Computational Modelling of the *Apatyādhikāra* in the *Aṣṭādhyāyī*

Himanshu Ayachit
Humanities and Social Sciences,
Indian Institute of Technology
Roorkee, India
himanshu_a@hs.iitr.ac.in

Pavankumar Satuluri
Humanities and Social Sciences
Indian Institute of Technology
Roorkee, India
pavankumar.satuluri@hs.iitr.ac.in

Abstract

The majority of Sanskrit computational tools focus on the inflectional morphology (*subanta* and *tiṅanta*), compounds (*samāsa*), and primary derivatives (*kṛdantas*). While secondary derivatives (*taddhitāntas*) remain computationally unexplored due to their vast coverage and complex semantic conditions. Most of the existing computational tools on Sanskrit morphology follow a "black-box" approach that only generates the final forms and does not show the entire derivational process. This paper presents a transparent rule-based model for *taddhitānta* word generation designed according to the internal logic of the *Aṣṭādhyāyī*. The model is focused on the *apatyādhikāra* domain of the *taddhita* section in *Aṣṭādhyāyī*. The system follows the *Pāṇinian* concepts of *utsarga-apavāda* (general-specific), which play an important role in conflict resolution. The model was tested on 175 examples collected from *Kāśikāvṛtti* and *Siddhāntakaumudī*, and *Pāṇinīyavyākaraṇodāharaṇakosaḥ*, covering all the rules from the selected domain. The model was highly accurate in generating the output, along with the entire derivational procedure (*prakriyā*) for the tested examples that were evaluated by the experts of *vyākaraṇa*. These results show that computational models can successfully reconstruct *Pāṇinian* grammar, which can be valuable for both linguistic research and pedagogical applications.

1 Introduction

Sanskrit is a rich language in terms of morphology (Kiparsky, 2009). *Pāṇini's Aṣṭādhyāyī* is a Sanskrit grammatical text that deals with the formation of words using different aphorisms (*sūtras*). It consists of approximately 4000 *sūtras*, which help us obtain different word forms. *Aṣṭādhyāyī* deals with all the grammatical components needed in a sentence (Joshi and Roodbergen, 1991; Kiparsky, 2009). It helps us derive various morphological components such as nouns (*subantas*), verbs (*tiṅantas*), compounds (*samāsas*), primary derivatives (*kṛdantas*), and secondary derivatives (*taddhitāntas*). The morphological components of Sanskrit can be further classified as inflectional and derivational. Inflectional morphology in Sanskrit consists of a) *subanta* (nouns) and b) *tiṅanta* (verbs). Derivational morphology in Sanskrit consists of a) *samāsa* (compounds), b) *kṛdanta* (primary derivatives), and c) *taddhitāntas* (secondary derivatives). The Sanskrit tradition articulates these morphological concepts with great precision, focusing primarily on the use of *dhātus* and their forms, *kārakas*, *kṛdantas*, and *samāsas*. Along with these, the *taddhita* section, which produces a wealth of word forms that contribute to sentence brevity, remains significantly underexplored computationally.

The *taddhita* section of the *Aṣṭādhyāyī* extends into the fourth and fifth *adhyāya*(chapter), and this is the largest section of *Aṣṭadhyāyī*, covering almost a quarter of the text, consisting of approximately 1125 *sūtras* (Bhate 1989). *Taddhitāntas* are formed by adding *taddhita* suffixes to existing nouns. These *taddhita* suffixes are added to nouns in various senses. The *taddhita* section begins with the rule, *taddhittāḥ* (A 4.1.76), which is the *adhikāra sūtra* (governing rule) for the whole section, and the *adhikāra* is applicable till the final aphorism of the fifth chapter,

i.e, *niṣpravāṇiśca* (A 5.4.160). The definition of *taddhita* is not provided by *Pāṇini* but is based on the rule *tasmai hitam*(A 5.1.5), which means 'beneficial for that' (Bhate 1989). Here, the word *'tat'*(that) refers to the nominal base to which the *taddhita* suffixes are added. Each rule in the *taddhita* section prescribes the addition of a suffix after a nominal stem denoting a certain meaning. Therefore, the *taddhita* section in *Aṣṭādhyāyī* lies under the morphological component of *Pāṇinian* grammar. However, upon closer examination of the *taddhita* section, we find that it provides morphological, syntactic, and semantic specifications (Bhate 1989). For instance, in the rule *tasya apatyam* (A 4.1.92), the morphological specification will be provided by a nominal stem selected in the place of the pronoun *'tad'*. The genitive ending in the pronominal form *tasya* provides us with the syntactic information that the suffix *'aṇ'* will be added after the nominal base ending in the genitive case. The word *apatyam* provides us with semantic information that suggests the derived form means 'offspring'.

Despite the linguistic importance of the *taddhita* section in Sanskrit grammar, computational modelling of this domain remains limited. Existing computational tools for Sanskrit morphology (Saṁsādhanī[1], Sanskrit Heritage[2], etc.) have significantly contributed to noun and verb generators, compound generators, *kṛt* generators, and sandhi processing. The *taddhita* section, due to its vast coverage, hierarchical organisation, and heavy reliance on semantic and lexical constraints, has not yet been implemented as a complete, transparent derivational system. In particular, no existing model on *taddhita* faithfully reconstructs the internal rule ordering, utsarga–apavāda hierarchy, gaṇa-based restrictions, and stepwise *prakriyā* discussed in the Pāṇinian tradition. This gap limits both computational accuracy and pedagogical utility, emphasizing the need for a systematic, glass-box computational approach to *taddhita* derivation.

To address this gap, the present work proposes a transparent, rule-based computational model for *taddhita* derivation, with a focused implementation of the *Apatyādhikāra* section of the *Aṣṭādhyāyī* (A 4.1.92–A 4.1.178). The system reconstructs the internal derivational logic of *Pāṇinian* grammar by integrating a structured, machine-readable database of *taddhita* rules, a *gaṇa*-based lexical repository, and a procedural *Prakriyā* Engine that applies grammatical operations in the order prescribed by the Indian Grammatical Tradition. The proposed model functions as a 'glass-box' system, explicitly displaying every intermediate stage of derivation, including suffix selection, *it-sañjā* processing, *āgama* insertion, *ādeśa* operations, *aṅgakārya, padakārya,* and *sandhi* operations, along with the corresponding *sūtra* references. The model is evaluated on a curated set of 175 *Apatyādhikāra* examples from traditional grammatical texts of Kāśikāvṛtti and Siddhāntakaumudī, achieving a high amount of accuracy, thereby demonstrating both the computational possibility of modelling the *taddhita* section and the pedagogical value of transparent derivational systems.

