Modelling the Grammatical Circle of the Pāṇinian System of Sanskrit Grammar

Anand Mishra

Department of Classical Indology
Ruprecht Karls University, Heidelberg, Germany
amishra@ix.urz.uni-heidelberg.de

Abstract. In the present article we briefly sketch an extended version of our previously developed model for computer representation of the Pāṇinian system of Sanskrit grammar. We attempt to implement an antecedent analytical phase using heuristical methods and improve the subsequent phase of reconstitution using the rules of *Aṣṭādhyāyī* by incorporating strategies for automatic application of grammatical rules.

Keywords: Sanskrit, Grammar, Pāṇini, Aṣṭādhyāyī, Computational Linguistics, Modelling

1 Introduction

In the following we propose a computer model for representing the grammatical process of the Pāṇinian system of Sanskrit grammar. In Sect. 2 we describe the circular nature of this grammatical process. It consists of first an analysis of a given linguistic expression into constituent elements and then its reconstitution using a given set of rules. This process, thus starts with a provisional statement and ends in a *saṃskṛta* or perfected expression.

A brief summary of our model for implementing the latter step of reconstitution of linguistic expressions using the rules of *Aṣṭādhyāyī* is presented in Sect. 3. The implementation of the preceding analytical step is now being attempted. The main approach here is to improvise heuristics to guess the constituent elements of a given expression. This is outlined in Sect. 4. The subsequent reconstitutive step as specified by *Aṣṭādhyāyī* is now proposed to include strategies for automatic application of rules in Sect. 5. The main program modules implementing all this are mentioned in Sect. 6.

2 Circular Nature of the Grammatical Process

The *Aṣṭādhyāyī* of Pāṇini is a device to produce linguistic expressions using its constituent elements. It prescribes a set of fundamental components which constitute the language, characterizes them using a number of attributes and specifies rules to form the final linguistic expressions. This process of constructing

linguistic expressions presupposes a process of analysis, as a result of which, Pāṇini[1] had at his disposal fundamental components like *bhū*, *ti*P[2] etc.

The analytical process, beginning with the *pada-pāṭha* of Vedic *mantra*s is not recorded in terms of rules of analysis. Thus, MADHAVA DESHAPANDE [3] notes that Pāṇini's grammar "does not provide us with analytical tools to go from texts to their interpretation, or from sentences to morphemes and phonemes. It presupposes the results of an analytical phase of scholarship, but is itself not representative of that analytical phase".

The *Aṣṭādhyāyī* records processes only of the second phase of a *grammatical circle* that begins with a provisional expression which a speaker formulates to express her/his intention (*vivakṣā*) and culminates in formation of a *saṃskṛta* expression. The first phase, that of analysis of a sentence in one or more *pada*s (*vākyavibhajyānvākhyāna*) and further a *pada* in its constituting components i.e. *prakṛti* and *pratyaya* etc. (*padavibhajyānvākhyāna*), precedes the process of synthesis.[3]

Thus, the grammatical process can be stated as consisting of the following steps:

1. Collection of *pada*s P_j from a (provisional) sentence S_i.

$$S_i = \{P_j\}_1^n \tag{1}$$

   Decomposition of a *pada* P_j into *prakṛti* (root) R and *pratyaya*s (affixes) $A_{1\cdots k}$.

$$P_j = \{R, A_{1\cdots k}\} \tag{2}$$

2. Combination of the constituent components into *pada*s and sentence.

$$\{R, A_{1\cdots k}\} \longrightarrow P'_j \tag{3}$$
$$\{P'_j\}_1^n \longrightarrow S'_i \tag{4}$$

The above steps comprise a circular process of first decomposing a (provisional) sentence into *imaginary*[4] components and then reassembling these components to form a *saṃskṛta* expression.

$$S_i \longrightarrow \{P_j\}_1^n \longrightarrow \{R, A_{1\cdots k}\} \longrightarrow \{P'_j\}_1^n \longrightarrow S'_i \tag{5}$$

Thus, modelling the grammatical process involves modelling this circular process of decomposition and then recombination, through which a provisional sentence is transformed into a *saṃskṛta* sentence.[5]

[1] Here, a reference to Pāṇini includes his predecessor, contemporary and successor grammarians as well.
[2] The *it* - markers are represented in SMALL CAPS.
[3] See BHATTACHARYA [1] Pp. 228.
[4] See BHATTACHARYA [1] Pp. 229.
[5] See HOUBEN [7] Pp. 48.

3 Representing the Grammatical Process of *Aṣṭādhyāyī*

This section briefly summarizes the basic data structure for modelling the grammatical process of *Aṣṭādhyāyī*.[6]

3.1 Fundamental Components

The building blocks of the language are collected and assigned a unique key in our database. For example, the phoneme /*a*/ has the key `a_0`, the *kṛt* suffix /*a*/ has the key `a_3` and the *taddhita* suffix /*a*/ is represented by the key `a_4`.

