Sanskrit Sentence Generator

Amba Kulkarni & Madhusoodana Pai J
Department of Sanskrit Studies
University of Hyderabad
apksh.uoh@nic.in, jmadhusoodan@gmail.com

Abstract

In this paper we describe a sentence generator for Sanskrit. Pāṇini's grammar provides the essential grammatical rules to generate a sentence from its meaning structure. The meaning structure is an abstract representation of the verbal import. It is the intermediate representation from which, using Pāṇini's rules, without appealing to the world knowledge, the desired sentence can be generated. At the same time, this meaning structure also represents the dependency parse of the generated sentence.

Keywords: Sanskrit, Sentence Generator, Pāṇini, Paninian Grammar, Computational Linguistics.

1 Introduction

Natural language generation (NLG) is the process of generating text from a meaning representation. It may be thought of as the reverse of natural language understanding (NLU). There has been considerably less focus in NLG than in NLU. Nevertheless, a generator is an essential component of any machine translation (MT) system. It is also needed in systems such as information summarization, question answering, etc. NLG systems are also being used by human writers to make the writing process efficient and effective (Galitsky, 2013). In the field of computational creativity, the interest does not lie any more on how a computer can generate creative pieces on its own but rather how such systems can be used to assist a person in a creative task. Poem machine by Hämäläinen () is an example of an online tool to generate Finnish poetry with a computationally creative agent. Automatic advertisement slogan generators (Iwama and Kano, 2018) are being used by Japanese.

NLG is also useful for second language learners. Second language learners can use such modules to generate sentences in a controlled way and learn the language at their own pace. For a classical language like Sanskrit which is for most of the people a second language and not the mother tongue, a computational aid can help a user in several ways. Some of the aspects where such an aid would be useful are listed below.

- Sanskrit is an inflectional language. That means the case suffixes (vibhakti-pratyayas) get attached to the stem (prātipadika/dhātu) and during the attachment some morpho-phonetic changes also take place. In some cases, one can't tell apart the stem and its suffix. This increases the load on memorization.
- Each Sanskrit noun has a gender which is independent of the sex or animacy of the referent. In Sanskrit, gender is an integral part of the nominal stem (prātipadika). That means one has to remember the gender of each nominal stem since the word forms differ with gender as well. The gender has no relation to the meaning/denotation of the word. For example wife in Sanskrit can be either a *patnī* in feminine gender or *dārā* in masculine gender or *kalatra* in neuter gender.
- The participants of an action are termed kārakas. The definitions of these kārakas are provided by Pāṇini which are semantic in nature. However, the exceptional cases make them syntactico-semantic. For example, in the presence of the prefix *adhi* with the verbs

śīṅ, *sthā* and *as*, the locus instead of getting the default adhikaraṇaṁ role gets a karma (goal) role and subsequently accusative case marker, as in *saḥ grāmam adhitiṣṭhati* (He inhabits/governs the village) where *grāma* gets a karma role, and is not an adhikaraṇaṁ.

- There are a set of words in whose presence a nominal stem gets a specific case marker. For example, in the presence of *saha*, the accompanying noun gets instrumental case suffix. The noun denoting the body part causing the deformity also gets an instrumental case suffix as in *akṣṇā kāṇaḥ* (one-eyed). Most of these rules being language specific, the learner has to remember all the relevant grammar rules.
- Sanskrit has a natural tendency to use passive (karmaṇi) with transitive verbs and impersonal passive (bhāve) with intransitive verbs. If the native language of a learner does not permit such usages, s/he finds it difficult to understand/construct sentences with such usages.
- There are also cases where the verbs in different pada (ātmanepada/ parasmaipada) have different meanings. A speaker, by mistake, if uses a wrong pada, the sentence may not convey the desired meaning. For example, the verb *bhuj* from rudhādi-gaṇa when used in the meaning of eating is always in ātmanepada while in the sense of *to rule* or *to govern* it is used in parasmaipada.[1]
- In the causative constructions, the semantics associated with certain participants is different for different sets of verbs. For example, for the verbs denoting motion, the causer is also a karman with respect to the causative action. And then in such cases, even a person who has studied grammar well gets confused in assigning proper case marker to the verbs. The confusion grows more if the senetnce is to be expressed in passive voice.

All these problems make the life of a Sanskrit speaker difficult. Even if a person has passive control, due to the above-mentioned problems, he either shies away from speaking / writing Sanskrit or ends up in speaking /writing wrong Sanskrit. Finally, the influence of mother tongue on Sanskrit speaking also results in wrong/nativized Sanskrit. A speaker who does not want to adulterate Sanskrit with the influence of his/her native language would like to have some assistance, and if it were by a mechanical device such as a computer, it would be advantageous.