The structure of this paper is as follows. In Section 2, we review the relevant theoretical and computational work related to Sanskrit morphology and *taddhita* derivation. Section 3 outlines the overall architecture of the proposed system, including the rule database, the lexical *gaṇa* repository, and the mechanism for rule selection, and a detailed account of the procedural *Prakriyā* Engine and the workflow for *taddhita* generation. Section 4 discusses the evaluation methodology and presents the results and the limitations of the current model. Section 5 discusses about the future work. Lastly, Section 6 wraps up the paper with a conclusion of the study.

[1] `https://sanskrit.uohyd.ac.in/scl/` (Accessed on 5th December 2025).
[2] `https://sanskrit.inria.fr/` (Accessed on 5th December 2025).

2 Related Work

2.1 Theoretical Framework

The *taddhita* section follows the core architecture of the *Aṣṭādhyāyī*, based on the principles of *anuvṛtti* and *utsarga-apavāda* (Bhate, 1987, 1989; Deo, 2007; Krishna and Goyal, 2016). The concept of *anuvṛtti* plays a crucial role in enhancing efficiency within the *Aṣṭādhyāyī*. Drawing upon information from previous rules effectively eliminates redundancy, allowing for a more streamlined approach to grammatical construction. This special feature of *anuvṛtti* helps *Pāṇini* to achieve brevity (Kulkarni, 2009). On the other hand, the *utsarga-apavāda* system plays a crucial role in shaping our understanding of *Aṣṭādhyāyī* by distinguishing between general and specific rules that facilitate the formation of a wide array of words (Cardona, 1970; Kiparsky, 1991; Deo, 2007). Apart from these two systems, the *taddhita* section has two more prominent features, i.e, *pratyayādhikāras* and *arthādhikāras*, which highlight *Pāṇini's* genius (Bhate 1989). Therefore, we can say that the *taddhita* section in the *Aṣṭādhyāyī* is mainly governed by three types of rules:

1. *Pratyayādhikāras*: The section governed by suffixes. (e.g., A 4.1.83 *prāg dīvyataḥ aṇ*)
2. *Arthādhikāras*: The section governed by meanings/ semantic aspects (e.g., A 4.1.92 *tasyāpatyam*).
3. Special rules addressing specific cases.

These levels form a hierarchical structure where affixation and semantics intersect. In the *taddhita* section, a single suffix may express multiple meanings (polysemy), and a single meaning is sometimes denoted by multiple suffixes. Building upon this linguistic foundation, Ashwini Deo (2007) demonstrated that the *taddhita* system, as a single-inheritance hierarchy, is similar to the concept of object-oriented structures in computational linguistics. Each rule in a particular domain (*adhikāra*) inherits semantic properties from the default rule at the higher level unless overridden by a more specific rule. So, this proves to be the combination of inheritance and *sāmānya-viśeṣa*. Furthermore, Deo states that the word form and the meaning are treated separately but are systematically connected through inheritance paths. This enables economy of representation where general rules encode defaults and specific rules add exceptions. This structure avoids redundancy and elegantly captures Sanskrit's derivational morphology.

The studies of Bhate (Bhate, 1989) and Deo (Deo, 2007) reveal that the *taddhita* section is not a random collection of suffixes but a lexical network connected to semantic indexing and governed by inheritance, hierarchy, and economy. While Bhate and Deo explain the theoretical organization of the *taddhita* rules, Amrith Krishna and Pawan Goyal (2015), in their study, bring the *taddhita* structure to computational form. The model is completely based on the object-oriented approach. It follows the inheritance and hierarchy of the *taddhita* system and also incorporates the conflict resolution process based on A 1.4.2 *vipratiṣedhe param kāryam* and the *asiddhatva* principle. The model achieves a high level of accuracy and follows the *Pāṇinian* procedure for *taddhita* generation. This model demonstrates that *Pāṇini's* design can be procedurally executed without altering its internal logic.

2.2 Computational Approaches to Sanskrit Morphology

Over the past few decades, several computational efforts have attempted to model different components of Sanskrit grammar, drawing inspiration from *Pāṇini's Aṣṭādhyāyī*. Amba Kulkarni's contributions have been instrumental in bridging traditional Indian linguistic theories and computational implementation. Kulkarni has developed a toolkit named 'Saṁsādhanī'[3] for the Sanskrit language. The toolkit consists of a morphological generator and analyzer, *sandhi* joiner

[3] `https://sanskrit.uohyd.ac.in/scl/` (Accessed on 5th December 2025).

and a *sandhi* splitter, compound generator (Satuluri and Kulkarni, 2013) that helps the user to generate and analyse various morphological components of the *Pāṇinian* grammar. Another major contribution is the Sanskrit Heritage Platform[4] developed by Gérard Huet. The system integrates lexical, morphological, and syntactic tools. The Heritage platform serves as a lexical as well as morphological analyser and a generator. It also includes a segmenter, which is capable of carrying out tasks like *sandhi viccheda*. The platform also contains the Heritage Sanskrit-French dictionary and a digital version of the Monier-Williams Sanskrit-English dictionary.

Significant efforts have been made to develop models based on the *Aṣṭādhyāyī*. Goyal et al. (2009), developed an *Aṣṭādhyāyī* simulator that is based on the concept of data spaces. The model focuses on *sūtras* like *pūrvatrāsiddham* (A 8.2.1), *asiddhavadatrābhāt* (A 6.4.22), and *ṣatvatukorasiddhaḥ* (A 6.1.86), which are crucial in determining rule ordering and conflict resolution. A more refined model was proposed by Subbanna and Varakhedi (2010), who designed a computational model based on the *Aṣṭādhyāyī*, which operates as a set of condition-action rules with a mechanism controlled by meta-rules. They employed the concepts of *vipratiṣedha* and *asiddhatva* (non-applicability) as a tool for conflict resolution by utilising a mathematical filter.

Apart from this, Pavankumar Satuluri (2015) developed a Sanskrit Compound Generator. This tool focuses on automatically generating Sanskrit compounds and follows the order of operations according to the Indian grammatical tradition. The generator focuses on the syntactic and semantic aspects of compound generation and also provides a provision for the user to provide extralinguistic information as needed. Patel and Katuri (2015) developed an open-source *subanta* generator called *Prakriyāpradarśinī*[5] which shows the subanta generation according to the *Pāṇinian* procedure. Amrith Krishna (2019) designed a data-driven Natural Language Processing model for a low-resource language like Sanskrit. The attempt successfully combines linguistic information from Sanskrit grammar, lexical networks, and distributional information into data-driven models for word segmentation and word formation. The model also performs tasks, such as compound type identification and identification of derivational nouns, which are essential for processing Sanskrit texts due to the typological characteristics of the language. The latest addition to these computational tools is Ambuda[6], a fast *prakriyā* generator capable of generating various noun and verb forms with minimal computing time.