Definition 1. *The collection of unique keys corresponding to the basic constituents of the language, we define as the set $\mathcal{F}$ of fundamental components.*

Remark 1. This set is further sub-divided in two disjoint sets consisting of the set of keys corresponding to the phonemes ($\mathcal{P}$) and the set containing the keys of the rest of the constituting elements ($\mathcal{M}$).

$$\mathcal{P} = \{\texttt{a\_0}, \texttt{i\_0}, \texttt{u\_0}, \ldots\} \tag{6}$$

$$\mathcal{M} = \{\texttt{bhU\_a}, \texttt{tip\_0}, \texttt{laT\_0}, \ldots\} \tag{7}$$

3.2 Attributes

The fundamental units of the language are given an identity by assigning a number of attributes to them. This includes the various technical terms introduced in the grammar as also the *it* -markers and *pratyāhāra*s.

Definition 2. *The collection of unique keys corresponding to the terms, which characterize a fundamental component, we define as the set $\mathcal{A}$ of attributes.*

Remark 2. Corresponding to the sets $\mathcal{P}$ and $\mathcal{M}$ we can decompose the set $\mathcal{A}$ into two disjoint sets $\mathcal{A}_\pi$ and $\mathcal{A}_\mu$, $\mathcal{A}_\pi$ being the set of unique keys of the attributes to the elements of $\mathcal{P}$ and $\mathcal{A}_\mu$ to elements of $\mathcal{M}$.

$$\mathcal{A} = \mathcal{A}_\pi \cup \mathcal{A}_\mu \tag{8}$$

$$\mathcal{A}_\pi = \{\texttt{hrasva\_0}, \texttt{udAtta\_0}, \texttt{it\_0}, \ldots\} \tag{9}$$

$$\mathcal{A}_\mu = \{\texttt{dhAtu\_0}, \texttt{pratyaya\_0}, \texttt{zit\_9}, \ldots\} \tag{10}$$

Remark 3. Any two of the four sets $\mathcal{P}, \mathcal{M}, \mathcal{A}_\pi, \mathcal{A}_\mu$ are mutually disjoint.

Given the mutually disjoint sets $\mathcal{P}$, $\mathcal{M}$ and $\mathcal{A}$, we represent a linguistic expression at any stage of its derivation through a *language component*, which is an ordered collection of *sound sets*. We first define a sound set and then a language component in terms of these sound sets.

[6] For a detailed description, see Mishra [9].

3.3 Sound Set ψ

Definition 3. *A sound set ψ is a collection of elements from sets $\mathcal{P}, \mathcal{M}$ and $\mathcal{A}$ having exactly one element from the set $\mathcal{P}$.*

$$\psi = \{\pi_p, \mu_i, \alpha_j | \pi_p \in \mathcal{P}, \mu_i \in \mathcal{M}, \alpha_j \in \mathcal{A}, i, j \geq 0\} \tag{11}$$

3.4 Language Component λ

Definition 4. *A language component λ is an ordered collection of at least one or more sound sets.*

$$\lambda = [\psi_0, \psi_1, \psi_2, \ldots \psi_n] \text{ such that } \|\lambda\| > 0 \tag{12}$$

A *language component* λ has as many *sound sets* ψ_i's as there are phonemes in that component. A *sound set* ψ contains a number of fundamental components and attributes. Those attributes which are common to a number of sound sets in a language component, become the attributes of that chain of sound sets. This chain could be a single phoneme, or a morpheme or more than one morphemes and even more than one words.

The process of formation is represented through a *process strip* σ, which is an ordered collection of a pair having its first entry as a rule number and second one to be a language component, which is achieved after application of this rule.

3.5 Process Strip σ

Definition 5. *A process strip σ is an ordered collection of pairs, where the first element of the pair is the number of a particular grammatical rule (e.g. $rule_p$) and the second element is a language component λ.*

$$\sigma = [(rule_p, \lambda_p), (rule_q, \lambda_q), \ldots] \tag{13}$$

There are two basic operations, *attribute addition* and *augmentation* which are applied to a language component. All the operations in *Aṣṭādhyāyī* e.g. substitution, reduplication, accentuation etc. are implemented using a combination of these two basic operations.