With these problems in mind, and also the possible applications in computational linguistics as mentioned above, we decided to build a Sanskrit sentence generator.

2 Approaches

Natural language generation is comparatively easier to handle than natural language understanding. NLU involves handling of ambiguities, whereas the main problem in NLG is selection of appropriate lexicon and syntax for expressions. In the late nineties of the last millennium, several NLGs were developed which were general purpose (Dale, 2000). But they were difficult to adopt to small task oriented applications. Two different methods were used to develop NLGs - rule based and template based. A rule based system can generate sentences without any restriction, provided the rules are complete. A template based generation on the other hand is delimited in its scope by the set of templates. A programme that sends individualized bulk mails is an example of template based generation. There have been efforts to mix the use of rule based and template based generation. The recent trend in NLG, as with all other NLP systems is to use machine learning algorithms using large databases.

With the availability of a full-fledged generative grammar for Sanskrit in the form of Aṣṭādhyāyī, it is appropriate to use a rule based approach for building the generation module. A lot of work in the area of Sanskrit Computational linguistics has taken place in the last decade, some of which is related to the word generators. So we decided to use the existing word generators and build a sentence generator, modelling only the sūtras that correspond to the assignment of case markers.

In the next section, we discuss our approach to building a sentence generator using rules

[1] bhujo'navane(1.3.66)

from kāraka and vibhakti sections of Pāṇini's Aṣṭādhyāyī. In the fourth section, we provide the implementation details. In the fifth section we discuss the interface while the usability of the sentence generator is reported in the last section.

3 Sentence Generator: Architecture

Pāṇini has given a grammar which is generative in nature. He presents a system of grammar that provides a step by step procedure to transform thoughts in the minds of a speaker into a language string. Broadly speaking one may imagine three mappings in the direction from semantics to phonology ((Bharati et al., 1994), (Kiparsky, 2009)). These levels are represented pictorially as in Figure 1.

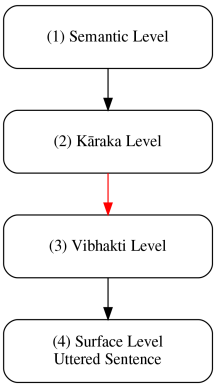

Figure 1: Levels in the Pāṇinian model

3.1 Semantic Level

This level corresponds to the thoughts in the mind of a speaker. The information is still at the conceptual level, where the speaker has identified the concept and has concretised them in his mind. The speaker, let us assume, for example, has witnessed an event where a person is leaving a place and is going towards some destination. For our communication, let us assume that the speaker has identified the travelling person as person#108, the destination as place#2019, and the action as move-travel#09. Also the speaker has decided to focus on that part of the activity of going where the person#108 is independent in performing this activity, and that the goal of this activity is place#2019. This establishes the semantic relations between person#108 and move-travel#09 as well as between place#2019 and move-travel#09. Let us call these relations sem-rel#1 and sem-rel#2 respectively. This information at the conceptual level may be represented as in Figure 2.

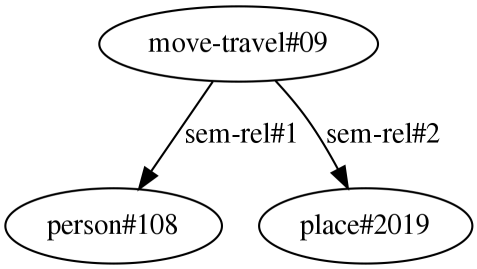

Figure 2: Conceptual representation of a thought

3.2 Kāraka Level

In order to convey this, now the speaker chooses the lexical items that are appropriate in the context from among all the synonyms that represent each of these concepts. For example, for the person#108, the speaker chooses a lexical term, say *Rāma*, among the synonymous words *{ayodhyā-pati, daśarathanandana, sītā-pati, kausalyā-nandana, jānakī-pati, daśa-ratha-putra, Rāma, ...}*. Similarly corresponding to the other two concepts, the speaker chooses the lexical terms say *vana* and *gam* respectively. With the verb *gam* is associated the pada and gaṇa information along with its meaning.

Having selected the lexical items to designate the concepts, now the speaker chooses appropriate kāraka labels corresponding to the semantics associated with the chosen relations. He also makes a choice of the voice in which to present the sentence. Let us assume that the speaker in our case decides to narrate the incidence in the active voice. The sūtras from Aṣṭādhyāyī now come into play. The semantic roles sem-rel#1 and sem-rel#2 are mapped to kartā and karma, following the Pāṇinian sūtras

- svatantraḥ kartā(1.4.54); which assigns a kartā role to Rāma.
- karturīpsitatamaṁ karma(1.4.49); which assigns a karma role to vana.

Let us further assume that the speaker wants to convey the information as it is happening i.e., in the present tense (vartamāna-kāla). Thus at the end of this level, the available information is as shown in Figure 3.

