Sanskrit And Computational Linguistics

Select papers presented
in
the **'Sanskrit and the IT World'** Section
at
the 16th World Sanskrit Conference

(28 June – 2 July 2015)
Sanskrit Studies Center,
Silpakorn University, Bangkok, Thailand

Editor
Amba Kulkarni

Pragun Publication
A D K Publishers Distributors Enterprise

First Published: 2015

ISBN: **978-93-80397-58-0**

Published by
Pragun Publication
(A D K Publishers Distributors Enterprise)
4224/1, Ansari Road, Daryaganj, New Delhi-110002, India
Ph: 011-41562573-77 Fax: 011-41562578
E-mail: pragun@dkpd.com
www.dkpd.com, www.pragunpublication.com

Printed at:
D K Fine Art Press P Ltd.
A-6, Nimri Colony, Ashok Vihar, Phase-IV
Delhi-110052, India

Contents

An XML formalization of the *Aṣṭādhyāyī*

Peter M. Scharf

Abstract: The present paper describes the structure of the XML formalization of Pāṇini's *Aṣṭādhyāyī* I have been constructing over the past two years so that it may be utilized by software engineers to produce a computational implementation of Pāṇini's grammar. The computational implementation will produce a comprehensive lexicon of Sanskrit from the basic elements listed in the *Dhātupāṭha*, *Gaṇapāṭha*, and the affixes provided in rules of the *Aṣṭādhyāyī*. The lexicon will contain not only the semantics and derivation of every form produced but also its immediate relations in a dependency hierarchy. The Pāṇinian lexicon such a computational implementation produces will serve as the core of a lexical table that coordinates headwords of numerous lexical resources in the Sanskrit Library's integrated dictionary. The lexicon will also prove useful in the development of Sanskrit parsing software by constraining homophonous speech forms and providing relation marking. Systematic computational surveys of the resulting lexicon may contribute to building a semantic network for Sanskrit and may provide useful insights to contemporary formal linguistics.

Keywords: Pāṇini, *Aṣṭādhyāyī*, XML, computational implementation

1 Introduction

Besides being a fascinating cultural expression of the fourth century BC, Pāṇini's *Aṣṭādhyāyī* contains detailed, specific information about the semantics, syntax, and lexicography of Sanskrit. Much of this information has been included in comprehensive Sanskrit lexica and descriptive grammars. Yet what has been included is not available in a computationally accessible form, nor has a thoroughly systematic and formal analysis been undertaken and made accessible for computational linguistic purposes. In addition, the grammar employs methods of description that have inspired modern generative grammar and continue to be of interest in that field. The efficacy and scope of many of the techniques used in the grammar have been the subject of investigation and debate by Indian linguists for millennia and modern scholars for decades. Yet these methods and techniques remain to be tested systematically. Information regarding the semantics, syntax, and lexicography Pāṇini describes remains incomplete, and debates regarding the techniques he used remain unresolved because of the extensive detail and complication involved. In order to conduct a thorough systematic analysis of the semantics, syntax, and lexicography of Sanskrit described by Pāṇini and to test the techniques utilized by him, it is necessary to model his grammar formally and implement it computationally.

Recent years have witnessed several projects that have implemented sections of Pāṇini's *Aṣṭādhyāyī* computationally or have begun the development of a complete implementation. Since these projects and the most pertinent discussions were just surveyed by Scharf, Goyal, et al. (2015: 165–70), a discussion of them will not be repeated here. A successful computational implementation of Pāṇinian grammar requires thorough knowledge of Sanskrit and the tradition of commentary on the intricate system of grammar founded by the Indian linguist as well as adequate expertise in computer science to employ suitable methods to model the linguistic system. Since the combination of expertise required is difficult to locate in a single individual, collaboration is demanded. Similarly, in order to build a digital Sanskrit library replete with computational analytic tools, several of my colleagues in the Sanskrit Computational Linguistics Consortium have found it productive to collaborate by distributing tasks to those with the requisite specialized skills and coordinating our results. The late Malcolm Hyman and I took a similar collaborative approach in early implementations that model Pāṇinian sandhi, declension, conjugation, and participle stem derivation. While my background in analytic philosophy and computer science is sufficient to enable me to communicate with computer scientists and formal and computational linguists, my expertise is in Sanskrit and Indian linguistics. Hence, I wrote rules in an XML framework using regular expressions and he wrote Perl software to convert the XML to executable code.

The XML framework and data-structure I describe below were developed over the past two years in frequent consultation with Ralph Bunker, a computer scientist who is developing a computational implementation of the framework, and build upon the structures employed in the implementation of voice determination described by Scharf, Goyal, et al. (2015). The framework itself, however, is meant to be independent of the computational software that implements it. Hence my description is independent of any particular computer language. I do not purport to explain Pāṇinian techniques of linguistic description or derivational procedures here. Nor do I attempt to justify the formalization by reference to Pāṇinian metarules or an explanation of Pāṇinian techniques. To do so would take far too much space. Rather I provide a description of the structure of the formalization of Pāṇinian rules and techniques in XML so that computer scientists may understand how to employ the formalization in an implementation of the *Aṣṭādhyāyī* and so that others who have a prior understanding of Pāṇinian techniques, whether they be Pāṇinian scholars or linguists, may understand how the formalization adequately represents the linguistic system it claims to represent. A companion paper to this one (T. Ajotikar, A. Ajotikar, and Scharf 2015) does explain some Pāṇinian techniques and illustrate how our formalization captures them in a number of cases. Other papers, such as Scharf (2014, 2015), consider individual major problems in Pāṇinian procedure brought to light by the attempt to formalize them. The present paper, in contrast, explains how the formalization captures the general sweep of Pāṇinian linguistic description: the representation of strings analyzed into components introduced under semantic and cooccurrence conditions which are subject to combination, replacement, deletion, and augmentation under additional semantic, cooccurrence, and phonetic conditions.