3.6 Attribute Addition

Let $\alpha \subset \mathcal{A} \cup \mathcal{M}$ and ψ be a sound set. Then attribute addition is defined as

$$h_{a\psi}(\psi, \alpha) = \psi \cup \alpha \tag{14}$$

Remark 4. This operation can be applied to a number of sound sets given by indices $[i, i+1, \ldots, j]$ in a given language component λ

$$h_{a\lambda}(\lambda, \alpha, [i, \ldots, j]) = [\psi_1, \ldots, \psi_i \cup \alpha, \ldots, \psi_j \cup \alpha, \ldots, \psi_n] \tag{15}$$

3.7 Augmentation

Let

$$\lambda = [\psi_1, \ldots, \psi_i, \psi_{i+1}, \ldots, \psi_n]$$
$$\lambda_k = [\psi_{1k}, \psi_{2k}, \psi_{3k}, \ldots, \psi_{mk}]$$

and i be an integer index such that $i \leq \|\lambda\|$, then augmentation of λ by λ_k at index i is defined as

$$h_g(\lambda, \lambda_k, i) = [\psi_1, \ldots, \psi_i, \psi_{1k}, \psi_{2k}, \psi_{3k}, \ldots, \psi_{mk}, \psi_{i+1}, \ldots, \psi_n] \tag{16}$$

3.8 Substitution

We define substitution in terms of the above two operations.

Let $[i, i+1, i+2, \ldots, j]$ be the indices of sound sets to be replaced in the language component $\lambda = [\psi_1, \ldots, \psi_i, \psi_{i+1}, \ldots, \psi_n]$.

Let $\lambda_k = [\psi_{1k}, \psi_{2k}, \psi_{3k}, \ldots, \psi_{mk}]$ be the replacement, then the substitution is defined as

$$h_s(\lambda, \lambda_k, [i, \ldots, j]) = h_g(h_{a\lambda}(\lambda, \{\delta\}, [i, \ldots, j]), \lambda_k, j) \tag{17}$$

where $\delta \in \mathcal{A}$ is the *attribute* which says that this sound set is no more active and has been replaced by some other sound set.

A rule of grammar is represented through a function f_q, which takes a process strip σ_p and adds a new pair $(rule_q, \lambda_q)$ to it where $rule_q$ is the number of the present rule, and λ_q is the new modified language component after application of one or more of the two basic operations defined above on the input language component λ_p.

$$f_q(\sigma_p) = \sigma_q \text{ where} \tag{18}$$
$$\sigma_p = [\ldots, (rule_p, \lambda_p)] \tag{19}$$
$$\sigma_q = [\ldots, (rule_p, \lambda_p), (rule_q, \lambda_q)] \tag{20}$$
$$\lambda_q = h_a, h_g(\lambda_p, \ldots) \tag{21}$$

A typical formative process begins with a *seed* element (usually a verbal root or nominal stem), and a chain of rules provided manually through a template is applied. At the end, a *saṃskṛta* expression is formed.

The system has been tested for different formative processes of *Aṣṭādhyāyī* and can be accessed online (`http://sanskrit.sai.uni-heidelberg.de`).

4 Heuristically Analyzing the Sanskrit Expressions

Aṣṭādhyāyī provides us with a collection $\mathcal{F}$ of fundamental elements which is a finite set, having limited entries. Using this set and another finite collection of rules, a substantially bigger set of linguistic expressions can be formed. To

specify this process, a number of meta-linguistic entities (collected in set $\mathcal{A}$ of attributes) as well as conventions are used.

This process presupposes another process of looking for these fundamental components in the expressions of the language. For example, by some process of analysis, it is ascertained that with *bhavati* the elements like *bhū* or *śa*P or *ti*P are associated.[7] As mentioned earlier, there are no recorded rules for this step. The question, which fundamental elements are associated with a particular expression, can however be approached heuristically.[8] The problem can be stated as follows:

Problem 1. Given a string S, search for the possible break-up tuples such that each tuple contains only fundamental elements which could be later used as seeds for reconstitution.

The above task is performed by an `Analyzer (A)` which aims at *guessing* the possible fundamental components constituting a given sentence S. It does not aim to ascertain the perfect break up in terms of fundamental constituents, but only some of the possible components, which could function as seeds for the subsequent step of reconstitution.

Example 1. We give an example first for a sub-problem, where our string S is a *pada*. Given $S = jayati$, A(S) should fetch us at least a tuple consisting of at least the verbal root *ji* and possibly the *ti*Ṅ suffix *ti*P as well.

$$\mathtt{A}(jayati) = [(ji, ti\mathrm{P}, \ldots), (e_1, \ldots), (e_2, \ldots), \ldots] \quad \text{where } e_i \in \mathcal{F} \tag{22}$$

In fact, it gives a list of possible decomposition tuples.

4.1 Some Observations on the Process of Analysis

Before we describe our approach for developing heuristics towards analysing an expression and give an example, we first mention a few observations as to the nature of this problem.

On the surface, it seems to be searching for a needle in a hay stack, but a closer look allows for such an adventure! For this purpose, certain features of the grammatical corpus and processes of *Aṣṭādhyāyī* can be made use of.

1. The set of fundamental elements $\mathcal{F}$ is finite. That means, we do not have to search infinite elements.
2. The order of fundamental elements in a tuple is also not random. Thus, (*upasarga*, *dhātu*, *vikaraṇa*, *pratyaya*) is one such (partial) order.

