

# Rule-Blocking and Forward-Looking Conditions in the Computational Modelling of Pāṇinian Derivation

Peter M. Scharf

Department of Classics,  
Brown University  
scharf@brown.edu

**Abstract.** Attempting to model Pāṇinian procedure computationally forces one to clarify concepts explicitly and allows one to test various versions and interpretations of his grammar against each other and against bodies of extant Sanskrit texts. To model Pāṇinian procedure requires creating data structures and a framework that allow one to approximate the statement of Pāṇinian rules in an executable language. Scharf (2009: 117-125) provided a few examples of how rules would be formulated in a computational model of Pāṇinian grammar as opposed to in software that generated speech forms without regard to Pāṇinian procedure. Mishra (2009) described the extensive use of attributes to track classification, marking and other features of phonetic strings. Goyal, Kulkarni, and Behera (2009, especially sec. 3.5) implemented a model of the asidhavadat section of rules (6.4.22-129) in which the state of the data passed to rules of the section is maintained unchanged and is utilized by those rules as conditions, yet the rules of the section are applied in parallel, and the result of all applicable rules applying exits the section. The current paper describes Scharf and Hyman's implementation of rule blocking and forward-looking conditions. The former deals with complex groups of rules concerned with domains included within the scope of a general rule. The latter concerns a case where a decision at an early stage in the derivation requires evaluation of conditions that do not obtain until a subsequent stage in the derivation.

## 1 Implementations of Sandhi and Inflection

Scharf and Hyman implemented Pāṇinian sandhi rules in a portable framework using modified regular expressions in an XML file to model Pāṇinian rules, as described in Scharf 2009: 118-120. Each rule is written as one or more XML rule tags each of which contains several parameters: source, target, lcontext, rcontext, optional, and c. The optional parameters lcontext and rcontext specify the left and right contexts for the replacement of the source by the target. The optional parameter `optional` specifies that the current state is to be duplicated and subsequent parallel paths created, one in which the rule is implemented and the other in which it is not. The parameter `c` (for comment) contains the

number of the Pāṇinian rule implemented by the rule tag. The implementation utilizes the Sanskrit Library Phonetic encoding scheme SLP1, described in Scharf and Hyman 2010, Appendix B, in which Sanskrit sounds and common phonetic features such as tones and nasalization are each represented by a single character.

The rule syntax utilizes a number of macros that model Pāṇinian structures. Macros are used to model Pāṇinian sound classes: *varṇa*, *varga*, *guṇa*, *vr̥ddhi*, *saṁprasāraṇa*, etc.; to create *pratyāhāras*: अक्, अण्, इक्, यण्, etc.; and to group sounds with common phonetic features: aspirated sounds, unaspirated sounds, voiced sounds, unvoiced sounds, etc. For example, the macros @(f) and @(x) in 1.1.9 vt. represent the *varṇas* ऋ and ॠ respectively. The macros @(eN) in 6.1.109 and @(ac) in 6.1.78 represent the *pratyāhāras* एङ् (monothongs) and अच् (vowels), respectively. Mappings are used to map sets of sounds onto corresponding sounds, such as short vowels onto long, and unvoiced stops onto voiced stops. Functions, such as *lengthen*, *guRate*, and *vfdDiize*, utilize the mappings to facilitate implementation of common operations, namely, the replacement of a vowel by its corresponding long vowel, *guṇa* vowel, or *vr̥ddhi* vowel, respectively.

Rules are not pre-selected by hand; rather they are triggered by the data that meets the conditions for the application of the rule. Yet rules are arranged in sequence and placed in blocks to ensure appropriate application of general rules and exceptions. In particular, negations and identical replacements that are exceptions select paths that avoid the application of rules of which they are negations and exceptions. Hyman wrote a Perl program that converts the XML file of regular expressions to Perl executable code. The model succeeds in encoding Pāṇinian rules in a manner that allows the rules that come into play to be tracked. Rule tracking has valuable research and pedagogical applications. Hyman (2009) describes the procedure by which the XML vocabulary to express Pāṇini's sandhi rules was developed and how a series of stages converts the rules not only into executable Perl code, but also into a network, and a finite state transducer. The latter, being extremely fast, will permit realtime web use of the models.

