A Sanskrit Compound Processor

Amba Kulkarni Anil Kumar

Department of Sanskrit Studies
University of Hyderabad
Hyderabad

June 21, 2012

- Sanskrit is very rich in compound formation.
  - vedavedāṅgatatvajña
  - pravaramukuṭamaṇimaricīmañjarīcayacarcitacaraṇayugula
  - jalādivyāpakapṛthivītvābhāvapratiyogipṛthivītvavatī

Sanskrit Compounds

- It is a single word (ekapadam).
- It has a single case suffix (ekavibhaktikam) with an exception of *aluk* compounds such as yudhiṣṭhiraḥ, where there is no deletion of case suffix of the first component.
- It has a single accent(ekasvaraḥ).
- The order of components in a compound is fixed.
- No words can be inserted in between the compounds.
- The compound formation is binary with an exception of dvandva and bahupada bahuvrīhi.
- Euphonic change (sandhi) is a must in a compound formation.
- Constituents of a compound may require special gender or number different from their default gender and number. e.g. pāṇipādam, pācikābhāryaḥ, etc.

Syntactic classification

supāṁ supā tiṅā nāmnā dhātunātha tiṅāṁ tiṅā |
subanteti vijñeyaḥ samāsaḥ ṣaḍvidhoḥ budheḥ ||

- Subanta (noun) + Subanta (noun) (*rājapuruṣaḥ*)
- Subanta (noun) + Tinanta (verb) (*paryyabhūṣayat*)
- Subanta (noun) + nāma (nominal base) (*kumbhakāraḥ*)
- Subanta (noun) + Dhātu (verbal root) (*kaṭaprū*)
- Tinanta (verb) + Subanta (noun) (*kṛntavicakṣaṇā*)
- Tinanta (verb) + Tinanta (verb) (*khādata*-modata)

Semantic classification

The Sanskrit compounds are classified semantically into four major types:

Tatpuruṣaḥ

(Endocentric with head typically to the right)

Bahuvrīhiḥ

(Exocentric)

Dvandvaḥ

(Copulative)

Avyayībhāvaḥ

(Endocentric with head typically to the left and behaves as an indeclinable)

Compound processor

1. Segmentation (Samāsapadacchedaḥ)
2. Constituency Parsing (Sāmarthyanirdhāraṇam)
3. Type Identification (Samāsabhedanirdhāraṇam)
4. Paraphrase generation (Vigrahavākyanirmāṇam)

Segmentation (Samāsapadacchedaḥ)

Split a compound into its constituents.

tapassvādhyāyaniratam

is segmented as

tapas-svādhyāya-niratam

Segmentation (Samāsapadacchedaḥ)

Split a compound into its constituents.

tapassvādhyāyaniratam
is segmented as
tapas-svādhyāya-niratam

Constituency Parsing (Sāmarthyaniradhāraṇam)

This module parses the segmented compound syntactically by pairing up the constituents in a certain order two at a time.

tapas-svādhyāya-niratam
is parsed as
<<**tapas-svādhyāya**>**-niratam**>

Segmentation (Samāsapadacchedaḥ)

Split a compound into its constituents.

tapassvādhyāyaniratam
is segmented as
tapas-svādhyāya-niratam

Constituency Parsing (Sāmarthyaniradhāraṇam)

This module parses the segmented compound syntactically by pairing up the constituents in a certain order two at a time.

tapas-svādhyāya-niratam
is parsed as
<<**tapas-svādhyāya**>**-niratam**>

Type Identification (Samāsabhedanirdhāraṇam)

- Decide the type of a compound at each node of composition.

<<**tapas-svādhyāya**>**Di-niratam**>**T7**

Segmentation (Samāsapadacchedaḥ)

Split a compound into its constituents.

tapassvādhyāyaniratam
is segmented as
tapas-svādhyāya-niratam

Constituency Parsing (Sāmarthyaniradhāraṇam)

This module parses the segmented compound syntactically by pairing up the constituents in a certain order two at a time.

tapas-svādhyāya-niratam
is parsed as
<<**tapas-svādhyāya**>**-niratam**>

Type Identification (Samāsabhedanirdhāraṇam)

- Decide the type of a compound at each node of composition.

<<**tapas-svādhyāya**>**Di-niratam**>**T7**

Paraphrase generation (Vigrahavākyanirmāṇam)

tapaḥ ca svādhyāyaḥ ca = tapassvādhyāyaḥ (= tat1)
gloss: penance and self-study
tasmin nirataḥ = tapassvādhyāyaniratah
gloss: who is constantly engaged in penance and self-study

Compound processor

1. **Segmentation (Samāsapadacchedaḥ)**
2. Constituency Parsing (Sāmarthyaniradhāraṇam)
3. Type Identification (Samāsabhedanirdhāraṇam)
4. Paraphrase generation (Vigrahavākyanirmāṇam)

Compound Segmenter

- The task of a segmenter is to split a given sequence of phonemes into a sequence of morphologically valid segments.