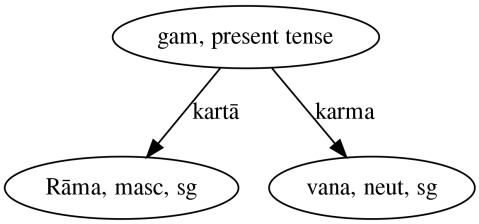

Figure 3: Representation in abstract grammatical terms

This information is alternately represented in simple text format as shown below.

word index	stem	features	role
1	Rāma puṃ	eka	kartā 3
2	vana napuṃ	eka	karma 3
3	gam parasmaipada bhvādi	vartamāna	kartari

The first field represents the word index which is used to refer to a word while marking the roles. The second field is the stem (with gender in case of nouns), the third field provides morphological features such as number, tense, etc. and the fourth field provides the role label and the index of the word with respect to which the role is marked.

3.3 Vibhakti Level

Now the sūtras from vibhakti section of Pāṇini's Aṣṭādhyāyī come into play. Vana which is a karma, gets accusative (dvitīyā) case marker due to the sūtra *karmaṇi dvitīyā (anabhihite) (2.3.2)*. Since the sentence is desired to be in active voice, kartā is abhihita (expressed), and hence it will get nominative (prathamā) case due to the sūtra - *prātipadikārtha-liṅga-parimāṇa-vacana-mātre prathamā(2.3.46)*. The verb gets a laṭ lakāra due to vartamāna-kāla (present tense) by the sūtra -*vartamāne laṭ(3.2.123)*. It also inherits the puruṣa (person) and vacana (number) from the kartā *Rāma*, since the speaker has chosen an active voice. Thus at this level, now, the information available for each word is as follows.

word index	stem	features	role
1	Rāma puṃ	eka	kartā 3
2	vana napuṃ	eka	karma 3
3	gam_1	vartamāna	kartari

Table 1: Input to Sentence Generator

word index	stem	morphological features
1	Rāma puṃ	eka prathamā
2	vana napuṃ	eka dvitīyā
3	gam parasmaipada bhvādi	laṭ prathama eka

3.4 Surface Level

With this information, now each pada is formed using the available word generator. Sandhi at the sentence level is optional. If the speaker intends, then the sandhi rules come into play and a sentence with sandhi is formed. Thus we get either *Rāmaḥ vanaṃ gacchati* or optionally *Rāmo vanaṅgacchati* as an output.

3.5 Sentence Generation: Input and Output

In the above architecture, there are three modules:

1. A module that maps the semantic information in the form of abstract concepts and abstract semantic relations into the linguistic elements viz. the nominal / verbal stem and syntactico-semantic relations
   We have not implemented this module yet. However we have conceptualised it as follows. A user interface is planned, to model this part, through which the speaker selects the proper lexical terms as well as declares his intention selecting the syntactico-semantic relations and the voice. The gender associated with the nominal stem is provided by the interface, and the user does not have to bother about it. The user only provides the nominal stem, chooses the number and its role with respect to the verb. In the case of verbs, the user selects the verb based on its meaning, and the information of pada and gaṇa is automatically picked by the interface, coding this information in the form of a subscript. User also chooses appropriate relations between the words. The user interface takes care of exceptional cases hiding the language specific information from the user. The output of this module is, for the example sentence under discussion, is as shown in the Table 1.
2. A module that maps the syntactico-semantic relations to the morpho-syntactic categories such as case marker and position (in the case of upapadas, for example)
   In this paper we describe this second module in detail that maps the syntactico-semantic relations into morpho-syntactic categories. The input to the generator is a set of quadruplets as shown in the Table 1. The first element provides the index, the second the stem, the third the morphological features and the last one the relation and the index of the second relata (viz. anuyogin). The current version recognises only the following expressions for stem-feature combinations, where '?' represents optionality, '*' is the Kleene operator for zero or more occurences.
   (a) {Noun}{Taddhita}?{Gender}{Vacana}?
   (b) {Upasarga}*{Verb}{Sanādi_suffix}{Kṛt_suffix}{Vacana}?
   (c) {Upasarga}*{Verb}{Sanādi_suffix}{prayoga}{lakāra}
   Number and Gender are not specified if it has an adjectival relation with other word.
   This representation is the same as the internal representation of the output of the Saṃsādhanī[2] parser. We call this representation, an intermediate form, or the meaning structure. It represents the verbal import of the sentence in abstract form, hiding the details of which

[2]http://scl.samsaadhanii.in/scl

linguistic unit codes what information.

3. A module that composes a surface form/word form from the morphological information.

This third module corresponds to the word generation. Given the morphological information, this module produces the correct form of the word. For this module, the word-generator developed in-house[3], which is also a part of Saṃsādhanī tools is being used. We decided to produce the output in unsandhied form. Hence, for this example, the output would be

Rāmaḥ vanaṃ gacchati.