2 Components of the grammar

Pāṇini's grammar consists of several components besides the *Aṣṭādhyāyī*. As shown in Figure 1, these include lists of phonological and lexical elements as well as rules and metarules. Our model of Pāṇinian grammar likewise consists of several components. These include the following elements:

1. A derivational data structure.
2. A set of rules that operates on the data structure.
3. A set of functions each of which conducts an operation common to some rules.
4. A set of input tables of basic elements.

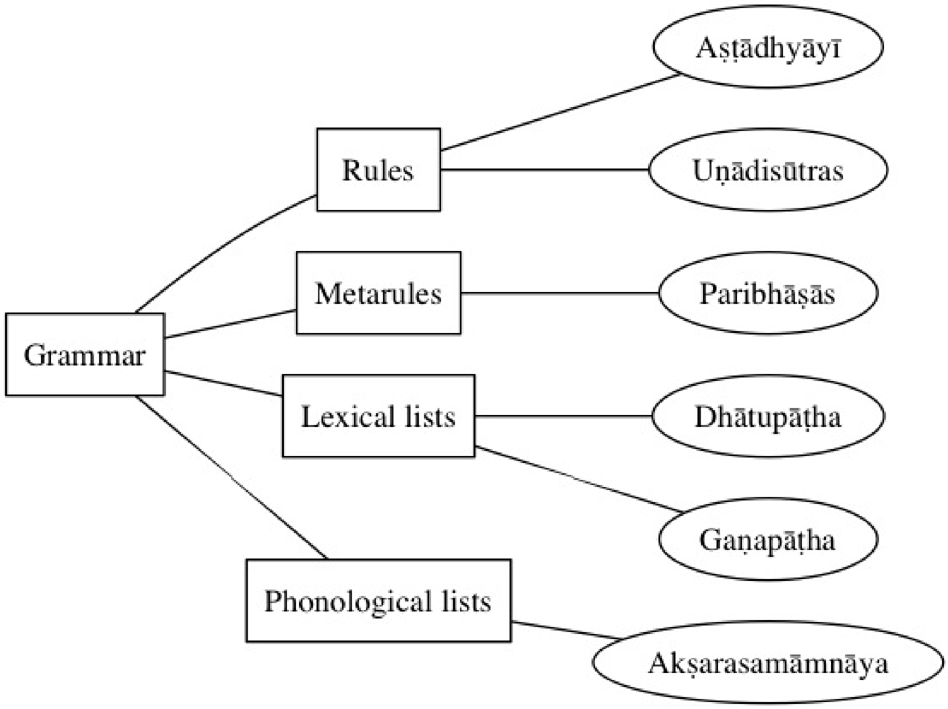

Figure 1
Components of Pāṇini's grammar

5. A set of produced tables of derived elements.
6. A set of display mechanisms.

All components of our grammar utilize the Sanskrit Library Phonetic basic (ASCII) encoding scheme whose basic segments are shown in Table 1 and which is described in full by Scharf and Hyman (2011: 151–58). The input tables consist of basic elements that are read but not altered by rules or functions. These include phonetic and lexical tables as well as a database of the *Aṣṭādhyāyī*. Besides the *Akṣarasamāmnāya*, the phonetic tables include a list of sounds, a table of sounds associated with phonetic properties, and a table of pratyāhāras and their expansions. Lexical tables include XML editions of the *Mādhavīyadhātuvr̥tti* (Scharf 2009), a list of additional roots mentioned in sūtras (*sautradhātu*s), and the *Gaṇapāṭha*. Scharf (2013) de-

Table 1
Basic Segments

अ a	आ ā	इ i	ई ī	उ u	ऊ ū
a	A	i	I	u	U

ऋ r̥	ॠ r̥̄	ऌ l̥	ॡ l̥̄
f	F	x	X
ए e	ऐ ai	ओ o	औ au
e	E	o	O

क् k	ख् kh	ग् g	घ् gh	ङ् ṅ
k	K	g	G	N
च् c	छ् ch	ज् j	झ् jh	ञ् ñ
c	C	j	J	Y
ट् ṭ	ठ् ṭh	ड् ḍ	ढ् ḍh	ण् ṇ
w	W	q	Q	R
		ळ् ḷ	ळ्ह् ḷh	
		L	|	
त् t	थ् th	द् d	ध् dh	न् n
t	T	d	D	n
प् p	फ् ph	ब् b	भ् bh	म् m
p	P	b	B	m

य् y	र् r	ल् l	व् v
y	r	l	v
श् ś	ष् ṣ	स् s	ह् h
S	z	s	h
ः ḥ	≍ ẖ	≍ h	ं ṁ
H	Z	V	M

scribed the database of the *Aṣṭādhyāyī* used. While rules of the *Aṣṭādhyāyī* themselves, such as rules that identify markers, rules to determine the pada of a root, and rules that condition the augment *i*, determine much of the information in the database of the *Aṣṭādhyāyī* and in the lexical tables, they do not determine the phonetic characteristic of the basic lexical items themselves or their specific markers. Nor are rules of the *Aṣṭādhyāyī* themselves sufficient to classify the various basic elements introduced or referred to in the text. Even affixes, despite the heading *pratyaya*, are not definitively distinguishable from other sounds by machine-usable criteria.

In contrast to the input tables, the produced tables contain tables of sounds with their features, pratyāhāras, and tables of affixes reconstituted using rules of the *Aṣṭādhyāyī*. Produced tables also include tables of name-value pairs, collections of the semantic conditions for affixes referred to as conditions for the introduction of other affixes, and collections of semantic conditions for the introduction of taddhita affixes. Produced tables include derived nominal bases (*prātipadika*), feminine stems, and derived verbal stems utilized in subsequent derivation, and tables of final derived words (*pada*) with their derivational histories. Display mechanisms include dictionaries of roots, nominal bases, stems, and words; and interactive research and educational tools.