Scharf and Hyman augmented the XML data structures created to model Pāṇinian sandhi to allow derivation of nominal stems, as described in Scharf 2009: 120-23. They introduced an additional parameter *morphid* in the XML rule tag that utilizes Scharf's (2002: 29-30) set of nominal inflection tags. They further enriched the XML structure utilized for nominal declension to model verbal conjugation. Scharf (2009: 123-125) described Scharf and Hyman's implementation of a computational model of Pāṇinian verbal inflection in a single cascade of rules that apply to whatever strings meet their conditions. The procedure begins with a single set of verbal terminations for all verbs, just as Pāṇini does, and introduces replacements on the basis of phonetic context. They widened the set of tags employed by the parameter *morphid* to utilize, in addition to nominal inflection tags, Scharf's (2002: 30-31) verbal inflection tags. They added two parameters to the rule tag: *lexid*, and *root*. The former allows reference to the class of the root in the Pāṇinian *Dhātupāṭha*. The latter allows reference to the original form of the root even when the previous rules have modified the input

string. The implementations of Pāṇinian inflection include rule tracking so that a derivational history of the form can be provided.

## 2 Participles

In 2008, Scharf and Hyman enriched the XML tagset further to allow derivation of participle stems. A `vlexid` parameter was added to allow reference to the class of the verbal root in the *Dhātupāṭha* and an `affix` parameter to allow reference to the affix in the form in which it was originally introduced. The implementation of 1.2.21 demonstrates both parameters. 1.2.21 is a negation rule that optionally denies क्-marking to the affixes क्त and क्तवत् that have the initial augment इ, when they occur after a verbal root with a penultimate short उ.

```
<rule source="k" target=""
lcontext="@ (begin) [@ (al)] *u [@ (hal)] #i: (:ta|tavat); [@ (it)] *"
rcontext="[@ (it)] *$" vlexid="^vt?1" lexid="^(ppp|pap)$"
affix="@ (nizWA)"
root="^[@ (char)] *~? [@ (char)] *u [@ (hal)]; ? [@ (char)] *$"
optional="yes"
c="1.2.21 udupaDAdBAvAdikarmaRoranyatarasyAm"/>
<!--Only roots with the stem-forming affix शप्, for now roots
of class 1 or 10, are subject to the negation of k-marking. Kā-
śikā: व्यवस्थितविभाषा चेयम् । तेन शङ्खिकरणानामेव भवति । Rule includes
vlexid.-->
```

The parameter `source` has the value of the क् to be deleted; the parameter `target` has the null string value. The parameter `lcontext` has the value of a regular expression matching strings preceding the `source` parameter. These include conditions specified in the ablative in the rule, namely, a string in which there is a penultimate short उ, as well as any markers that might occur prior to the marker क् to be deleted. The pound sign in the string marks the morpheme boundary between the stem and the affix. This is followed by the initial augment इ separated from the rest of the affix by a colon. The subsequent parenthesis matches either `ta` or `tavat` representing the phonetic strings त or तवत् respectively. A semicolon separates the phonetic form of the affix from its markers, regardless of whether they are initial or final. The value of the `lcontext` parameter ends with an expression including a macro reference that matches any possible marker.

```
<macro name="it" value="ufkNcYRtnprlS" c="1.3.2-8"/>
```

The `rcontext` parameter in the implementation of 1.2.21 again has the value of any marker that might follow क् represented in the `source` parameter. Now it is possible that the string त belongs to some affix other than क्त. It is also possible that an affix has a final marker क् rather than an initial one. In order to ensure reference to the desired affix, the original form of the affix is included as the value of the `affix` parameter. Here the location of markers as initial or final

is preserved. The affixes in question are introduced with an initial marker क् and are termed निष्ठा. Initial and final markers are separated from the phonetic form of the affix by a grave accent and a semicolon respectively. A macro implements the classification rule as follows:

```
<macro name="nizWA" value="k`ta|k`tavat;u"
c="1.1.26 ktaktavatU nizWA"/>
```

The macro name nizWA is then employed in the implementation of 1.2.21, just as the technical term निष्ठा is in 1.2.19 from where it recurs. Recurring terms are explicitly stated in XML implementations of rules. The option parameter has the value yes which initiates two streams of derivation from this point forward, one in which the rule is applied and one in which it is not.