Taken together, existing computational approaches suggest that it is possible to model Sanskrit morphology and aspects of *Pāṇinian* derivation. However, none of the available systems provide a complete, glass-box implementation of a full *taddhita* domain that simultaneously includes hierarchical rule selection, lexical *gaṇa* constraints, semantic conditioning, and transparent derivational order. This gap motivates the present work, which aims to computationally reconstruct the *Apatyādhikāra* in a manner that is both faithful to the grammatical tradition and transparent in its operational details.

3 System Architecture

The proposed system is designed as a transparent, rule-based generator for *taddhita* derivation, with a focused implementation of the *Apatyādhikāra* section of the *Aṣṭādhyāyī*. The architecture based on the internal structure of the *taddhita* section, consisting of *Pratyayādhikāras* and *Arthādhikāras*. It consists of three core components: (i) a *Taddhita* Rule Database, (ii) a *Gaṇa*-based Lexical Repository, and (iii) a procedural *Prakriyā* Engine. Together, these components enable systematic rule selection, controlled derivation, and stepwise generation of *taddhitānta* forms.

[4] `https://sanskrit.inria.fr/` (Accessed on 5th December 2025).
[5] `https://api.sanskritworld.in/` (Accessed on 5th December 2025).
[6] `https://ambuda.org/` (Accessed on 5th December 2025).

3.1 Design Philosophy

The model is based on the assumption that *Pāṇinian* grammar is completely procedural and hierarchical in nature. The proposed model follows a rule-ordered framework that represents traditional *prakriyā*-based analysis, not just viewing derivation as a complex transformation from input to output. This approach focuses on transparency, interpretability, and alignment with grammatical tradition.

Another main focus is to transparently model how rules interact. Knowing that the *taddhita* derivation is highly influenced by semantic conditions, lexical constraints, and *utsarga–apavāda* relationships, the framework is designed to clearly highlight rule precedence and applicability at each step. Therefore, the system acts as a glass-box model, revealing the final output as well as the grammatical reasoning that leads to it.

3.2 Overview of the Architecture

The system accepts a nominal base (*prātipadika*) and a semantic specification (such as *apatyam*) as input. The Rule Engine begins by filtering the rule database according to semantic, lexical, and morphological criteria to select the correct *taddhita* rule. After selecting a rule, the *Prakriyā* Engine executes the specific grammatical operations in a particular order to produce the final derived form, including all the intermediate steps. In this way, the output provides not only the final *taddhitānta* form but also a detailed derivational analysis that resembles the *Pāṇinian prakriyā.*

3.3 *Taddhita* Rule Database

The *taddhita* rule database currently consists of nearly 60 rules beginning from *prāgdīvyato'ṇ* (A 4.1.83), which is the beginning of the *'aṇ' pratyayādhikāra*, and it extends to the rule *sālvāvayavapratyagrathakalakūṭāśmakādiñ* (A 4.1.173) covering all the major rules of the *apatyādhikāra* section, which is the first *arthādhikāra* of the *pratyayādhikāra prāgdīvyato'ṇ.* The entire stretch consists of nearly 90 rules, out of which 60 rules have been programmed. We have skipped nearly 30 rules because the section contains many rules dealing with *saṁjñās* (such as *tadrāja saṁjñā*), *pratyaya luk* and requires some complex semantic and extralinguistic conditions. In the current database, we handle three semantic relations: *apatyam, gotrāpatyam* and *yuvāpatyam.*

To simplify machine-readable processing, every rule from the *Aṣṭādhyāyī* is represented as a structured dictionary object. This framework organizes the various conditions of *Pāṇinian sūtras* into five main attributes. This helps the Rule Engine to evaluate the conditions properly. The rules have been encoded in wx notation[7].

1. ***Sūtra*** **(Rule Identifier)**

   Type: String

   This string stores the unique reference index and the traditional Sanskrit text of the rule (e.g., "4.1.112 *SivAxiByaH aN"*). This feature acts as the basic element for the rule and allows the system to track the exact source of a derivational step in the final output (*prakriyā*).

2. ***Pratyaya*** **(suffix)**

   Type: String

   It specifies the unique suffix prescribed by the rule (e.g., "aN"). This value is stored in its *upadeśa* (instructional) form. The meta-linguistic markers (*anubandhas*), such as N or k, are preserved for later use, as they trigger subsequent grammatical operations (e.g., the *ṇit* and *kit* suffixes trigger *ādi vṛddhi*).

[7] `https://en.wikipedia.org/wiki/WX_notation` (Accessed on 5th December 2025).

3. ***Artha* (Semantic Relations)**

   Type: String

   This attribute defines the specific semantic domain in which the rule is valid (e.g., "*apawyam*" for offspring). This acts as a semantic filter. During the derivation process, the engine compares this value with the user's intended meaning. If the user request is out of the semantic domain (e.g., *samūha* or "collection"), it is automatically disqualified. This prevents semantic overgeneration. Currently, the model handles three semantic domains: *apatyam, gotrāpatyam*, and *yuvāpatyam*.

4. **Conditions**

   Type: List of Tuples [(Operator, Operand)]

   It contains a dynamic list of certain logical conditions required to trigger the rule. Each condition is stored as a tuple that contains an operator and a target value.

   Example: [(*"in_gaṇa", "SivAxi"*)]

   In this example, the operator **"in_gaṇa"** instructs the engine to check the Lexical Repository and verify if the input stem exists in the **Śivādi** list. There are other operators like morphological checks (e.g., ends_with) or phonological conditions (e.g., is_vowel_initial). This flexible structure allows the system to model complex **Pāṇinian** conditions without changing the core logic of the engine.

5. **Advanced Features**

   Apart from regular conditional checks, the rule schema includes special features to deal with complex morphological transformations such as *āgama* (augment insertion), *ādeśa* (substitution), and *vikalpa* (optionality). These attributes allow the rule engine to strictly maintain linguistic accuracy.

   5.1 ***Āgama* (Augment)**

   Type: Tuple (Augment, Marker_Type)

   It specifies any phonological augment prescribed by the rule (e.g., (***"kuk," "yuṭ"***)). This information of *āgamas* is then passed to the *prakriyā* engine. Based upon the augment, it triggers the *Pāṇinian* placement logic defined in A 1.1.46 *ādyantau ṭakitau*. This explicit encoding ensures that augments like *yuṭ* (in *Chāgyāyani*) or *kuk*(in *vākinakāyani*) are inserted at the correct linguistic positions before any sandhi operations occur.

   5.2 ***Ādeśa* (Substitution)**

   Type: Dictionary or Tuple

   This attribute triggers particular substitutions where the application of a suffix triggers a modification in the *prakṛti*. While most substitutions are handled by the phonological engine, specific rules require unique replacements. This attribute allows a rule to carry its own local modification logic (e.g., replacing the final vowel of a specific stem) without requiring a universal rule change. This helps in effectively modeling the *nipātana* (irregular form) conditions found in the *Aṣṭādhyāyī*.