3 Phonetics

Although the *Aṣṭādhyāyī* contains a number of rules that classify sounds and construct abbreviatory terms, and is accompanied by the *akṣarasamāmnāya*,[1] subsequent rules rely on more information than is explicitly provided. Pāṇini's rules rely upon knowledge of the properties of sounds that the *Aṣṭādhyāyī* itself does not exhaustively instruct; it merely introduces certain terms used for a number of useful properties and classifies sounds based upon known phonetic features. The formalization therefore employs an additional table of sounds associated with phonetic properties to provide the missing information. Table 2 shows a small snippet of our table of sounds associated with phonetic properties. Our table is not minimal; it includes as attributes of each sound all of the sound properties necessary to uniquely identify that sound. In Pāṇini's system some of these properties are derivable and

[1]Petersen (2004, 2008, 2009) has recently analyzed the *Akṣarasamāmnāya* and (2010) pratyāhāras.

```
<phone>
  <Sabda>a</Sabda>
  <slp1>a/</slp1>
  <sTAna>kaRWya</sTAna>
  <AByantara>vivfta</AByantara>
  <Goza>Gozavat</Goza>
  <prARa>alpaprARa</prARa>
  <AnunAsikya>ananunAsika</AnunAsikya>
  <AyAma>hrasva</AyAma>
  <svara>udAtta</svara>
</phone>
```

Table 2
XML table of sounds and their properties

some are dispensable. For example, the sound *á* in our table includes the phonetic features *hrasva* 'short' and *udātta* 'high-pitched' even though *A.* 1.2.27 provides that the sound *á* is termed *hrasva*, and *A.* 1.2.29 that it is termed *udātta.* Likewise our table includes the phonetic features *ghoṣa-vat* 'voiced' and *alpaprāṇa* 'unaspirated' even though Pāṇini does not rely upon the voicing and aspiration properties of vowels because he uses classes designated by abbreviatory terms formed from the *Sivasūtra*s (*pratyāhāra*s) to limit sound selection instead. Yet no Pāṇinian rule tells us that the sound *a* is *kaṇṭhya* 'velar' and *vivṛta* 'open'. Nor do rules that distinguish conditions necessary for a sound to qualify to be termed *hrasva*, *udātta* and *ananunāsika* 'unnasalized' explain to us that the sound *a* is associated with such conditions as the time of a short *u*, high tone (*uccais*), and lack of accompaniment by the nose. But the *Aṣṭādhyāyī* assumes and requires such knowledge for the proper application of its phonetic classificatory rules such as *A.* 1.2.27 *ūkālo 'j jhrasvadīrghaplutaḥ*, *A.* 1.2.29 *uccair udāttaḥ*, and *A.* 1.1.8 *mukhanāsikāvacano 'nunāsikaḥ* to distribute phonetic terms regarding the features of length, pitch and nasality properly. To formalize these phonetic classificatory rules strictly, we would have to depend upon some independent data source for this information in any case in order to provide sufficient conditions for these rules. Instead, we provide a simple phonetic properties table that associates each sound with a list of phonetic properties sufficient to uniquely identify each sound, let Pāṇini's rules draw from it the distinctions they need, and regenerate tables of sounds with their properties in the form in which subsequent rules depend upon them.

Hence, for example, since *A.* 1.1.2 *aeḍ guṇaḥ* provides that the sounds *a*, *e*, and *o* are termed *guṇa*, our formalization of this rule adds the attribute `bala="guRa"` to the list of attributes associated with these sounds.

Let's consider a more complicated example concerned with creating the conditions that allow rules to recognize sounds as homorganic (*savarṇa*) with each other. A dozen rules explicitly rely on this feature while numerous others rely upon it to determine replacements that are featurally closest. *A.* 1.1.9 *tulyāsyaprayatnaṁ savarṇam* states that each sound that has the same internal effort (*ābhyantaraprayatna*) at the same place of articulation (*sthāna*) belongs to the same class (*varṇa*), and *A.* 1.1.10 *nājjhalau* disallows vowels (*ac*) and consonants (*hal*) from belonging to the same class. Our formalization of these rules endows each sound with a `varRa` attribute based upon its association with `sTAna` and `AByantaraprayatna` attributes in the phonetic properties input table and additionally based upon its inclusion among the sounds that match one of the regular expression macros `@(ac)` or `@(hal)` that formalize pratyāhāras. The first `div` element in the following snippet of the formalization of *A.* 1.1.9 endows the velar stops *k*, *kh*, *g*, *gh*, and *ṅ*, with the attribute `varRa="kaRWya.spfzwa"`:

```
<div>
  <uddeSya sTAna="kaRWya" AByantaraprayatna=
    "spfzwa"/>
  <AdeSa varRa="kaRWya.spfzwa"/>
</div>
…
<div>
  <uddeSya sTAna="kaRWya" AByantaraprayatna=
    "vivfta"/>
  <AdeSa varRa="kaRWya.vivfta"/>
</div>
```

In exception to the second `div` element in the above snippet, the first `div` in the following snippet of the formalization of *A.* 1.1.10 endows the vowel *a* with the attribute `varRa="kaRWya.vivfta.ac"`, and the second `div` endows the consonant *h* with the attribute `varRa="kaRWya.vivfta.hal"`:

```
<div>
  <uddeSya sTAna="kaRWya" AByantaraprayatna=
    "vivfta" phone="^[@(ac)]$">
    <or>
      <attribute not="yes" sTAna="tAlavya"/>
      <attribute not="yes" sTAna="ozWya"/>
    </or>
  </uddeSya>
  <AdeSa varRa="kaRWya.vivfta.ac"/>
</div>
…
<div>
  <uddeSya sTAna="kaRWya" AByantaraprayatna=
    "vivfta" phone="^[@(hal)]$"/>
  <AdeSa varRa="kaRWya.vivfta.hal"/>
</div>
```

The two `attribute` elements in the first `div` prevent it from endowing the diphthongs `e`, `o`, `ai`, and `au` with the attribute `varRa="kaRWya.vivfta.ac"`. Table 3 shows a snippet of the XML table that results from these and other phonetic classification rules. The table permits accessing the sounds that correspond to other sounds due to featural similarities.