The focus of this paper is on the second module viz. morphological spellout rules.

4 Morphological spellout module

There are 3 major tasks that are carried out in this module.

1. Assigning case marker to the substantive based on its syntactico-semantic role,
   In Pāṇini's grammar we come across 3 different types of case marker assignment. They are
   (a) case marking for a kāraka relation,
   (b) case marking in the presence of certain words called upapadas,
   (c) case marking expressing the noun-noun relations
   All these sūtras are found in the third section of the second chapter of Aṣṭādhyāyī from 2.3.2 till 2.3.50.
2. Inheriting morphological features of the adjectives from their heads, and
3. Assigning morphological features for finite verbs such as person and number, and
4. Assigning lakāra corresponding to the tense, aspect and modality of the verb.

Now we explain each of these steps below.

4.1 Assigning case marker

For generating the substantial forms, we need the case marker corresponding to the kāraka role. The default cases for kartā, karma, karaṇaṁ, sampradānaṁ, apādānaṁ and adhikaraṇaṁ are 3,2,3,4,5,and 7 respectively, provided the kāraka is an-abhihita (not expressed). When the kartā (karma) is expressed by the verbal suffix, then kartā (karma) gets the nominative case suffix by *prātipadikārthaliṅgaparimāṇavacanamātre prathamā*(2.3.46). Similarly, in the case of causatives, the case markers get decided based on the semantics of the verbal roots. For example, the sūtra *gatibuddhipratyavasānārthaśabdakarmākarmakāṇāmaṇi kartā sa ṇau* (1.4.52) assigns a karma role and hence accusative case suffix to the prayojya-kartā, if the verb has one of the following meaning - motion, eating, knowledge or information related, or it is a verb with literary work as a karma or it is an intransitive verb. We have summarized all these rules in Appendix A.

For other kārakas viz. karaṇaṁ, sampradānaṁ, apādānaṁ and adhikaraṇaṁ, the case assignment is pretty straightforward. However, there is some problem, from the user's perspective, in the selection of a kāraka. We illustrate this problem with examples.

1. In the presence of the prefix *adhi* with the verbs *śīṅ*, *sthā* and *as*, the locus instead of getting the default adhikaraṇaṁ role, gets a karma (goal) role, as in *saḥ grāmam adhitiṣṭhati* (He inhabits/governs the village) where *grāma* gets a karma role, and is not an adhikaraṇaṁ. Now this is an exception to the rule, and only the native speaker of Sanskrit might be aware of this phenomenon. The user, based on his semantic knowledge, would consider *grāma* a locus, and the generator then will fail to generate the correct form.

2. Another problem is with cases of exceptions under apādānaṁ and sampradānaṁ. For a verbal root *bhī* to mean *to be afraid of*, according to Pāṇini's grammar, the source of fear is termed apādānaṁ. But this is not obvious to a user who has not studied Pāṇini's grammar. He may treat it as a cause. Similarly, in the case of motion verb *gam*, the destination, according to the Pāṇini's grammar is a karma, but due to the influence of native language such as Marathi or Malayalam, the speaker may think it as an adhikaraṇaṁ.

[3]http://scl.samsaadhanii.in/scl

Another case is of the relation between two nouns such as part and whole, kinship relations, or relation showing the possession, as in *vṛkṣasya śākhā* (the branches of a tree), *Daśarathasya putraḥ* (son of Dasharatha) and *Rāmasya pustakam* (Rama's book). In all these cases Sanskrit uses a genitive case. Pāṇini does not discuss the semantics associated with all such cases, neither he proposes any semantic role in such cases. He deals with all such cases by a single rule *ṣaṣṭhī śeṣe (2.3.50)* assigning a genitive case in all the residual cases. While for analysis purpose, it is sufficient to mark it as a generic relation, for the generation purpose, the user would like to specify the semantics associated with it as part-and-whole-relation, or kinship, etc.

Hence in all such cases, we plan[4] to provide templates of expectancies for such verbs and internally they are mapped to the Pāṇinian labels. The set of tags providing the role labels and other relations are provided in Appendix A. These tags were found to be appropriate for both analysis as well as generation (Kulkarni, 2019). This tagset essentially consists of the kāraka roles which account for the direct participants in the activity, other tags such as hetu (cause), prayojanaṁ (purpose), kriyāviśeṣaṇaṁ (adverb), etc. which indicate the modifiers of the action, tags such as pūrvakāla (precedence) showing the relation between sub-ordinate clause with the main clause, and tags marking the relations between nouns such as adjectival relation, etc. All these relations are semantic in nature.