   5.3 ***Vikalpa*** (**Optionality**)

   Type: Boolean (True/False)

   Indicates whether the rule is prescriptive (nitya) or optional (vibhāṣā / anyatarasyām). When set to True, this flag instructs the engine that the rule prescribes vibhāṣā. In the generation process, this allows the system to produce multiple valid outputs. This helps us to capture the vast range of valid usage accepted by grammatical tradition.

   5.4 **Explanatory Notes (display_note)**

Type: String (Optional)

A custom text field containing pedagogical information (e.g., "Specific exclusion/inclusion due to semantic constraint"). Unlike the procedural attributes, which drive the logic, this attribute is purely informational. When the Rule Engine selects a rule with this tag, the string is inserted directly into the derivation trace (*prakriyā*). This allows the "Glass-Box" system to provide human-readable context for complex rule applications or specific exclusions. This helps in a better understanding of the generated output for students and researchers.

Example 1: Rule representation in Rule database

```
Rule {
  sūtra    : "4.1.158 vAkinAxInAm kuk ca"
  pratyaya : PiF          // Phiñ
  artha    : apatyam      // Semantic scope: offspring

  conditions :
      in_gana(vAkinAxi)   // Stem must belong to Vākinādi gaṇa

  āgama :
    augment  : kuzk       // kuk
    target   : prakṛiti   // Applied to the base stem
    position : suffix    // Kit-marker conditioned augmentation

  display_note :
    "(uxIcAmAcAryANAM mawena)"   // Contextual note for the user
}
```

3.4 *Gaṇa*-based Lexical Repository

Lexical conditioning plays a crucial role in the *taddhita* section. This is because many rules apply only to stems belonging to specific *gaṇas* (e.g., A 4.1.158 applies only to the stems belonging to *vākinādi gaṇa*). To address this, the system comprises of a *Gaṇa*-based Lexical Repository that stores *gaṇapāṭha* lists as machine-readable lexical sets. During rule selection, the Rule Engine checks this repository to verify whether the input satisfies *gaṇa*-specific conditions. This mechanism ensures accurate modeling of lexically restricted rules and prevents overgeneration. The repository is implemented as a key-value hash map, where the keys represent the traditional *Gaṇa* names (e.g., *gargAxi, SivAxi*) and the values are implemented as Hash Sets that contain nominal stems (*prātipadikas*).

This design choice offers two major advantages:

1. **Lookup Efficiency:** The use of hash sets helps us to check whether the word is present in a specific gaṇā (e.g., "Is 'vAkina' in 'vAkinAxi'?") in constant time, regardless of the list's size. This is essential for performance, as the Rule Engine must search through these lists multiple times during the derivation of large corpora.

2. **Handling of Open Lists (*Ākṛtigaṇa*)**: *Pāṇini's gaṇapātha* has two types of *gaṇas*, *paṭhita-gaṇa*(closed lists) and *ākṛti-gaṇa*(open-ended lists). The repository architecture allows for the dynamic expansion of open lists without altering the core codebase. This enables the system to accommodate new words as they are encountered or defined by the user.

The Lexical Repository functions as a passive validator for the Rule Engine. When a rule states a lexical condition (e.g., [("in_gaṇa", "vAkinAxi")]), the engine checks the repository. If the input stem is found within the specified set, the condition is set to True and allows the

apavāda rule to trigger. If the check fails, the engine bypasses the *apavāda* rule and defaults to the *utsarga* rule. This helps in strictly maintaining the *Pāṇinian* structure through lexical validation.

Example 2: Representation of a *gaṇa*-based lexical class

```
Gaṇa {
  name  : vAkinAxi

  stems :
    - vAkina
    - gAreXa
    - kArkata
    - kAka
    - laMkA
}
```

3.5 Rule Selection Mechanism

The core intelligence of the system lies in the Rule Engine. The Rule Engine implements a "Filter-Rank-Select" algorithm which is based on the *Pāṇinian* principle of *Utsarga-Apavāda* (General-Exception Hierarchy). Compared to statistical models that predict morphology based on probability, this engine operates systematically by selecting the single most appropriate rule based on the specificity hierarchy. The rule selection process is triggered when the user provides a nominal base (*prakṛti*) and a target semantic meaning (*artha*). The engine navigates through the rule database using the following logic:

1. ***Arthādhikāra* Check (Semantic Filtering):** The engine first eliminates any rule whose *artha* attribute does not match the user's input. For example, if the user requests an *apatyam* (offspring) derivation, rules belonging to the *gotrāpatyam* and *yuvāpatyam* domains are immediately filtered out. This strictly regulates the semantic boundaries defined by *Pāṇini's arthādhikāras*[8].

2. **Condition Validation:** For the remaining semantically valid rules, the engine evaluates their specific conditions against the input stem. This involves two parallel checks:

   - **Lexical Check:** Asking the *Gaṇa* Repository to verify membership in specific lists (e.g., is the stem in *Gargādi*?).
   - **Phonological Check:** Verifying phonological properties (e.g., does the stem end in *'a'*?).

3. ***Utsarga-Apavāda* (Hierarchical Prioritization):** The crucial step is the ordering of valid rules. The database is structured such that the *Apavāda* rules(exception rules) are evaluated before the *Utsarga* rules (general rules).

   - **Step A:** The engine checks for high-specificity rules (e.g., *Gargādibhyo yañ*). If the conditions are met, this rule is selected immediately, blocking all other rules.
   - **Step B:** If no specific exception is found, the engine falls back to the general rule (e.g.,*tasyāpatyam*).

   This linear, organized search ensures that the system follows the *"utsarga-apavāda"* (Principle of Specificity Hierarchy) effectively.