Just as phonetic classification rules rely on knowledge of phonetic features not taught in the *Aṣṭādhyāyī*, rules that produce pratyāhāras rely upon the prior existence of some of them. The pratyāhāra ***hal*** is required to identify final markers in the *Akṣarasamāmnāya* before *A.* 1.1.71 *ādir antyena sahetā* can apply to form any pratyāhāra. While commentators offer clever interpretations that avoid circularity, we deem rather that this type of circularity was not an issue for the early grammarians. They considered that speech was abiding (*nitya*), and, as long as rules reconstitute speech forms, an axiomatic, sequential application is not strictly required.

```
<cells>
  <cell row='kaRWya' col='spfzwa1'>k
    <attribute varRa="kaRWya.spfzwa" Goza="aGoza" prARa="alpaprARa"/>
  </cell>
  <cell row='kaRWya' col='spfzwa2'>K
    <attribute varRa="kaRWya.spfzwa" Goza="aGoza" prARa="mahAprARa"/>
  </cell>
  …
  <cell row='kaRWya' col='Uzman'>h
    <attribute varRa="kaRWya.vivfta.hal"/>
  </cell>
  <cell row='kaRWya' col='hrasva'>a
    <attribute varRa="kaRWya.vivfta.ac" AyAma="hrasva"/>
  </cell>
  …
  <cell row='kaRWya' col='guRa'>a
    <attribute varRa="kaRWya.vivfta.ac" bala="guRa"/>
  </cell>
  …
</cells>
```

Table 3
Enriched XML sound classification table

4 The derivational data structure

The derivational data structure consists of one or more cooccurring streams, one of which is the main stream. Each stream consists of a sequence of states. Each state consists of a phonetic string and a list of attributes. Both streams and states can be given names that can be used to refer to them during the evaluation of a rule. Each attribute consists of a class, a value, and has a range consisting of an offset and length which generally associates it with a substring of the phonetic string. Attributes whose ranges do not point to a substring of the phone, are pending such association. An attribute can also have a property list which contains name-value pairs. A property value can either be a string, the identifier of an attribute, or an array of strings and attributes. The identifier of an attribute is its position in the list of attributes of a state. This identifier remains constant for the lifetime of the attribute in the stream. Figure 2 shows a sketch of a rule and its corresponding data structure in the default main stream.

```
<rule n="sUtra number" s="sUtra" type="sUtra type">
  <element phone="abc"/>
  <element attribute="value"/>
  <element attribute="value" phone="abc"/>
  <element attribute="value[property=pvalue]"
    phone="abc"/>
  …
</rule>

phone=abc
0 1 a-attribute
0 3 abc-attribute
```

Figure 2
Sketch of a rule and data structure

5 Rules

Rules are composed in an XML file which conforms to a DTD and a schema. While the DTD is sufficiently general to permit the file of rules to be validated against it by general XML validation software, the schema is extensively detailed and requires validation software written specifically for this purpose. Figure 3 shows the structure of the rules file. The root element of the file rules.xml is the element `grammar` which has as its children the elements `constants` and `rules`. The latter has as its children one rule element for each sūtra of the *Aṣṭādhyāyī.* Complex rules may have subdivisions in `div` elements. Each rule organizes a set of regular expressions and attributes into a tree consisting of XML elements. XML elements may contain a phone attribute that refers to a subsegment of the phonetic string in the data structure, and attributes of that subsegment.

```
<grammar>
  <constants>
    <constant name="constantName" value="sUtraNumber"/>
    …
  </constants>
  <rules>
    <rule n="A1.1.1" s="vfdDirAdEc" type="saYjYA">
      <uddeSya phone="(?:A|[@(Ec)])" flags="g"/>
      <AdeSa bala="vfdDi"/>
    </rule>
    …
    <rule n="A1.1.1" s="vfdDirAdEc" type="saYjYA">
      <conditionElement/>
      …
      <actionElement/>
    </rule>
    <rule n="sUtraNumber" s="sUtra" type="sUtraType">
      <div>
      …
      </div>
      …
    </rule>
    …
</rules>
</grammar>
```

Figure 3
XML rules file structure

XML elements are of two major classes: condition elements and action elements. Action elements include the following: `AdeSa`, `atideSa`, `call` and their children; all the rest are condition elements. Condition elements evaluate the current state of the main stream unless reference is explicitly made to another prior state or another stream. Condition elements may also evaluate possible subsequent states of the main stream by utilizing a `try` element. Building upon the look-ahead mechanism described by Scharf (2010), a `try` element instructs the derivation engine to create a hypothetical stream, to proceed with the derivation in that stream, to evaluate a condition in a particular subsequent state of that stream, and either to keep or discard the stream based upon that evaluation. The action element `AdeSa` affects the current state of the main stream; it may affect other streams only to the extent of copying constituents of them into the main stream and discarding them. An `atideSa` element replaces a phonetic string or attribute with another for the duration of specific rules designated in the scope element. A `call` element invokes another rule.

5.1 Element types

There are three types of elements:

1. normal elements
2. attributive elements
3. logical elements

The relation of the child element to its parent is determined by its type. Normal elements define substrings of the range of the phone of their parent. Attributive elements define additional attributes of their parent. Attributive elements house attributes that properly belong to the parent but cannot be put in the parent element due to limits of XML (no two attributes of the same name can occur in a single element), or to organizational motivations (attributes of a certain type may be classed together by limiting them to a specific element, e.g. attributes that define semantic conditions all appear in an `arTa` element). Attributive elements include the following: `attribute`, `arTa`, and `cooccur`. The last refers to an attribute of its parent that points to a cooccurring stream and describes a string in that steam. Logical elements include the elements `and`, `or`, and `not`, which represent Boolean operators. Logical elements, which may house any combination of normal and attributive elements, pass the characteristics of their children on to their parent. All other elements are normal elements.

Every rule must contain a source element (`uddeSya` or `sTAnin`). The range of the source element(s) determine the range to which the action elements apply. Most rules have exactly one source element as does for example the formalization of *A.* 1.1.1 in Figure 3. No rule can contain both an `uddeSya` and a `sTAnin`, but some rules may contain two `sTAnin` elements. The rules that require two substituends are those that occur under the heading *A.* 6.1.84 *ekaḥ pūrvaparayoḥ* which states that a single replacement occurs in place of two successive substituends, a preceding one and a following one. If a rule contains two `sTAnin` elements, one will be the child of a `pUrva` element, the other a child of a `para` element, and the `pUrva` and `para` elements must define contiguous ranges.