[7] The examples here are at *pada* level and not at *vākya* level, although the unit of analysis (as well as synthesis) is a sentence. This is because of simplicity and also it does not amount to any loss of generality.

[8] I am also working on the possibilities to incorporate some statistical methods, but it is too early to report about it.

3. Simultaneous presence of certain fundamental elements within a tuple is also restricted. For example, while analysing a *pada*, both a *ti*ṄIT suffix as well as a *s*UP suffix can not occur simultaneously.
4. Certain attributes of the fundamental elements, like *avasāna*, *s*UP, *ti*ṄIT indicate *pada* boundaries. This is helpful to identify more than one *pada*s within a character string.
5. A dictionary of possible surface forms (as keys) and corresponding original elements (as values) provides a connection between phoneme chains on the surface level to the fundamental elements, which may have contributed to it. Here, a special sub-set is of those elements, like *dhātu*s or *ti*ṄIT suffixes which are more abundantly present.
6. Consonants are less prone to phonetic changes.
7. The replacement rules (*ādeśa-sūtra*s) can be reversed, and these reversed rules can be used to gain the replaced element. Thus, for example, the replacement rule **thā aḥ se** (3.4.080)[9] replaces the whole of *thās* of a *ṭit lakāra* with *se*. So in case, *se* is identified, the reversal of this rule will be used to check the possibility of *thās* here.
8. Reverse engineering of certain standard *vidhi*s, e.g. reduplication or *ṣatva vidhi* etc. brings us closer to the original element.

Finally it should be mentioned that it is for the teleological purpose of providing *seeds* for the subsequent step of reconstitution, with which this phase is concerned and not to provide a correct and complete decomposition of a sentence or a word. In fact an imprecise break up of an incorrect *pada* can only possibly lead to the *saṃskṛta* form.

4.2 The General Process of Analysis

Given a character string, the general process of analysis involves in guessing its possible break ups in terms of elements of the set $\mathcal{F}$ of fundamental constituents. It consists of the following steps:

1. Take a possible break up of character string.
2. Try to find the corresponding elements associated with these sub-strings using the dictionary which maps surface forms to fundamental elements.
3. Try to find the possible replaced elements using reverse replacement rules in a given tuple of fundamental elements.
4. Check for Pāṇinian consistency of this tuple.
5. If consistent, then add to the list of break up tuples.
6. Repeat the previous steps for a new initial break up.

We illustrate the above process through a couple of examples.

Example 2. Consider *pavete* to be the input string. We first try to guess the possible *ti*ṄIT suffix. For that, we look up in the the dictionary which maps surface forms to fundamental elements for *ti*ṄIT suffixes. This dictionary looks like

$$\{ti : [ti\text{P}, \ldots], taḥ : [tas, \ldots], \ldots, te : [ātām, \ldots], \ldots\}$$

[9] Numbers in brackets refer to the rule number in *Aṣṭādhyāyī*.

We take only those break up strings which can possibly be associated to some *ti*Ṅ element. Thus, the possible break up is restricted through the keys of the dictionary of surface forms to fundamental elements. In this case, we break the string as *pave te* and associate *ātām* to *te*. We represent this as follows

$$[(pave)(te : \bar{a}t\bar{a}m)]$$

Next we look at the leading part (*pave*) of the provisional break up, which has the possibility that it may contain the verb. Here we look first in the list of verbs beginning with *p*. These are [*pac*I, *paṭ*A, ..., *puṣ*A, *pū*Ṅ, *pū*Ñ, ...]. We now use the rules for reverse replacement. This is guided by the standard replacement series in verbs. One such series is $\bar{u} \rightarrow o \rightarrow av$ replacements. The character string *av* motivates the reverse replacement, and applying this, we come from (*pave*) to (*pūe*). We associate now the verbs *pū*Ṅ as well as *pū*Ñ to the sub-string *pū*. We thus have,

$$[(p\bar{u} : p\bar{u}\dot{\mathrm{N}}, p\bar{u}\tilde{\mathrm{N}})(e)(te : \bar{a}t\bar{a}m)]$$

We now collect the decomposition tuples. These are,

$$[(p\bar{u}\dot{\mathrm{N}}, \bar{a}t\bar{a}m), (p\bar{u}\tilde{\mathrm{N}}, \bar{a}t\bar{a}m), \ldots]$$

Now the Pāṇinian consistency of the tuples are checked. In this case the tuples are (*dhātu*, *pratyaya*) tuples. So the order of elements within a tuple is correct. Moreover, within a tuple, there is no simultaneous presence of mutually exclusive pairs of fundamental elements. For example, no *s*UP and *ti*Ṅ suffixes are present simultaneously. Thus, these two tuples are added to the list of other possible break up tuples of fundamental elements.

$$L = [\ldots, (p\bar{u}\dot{\mathrm{N}}, \bar{a}t\bar{a}m), (p\bar{u}\tilde{\mathrm{N}}, \bar{a}t\bar{a}m), \ldots]$$

The process is repeated for other possible character break ups (as long as there is such a possibility). The tuples are ranked according to the richness of information they contain for the subsequent process of reconstitution. Thus, those tuples, having a *dhātu* or *prātipadika* and a *ti*Ṅ or *s*UP suffix are ranked higher than those having only a *dhātu* etc.