Compound Segmenter

- The task of a segmenter is to split a given sequence of phonemes into a sequence of morphologically valid segments.
- The compound formation involves a mandatory sandhi.

Compound Segmenter

- The task of a segmenter is to split a given sequence of phonemes into a sequence of morphologically valid segments.
- The compound formation involves a mandatory sandhi.
- Each sandhi rule is a triple *(x, y, z)* where *y* is the last letter of the first primitive, *z* is the first letter of the second primitive, and *x* is the letter sequence resulting from the euphonic combination.

Compound Segmenter

- The task of a segmenter is to split a given sequence of phonemes into a sequence of morphologically valid segments.
- The compound formation involves a mandatory sandhi.
- Each sandhi rule is a triple *(x, y, z)* where *y* is the last letter of the first primitive, *z* is the first letter of the second primitive, and *x* is the letter sequence resulting from the euphonic combination.
- For analysis, we reverse these rules of sandhi and produce *y + z* corresponding to a *x*.

Compound Segmenter

- The task of a segmenter is to split a given sequence of phonemes into a sequence of morphologically valid segments.
- The compound formation involves a mandatory sandhi.
- Each sandhi rule is a triple *(x, y, z)* where *y* is the last letter of the first primitive, *z* is the first letter of the second primitive, and *x* is the letter sequence resulting from the euphonic combination.
- For analysis, we reverse these rules of sandhi and produce *y + z* corresponding to a *x*.
- Only the sequences that are morphologically valid are selected.

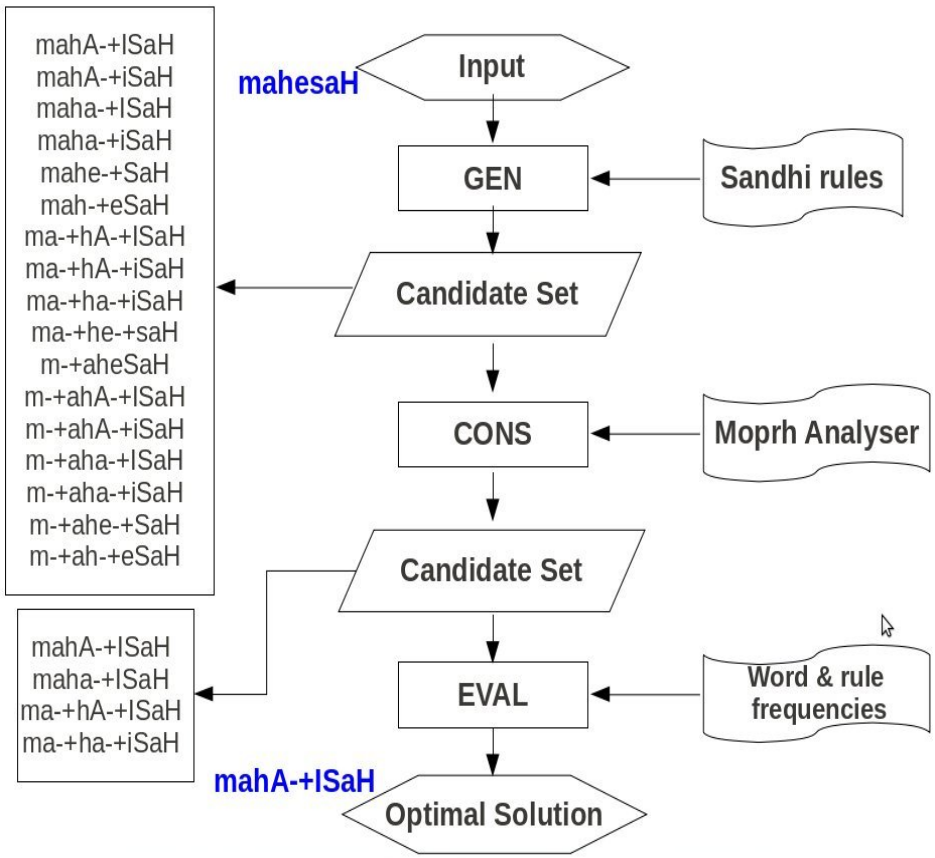
Compound Segmenter

- The task of a segmenter is to split a given sequence of phonemes into a sequence of morphologically valid segments.
- The compound formation involves a mandatory sandhi.
- Each sandhi rule is a triple *(x, y, z)* where *y* is the last letter of the first primitive, *z* is the first letter of the second primitive, and *x* is the letter sequence resulting from the euphonic combination.
- For analysis, we reverse these rules of sandhi and produce *y + z* corresponding to a *x*.
- Only the sequences that are morphologically valid are selected.
- We follow GENerate-CONstrain-EVALuate paradigm attributed to the Optimality Theory for segmentation.