Figure 4 shows the formalization of the first single-replacement operational rule. *A.* 6.1.87 *ād guṇaḥ (aci)* provides that a single guṇa vowel occurs in place of a vowel of the class *a* and a following vowel. In Figure 4 the `pUrva` element describes the preceding string and the `para` element describes the succeeding string. The regular expression that is a value of the phone attribute in the `sTAnin` element in the former describes a vowel of class *a* and the `sTAnin` element in the latter describes any vowel. The fact that the `sTAnin` elements are children of the `pUrva` and `para` elements indicates that they describe substrings of the strings described by their parent elements. The `locus` attributes in the `sTAnin` elements fix those substrings at the end and beginning of those parent strings.

```
<rule n="A6.1.87" s="AdguRaH" type="viDi">
  <pUrva phone="^[@(al)]*([@(a)])">
    <sTAnin phone="[@(a)]" locus="anta"/>
  </pUrva>
  <para phone="([@(ac)])[@(al)]*$">
    <sTAnin phone="[@(ac)]" locus="Adi"/>
  </para>
  <AdeSa execute="guRa(para.sTAnin.phone)"
    saYjYA="antAdi"/>
</rule>
```

Figure 4
XML formalization of A. 6.1.87

A. 6.1.85, articulates a principle followed in the section headed by *A.* 6.1.84: a single replacement behaves as if it is both the end of the preceding speech unit and the beginning of the succeeding speech unit. In the formalization, the presence of the attribute `saYjYA="antAdi"` in the `AdeSa` element instructs the implementing software not only to replace both the substring defined by the `sTAnin` element that is a child of the `pUrva` element and the `sTAnin` element that is a child of the `para` element together (not each of them individually) with one single replacement, but also to adjust the ranges of all the units that contain either the preceding sound or the following sound to contain the replacement. The ranges of those containing elements will thus overlap.

5.2 Element order

Elements in rules are ordered in such a way that previous elements describe ranges of the root phone to the left of subsequent elements; subsequent elements describe ranges to the right of previous elements. A sequence of sibling elements describes a sequence of ranges in the range of their parent element. The range of each sibling is not necessarily contiguous with the preceding and subsequent sibling. If a sibling contains an `n` attribute or `locus` attribute its range is constrained within its domain. As shown in Figure 4, the attribute-value pair `locus="Adi"` constrains the range of the element to the left boundary of the domain of its parent element, and the attribute-value pair `locus="anta"` constrains the range of the element to the right boundary of the domain of its parent element. An `n` attribute whose value is an integer sequential with the integer value of a preceding element constrains the range of the element to have the range of the preceding element as its left context; otherwise, if an element does not have an `n` attribute, it does not have to be contiguous with the preceding element. Two elements are constrained to the immediate context of the source element. The range of a `pUrva` element serves as the immediate left context of the domain of a subsequent element. The range of a `para` element is the immediate right context of the domain of a preceding element. If an element contains the attribute `type=overlapping`, its domain overlaps the range of a preceding sibling. The overlapping segment is defined by the range of the required attribute `saYjYA=antAdi`. Hence the left boundary of the element with the attribute `type=overlapping` is the offset of the attribute `saYjYA=antAdi`. These attributes are used, for example, in the formalization of *A.* 6.1.103 *tasmāc śaso naḥ puṁsi* which provides that a replacement takes place immediately following the long vowel that is the single replacement provided by *A.* 6.1.102. The attributes are necessary to identify the particular long vowel provided by *A.* 6.1.102 rather than any long vowel in the string.

5.3 Regular expressions

A regular expression applies to a particular state in a particular stream in the derivational data structure specified by stream, state, and other attributes in the element in which it occurs; the default is the current state of the main stream. Regular expressions are utilized in the `phone` attribute and in the values of properties. The domain to which a regular expression in the value of a property applies is the value of a property in a particular class, in a particular attribute. The domain to which a regular expression in a `phone` attribute applies is the range of the phone. The attributes and phones in elements higher in the rule tree define a range that limits the domain of the phone to which the attributes and phones in lower elements in the tree apply. The domain of a regular expression in a `phone` attribute in an element that is the immediate child of a `rule` element is the root phonetic string.

The order of application of attributes is different for condition elements and action elements. In condition elements, the attributes that accompany the `phone` attribute in an element are applied first to limit the range in which the regular expression in that phone applies. In action elements, the regular expression in a `phone` attribute or the reference expression in a `range` attribute applies first to locate the range to which other attributes apply. Reference to regular expressions in an action element must be unambiguous; an action element cannot operate on phones that match different parts of the current string.

The domain to which a regular expression in the `phone` attribute of a condition element applies is determined by its parent element and by the other attributes in which the `phone` attribute occurs. If there are no other attributes, the range of the phone is the range of the parent element. If there are other attributes, the range of those attributes must fall within the range of the parent. The range of the phone is the range defined by the other attributes.

From the regular expressions in `phone` attributes of elements with contiguous ranges, a single regular expression is built. The right end of the regular expression of the preceding sibling meets the left end of the regular expression of the succeeding sibling. If siblings are not constrained to be contiguous, `.*`, which permits any number of characters to intervene, is inserted between them. Source elements within those elements determine groups within that regular expression. For example, in the formalization of *A.* 6.1.87 in Figure 4 the regular expression

`^[@(al)]*([@(a)][@(ac)])[@(al)]*$` is formed from the two regular expressions `^[@(al)]*([@(a)])` and `([@(ac)])[@(al)]*$` in the `pUrva` and `para` elements respectively. The two groups indicated in these two regular expressions are combined in a single group in the constructed regular expression in accordance with the single replacement principle.

If a rule is meant to apply more than once to substrings identified by regular expressions within a parent string, the attribute-value pair `flags="g"` accompanies the phone attribute which will have a regular expression absent the caret and dollar sign that anchor it at the left and right, e.g. in the `uddeSya` element in Figure 3. If the domain of a regular expression is not anchored at the left boundary of the domain of its parent or in the root domain, it should be checked repeatedly beginning at each character within the domain defined by its parent and left sibling until a match is found. A caret at the beginning of a regular expression and dollar sign at the end anchor the bounds of the string to be sought at the beginning and end of its range as defined by its parent or other attributes.