Now the *Kāśīkā* on 1.2.21 states that the option is specifically distributed (व्यवस्थित) in that the rule applies only when the affixes occur after roots to which the stem-forming affix (*vikarāṇa*) शप् will be introduced (व्यवस्थितविभाषा चेयम् । तेन शङ्खिकरणानामेव भवति ।). In order to limit the rule in accordance with the *Kāśīkā*'s statement, the rule includes the parameter vlexid which has the value of the lexical tag associated with the verbal root in our digital *Dhātupāṭha*. Pertinently, vlexid contains the roots' class. The example shown of the implementation of 1.2.21 demonstrates the utility of the two new parameters vlexid and affix.

### 3 Blocking

Scharf and Hyman enriched the structure of the xml file containing rules for participle derivation to allow complex blocking relations. Pāṇini formulates general rules (उत्सर्ग) and exceptions (अपवाद). Exceptions take precedence over their related general rule. Where the exception alters the string so that it no longer meets the conditions for the application of the general rule, it is easy to implement exceptions by simply ordering the rules in such a way that the exceptions have the opportunity to apply first. This is achieved by placing them earlier in the cascade of rules. However, rule ordering alone is insufficient to capture the structure of Pāṇini's blocking relations where the exceptions do not alter the string. This is particularly obvious where the exception takes the form of a negation. In previous versions of their framework, Scharf and Hyman rewrote general rules that had negative exceptions to reflect the narrower scope of application in the conditions of the general rule itself. Such a procedure makes it more difficult to implement accurate rule tracking. Hence we have enriched the framework of the XML rule file to reflect blocking.

Let's take, as an example, rules concerning the provision of the initial augment इ. 7.2.35 provides that the augment इ occurs as the initial part of an ārdhadhātuka affix that begins with a consonant other than य्.

```
<rule source="" target="i:" lcontext="#"
rcontext="[@(val)][@(al)]*@(anta)" affix="^(@(ArDaDatuka))$"
c="7.2.35 ArDaDatukasyeq valAdeH"/>
```

For the purposes of the participial derivation in this ruleset, the macro name **ArDaDatuka** refers to any one of several affixes that form what are typically called participals as follows:

```
<macro name="ArDaDatuka"
value="k`vas;u|k`Ana;c|k`ta|k`tavat;u|k`tvA|tum;un|@(kftya)"
c="3.4.114 ArDaDatukaM SezaH, 3.4.115 liw ca"/>
```

The macro **kftya** again refers to what are called gerundives:

```
<macro name="kftya"
value="tavya|tavya;t|anIya;r|ya;t|k`ya;p|R`ya;t"
c="3.1.95 kftyAH prANRvulaH"/>
```

The general rule above accounts for the **इ**-augment in the infinitive **लवितुम्** (< लू 'cut' + **इ**-तुम्) and the gerundive stem **लवितव्य** (< लू 'cut' + **इ**-तव्य), for example.

Several negations that disallow the addition of the initial augment **इ** are exceptions to 7.2.35. Their domains are entirely included within the domain of the general rule; hence these negations would have no scope of application if they were not given precedence over 7.2.35. For example, 7.2.8 provides that the initial augment **इ** does not occur in affixes termed *kṛt* that begin with a voiced consonant other than **य्** or **ह्**. The domain of 7.2.8 is included within the domain of 7.2.35 because all *kṛt*-affixes that begin with a voiced consonant other than **य्** or **ह्** are also *ārdhadhātuka* affixes that begin with a consonant other than **य्**. Obviously the set of voiced consonants excluding **य्** and **ह्** is a subset of the set of consonants excluding **य्**. Moreover, most *kṛt*-affixes are *ārdhadhātuka*; only eight *kṛt*-affixes are termed *sārvadhātuka* rather than *ārdhadhātuka* due to being marked with **श्**, but they all begin with a vowel, in particular with **अ** or **आ**.<sup>1</sup> Since the domain of 7.2.8 is entirely included within the domain of 7.2.35, 7.2.8 is an exception to 7.2.35 and takes precedence over it. The XML rule 7.2.8 is formulated as follows:

```
<rule source="" target="" lcontext="#"
rcontext="[@(vaS)][@(aI)]*:[@(it)]*k[@(it)]*"$
affix="^(@(kft))$" c="7.2.8 neqvaSi kfti"/>
```

The rule includes reference in the **affix** parameter to the macro **kft**, which lists several *kṛt*-affixes relevant to participle formation, including absolutive and non-Vedic infinitives.