[8] *gotrāpatyam* and *yuvāpatyam* are the sub-classes for the *apatya arthādhikāra*, but are special semantic relations different than *apatya*. Hence, we have made a separate *artha* input for *gotrāpatyam* and *yuvāpatyam*

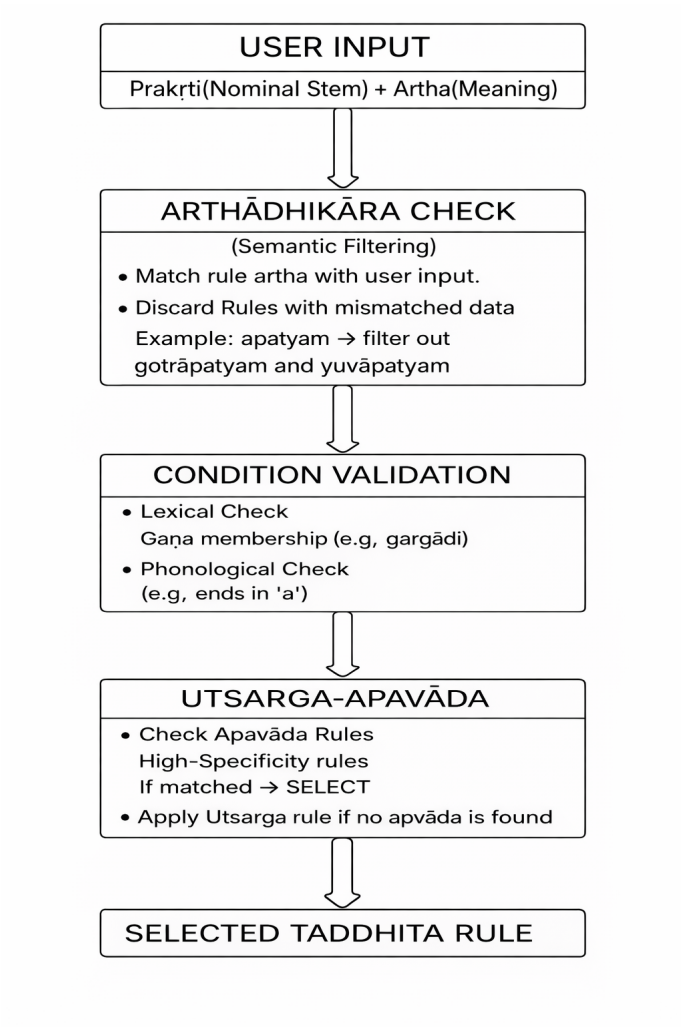

Figure 1: Rule Selection Mechanism

3.6 *Prakriyā* Engine

The *Prakriya* Engine is one of the most important components of the model. It carries out the crucial task of generating the whole *taddhita prakriyā* according to the *Pāṇinian* procedure. The *Prakriyā* Engine functions as a state machine that transforms the input through a sequence of grammatical operations that follow a strict procedural order of the *Aṣṭādhyāyī*.

The *Prakriyā* Engine receives the information of the input (*prakṛti + artha*) and the selected rule by the Rule Engine, and then carries out the derivation in six different stages. The order in which the rules are applied is pre-decided manually according to the Aṣṭādhyāyī. Within a specific type of operation, which rule to trigger is decided by the machine automatically based on the conditions

1. ***Prakṛti Ādeśā* (Stem Substitution):**

   There are some specific rules that prescribe *ādeśa* of the *prakṛti* along with a *taddhita* suffix. In such cases, the *prakriyā* engine applies the prescribed *ādeśa* to the *prakṛti*. The *prakṛti ādeśa* attribute consists of two different methods:

   - Internal Replacement: Some rules, like A 4.1.97[9], prescribe *ādeśās* to a particular part of the nominal stem. It replaces a specified part of the *prakṛti* as prescribed by the rule.

[9] A 4.1.97 *sudhāturakaṅ ca*, prescribes *akaṅ ādeśā* for the word *sudhātṛ*. Replaces *'ṛ'* with *'ak'*

- Full Replacement: Some rules, such as A 4.1.116[10], prescribe the *ādeśā* for the entire *prakṛti*. It replaces the entire nominal stem with the prescribed *ādeśā* according to the rule.

2. ***Āgama* (Augment):**

   In the next stage, the engine evaluates whether the selected rule prescribes some *āgama*. The *āgama* information is stored in the rule itself. As mentioned earlier, the *āgamas* are distinguished as *'ṭit'* and *'kit' āgamas*. The placement of the *āgamas* is based on the rule A 1.1.46[11]. At this stage, only the prescribed *āgama* is identified, the insertion of *āgamas* is done at a later stage.

3. ***It-saṁjñā lopa* (Processing of Markers):**

   This section is built on the basis of all the *it-saṁjñā* related rules (A 1.3.2-A 1.3.8), and the engine is completely capable of handling the process of *it-saṁjñā* on the *prakṛiti*, added *taddhita* suffix, and *āgamas*. The information of *it-varṇas*(deleted markers) is stored for future operations like *ādivṛddhi* and correct placement of *āgamas*.

4. ***Subluk and bha/pada Saṁjñā* (Deletion of cases and State Assessment):**

   The next step in *taddhita* generation is the *vibhakti lopa* or *subluk*. At this stage, the case suffixes are deleted, thus exposing the *prakṛti* to the *taddita* suffix. As the *prakṛti* is exposed to the *taddhita* suffix, the next operation is to assign *'bha'* or *'pada' saṁjñā*. If the suffix begins with a vowel or *'y,' bha-saṃjñā* is assigned according to the rule A 1.4.18 *yaci bham*. If *bha saṁjñā* is not assigned, the engine assigns the *pada saṁjñā*.

5. **Core Derivational Operations:**

   This stage consists of a set of operations that take place on the *prakṛti* as well as the *taddhita* suffix.

   - ***Pratyaya Adeśā:*** Certain *taddhita* suffixes undergo substitution (eg *'ṭha'* is replaced by *'ika'*). This operation is performed on the basis of various rules like A 7.1.2[12] and A 7.3.50[13].
   - ***Āgama* Placement:** The *āgama* identified as *'ṭit'* is placed at the beginning of the suffix, and the *'kit' āgama* is placed at the end of the *prakṛti* according to A 1.1.46 *ādyantau ṭakitau*. For instance, in *cāgyāyani*, '*yuṭ*' *āgama* is prescribed, and it is placed at the beginning of the suffix as it is a *'ṭit' āgama*. Similarly, in *vakinakāyani*, '*kuk*' *āgama* is prescribed, and it is placed at the end of the *prakṛti* as it is a *'kit' āgama*.
   - ***Adivṛddhi***: The first vowel of the *prakṛti* undergoes *vṛddhi*, if the stored deleted markers from the suffix are either *'ṇ', 'ñ' or 'k',* according to the rules A 7.2.117[14] and A 7.2.118[15].
   - ***Aṅga and Pada Kārya:*** Based on the assigned *'bha'* or *'pada' saṁjñā*, the engine carries out various operations on the *prakṛti*. If the *prakṛti* has *bha-saṁjñā, aṇga* operations are carried out according to A 6.4.148[16], and if the *prakṛti* has *pada-saṁjñā, pada* operations are carried out according to A 8.2.23[17].

[10] A 4.1.116 *kanyāyāḥ kanīna ca*, prescribes full replacement of the stem *'kanyā'* with *'kanīna'*

[11] *ādyantau ṭakitau*. If the marker is *ṭit*, the augment is attached to the beginning of the *prakṛiti*. If the marker is *kit*, the augment is attached to the end of the *prakṛti*.