Example 3. Consider the case where the input string (for a *pada*) has four or more consonants. We look for the possibility whether it is the case of reduplication, specially because of the suffix *sa*N.

1. Look whether there is consonant *s* or *ṣ* in the input string
2. Use the heuristics for deciding the *ti*Ṅ endings for a *pada* (see previous example) and check if *s* or *ṣ* appear before these.
3. Get the part before *s* or *ṣ* and check it with heuristics for reduplication.

Now the heuristics of reduplication is implemented taking care of the process of reduplication in *Aṣṭādhyāyī*. Thus, let us consider the case, where input has three consonants.

1. The probability of a root with two consonants is high.
2. Get the list of roots having the consonants as the last two consonants of input. (Here, also the roots which undergo changes due to *ṇatva* or *ṣatva vidhi* etc.)
3. Check if the first consonant of input could be a reduplicated consonant e.g. pairs like *j*-*g* or *c*-*k* etc.
4. If the last consonant is *r* or *l* then consider the possibility of *ṛ* or *ḷ*.

Consider now the input string: *titikṣate*. The process of analysis is briefly sketched below.

1. Consider a possible string break up: *titik ṣa te*
2. Check heuristics for *tiṄ* and assume that it returns (*ta*, *ātām*...)
3. Split at *s* or *ṣ*: *titik ṣate*
4. Send the first half before *s* or *ṣ* to check for reduplication.
5. We have here three consonants: *t t k*
6. Search for the roots with consonants *t k* (as also the possible variants *t j*, *t g* etc.). It returns roots like *tik*A, *tig*A, *tij*A, *tuj*A, *tuj*I etc.
7. Check the first consonant of the input, if it is according to the reduplication rules.
8. Now reduce the choice further using other heuristics (e.g. looking at the vowels between the two consonants of the root).
9. Thus, the output of this heuristics is: [(*tik*A, *sa*N, *ta*), (*tig*A, *sa*N, *ta*), (*tij*A, *sa*N, *ta*), ...]

5 Forming *Saṃskṛta* Expressions Using the Rules of *Aṣṭādhyāyī*

Given a break up tuple, the process of forming the *saṃskṛta* expression(s) is regulated by the rules of *Aṣṭādhyāyī*. We developed a model for constituting linguistic expressions beginning with seed elements and through manual prescription of rule order using templates (see Sect. 3). We now propose to include strategies for automatic application of rules.

5.1 An Extended Model for Forming *Saṃskṛta* Expressions

This part of the grammatical process is being implemented in the `Synthesizer` module, which takes as input a tuple of fundamental elements. This tuple is gained by the preceding step of analysis. All the constituent elements for formation of a particular expression are not provided in this input tuple, but only the *seed* elements, e.g. verbal root or nominal stem and if possible *tiṄ* or *s*UP suffixes etc. This initial tuple contains partial information about the *vivakṣā* or intention of the speaker.

Further, the input tuple is *consistent*. Consistency means that it contains only those elements which can occur simultaneously. Moreover, the ordering of

these elements is according to the Pāṇinian principles. For example, the element corresponding to *dhātu* must precede the *pratyaya* or suffix element.

In the `Synthesizer`, the elements of this input tuple are taken as *seeds* and a number of appropriate rules are applied with the goal of reproducing the *saṃskṛta* form of the original expression. For this purpose, the data structure and corresponding operations are described in Sect. 3. The entire process is simulated using the process strip. All the information which is required to assess the conditions for application of a particular rule is stored in this process strip, which stores not only the current stage of formation but also the previous stages.

The question, which rule must be applied next, was resolved thus far by prescribing a template based approach in which the order of rules to be applied was stated manually. We now propose strategies for automatic application of rules. For this, we introduce stable and transitional λ - states.

5.2 Stable and Transitional λ - States

At any stage of formation, the fundamental components together with their attributes are represented in a language component λ. Given such a language component, we first try to bring it in a *stable* λ - state by applying certain rules which we call stabilizing rules.

The purpose of this step is to prepare the current λ - state for assessing the 'cause of application' or *nimitta* of those transitional rules which bring about a transition of λ - state. We first specify what we mean by stabilizing and transitional rules.