If a search fails to match with ranges defined by the first match for attributes of elements, again the attempt to match should continue with subsequent matches of those attributes until the search domain is exhausted. Once a match is found the rule is not repeated unless the attribute value pair `flags="g"` accompanies the phone attribute. We indicate greedy or non-greedy in regular expressions. A '?' indicates non-greedy. Our regular expressions do not employ nested groups.

The name of the range of a regular expression is the path from the rule element to the `phone` attribute whose value is the regular expression. (E.g. `rule.avayavin.sTAnin.phone`; an abbreviated name dropping the rule element is used in rules.xml: `avayavin.sTAnin.phone`.)

5.4 Cooccurrence conditions

A `cooccur` element describes a string in a stream other than the stream in which it occurs. The string it describes has a syntactic relation with the string described by the parent element of which the `cooccur` is the child. The attribute-value pairs `saYjYA="upapada"` and `saYjYA="upasarga"` in a `cooccur` element specify that the syntactic relation of the string described by the `cooccur` element with its parent element is a subordinate dependency relation. In the absence of one of these pairs, the dependency relation is undefined. A string described by a `cooccur` must have an attribute set in the initial conditions that tells what it is a `cooccur` of, i.e. to what it bears a relation. The relatum is indicated by the attribute value pair `r="x"`, 'related to *x*', where *x* is the identifier of an attribute created previously in the initial conditions.

Figure 5 shows the formalization of a typical rule requiring a `cooccur` element. *A.* 3.2.1 *karmaṇy aṇ* provides that the affix *aṇ* occurs after a verbal root (*dhātu*) on the condition that an agent is to be denoted and a direct object (*karman*) occurs as a subordinate term (*upapada*) connected with it. The fact that the subordinate term denotes a participant in action is indicated in the formalization by making the `cooccur` element a child of the `pUrva` element which is supplied with the attribute-value pair `saYjYA="DAtu"`. That the particular participant is a direct object, i.e. the kāraka termed *karman*, is indicated by supplying the `cooccur` element with an `arTa` element with the attribute-value pair `kAraka="karman"`.

```
<rule n="A3.2.1" s="karmaRyaR" type="viDi">
  <pUrva saYjYA="DAtu">
    <cooccur saYjYA="upapada">
      <arTa kAraka="karman"/>
    </cooccur>
  </pUrva>
  <AdeSa saYjYA="pratyaya[slp=aR]"/>
</rule>
```

Figure 5
The XML formalization of A. 3.2.1

5.5 Logic

All elements and conditions are cumulative, i.e. connected by a logical 'and' unless the elements appear as siblings that are immediate children of an `or` element. An `and` element groups cumulative conditions beneath an `or` element. The `not="yes"` attribute-value pair in an `attribute` element negates the other attributes within the same element. The `not="yes"` attribute-value pair in an `and` element that houses elements `x` and `y` means 'it is not the case that *x* and *y*', i.e. $\neg(x\&y)$. An exclamation point before a property in a property list indicates the absence of that property, e.g. in *A.* 1.3.4 `!bare` refers to a basic element not already stripped of its markers.

Elements that appear as siblings that are immediate children of an `or` element in `AdeSa` or `atideSa` elements indicate the alternate application of each. Their significance is the same as if two independent rules, each of which had just one of those siblings, applied alternatively. For example, *A.* 3.1.133 *ṇvultṛcau* provides that the affixes ***ṇvul*** and ***tṛc*** occur alternatively in the sense of an agent (*kartṛ*) after any verbal root (*dhātu*). The rule mentions the two affixes in a dvandva compound yet certainly does not intend that both affixes apply after a root at once. The XML formalization of the rule mentions each affix in the form in which it is originally taught in the sūtra as the value of the `upadeSa` property of the `pratyaya` value of the `saYjYA` attribute in an `AdeSa` element. To indicate that they alternate, the two `AdeSa` elements are made siblings of an `or` element.

5.6 Numbering and referencing

Rules are numbered *Aa.p.s*, where *a* is an integer from 1–8, *p* an integer from 1–4, *s* an integer from 1–223. An integer (currently only 1–4) in an optional set of square brackets immediately following a rule number refers to the `div` number in an `n` attribute within a rule element. For example, `A3.1.74[1]` indicates the `div` numbered with the attribute-value pair `n="1"` under *A.* 3.1.74. The elements `cooccur`, `avayava`, and `scope` are likewise numbered by `n` attributes and referenced by a digit in square brackets immediately after the element name in a path. Paths are indicated by element, attribute, value, property, and property value names separated by a period. The formalization of rules that indicate the semantic conditions under which taddhita affixes are provided provide the paths to elements that house these conditions. For example, *A.* 4.1.92 states that certain affixes occur in the sense of an offspring (*apatya*). The formalization indicates the semantic condition by putting `apatya` as the value of the attribute `santati` of an `arTa` element. Since the meaning belongs to the nominal base of the second word mentioned in the sūtra, the `arTa` element is situated in an `avayava` element in the second `cooccur` stream as follows:

```
<cooccur stream="2">
  <avayava saYjYA="prAtipadika" locus="Adi">
    <arTa santati="apatya"/>
  </avayava>
</cooccur>
```

The path of the semantic condition passed to the rules that provide affixes in this meaning is `cooccur[2].avayava.arTa. other`.

The numbering of `div`, `avayava`, and `cooccur` elements constrains their order. The order of `div` elements indicates the sequence of their implementation. The order of `avayava` elements indicates the uninterrupted sequence of occurrence in the phonetic string of the segments they describe.

The values of attributes are hierarchical. Subordinate values are indicated after a period. Values without specified subordinate values include any subordinate value as well as no subordinate value. An asterisk indicates any value.