[12] A 7.1.2 *āyaneyīnīyiyaḥ phaḍhakhachaghāṁ pratyayādīnām*

[13] A 7.3.50 *ṭhasyekaḥ*

[14] A 7.2.117 *taddhiteṣvacāmādeḥ*

[15] A 7.2.118 *kiti ca*

[16] A 6.4.148 *yasyeti ca*

[17] A 8.2.23 *saṁyogāntasya lopaḥ*

6. **Final Phonological Operations:**

   At the final stage of the derivation, small phonological operations like sandhi of the *prakṛti* and the suffix, *ṇatva* (changing of *'n'* to *'ṇ'*), are carried out.

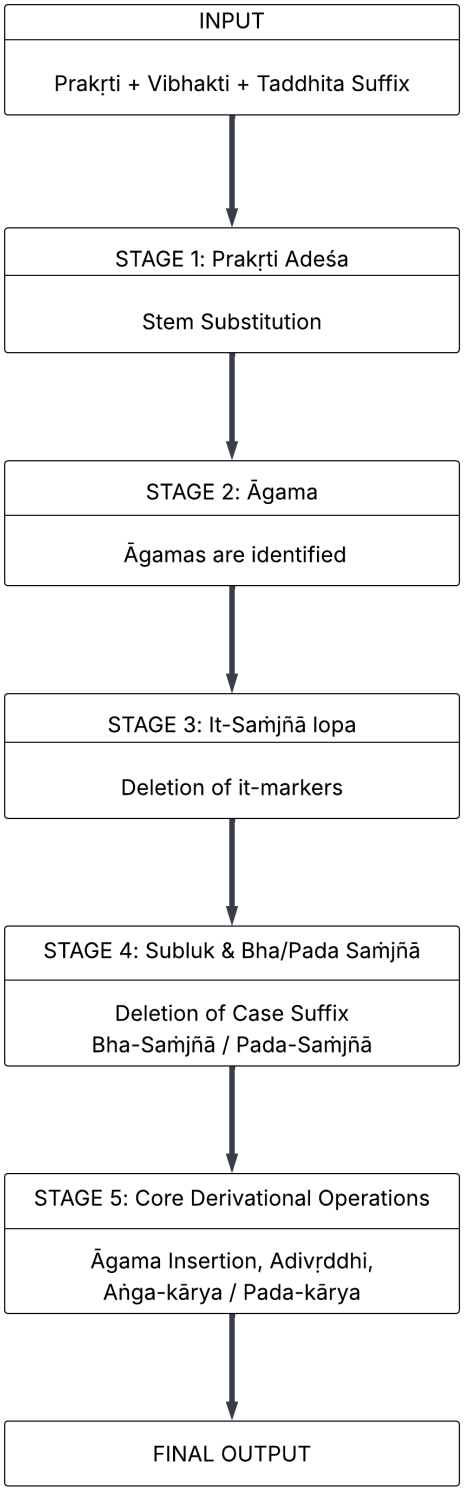

Figure 2: Prakriyā Engine Workflow

4 Evaluation and Results

This section highlights the evaluation methodology and the results of the proposed *taddhita* generator model.

4.1 Dataset for the Experiment

The model was tested on the dataset of 175 *taddhitānta* forms related to the *Apatyādhikāra* section(A 4.1.83 - A 4.1.178) of the *Aṣṭādhyāyī*. The examples were taken from the traditional grammatical texts of *Kāśikāvṛtti* and the *Siddhāntakaumudī*. To test each rule, a minimum of five examples were tested from the traditional texts, except for the rules that deal with one or two specific cases. For the rules requiring the *gaṇapātha*, random words were tested from that particular *gaṇa*. Each tested example consisted of a nominal base and a semantic relation (*apatyam, gotrāpatyam, yuvāpatyam*). For each input, the system generated a *taddhitānta* form along with the complete *prakriyā*, providing an instance for the *Pāṇinian* rules that were applied during the entire process.

4.2 Evaluation Criteria

The generated output was saved in a log file and evaluated by the experts of *Pāṇinian* grammar. The output was considered correct if it satisfied both of the following conditions:

1. *Prakriyā* (Derivational Procedure): The entire *prakriyā* of the derivation was examined thoroughly, considering the proper selection of rule, *āgama, ādeśa,* and the order of operations according to the *Aṣṭādhyāyī*.

2. Final Output: The final output form generated by the system was evaluated against the examples of the traditional grammatical texts.

4.3 Results

The system was able to generate accurate *taddhitānta* forms for all 175 examples within the *apatyādhikāra* domain, achieving 100% accuracy, demonstrating the effectiveness of the proposed rule-based approach that models the *Apatyādhikāra* domain of the *taddhita* section. The reason for this high accuracy is the arrangement of *utsarga-apavāda* rules in the *Aṣṭādhyāyī*. This arrangement ensures that if there is no specific rule for a situation, the general rule applies. The high accuracy indicates that the combination of structured rule representation according to *utsarga-apavāda*, *gaṇa*-based lexical dictionary, and a stepwise-designed *prakriyā* closely aligns with the grammatical principles encoded in the *Aṣṭādhyāyī*.

4.4 Illustrative Examples

The output was converted from WX to devanāgari for better readability of the users.

Derivation for: उपगु (अपत्यम्) with suffix अण्

0. Input: **उपगु (अपत्यम्)**
1. Initial Form: **उपगु + ङस् + अण्** ..... (by 4.1.92 **तस्यापत्यम्**),
   (**तद्धितसंज्ञा** assigned by 4.1.76 **तद्धिताः**)
2. **उपगु + ङस् + अण्** ..... (**प्रातिपदिकसंज्ञा** assigned by 1.2.46 **कृत्तद्धितसमासाश्च**)
3. **उपगु + अस् + अ** ..... (**इत्-लोपः** by 1.3.8 **लशक्वतद्धिते**, 1.3.3 **हलन्त्यम्**)
4. **उपगु + अ** ..... (**विभक्ति (अस्) लुक्** by 2.4.71 **सुपो धातुप्रातिपदिकयोः**)
5. **उपगु इत्यस्य भ-संज्ञा** ..... (by 1.4.18 **यचि भम्**)
6. **औपगु + अ** ..... (**आदिवृद्धिः** by 7.2.117 **तद्धितेष्वचामादेः**)
7. **औपगो + अ** ..... (**अङ्ग-कार्यम्** by 6.4.146 **ओर्गुणः**)
8. **औपगव** ..... (**सन्धिः** by 6.1.78 **एचोऽयवायावः**)