Stabilizing Rules There are certain rules in *Aṣṭādhyāyī* (specially most of the definition rules), which need to be applied to a λ - state in order to add more grammatical information which is necessary for a progressive flow of the process of formation of linguistic expressions.

For example, if a new element is introduced in the previous step, which contains the phoneme $/\bar{a}/$, then the application of rule **vṛddhirādaic** (1.1.001) adds the information that it also has the attribute *vṛddhi*, which may be required for subsequent application of other rules. Similar rules which bring about some kind of attribute addition are what we call stabilizing rules.[10]

We collect these stabilizing rules seperately and define this set as follows:

Definition 6. *The set of stabilizing rules $\mathcal{R}_f$ is the set of those characterizing rules in* Aṣṭādhyāyī*, which fulfill the condition that the application of any rule belonging to this set on a language component is not depended upon the results of application of any other rule of this set.*

For example, the characterizing rules **vṛddhirādaic** (1.1.001) and **adeṅguṇaḥ** (1.1.002) belong to the set of stabilizing rules $\mathcal{R}_f$.

[10] In fact many fundamental elements have certain attributes which are *static*, i.e. they are always associated with that particular fundamental element. For example, with the element $/a/$ the attribute aC (specifying that it is a vowel) is always attached.

Having defined the set of stabilizing rules, we can now speak of a *stabilizing process* ($\longrightarrow$) which brings a language component λ to a stable language component λ' by applying the stabilizing rules from the rule set $\mathcal{R}_\int$.

$$\lambda_i \longrightarrow \lambda'_i \tag{23}$$

Transitional Rules Those rules which do not belong to the set of stabilizing rules, we call transitional rules. The effect of the application of a particular rule belonging to this set has a consequence for the application of some other rule belonging to this same set. Barring those characterizing rules grouped under $\mathcal{R}_\int$, all the other rules, we put in this group.

The process of transition of λ - states caused by application of transitional rules can now be considered as a *transitional process* ($\Longrightarrow$).

$$\lambda_i \Longrightarrow \lambda_{i+1} \tag{24}$$

The General Formative Process The general Pāṇinian process of formation of linguistic expressions can now be presented as an incremental increase of process strip σ through a transitional phase and then stabilization of the strip through a stabilizing phase, whereby the two phases always alternate.

$$[\ldots, ([rule_{p_i}], \lambda_p \longrightarrow \lambda'_p)] \Longrightarrow [\ldots, ([rule_{p_i}], \lambda'_p), ([rule_{q_i}], \lambda_q \longrightarrow \lambda'_q)] \tag{25}$$

5.3 Executing the Stabilizing Process

The stabilizing phase ($\lambda_p \longrightarrow \lambda'_p$) is executed every time by applying the rules from the set $\mathcal{R}_\int$. This helps in characterizing the current situation of the language component.

Example 4. For example, if a morpheme like ŚaP is added in the previous transitional phase, the following stabilizing phase adds the attributes like *hrasva*, *guṇa*, *śit*, *pit*, *sārvadhātuka* etc. to the sound set corresponding to the phoneme /*a*/ of ŚaP.

5.4 Executing the Transitional Process

Given a stable λ - state within a process strip, the main challenge here is to decide as to which rule should next be applied to proceed through the transitional step. A correct and definite answer to this problem is the key for automatic application of rules in the process of formation of linguistic expressions according to *Aṣṭādhyāyī*. The problem can be divided into two sub-steps.

1. What are the possible rules which could be applied?
2. Given more than one possibilities, how to choose the correct one?

Assessing a λ - State A sub-module `Assessor` assesses a given λ - state and gives a tuple of list of rules, which could be applied for a transitional process. This tuple is then sent to `ConflictResoluter`, another sub-module, which evalutes the question of conflicting rules and fetches those options which may lead to correct results. If there are more than one possibilities, then all are pursued in a parallel manner.

The `Assessor` Module There are two guiding principles here to decide which rules can possibly now be applied:

1. Assessment of the intention (*vivakṣā*) of the speaker.
2. Assessment of the thus far evolution of formative process, i.e. assessment of the input process strip.

Assesing the Intention of the Speaker One way to assess the intention of the speaker is to provide a user interface at this level. But for now, we depend on our heuristic analysis of the original provisional input by the speaker.

Example 5. If *bhavati* is what the speaker inputs and if our analysis provides us with one such tuple like (*bhū*, *ti*P) then we can guess some of the information as to what she/he wants to convey. Thus, at some stage when the question arises which *ti*Ṅ suffix is to be attached, then the entry in the input tuple can give us the information.

Example 6. Given the initial tuple (*tij*A, *sa*N, *ta*), and the situation that we have just the *dhātu* in the stable $\lambda - state$ (λ'), the question as to which way to proceed could be answered by looking at the presence of *sa*N.