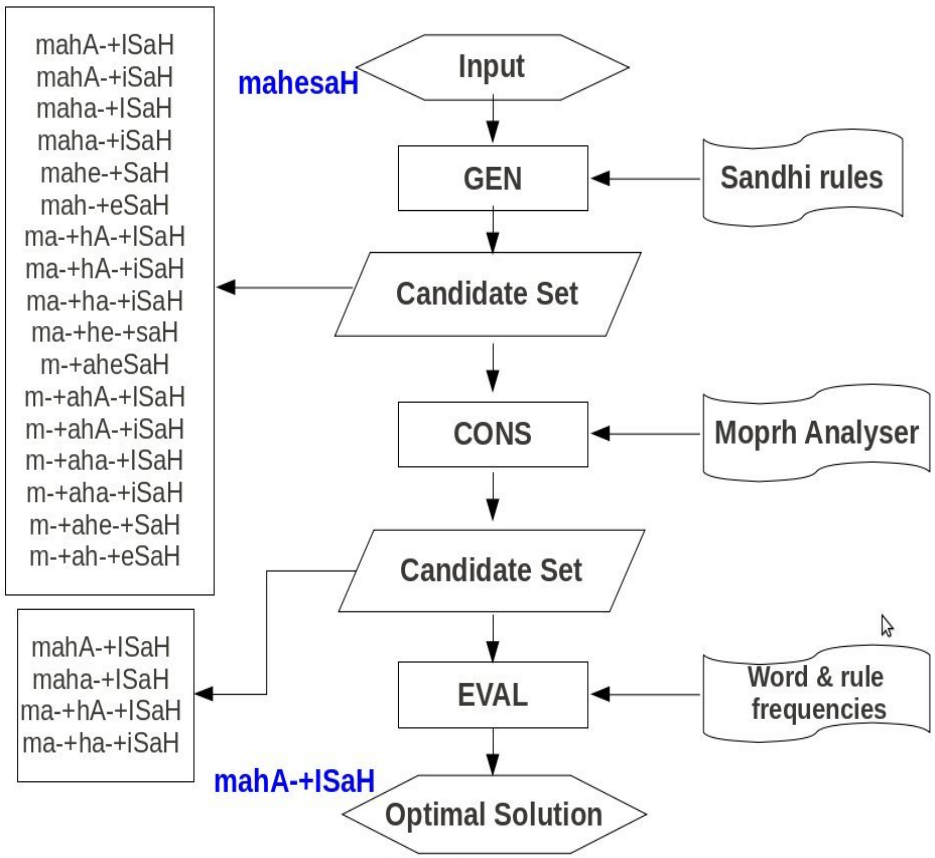
Compound Segmenter

- The task of a segmenter is to split a given sequence of phonemes into a sequence of morphologically valid segments.
- The compound formation involves a mandatory sandhi.
- Each sandhi rule is a triple *(x, y, z)* where *y* is the last letter of the first primitive, *z* is the first letter of the second primitive, and *x* is the letter sequence resulting from the euphonic combination.
- For analysis, we reverse these rules of sandhi and produce *y + z* corresponding to a *x*.
- Only the sequences that are morphologically valid are selected.
- We follow GENerate-CONstrain-EVALuate paradigm attributed to the Optimality Theory for segmentation.
- The Optimality Theory basically addresses the issue of generation.