6 Rule processing

The rules of the *Aṣṭādhyāyī* do not produce speech forms out of nothing; rules apply when their conditions are met. These conditions must be presented before any operations can apply. Hence our XML formalization presumes that a set of initial conditions is supplied. Initial conditions include both strings and attributes. Strings introduced initially consist of basic elements selected from the *dhātupāṭha* or *gaṇapāṭha*, and attributes introduced initially include semantic conditions and syntactic relations. Products derived by previous cycles of the implementation, consisting of datasets of strings and their attributes, may be introduced in subsequent initial conditions. For example, the set of kr̥t-derivates, each with its attributes accumulated in the course of its derivation, may supply some of the initial conditions for taddhita rules. Subsequent rules and actions inherit the state produced by the preceding rule or preceding action within the same rule. The attributes and phones in the data do not have to be passed as parameters.

Attributes may be introduced in the initial conditions either with a specific range or with a pending range (* *). An attribute that is pending is captured by a string that is introduced in an action element when the action element contains an attribute `vAcya` whose value is the class and value of the attribute, e.g. `vAcya="kAraka=kartf"`. Attributes are inherited by their replacements. So are markers, which are properties. Phonetic feature properties such as accent, nasalization, etc., however, are not inherited.

Attributes in the `rule`, `div`, or `AdeSa` elements (the last occurring only within a `case` element) indicate rule dynamics by referring to the number of the `rule` or `div` element (see §5.6) that is the target of the specified rule relation. Rule relations are indicated by the attributes `apodita`, `pratizidDa`, `niyata`, and `SezaTo`. The values of an `apodita` attribute indicate those rules to which this rule is an exception. The values of a `pratizidDa` attribute indicate those rules to which this rule is a negation. The values of a `niyata` attribute indicate those rules to which this rule is a restriction. Lastly, the values of a `SezaTo` attribute indicate those rules to which this rule serves as a remainder. The remainder rule functions with respect to the listed rules as an otherwise condition in a case statement does with respect to the preceding cases. Exceptions and negations apply instead of the rules of which they are exceptions or negations.

Restrictions apply in tandem with the rules they restrict. For example, *A.* 1.4.99–102 *laḥ parasmaipadam* etc. classify verbal terminations (*tiṅ*) according to their pada, person, and number by designating them by the terms *parasmaipada* and *ātmanepada*, *prathama* etc., and *ekavacana* etc. These classificatory rules can be applied prior to processing any particular initial conditions. Rules such as *A.* 3.2.123 *vartamāne laṭ*, which provides the affix ***laṭ*** after a verbal root if the action denoted by that root takes place in present time, introduce an abstract *l*-affix endowed with specific markers. *A.* 1.3.12–93 *anudāttaṅita ātmanepadam* etc., *A.* 1.4.105–108 *yuṣmady upapade …madhyamaḥ* etc., and *A.* 1.4.21–22 *bahuṣu bahuvacanam* etc. provide restrictions to the pada, person, and number of the verbal terminations to be introduced as replacements for the *l*-affix by *A.* 3.4.78 *tiptasjhi …iḍvahimahiṅ*. In the XML formalization, the restriction is indicated by the presence of the attribute-value pair `niyata="A3.4.78"` in the rule element of each of these rules. Each restrictive rule that applies adds to the data (see §4) an attribute whose range is the *l*-affix provided by rules such as *A.* 3.2.123. When the *l* is replaced by *A.* 3.4.78, the verbal termination with the corresponding attributes is selected.

Rules that restrict the selection of nominal terminations work similarly in tandem with the rule that provides them generally. In contrast to restrictions to the rule that introduces verbal terminations which are provided as replacement for an abstract *l*-affix to which restrictive attributes can be attached however, Pāṇini has not employed any such abstract nominal termination to which the restrictive attributes *vibhakti* and *number* of nominal terminations can be attached. Hence, in order to handle the restrictions in a similar fashion, in the XML formalization, the restricted rule *A.* 4.1.2

svaujas ...ṅyossup is divided into two `divs`. The first, *A.* 4.1.2[1], adds a zero-length place-holder before the restrictive rules apply. The restrictive rules associate attributes with the zero-length place-holder after which *A.* 4.1.2[2] selects the corresponding nominal termination.

7 An example

Space does not permit showing the complete derivation of a form in full. However, let us sketch a couple of phases in the simple derivation of the past passive participle stem *kr̥ta*. At least a couple of initial semantic conditions are required. They are selected from a menu, and corresponding attribute-value pairs are introduced in the main stream of the data:

1. * * phala=karaṇa
2. * * kāla=bhūta

The first indicates the basic sense of the root, and the second indicates the time of the action denoted by the root. The range of the attributes is indicated as pending (see §6). Of the two roots that have the sense of *karaṇa* in Scharf's (2009) canonical edition of the *Mādhavīyadhatuvr̥tti*, we select the root ***ḍukr̥ñ***. After root selection, the phonetic string of the root is introduced, and the initial semantic attributes `Pala` and `kAla` are associated with it by fixing their ranges to the offset and length of the phonetic string of the root. Skipping the steps of stripping the markers from the verbal root and interpreting its root accent described by Scharf (2009: Introduction §IIIA2, §IIIA3a), which in any case can be performed tangentially to any particular derivation, we arrive at the following state, with changes made by rules shown in grey:

1. phone=kf
2. 0 2 Pala=karaRa
3. 0 2 kAla=BUta
4. 0 2 saYjYA=DAtu[slp=qukf\Y, upadeSa=qukfY, bare=kf, svara=anudAtta]
5. 0 2 it=qu
6. 0 2 it=Y

We illustrate just one rule: *A.* 6.1.161 *dhātoḥ* (*anta udāttaḥ* 159), which replaces the final vowel of a root by a high-pitched vowel. The rule is formalized as follows:

```
<rule n="A6.1.162" s="DAtoH" type="viDi">
  <avayavin saYjYA="DAtu" phone=
    "^[@(al)]*([@(ac)])[@(hal)]*$">
    <sTAnin range="\1"/>
  </avayavin>
  <AdeSa svara="udAtta"/>
</rule>
```