Assesing the Stable $\lambda - state$ The assessment of the stable λ - states in a process strip is based upon the observations as to what are the elements which are currently present and what could be the next introduction (*āgama*) or substitution (*ādeśa*). Here, certain guidelines for the general flow of the formative process are first taken into consideration.

For example, in the situation where only a *dhātu* is present and the input tuple has a *ti*Ṅ suffix, the general flow would be to apply rules for introduction of *lakāra*. If there is already a *lakāra* and no *ti*Ṅ substitute, then rules for such a substitution are applied.

The observations regarding the general order of introduction of fundamental elements are stored in terms of defining a partial order of these elements. This partial ordering aims to provide the answer to the question as to which element should next be introduced.

Certain morphemes give rise to a scope of applying certain rules once they are introduced. This observation is collected in the form of a special dictionary, where the possible rules, which could subsequently be applied, are listed. For example *l*IṬ, *sa*N etc. trigger reduplication, and so the rules which bring about reduplication are listed with these morphemes.

Conflict Resolution The successive filtering of possible candidates should normally provide a definite answer to the question of rule application. But in some cases, there are conflicting claims. Here we follow the solutions according to the principles laid down by JOSHI-ROODBERGEN [5]. One such principle for a 'one-way conflict' is that in case of rules not belonging to *asiddhakāṇḍa*, "that rule is to be applied first, which destroys the *nimitta* of the other rule, or which changes the phonetic form to which the other rule was to become applicable".[11]

The `ConflictResoluter` Module The conflict resolution algorithms are implemented in the module `ConflictResoluter` which gets as input, the process strip together with a tuple of list of conflicting rules. It checks the applicability of these rules and returns the one which should be applied. We briefly show its functioning by way of one example.

Example 7. Consider the reconstitutive process of *dudyūṣati* and let the process strip correspond to the stage[12]

$$\sigma_p = di\bar{u} + sa\text{N} + \acute{S}a\text{P} + ti\text{P}$$

At this stage, the `Assessor` proposes two possibilities ([6.1.009], [6.1.077]). The first one is the rule **san yaṅ oḥ** (6.1.009) which calls for reduplication and the second one is the rule **ikaḥ yaṇ aci** (6.1.077) which prescribes substitution of $\underline{ya}\text{N}$ respectively in place of iK. The `ConflictResoluter` now checks as follows:

1. Take the process strip σ_p and apply **san yaṅ oḥ** (6.1.009). This means applying the process of reduplication. This gives the extended process strip with the new λ - state as

$$\sigma_q = di + di\bar{u} + sa\text{N} + \acute{S}a\text{P} + ti\text{P}$$

2. Now this new λ - state is stabilized and then checked using `Assessor` if the conflicting rule (6.1.077) is still applicable. This is the case here. So, the application of this rule neither changes the *nimitta* of the conflicting rule nor does it bring about any change in the phonetic form of the elements to which the other rule is to be applied.
3. The other rule **ikaḥ yaṇ aci** (6.1.077) is now applied to the process strip σ_p. This gives the result

$$\sigma_q = dy\bar{u} + sa\text{N} + \acute{S}a\text{P} + ti\text{P}$$

4. The resulting new λ - state is assessed and it shows that the phonetic form of the element to which the other rule is to be applied has been changed.

[11] JOSHI-ROODBERGEN [6] Pp. X.

[12] The process strip actually is a complex data structure which is expressed in terms of language components, which in turn is a list of sound sets (see Sect. 3), but for the sake of simplicity, we write it here in this form.

5. This results in selection of the rule **ikaḥ yaṇ aci** (6.1.077) for application at this stage, because both the rules do not belong to the range of rules of *asiddhakāṇḍa*.

Similarly, other principles of conflict resolution are implemented. We enunciate below the general architecture of the computer implementation of the modelling process. At the moment, the modules are in the programming phase and testing of a wider range of examples are needed before the system could be put to use.

6 Computer Implementation of the Model

The entire process is divided into four main modules besides a specially designed `Database` of the fundamental components $\mathcal{F}$ and attributes $\mathcal{A}$ of Pāṇinian Grammar. These four main modules are:

1. `Input`
2. `Analyzer`
3. `Synthesizer`
4. `Output`

Each of them contain a number of other sub-modules. We sketch briefly the main ones below.