One more set of relations between nouns is due to the upapadas (accompanying words). In the presence of an upapada, the accompanying word gets a specific case marker. For example, in the presence of *saha*, the accompanying word gets an instrumental case. This is again language specific, and hence non-native speakers of Sanskrit may go wrong in speaking sentences that involve upapadas. Pāṇini has not provided any semantic interpretation associated with such upapadas. (Kulkarni, 2019) has provided a semantic classification of these upapadas (See Appendix A).

Handling Causatives: In Sanskrit a causative suffix (ṇic) is added to the verbal root to change the sentence from non-causative to causative. In kartari ṇic prayoga, the prayojakakartā being expressed by the verbal suffix gets nominative case. If the verb is transitive, the karma gets dvitīyā vibhakti by anabhihite karmaṇi dvitīyā. The prayojyakarma however behaves in a different way with different verbs. Next, in the case of karmaṇi ṇic prayoga, karma being abhihita gets nominative case and prayojakakartā gets instrumental case. Now when the verb is dvikarmaka, which of the two karmas is expressed and which is unexpressed is decided on the basis of the verbal root. In the case of verbal roots *duh, yāc, pac, daṇḍ, rudhi, pracchi, chī, brū, śāsu, jī, math, muṣ* mukhyakarma gets accusative case and gauṇakarma gets nominal case. In the case of verbal roots *nī, hṛ, kṛṣ, vah* gauṇakarma gets accusative case and mukhyakarma gets nominal case [5]. Following Pāṇini's grammar, we have classified the verbs into semantic classes as below.

- akarmaka (intransitive)
- sakarmaka (transitive)
  - verbs in the sense of to motion, knowledge or information, eating and the verbs which have literary work as their object
    - ∗ verbs in the sense of motion
- dvikarmaka (ditransitive)-type 1
- dvikarmaka (ditransitive)-type 2

This list then takes care of the proper vibhakti assignment in all the type of causatives. See AppendixA for the summary of all rules.

4.2 Handling adjectives

Consider the following input to the system, which has viśeṣaṇa in it.

[4]The work is in progress, and hence is not being reported.

[5]*pradhānakarmaṇyākhyeye lādīnāhurdvikarmaṇām . apradhāne duhādīnām ...(akathitaṃ ca (Mahābhāṣyam))*

word index	stem	features	role
1	vīra		viṣeṣaṇam 2
2	Rāma puṃ	eka	kartā 3
3	vana napuṃ	eka	karma 3
4	gam_1	vartamāna	kartari

Table 2: example with adjective

Note here that no morphological features have been provided for the viśeṣaṇaṁ. In order to generate the correct word form of the word vīra, we need its gender, number, and case (liṅga, vacana, vibhakti). Only information available to the generator from the user that *vīra* is a viśeṣaṇaṁ of the second word. The required information is inherited from the parent node i.e. the viśeṣya. If the adjective is a derived participle form of a verb, which itself may have kāraka expectancies, we provide the necessary verbal root and the participle suffix also as input parameters for generation. For example, in Table 3, vyūḍhaṁ is an adjective of pāṇḍavānīkaṁ, and the stem and the features for it are provided as *vi+vah1* and *bhūtakarma* respectively.

4.3 Handling finite verbs

In the case of verb form generation, the verb form generator needs the information of

- pada,
- gaṇa,
- puruṣa,
- vacana, and
- lakāra.

to generate the verb form.

Pāṇini has given sūtras to assign lakāras for different tense and mood. For example -*vartamāne laṭ(3.2.123)*. These sūtras are implemented as a hash data structure that maps the tense and mood to the lakāra. The voice determines the person and number of the verbal form. If the voice is kartari (karmaṇi), then the person and number information is inherited from the kartā(karma). In the case of impersonal passive (bhāve), the person and number are assigned the values third (prathama-puruṣa) and singular(eka-vacana) respectively. A note on the information of puruṣa is in order. As we notice, the information of person is not provided with a noun stem in the input. Then from where does the machine get this information? Here we use Pāṇini's sūtras:

- yuṣmadyupapade samānādhikaraṇe sthāninyapi madhyamaḥ(1.4.105).
- asmadyuttamaḥ(1.4.107).
- śeṣe prathamaḥ(1.4.108).

Next comes the information about pada and gaṇa. We notice that, though the majority of the verbs belong to a single gaṇa, there are several dhātus which belong to more than one gaṇa. For example the very first dhātu in the dhātupāṭha viz *bhū* belongs to two different gaṇas viz bhvādi and curādi. It is the meaning which distinguishes one from the other. *Bhū* in bhvādigaṇa is in the sense of sattāyām (to exist) and the one in the curādigaṇa is in the sense of prāptau (to acquire). A detailed study of the verbs belonging to different gaṇas is carried out by (Shailaja, 2014). She has indexed these dhātus for distinction. The verb generator of Saṃsādhanī uses these indices to distinguish between these verbs. The speaker, on the other hand, would not be knowing these indices. So we provide a user interface to the user wherein the user can select the dhātu, gaṇa and its meaning, and the interface assigns a unique desired index automatically.

