automatic generation of sanskrit compounds: A study. In The recent developments in Sanskrit Computational Linguistics', at SALA-30, Hyderabad.

[Scharf2009] Peter Scharf. 2009. Modeling pāṇinian grammar. In Sanskrit Computational Linguistics, First and Second International Symposia, Rocquencourt, France, pages 95–126. Springer.

[Sharma2002] Rama Nath Sharma. 2002. The Aṣṭādhyāyi of Pāṇini - Vol.1 : Introduction to the Aṣṭādhyāyi as a Grammatical Device. Munshiram Manoharlal Publishers Pvt. Ltd., New Delhi.

[Staal1965] J. Frits Staal. 1965. Context-sensitive rules in pāṇini. In Foundations of Language 1, pages 63–72.

[Subbanna and Varakhedi2009] Sridhar Subbanna and Shrinivasa Varakhedi. 2009. Computational structure of the aṣṭādhyāyī and conflict resolution techniques. In Sanskrit Computational Linguistics, Third International Symposium, Hyderabad, India, pages 56–65. Springer.

[Subbanna and Varakhedi2010] Sridhar Subbanna and Shrinivasa Varakhedi. 2010. Asiddhatva principle in computational model of aṣṭādhyāyī. In Sanskrit Computational Linguistics, Fourth International Symposium, Delhi, India, pages 231–238. Springer.

[Szallies1997] Constantin Szallies. 1997. On using the observer design pattern. XP-002323533,(Aug. 21, 1997), 9.