⟶ **FINAL STEM: औपगव**

Derivation for: गर्ग (गोत्रापत्यम्) with suffix यञ्

0. Input: **गर्ग (गोत्रापत्यम्)**
1. Initial Form: **गर्ग** + **ङस्** + **यञ्** ..... (by 4.1.105 **गर्गादिभ्यः यञ्**),
   (**तद्धितसंज्ञा** assigned by 4.1.76 **तद्धिताः**)
2. **गर्ग** + **ङस्** + **यञ्** ..... (**प्रातिपदिकसंज्ञा** assigned by 1.2.46 **कृत्तद्धितसमासाश्च**)
3. **गर्ग** + **अस्** + **य** ..... (**इत्-लोपः** by 1.3.8 **लशक्वतद्धिते**, 1.3.3 **हलन्त्यम्**)
4. **गर्ग** + **य** ..... (**विभक्ति (अस्) लुक्** by 2.4.71 **सुपो धातुप्रातिपदिकयोः**)
5. **गर्ग इत्यस्य भ-संज्ञा** ..... (by 1.4.18 **यचि भम्**)
6. **गार्ग** + **य** ..... (**आदिवृद्धिः** by 7.2.117 **तद्धितेष्वचामादेः**)
7. **गार्ग्** + **य** ..... (**अङ्ग-कार्यम्** by 6.4.148 **यस्येति च**)

⟶ **FINAL STEM: गार्ग्य**

Derivation for: कन्या (अपत्यम्) with suffix अण्

0. Input: **कन्या (अपत्यम्)**
1. Initial Form: **कन्या** + **ङस्** + **अण्** ..... (by 4.1.116 **कन्यायाः कनीन च**),
   (**तद्धितसंज्ञा** assigned by 4.1.76 **तद्धिताः**)
2. **कन्या** + **ङस्** + **अण्** ..... (**प्रातिपदिकसंज्ञा** assigned by 1.2.46 **कृत्तद्धितसमासाश्च**)
3. → **कनीन** + **ङस्** + **अण्** ..... (by 4.1.116 **कन्यायाः कनीन च** – **कनीन आदेशः**)
4. **कनीन** + **अस्** + **अ** ..... (**इत्-लोपः** by 1.3.8 **लशक्वतद्धिते**, 1.3.3 **हलन्त्यम्**)
5. **कनीन** + **अ** ..... (**विभक्ति (अस्) लुक्** by 2.4.71 **सुपो धातुप्रातिपदिकयोः**)
6. **कनीन इत्यस्य भ-संज्ञा** ..... (by 1.4.18 **यचि भम्**)
7. **कानीन** + **अ** ..... (**आदिवृद्धिः** by 7.2.117 **तद्धितेष्वचामादेः**)
8. **कानीन्** + **अ** ..... (**अङ्ग-कार्यम्** by 6.4.148 **यस्येति च**)

⟶ **FINAL STEM: कानीन**

4.5 Limitations

Despite achieving a 100% verification accuracy on the tested examples, the model has some limitations. The current model focuses only on the *apatyādhikara* domain of the entire *taddhita* section. Other semantic domains like *tasya samūhaḥ* or *tasya nivasaḥ* have not yet been implemented. Extending the scope beyond *apatyādhikāra* will require more rule encoding. We have not yet integrated the available *sandhi* module of *Samsādhanī*. The current model has some helper functions for sandhi processing that carry the necessary sandhi operations for *taddhita* generation. Another limitation of the model is due to the *ākṛtigaṇas* in *Pāṇinian* grammar. *Ākṛtigaṇas* are open-ended lists provided by *Pāṇini* in the *gaṇapātha*. This can cause a derivational error due to the absence of a particular word in the prescribed list. Another key limitation of the model is that it cannot handle the rules that require complex semantic, pragmatic, or extralinguistic information. There are certain taddhita derivations that are based on speaker intention and world knowledge. Handling such cases is beyond the scope of the current model.

5 Future Work

The present work has a vast scope for further research. The most immediate direction is the expansion of the rule database to the remaining *arthādhikaras* that come under the *prāgdīvyato'ṇ*

section. This extension beyond *apatyādhikāra* will allow us to test our model on other *arthādhikāras*, providing a good amount of tested corpus. Another important direction is to integrate the available *sandhi* module of *Saṁsādhanī* into the *prakriyā* engine. This may enable us to achieve high accuracy even after the expansion of the rule database. Efforts can also be made to process the examples requiring complex semantic and extralinguistic information, which will improve the efficiency of the model.

6 Conclusion

This paper presents an overview of a transparent, rule-based computational model for the generation of *taddhitāntas*, especially focusing the *apatyādhikāra* domain of the *taddhita* section of the *Aṣṭādhyāyī*. This model follows a glass-box architecture, which shows the entire derivational process along with the output and also provides information about the grammatical rule applied at each stage of the derivation. This makes it unique from other statistical or black-box models. The model follows the *Aṣṭādhyāyī* by replicating the principles of *utsarga-apavāda* for conflict resolution, lists of *gaṇas*, and a staged *prakriyā* engine following the accurate order of operations according to the *Pāṇinian* procedure.

The evaluation shows that the model has a high accuracy rate on the test cases taken from the traditional texts in the *apatyādhikāra* domain. This proves the ability of the model to handle all the grammatical operations smoothly and derive correct grammatical forms.

Overall, the study contributes to filling a major gap in the field of Sanskrit Computational Linguistics by providing a base for *taddhita* generation, which seems to be the largest section of the *Aṣṭādhyāyī*. It also provides a solid foundation for extending the scope for the entire *taddhita* section of *Pāṇinian* grammar.

References

K. V. Abhyankar. 1860. *Vyakarana Mahabhasya with text and Marathi Translation (Part IV).* Deccan Education Society, Pune.

V. S. Abhyankar. 1965. *Siddhāntakaumudī with the Commentaries Bālamanoramā and Tattvabodhinī.* Chowkhamba Sanskrit Series Office, Varanasi.

Tanuja Ajotikar, Anuja Ajotikar, and Peter M. Scharf. 2016. Some issues in formalizing the Aṣṭādhyāyī. In *Sanskrit and computational linguistics: Select papers presented in the 'Sanskrit and the IT world' section at the 16th World Sanskrit Conference*, pages 103–124.

Saroja Bhate. 1989. *Pāṇini's Taddhita Rules.* University of Poona, Pune.

Saroja Bhate. 2006. The meaning-adhikāras in the Taddhita section of the Aṣṭādhyāyī: An analysis. *Indo-Iranian Journal*, 49(2):93–110.

George Cardona. 1970. On Pāṇini's rules of grammar. *Journal of the American Oriental Society*, 90(1):40–74.

George Cardona. 1997. *Pāṇini: A survey of research.* Motilal Banarsidass, Delhi.

Ashwini Deo. 2006. Derivational morphology in inheritance-based lexica: Insights from Pāṇini. In *Proceedings of the 28th Annual Meeting of the Berkeley Linguistics Society*, volume 28, pages 75–86.

Madhav Deshpande. 1993. *The philosophy of grammar and linguistics in ancient India.* Motilal Banarsidass, Delhi.