If a verb has ubhayapada both the parasmaipada and ātmanepada forms would be generated. Otherwise only the form with associated pada would be generated. Certain verbs use different padas to designate different meanings. For example, the verb *bhuj* has two meanings viz. *to eat* and *to rule* or *to govern*. In the sense of *to eat*, the verb has only ātmanepada forms and in the sense of *to govern*, it has only parasmaipada forms. In such cases, the user interface hides all

these complexities from the user.

4.4 Evaluation

In order to evaluate the coverage, a list of around 1000 sentences is manually collected covering a wide range of syntactic phenomenon and also verbs with different expectancies. Each sentence is parsed with the available parser and the parsed output, which is the same as the meaning representation or the semantic input for the generation, is manually verified. This semantic representation is given to the generator as an input.

There were a few challenges in the evaluation. In the absence of a taddhita (secondary derivatives) word generator, we provide the nominal stem formed by affixing the taddhita suffix. For example, we directly provide the stem śaktimat instead of śakti + matup. Similarly in the absence of a handler for feminine suffix, we provide the stem formed after the addition of feminine suffix as in anarthā (which is formed by adding a feminine suffix to anartha). In order to handle the out of vocabulary words, we developed a morphological analyser that assigns the default paradigm for the generation of such words.

5 Sanskrit Sentence Generator: Interface

The Graphical User Interface (GUI) of the Sanskrit Sentence Generator facilitates a user to provide the required input in a prescribed form. As mentioned earlier, all the language specific details such as the gaṇa, pada information of a verb, or the gender of a nominal stem are hidden from the user. The user just selects the appropriate nominal / verbal stem and the grammatical relations among the words. Figure 4 shows the generator interface for the following input.

word index	stem and features	relation
1	dṛś1	pūrvakālaḥ 11
2	tu	sambandhaḥ 1
3	pāṇḍava-ānīka {puṃ eka}	karma 1
4	vi+vah1 {bhūtakarma}	viśeṣaṇam 3
5	duryodhana {puṃ eka}	kartā 11
6	tadā	kālādhikaraṇam 11
7	ācārya {puṃ eka}	karma 8
8	upa_sam+gam1	pūrvakālaḥ 11
9	rājan	abhedaḥ 5
10	vacana {napuṃ eka}	karma 11
11	brū1 {anadyatanabhūtaḥ}	kartari

Table 3: Input for the generator

We have also provided another interface. This interface takes the input from the Sanskrit parser. It allows us to test the completeness of both parser as well as the generator at the sentence level. This interface takes the machine internal representatin of the parser's output (which is the same as shown in the Table 1) and feeds it to the generator. The overall architecture of our generator (and parser) is as shown in Figures 5 and 6.

6 Conclusion

Pāṇini's grammar provides a grammatical framework for generation. While the complexity of Sanskrit generation lies at the word level, the sentence generation is pretty straightforward. The only challenge in designing the generator was in deciding the granularity of the semantic relations appropriate for both analysis and generation. We wanted to make sure that the grammatical relations used are universal in nature, without carrying any baggage of the language idiosyncrasy. Having confirmed that this tagset is appropriate for both generation and analysis (Kulkarni, 2019), we can now open it for other languages as well; to start with the Indian languages. Now