6.1 Database

Besides having a repository of the fundamental components $\mathcal{F}$ and attributes $\mathcal{A}$ in *Aṣṭādhyāyī*, there are a few special dictionaries and lists of elements and attributes serving specific purposes. It includes

1. A dictionary which maps surface forms to fundamental elements, which is used by the `Analyzer` module. It looks like:

$$\{ti : [ti\text{P}, \ldots], taḥ : [tas, \ldots], \ldots, te : [ātām, \ldots], \ldots\}$$

2. A set of pairs whose elements exclude each other within an analysis tuple. E.g. $\{(s\text{UP},\ ti\dot{\text{N}}),\ (l\text{AṬ},\ l\text{AṄ}),\ \ldots\}$
3. A list of acceptable partial orders within an analysis tuple. E.g. [(*upasarga*, *dhātu*, *vikaraṇa*, *pratyaya*), (*dhātu*, *sa*N, *vikaraṇa*), ...]
4. A dictionary, used by `Assessor`, of fundamental elements as keys and list of rules which are possibly applied when this element is introduced as values.
5. A number of special subsets of the set of fundamental elements or attributes for the sake of identification of respective elements. For example, the set of phoneme attributes, or the set of morpheme attributes etc.

6.2 Input

This module, as the name suggests, takes care of the user interface for entering a provisional sentence S. After an initial processing, e.g. checking the non-occurence of phonemes not belonging to the language, the input is passed to the `Analyzer`.

6.3 Analyzer (See Sec. 4)

Given a sentence S, the `Analyzer` aims at *guessing* the possible fundamental components constituting it. The `Analyzer` functions heuristically and suggests first a number of possible break ups. These possibilities are then ranked based upon certain constraints and provide *seeds* for `Synthesizer`.

Thus, given a string S, the `Analyzer (A)` produces a ranked list D of possible decompositions, represented as tuples containing the fundamental components constituting S.

$$\mathtt{A}(S) = D = [(e_1, e_2, \ldots), (e_3, \ldots), (e_4, \ldots), \ldots] \quad \text{where } e_i \in \mathcal{F} \tag{26}$$

6.4 Synthesizer (See Sec. 5)

Given an analysis tuple $t = (e_1, e_2, e_3, \ldots)$, the `Synthesizer (Z)` now applies a series of rules from *Aṣṭādhyāyī*, which are collected in a rule set $\mathcal{R}$, and produces the final expression S'.

$$\mathtt{Z}((e_1, e_2, e_3, \ldots)) = S' \quad \text{where } e_i \in \mathcal{F} \tag{27}$$

For the purpose of assessing a given stage during the process of formation, it uses the `Assessor` module, which outputs a tuple of lists of rules which could be applied at that stage. In case of a conflict of rules, the `ConflictResoluter` module tries to resolve the clash. Otherwise all the possibilities are checked in a parallel manner.

6.5 Output

This module outputs the reconstituted sentence S' as well as the original provisional sentence S. It also provides a step-by-step process of constitution of the final expression beginning with its elemental parts and grammatical information gathered during the course of formation.

References

1. Bhattacharya, R.S. *Pāṇinīya Vyākaraṇa kā Anuśīlana.* Indological Book House. Varanasi. 1966.
2. Böhtlingk, Otto von. *Pāṇini's Grammatik.* Olms, Hildesheim. Primary source text for our database. 1887.
3. Deshpande, Madhav M. Semantics of Kārakas in Pāṇini: An Exploration of Philosophical and Linguistical Issues. *Sanskrit and Related Studies: Contemporary Researches and Reflections.* (eds. Matilal B.K., and Purusottama Bilimoria): 33-57. Delhi: Sri Satguru Publications. 1990.
4. Dīkṣita, Puṣpā. *Aṣṭādhyāyī sahajabodha.* Vols. 1-4. Pratibha Prakashan, Delhi, India. 2006-07.
5. Joshi, S. D. and Roodbergen, J. A. F. On siddha, asiddha and sthānivat *Annals of the Bhandarkar Oriental Research Institute.* Vol. LXVIII, Poona, 1987, p.541-549.

6. Joshi, S. D. and Roodbergen, J. A. F. *The Aṣṭādhyāyī of Pāṇini.* With Translation and Explanatory Notes. Vol. II. Sahitya Akademi, New Delhi, 1993.
7. Houben, Jan. E. M. 'Meaning statements' in Pāṇini's grammar: on the purpose and context of the Aṣṭādhyāyī *Studien zur Indologie und Iranistik 22:23-54.* 1999 [2001].
8. Katre, Sumitra M. *Aṣṭādhyāyī of Pāṇini.* Motilal Banarsidass, Delhi, India. 1989.
9. Mishra, Anand. Simulating the Pāṇinian System of Sanskrit Grammar *Proceedings of the First International Sanskrit Computational Linguistics Symposium, Pp. 89-95.* Rocquencourt, 2007.
10. Śāstrī, Cārudeva. *Vyākaraṇacandrodaya.* Vols. 1-5. Motilal Banarsidass, Delhi, India. 1971.
11. Vasu, Srisa Chandra and Vasu, Vaman Dasa. *The Siddhānta-Kaumudī of Bhaṭṭojī Dīkṣita.* Vols. 1-3. Panini Office, Bhuvanesvara Asrama, Allahabad, India. Primary source text for *prakriyā.* 1905.