The rule processor finds an attribute `DAtu` in the attribute list, applies the regular expression in the `phone` attribute of the `avayavin` element to the string demarcated by the range of that attribute, and writes the attribute-value pair `svara=udAtta` with the range of the group referenced by the `sTAnin` element at the bottom of the attribute list. The resulting data state, with changes made by the rule shown in grey, is as follows:

1. phone=kf
2. 0 2 Pala=karaRa
3. 0 2 kAla=BUta
4. 0 2 saYjYA=DAtu[slp=qukf\Y, upadeSa=qukfY, bare=kf, svara=anudAtta]
5. 0 2 it=qu
6. 0 2 it=Y
7. 1 1 svara=udAtta

8 Conclusion

Speakers of a language understand word-sense correspondences and distinctions. Descriptive grammars and lexicons describe these relationships casually. Contemporary linguists formalize these relationships through techniques such as generative and transformational grammars. Pāṇini likewise formalized these relationships in his generative grammar. By introducing the appropriate affixes under specific conditions Pāṇini formally associates the speech form with its meaning. Our systematic computational implementation of the *Aṣṭādhyāyī* makes Pāṇini's formal word-sense associations available in the form of a lexicon and for use in a parser.

The author is currently working with Ralph Bunker to implement the XML grammar described here in Java. Once implemented, we expect that it will be able to assist in investigating difficult issues regarding Pāṇinian methodology. For example, it may be employed as a benchmark against which to test such issues as theoretical differences regarding rule formulation and automated rule selection. The comprehensive lexicon produced from the implementation may provide data to assist in determining the relationship between the linguistic description inherent in the *Aṣṭādhyāyī* and various texts, genres, and dialects. This in turn may assist in dating and locating texts.

References

Ajotikar, Tanuja, Anuja Ajotikar, and Peter M. Scharf. 2015. “Some issues in the computational implementation of the Aṣṭādhyāyī.” In: *Sanskrit and Computational Linguistics: Proceedings of the ‘Sanskrit and the IT World’ section of 16th World Sanskrit Conference.* 16th World Sanskrit Conference. (Sanskrit Studies Centre, Silpakorn University, Bangkok, June 28–July 2, 2015). Ed. by Amba Kulkarni. New Delhi: D. K. Publishers.

Jha, Girish Nath, ed. 2010. *Sanskrit computational linguistics: 4th International Symposium, New Delhi, India, December 2010, Proceedings.* (Dec. 10–12, 2010). Lecture Notes in Artificial Intelligence 6465. Berlin; Heidelberg: Springer-Verlag.

Petersen, Wiebke. 2004. “A Mathematical Analysis of Panini’s Siva-Sutras.” *Journal of logic, language and information* 13.4: 471–89.

—. 2008. “Zur Minimalität von Pāṇinis Śivasūtras: eine Untersuchung mit Methoden der formalen Begriffsanalyse.” Ph.D. dissertation. Düsseldorf.

—. 2009. “On the Construction of Śivasūtra-Alphabets.” In: *Sanskrit computational linguistics: third international symposium, Hyderabad, India, January 2009, proceedings.* Ed. by Amba Kulkarni and Gérard Huet. Lecture Notes in Artificial Intelligence 5406. Berlin; Heidelberg: Springer-Verlag, pp. 79–98.

—. 2010. “On the generalizability of Panini’s pratyahara-technique to other languages.” In: *Sanskrit computational linguistics: 4th International Symposium, New Delhi, India, December 2010, Proceedings.* (Dec. 10–12, 2010). Ed. by Girish Nath Jha. Lecture Notes in Artificial Intelligence 6465. Berlin; Heidelberg: Springer-Verlag, pp. 21–38.

Scharf, Peter M. 2009. *Mādhavīya Dhātuvṛtti canonical index.* Providence: The Sanskrit Library. URL: `http://sanskritlibrary.org`.

—. 2010. “Rule-blocking and forward-looking conditions in the computational modeling of Pāṇinian derivation.” In: *Sanskrit computational linguistics: 4th International Symposium, New Delhi, India, December 2010, Proceedings.* (Dec. 10–12, 2010). Ed. by Girish Nath Jha. Lecture Notes in Artificial Intelligence 6465. Berlin; Heidelberg: Springer-Verlag.

Scharf, Peter M. 2013. "An analytic database of the *Aṣṭādhyāyī*." In: *Proceedings of the Fifth International Sanskrit Computational Linguistics Symposium.* (IIT Bombay, Mumbai, Jan. 4–6, 2013). Ed. by Malhar Kulkarni and Chaitali Dangarikar. New Delhi: D. K. Printworld.

—. 2014. "Are taddhita affixes provided after prātipadikas or padas?" In: Pāṇini and the Pāṇinīyas of the 16th–17th century C.E., trosième atelier du projet ANR PP16-17. (Institut Français de Pondichéry, Pondicherry, Oct. 14–16, 2014). Paper presented at the Troisième Atelier du Projet ANR PP16–17 (Pāṇini et les Pāṇinéens des XVI^{e}–XVII^{e} siècles) accueilli par EFEO, IFP, EPHE, Pondichéry, 14–16 octobre 2014. In preparation.

—. 2015. "On the status of nominal terminations in aluk and upapada compounds." In: *Proceedings of the Vyākaraṇa section of 16th World Sanskrit Conference.* 16th World Sanskrit Conference. (Sanskrit Studies Centre, Silpakorn University, Bangkok, June 28–July 2, 2015). New Delhi: D. K. Printworld.

Scharf, Peter M., Pawan Goyal, Anuja Ajotikar, and Tanuja Ajotikar. 2015. "Voice, preverb, and transitivity restrictions in Sanskrit verb use." In: *Sanskrit syntax: Selected papers presented at the seminar on Sanskrit syntax and discourse structures, 13-15 June 2013, Université Paris Diderot, with a bibliography of recent research by Hans Henrich Hock.* Ed. by Peter M. Scharf. New Delhi: D. K. Printworld; Providence: The Sanskrit Library, pp. 157–201.

Scharf, Peter M. and Malcolm D. Hyman. 2011. *Linguistic issues in encoding Sanskrit.* Delhi: Motilal Banarsidass.