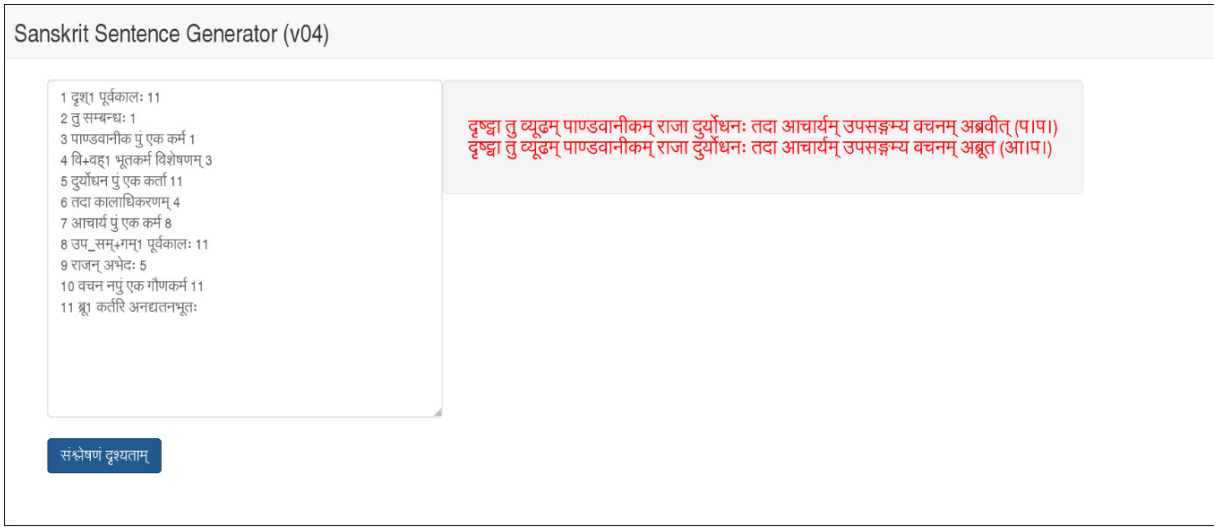

Figure 4: Generation of a Shloka from its analysis

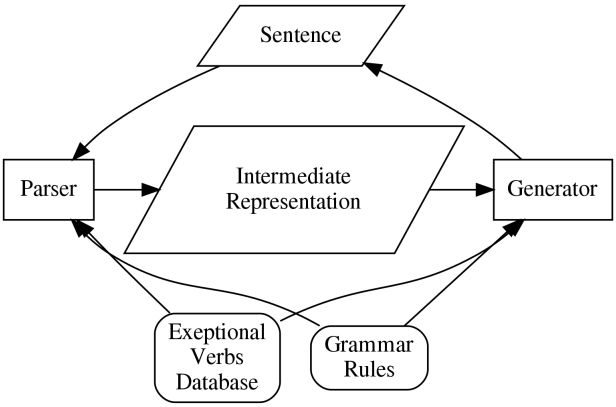

Figure 5: parser-generator: inverse operations

we are in the process of designing a user interface that hides the language and grammar specific details from the user and allows him to provide the input purely in semantic form.

Having said this, now we list some advantages and limitations of our generator.

1. This generator can be plugged in to a machine translation system.
2. It acts as a useful aid to the non-native speakers of Sanskrit to write in Sanskrit effectively guaranteeing grammatically correct sentences.
   - One need not memorize the word forms and the gender of the nominal stems
   - No need to remember all the special rules assigning case suffix to a noun representing the specific kāraka role.
   - With a single keystroke, one can generate passive constructs which are predominantly found in Sanskrit literature, with which a non-native speaker may not be at ease with.
   - The generator does not dictate any word order. So one may generate a sentence in any word order as one desires. In the future, it should also be possible to provide a generator that will help the user to render the text in a chosen prosodic meter.
3. The generator is useful for testing the parser performance as well. Since both the modules are developed independently, testing helps in mutual improvement of the systems.
4. The major contribution of the development of this module was in identifying some morpho-syntactic relation labels such as those due to upapadas (Kulkarni, 2019).

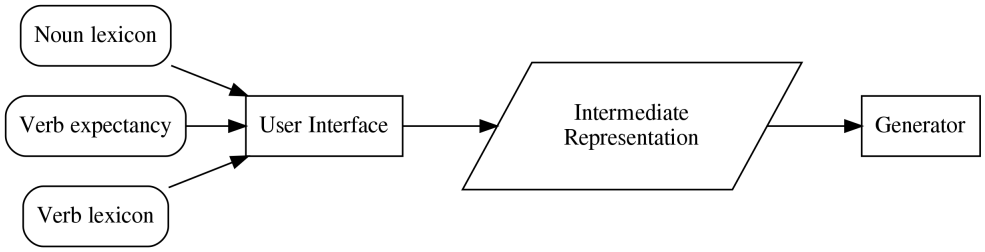

Figure 6: User interface

5. One disadvantage of this generator is the amount of information one has to provide for generation in a particular format.
6. While most of the relation labels are semantic in nature, one may need some initial training for the proper use of some relational tags.
7. One also needs some training in specifying the use of conjuncts and disjuncts since the current implementation is dominated by the syntax of Sanskrit(Panchal and Kulkarni, forthcoming). More research is needed to arrive at a uniform treatment of the conjuncts across languages.

References

[Bharati et al.1994] Akshar Bharati, Vineet Chaitanya, and Rajeev Sangal. 1994. *Natuaral Language Perscpective - A Paninian Perspective*. Prentice Hall of India.

[Cardona2007] George Cardona. 2007. On the Structure of Pāṇṇini's System. *Sanskrit Computational Linguistic*, 1&2:1–31.

[Dale2000] Ehud Dale, Robert; Reiter. 2000. *Building natural language generation systems*. Cambridge University Press, Cambridge, U.K.

[Galitsky2013] Boris Galitsky. 2013. A web mining tool for assistance with creative writing. In *Advances in Information Retrieval. Lecture Notes in Computer Science. Lecture Notes in Computer Science. 7814*.