<sup>1</sup> शतृ (3.2.124, 3.2.130), शानच् (3.2.124), शानन् (3.2.128), चानश् (3.2.129), खश् (3.2.28, 3.2.83), श (3.1.137, 3.3.100), and शध्यै and शध्यैन् (3.4.9). Note that the vikaraṇas श्रु (3.1.73, 3.1.82), श्रम् (3.1.78), and श्रा (3.1.81, 3.1.83) are taught before the heading धातोः in 3.1.91; hence they are not subject to being termed *kṛt* according to the *Kāśikā* on 3.1.93 which limits the rule to the range of the heading धातोः in 3.1.91 (अस्मिन् धात्वधिकारे, etc.). Although these vikaraṇas are not subject to the **इ**-augment's negation by 7.2.8, they are not subject to its provision by 7.2.35 or any other rule either.

```

<macro name="kft"
value="S`at;f|S`Ana;c|S`Ana;n|c`Ana;S|k`vas;u|k`Ana;c|k`ta|
k`tavat;u|k`tvA|tum;un|@(kftya)"
c="3.1.93 kfdatiN"/>

```

The exception 7.2.8 does nothing to change the conditions that would allow the general rule 7.2.35 from applying subsequently; it replaces nothing by nothing at the beginning of the affix. Now, if 7.2.8 were placed prior to 7.2.35 in the cascade of rules without any other restriction, 7.2.35 would proceed to add the  $\mathfrak{K}$ -augment there. This is not desired. 7.2.8 should prevent 7.2.35 from applying. To achieve this, Hyman created an XML structure similar to an **otherwise** statement found in some programming languages. The structure groups rules within a block that contains two subsections: a **try** section and an **otherwise** section, as shown:

```

<block>
  <try>
    <rule/>
    <rule/>
    ...
  </try>
  <otherwise>
    <rule/>
    <rule/>
    ...
  </otherwise>
</block>

```

To effect the blocking of 7.2.35 by 7.2.8, then we write 7.2.8 in the **try** block and 7.2.35 in the **otherwise** block as follows:

```

<block>
  <try>
    ...
    <rule source="" target="" lcontext="#"
      rcontext="[@(vaS)][@(al)]*[@(it)]*k[@(it)]* $"
      lexicid="^(@(kft)) $" c="7.2.8 neqvaSi kfti"/>
    ...
  </try>
  <otherwise>
    <rule source="" target="i:" lcontext="#"
      rcontext="[@(val)][@(al)]*@(anta)"
      affix="^(@(ArDaDAtuka)) $"
      c="7.2.35 ArDaDAtukasyeq valAdeH"/>
  </otherwise>
</block>

```

All instances of exceptions whose domains are totally included within the domain of their related general rule can be handled similarly. In instances of

partial blocking, the rule that contains a domain partially included within the domain of a related general rule must be split into a rule with a totally included domain and a rule with an excluded domain. Only the rule with the totally included domain will be placed in the `try` block.<sup>2</sup>