Pawan Goyal, Amba Kulkarni, and Gérard Huet. 2009. Computer simulation of Aṣṭādhyāyī: Some insights. In *Proceedings of the 5th International Sanskrit Computational Linguistics Symposium*, pages 71–83.

François Grimal, V. Venkataraja Sarma, and S. Lakshminarasimham. 2015a. *Pāṇinīyavyākaraṇodāharaṇakośaḥ Vol. IV: Taddhitaprakaraṇam (Part 1: Aṃśakaḥ - Pāriṣadaḥ).* Rashtriya Sanskrit Vidyapeetha and Institut français de Pondichéry, Tirupati and Pondicherry.

François Grimal, V. Venkataraja Sarma, and S. Lakshminarasimham. 2015b. *Pāṇinīyavyākaraṇodāharaṇakośaḥ Vol. IV: Taddhitaprakaraṇam (Part 2: Pāriṣadam - Hrasiṣṭhaḥ)*. Rashtriya Sanskrit Vidyapeetha and Institut français de Pondichéry, Tirupati and Pondicherry.

Gérard Huet, Amba Kulkarni, and Pawan Goyal, editors. 2009. *Sanskrit Computational Linguistics (Vols. 1–2)*. Springer, Berlin.

Girish Nath Jha, Amba Kulkarni, and S. Shukla. 2009. Inflectional morphology analyzer for Sanskrit. In *Proceedings of the 1st International Sanskrit Computational Linguistics Symposium*, pages 1–10.

S. D. Joshi and J. A. F. Roodbergen. 1991. *Pāṇini's Aṣṭādhyāyī with translation and explanatory notes (Vols. 1–6)*. University of Pune, Pune.

S. D. Joshi and J. A. F. Roodbergen. 2011. *The Aṣṭādhyāyī of Pāṇini with translation and explanatory notes (Vol. 14)*. Sahitya Akademi.

Paul Kiparsky. 1991. Elsewhere in phonology. In S. Hargus and E. M. Kaisse, editors, *Studies in Lexical Phonology*, pages 81–106. Academic Press, San Diego, CA.

Paul Kiparsky. 2009. On the architecture of Pāṇini's grammar. In Gérard Huet and Amba Kulkarni, editors, *Sanskrit computational linguistics*, pages 33–94. Springer, Berlin.

Amrith Krishna and Pawan Goyal. 2016. Towards automating the generation of derivative nouns in Sanskrit by simulating Pāṇini. In *Language Resources and Evaluation Conference (LREC 2016)*, pages 1–7.

Amrith Krishna. 2019. *Addressing language specific characteristics for data-driven modelling of lexical, syntactic and prosodic tasks in Sanskrit*. Ph.D. thesis, Indian Institute of Technology Kharagpur.

Amba Kulkarni and D. Shukla. 2009. Sanskrit morphological analyser: Some issues. In *Proceedings of the 1st International Sanskrit Computational Linguistics Symposium*, pages 200–211.

Malhar Kulkarni. 2009. Phonological overgeneration in the Pāṇinian system. In *Proceedings of the 1st International Sanskrit Computational Linguistics Symposium*, pages 120–128.

Amba Kulkarni. 2013. Sanskrit and computational linguistics. In *Language Technology for Indian Languages*, pages 71–90. Springer, Berlin.

Anand Mishra. 2009. Simulating the Pāṇinian system of Sanskrit grammar. In *Proceedings of the 1st International Sanskrit Computational Linguistics Symposium*, pages 155–167.

Anand Mishra. 2013. *Modelling Aṣṭādhyāyī: An approach based on the methodology of ancillary disciplines (Vedāṅga)*. Rashtriya Sanskrit Sansthan.

D. Patel and S. Katuri. 2015. Prakriyāpradarśinī: An open-source subanta generator. In V. Chaitanya and Amba Kulkarni, editors, *Sanskrit and Computational Linguistics: Selected Papers from the 16th World Sanskrit Conference*. Rashtriya Sanskrit Sansthan.

Pavankumar Satuluri and Amba Kulkarni. 2013. Generation of Sanskrit compounds. In *Proceedings of the 5th International Sanskrit Computational Linguistics Symposium*, pages 22–31.

Pavankumar Satuluri and Amba Kulkarni. 2014. Extra-linguistic information needed for automatic generation of Sanskrit compounds: A study. *International Journal of Dravidian Linguistics*, 43(2):59–76.

Pavankumar Satuluri. 2015. *Sanskrit compound generation: With a focus on the order of operations*. Ph.D. thesis, University of Hyderabad.

Peter Scharf. 2009. Modeling Pāṇinian grammar. *Journal of Indian Philosophy*, 37(1):23–53.

Peter Scharf. 2010. Rule selection in the Aṣṭādhyāyī, or is Pāṇini's grammar mechanistic? *Journal of Indian Philosophy*, 38(6):551–579.

Peter M. Scharf. 2016. An XML formalization of the Aṣṭādhyāyī. In *Sanskrit and computational linguistics: Select papers presented at the 16th World Sanskrit Conference*, pages 77–102.

Peter M. Scharf. 2017. A computational implementation of pāṇini's derivational morphology of sanskrit. In Eleonora Litta and Marco Passarotti, editors, *Proceedings of the Workshop on Resources and Tools for Derivational Morphology (DeriMo)*, pages 93–104, Milano, Italy. EDUCatt.

R. N. Sharma. 2002. *The Aṣṭādhyāyī of Pāṇini: Vol. 1—Introduction to the Aṣṭādhyāyī as a grammatical device*. Munshiram Manoharlal Publishers.

J. F. Staal. 1965. Context-sensitive rules in Pāṇini. *Foundations of Language*, 1(1):83–93.

S. Subbanna and S. Varakhedi. 2009. Computational structure of the Aṣṭādhyāyī and conflict resolution techniques. In *Proceedings of the 1st International Sanskrit Computational Linguistics Symposium*, pages 300–311.

K. Subrahmanyam. 1999. *Four vṛttis in Pāṇini*. Rashtriya Sanskrit Vidyapeetha.

J. S. L. Tripathi and S. Malaviya. 1988. *Kāśikā of Vāmana and Jayāditya with Nyāsa or Vivaraṇa Pañjikā and Padamañjarī with Bhāvabodhinī (Vol. V)*. Tara Book Agencies.

J. S. L. Tripathi and S. Malaviya. 1989. *Kāśikā of Vāmana and Jayāditya with Nyāsa or Vivaraṇa Pañjikā and Padamañjarī with Bhāvabodhinī (Vol. VI)*. Tara Book Agencies.

S. C. Vasu. 1962. *The Aṣṭādhyāyī of Pāṇini (Reprint)*. Motilal Banarsidass.