[Hämäläinen] Mika Hämäläinen. Poem Machine - a Co-creative NLG Web Application for Poem Writing. *Department of Digital Humanities, University of Helsinki*.

[Iwama and Kano2018] Kango Iwama and Yoshinobu Kano. 2018. Japanese advertising slogan generator using case frame and word vector. In *Proceedings of The 11th International Natural Language Generation Conference, Japan*, pages 197–198, Japan, November. Association for Computational Linguistics.

[Joshi2009] S. D. Joshi. 2009. Background of the Aṣṭādhyāyī. *Sanskrit Computational Linguistic*, 3:1–5.

[Kiparsky2009] Paul Kiparsky. 2009. On the Architecture of Pāṇṇini's Grammar. *Sanskrit Computational Linguistic*, 1&2:32–94.

[Kulkarni2019] Amba Kulkarni. 2019. Appropriate Dependency Tagset for Sanskrit Analysis and Generation. In *Proceedings of Sanskrit in China International Conference 2019: Sanskrit on Paths*. forthcoming.

[Panchal and Kulkarniforthcoming] Sanjeev Panchal and Amba Kulkarni. forthcoming. Ca-śabdayuktavākyaviśleṣaṇam. In Gauri Mahulikar, editor, *Proceedings of NFSI*. Chinmaya Vishvavidyalaya, Veliyanad.

[Pande1992] Gopal Dutt Pande. 1992. *Aṣṭādhyāyī of Pāṇini*. Chaukhamba Surbharti Prakashan, Varanasi.

[R and P2017] Perera R and Nand P. 2017. Recent Advances in Natural Language Generation: A Survey and Classification of the Empirical Literature. *Computing and Informatics*, 36 (1):1–32.

[Ramakrishanamacharyulu2009] K.V. Ramakrishanamacharyulu. 2009. Annotating Sanskrit Texts Based on Śābdabodha Systems. *Sanskrit Computational Linguistics*.

[Rao1969] Veluri Subba Rao. 1969. *The Philosophy of a Sentence and its Parts*. Munshiram Manorharlal Publishers, New Delhi.

[Shailaja2014] N. Shailaja. 2014. *Comparison of Paninian Dhātuvṛttis*. Ph.D. thesis, Department of Sanskrit Studies, University of Hyderabad.

A Tagset of Dependency Relations

- **Kāraka-sambandhāḥ**
- kartā
  - prayojaka-kartā
  - prayojya-kartā
- karma
  - mukhya-karma
  - gauṇa-karma
  - vākya-karma
- karaṇam
- sampradānam
- apādānam
- adhikaraṇam
  - kāla-adhikaraṇam
  - deśa-adhikaraṇam
  - viṣaya-adhikaraṇam
- **Kāraketara-sambandhāḥ**
  - **Kriyā-kriyā-sambandhāḥ**
    - pūrva-kālaḥ
    - vartamāna-samāna-kālaḥ
    - bhaviṣyat-samāna-kālaḥ
    - bhāvalakṣaṇa-pūrva-kālaḥ
    - bhāvalakṣaṇa-vartamāna-samāna-kālaḥ
    - bhāvalakṣaṇa-anantara-kālaḥ
    - sahāyaka-kriyā
  - **Kriyā-sambandhāḥ**
    - sambodhyaḥ
    - hetuḥ
    - prayojanam
    - kartṛ-samānādhikaraṇam
    - karma-samānādhikaraṇam
    - kriyāviśeṣaṇam
    - pratiṣedhaḥ
  - **Nāma-nāma-sambandhāḥ**
    - śaṣṭhī-sambandhaḥ
    - aṅgavikāraḥ
    - vīpsā
    - viśeṣaṇam
    - sambodhana-sūcakam
    - vibhaktam
    - avadhiḥ
    - abhedaḥ
    - lyapkarmādhikaranam
    - nirdhāraṇam
    - atyanta-saṃyogaḥ
    - apavarga-sambandhaḥ
    - vakyakarmadyotakaḥ
- **Upapada-sambandhāḥ**
  - sandarbhabinduḥ
  - tulanābinduḥ
  - viśayādhikaraṇam
  - nirdhāraṇam
  - prayojanam
  - udgāravācakaḥ
  - saha-arthaḥ
  - vinā-arthaḥ
  - svāmī
  - srotaḥ
- **Vākyetarasambandhāḥ**
  - anuyogī
  - pratiyogī
  - nitya-sambandhaḥ
- **Samuccayādisambandhāḥ**
  - samuccitaḥ
  - samuccaya-dyotakaḥ
  - anyataraḥ
  - anyatara-dyotakaḥ

Note: The bold entries are the headings and do not indicate relation labels