## 4 Look Ahead

Some rules in the *Aṣṭādhyāyī* apply only under conditions that are not created until subsequent rules apply. Let's consider the case of 7.2.67, necessary for the derivation of perfect active participles. Rules in the अङ्ग-section (6.4.1–7.4.97) apply before rules for doubling and related stem-internal changes (6.1.1 etc.). 1.1.59 द्विवचने ऽचि provides that the replacement of a vowel whose replacement is conditioned by a vowel-initial affix has the status of its substituent for the purpose of doubling in the section of rules headed by 6.1.1 एकाचो द्वे प्रथमस्य (See Cardona 1997: 60). This implies that the augment इ precedes vowel deletion which in turn precedes doubling. Likewise, rules providing the augment इ to the beginning of an affix apply before rules that change the stem prior to the augmented affix. Hence 7.2.67 applies to add the augment इ to the affix -ङ्सु before 6.4.98 applies to delete the penultimate vowel of the root before vowel-initial affixes. The augment must be provided first because without it, the affix would not be vowel-initial. 6.4.98 in turn applies prior to 6.1.8 लिटि धातोरनभ्यासस्य. However, 7.2.67, the rule that provides the augment, includes among its conditions that the root be single-syllabled, and the *Kāśikā* explains that this refers to the root after doubling has applied. But doubling can't apply until after the augment इ is added, which conditions deletion of the penultimate vowel. It is not possible to evaluate the condition of being a single-syllabled doubled root at the time the augment is added. It is not possible to evaluate a state of affairs brought about subsequent to a rule as a condition at the time of application of that rule. Hence it is necessary to apply the rule tentatively, then trace the derivation subsequent to the application of the rule to the point where its conditions can be evaluated before deciding whether to apply the rule or not. Our XML framework implements look ahead to achieve this by deriving the form both with and without the rule application, then abandoning the rejected line of derivation at the point the decision is made.

Let's trace the derivation of the perfect active participle of the root गम् 'go' to illustrate our implementation. The Pāṇinian derivation is shown in Table 1; the Scharf-Hyman derivation in Table 2.

The Scharf-Hyman framework does not at present implement sthānivadbhāva; it does not reintroduce replaced sounds at certain points where they are required in the conditions of subsequent rules. Instead, for the time being, rules are modified to include the replacements in the conditions of the subsequent rule. Hence our derivation of the perfect active participle of गम् differs from the Pāṇinian

<sup>2</sup> See Scharf (forthcoming) for a thorough examination of principles for determining rule precedence in cases of conflict between rules that have independent as well as overlapping domains (विप्रतिषेध).

**Table 1.** Pāṇinian derivation

Perfect active participle of the root गम्

1	गम्(लृ)	MDhP 1.702	गम् सृप् गतौ
2	गम्-ल्(इट्)[लिट्]	3.2.115	परोक्षे लिट्
3	गम्-वस्(कु)	3.2.107	ऋसुश्च
4	गम्-इःवस्(क्)	7.2.67	वस्वेकाजाद्वसाम्
5	गम्-इःवस्(क्)	6.4.98	गमहनजनखनघसां लोपः किङित्यनडि
6a	गम्-इःवस्	1.1.59	द्विवचने ऽचि
6	गम्-इःवस्	6.1.8	लिटि धातोरनभ्यासस्य
7	गम्-इःवस्	7.4.60	हलादिः शेषः
8	जग्म्-इःवस्	7.4.62	कुहोश्चुः
9	जग्मिवस्		Delete morpheme boundaries

**Table 2.** Scharf-Hyman derivation

Perfect active participle of the root गम्

1	गम्(लृ)	MDhP 1.702	गम् सृप् गतौ
2	गम्-ल्(इट्)[लिट्]	3.2.115	परोक्षे लिट्
3	गम्-वस्(कु)	3.2.107	ऋसुश्च
4	गम्-इःवस्(क्)	7.2.67	वस्वेकाजाद्वसाम्
5	गम्-इःवस्(क्)	6.1.8	लिटि धातोरनभ्यासस्य
6	गम्-इःवस्(क्)	6.4.98	गमहनजनखनघसां लोपः किङित्यनडि
7	गम्-इःवस्	7.4.60	हलादिः शेषः
8	जग्म्-इःवस्	7.4.62	कुहोश्चुः
9	जग्म्-इःवस्	7.2.67	वस्वेकाजाद्वसाम्
10	जग्मिवस्		Delete morpheme boundaries

derivation. While Pāṇini implements penultimate vowel deletion by 6.4.98 (Table 1, step 5) prior to doubling by 6.1.8 (Table 1, step 6), we implement doubling by 6.1.8 first (Table 2, step 5) and then delete the penultimate vowel by 6.4.98 (Table 2 step 6), modifying the condition for deletion to accomodate the doubled root. To avoid dealing with the issue of sthānivadbhāva, deletion of the penultimate vowel is implemented after doubling by including the parameters to match a doubled root in the conditions for the application of 6.4.98. The `root` parameter compensates for the overbroad `lcontext` parameter to ensure the application of the rule only to the proper roots.

```
<rule source="a" target=""
lcontext="@ (begin) (?: [ghjKG] a[mns] [ghjKG]) "
rcontext="[mns] # (?! a; N@ (anta)) [ @ (ac) ] [ @ (al) ] * @ (kNit) "
root="^(g\am;/x|h\an;/a|j/an;/I|j\an;/a|j/an;/I|K/an;~u|G/as;/x)$"
c="6.4.98 gamahanajanaKanaGasAM lopaH kNityanaNi"/>
```



In either derivation, 7.2.67 at step 4 must anticipate the state of the derivation at a subsequent step (at least step 6 in our derivation or step 7 in the Pāṇinian derivation). Our framework implements 7.2.67 in two XML rules to separate the single-syllable condition, which requires look-ahead, from conditions that don't require look-ahead. The single-syllable portion is written as follows:

```
<rule source="" target="i:"
lcontext="^(?:@upasarga-|) [@(hal)]* [@(ac)] [@(hal)]*#"
rcontext="vas@(anta)" affix="^(k`vas;u|S`at;f)$"
root="@ (DpAdi) [@(hal)]* [^(/\\) [@(ac)] [@(hal)]*@(Dpanta)"
c="7.2.67 vasvekA jAdGasAm"/>
```

At step 4 we begin two lines of derivation, one with the augment and one without. Only the one with is shown in Table 2. At step 9, we evaluate the condition in 7.2.67 and, finding that the string qualifies for augmentation, throw out the derivation without the augment.

We look forward to utilizing the enriched framework created for participle derivation in a revised, more faithful model of Pāṇinian inflection and hope to go on to implement derivational morphology generally.

## References

1. Cardona, G.: Pāṇini: His Work and its Traditions. Vol. I, Background and Introduction, 2nd edn. Motilal Banarsidass, Delhi (1997)
2. Goyal, P., Kulkarni, A., Behera, L.: Computer Simulation of *Aṣṭādhyāyī*: Some Insights. In: Huet, G., Kulkarni, A., Scharf, P. (eds.) *Sanskrit Computational Linguistics 2007/2008*. LNCS (LNAI), vol. 5402, pp. 139–161. Springer, Heidelberg (2009)
3. Huet, G., Kulkarni, A., Scharf, P. (eds.): *Sanskrit Computational Linguistics 2007/2008*. LNCS (LNAI), vol. 5402. Springer, Heidelberg (2009)
4. Hyman, M.: From Pāṇinian Sandhi to Finite State Calculus. In: Huet, G., Kulkarni, A., Scharf, P. (eds.) *Sanskrit Computational Linguistics 2007/2008*. LNCS (LNAI), vol. 5402, pp. 253–265. Springer, Heidelberg (2009)
5. Mishra, A.: Simulating the Pāṇinian System of Sanskrit Grammar. In: Huet, G., Kulkarni, A., Scharf, P. (eds.) *Sanskrit Computational Linguistics 2007/2008*. LNCS (LNAI), vol. 5402, pp. 127–138. Springer, Heidelberg (2009)
6. Scharf, P. M.: *Rāmopākhyāna*—the Story of Rāma in the Mahābhārata: An Independent-study Reader in Sanskrit. Routledge Curzon, London (2002)
7. Scharf, P.: Modeling Pāṇinian Grammar. In: Huet, G., Kulkarni, A., Scharf, P. (eds.) *Sanskrit Computational Linguistics 2007/2008*. LNCS (LNAI), vol. 5402, pp. 95–126. Springer, Heidelberg (2009)
8. Scharf, P.: Rule selection in the *Aṣṭādhyāyī* or Is Pāṇini's grammar mechanistic? In: *Proceedings of the 14th World Sanskrit Conference*, Kyoto University, Kyoto, September 1-5 (2009) (forthcoming)
9. Scharf, P., Malcolm, H.: *Linguistic issues in encoding Sanskrit*. The Sanskrit Library, Providence. Motilal Banarsidass, Delhi (2010)