Flow-chart represetation of Compound Segmentation



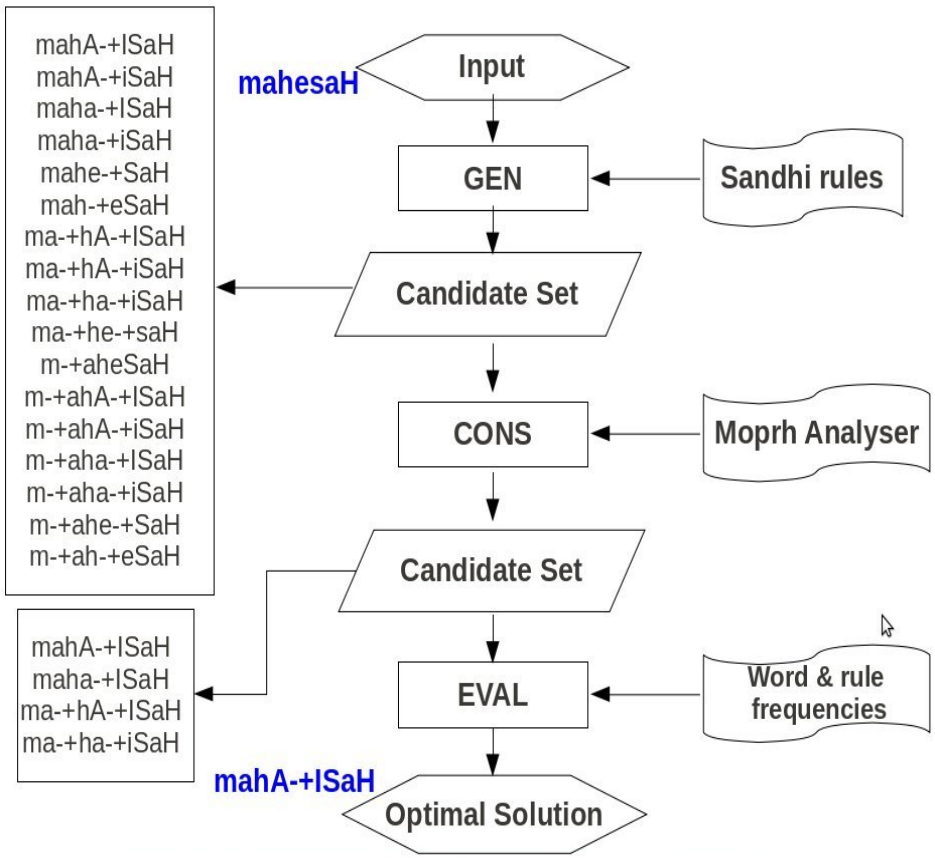
Flow-chart represetation of Compound Segmentation



The basic outline of the algorithm is:

1. Recursively break a word at every possible position applying a sandhi rule and generate all possible candidates for the input. (17 segments)

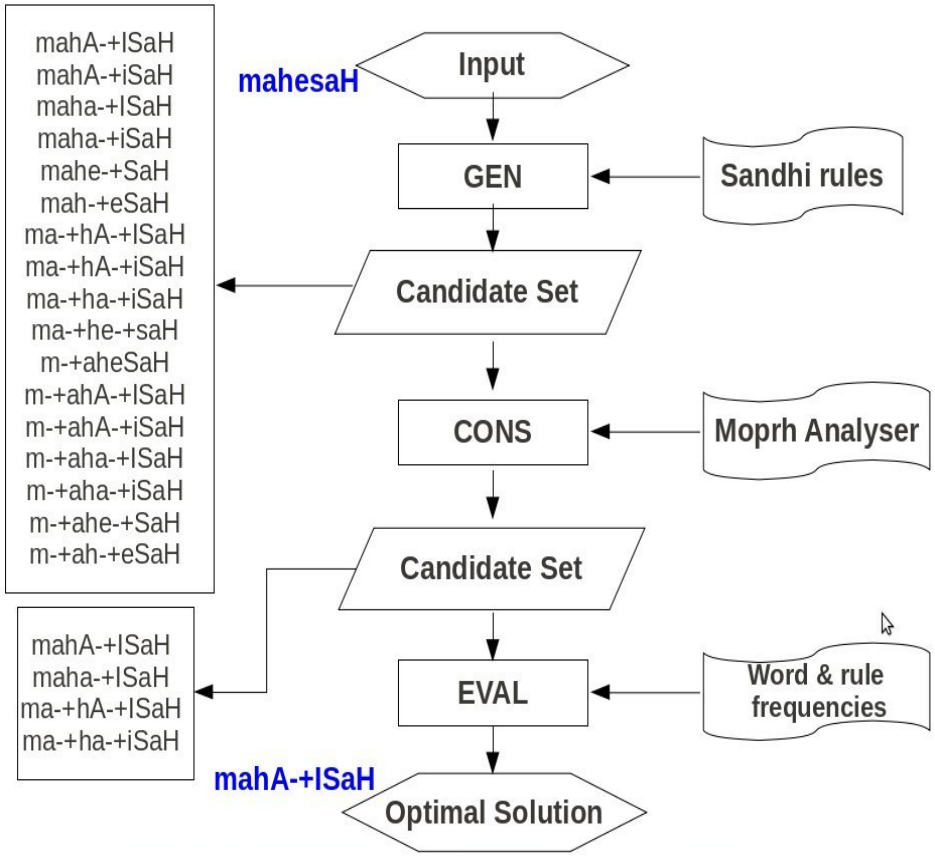
Flow-chart represetation of Compound Segmentation



The basic outline of the algorithm is:

1. Recursively break a word at every possible position applying a sandhi rule and generate all possible candidates for the input. (17 segments)
2. Pass the constituents of all the candidates through the morph analyser.

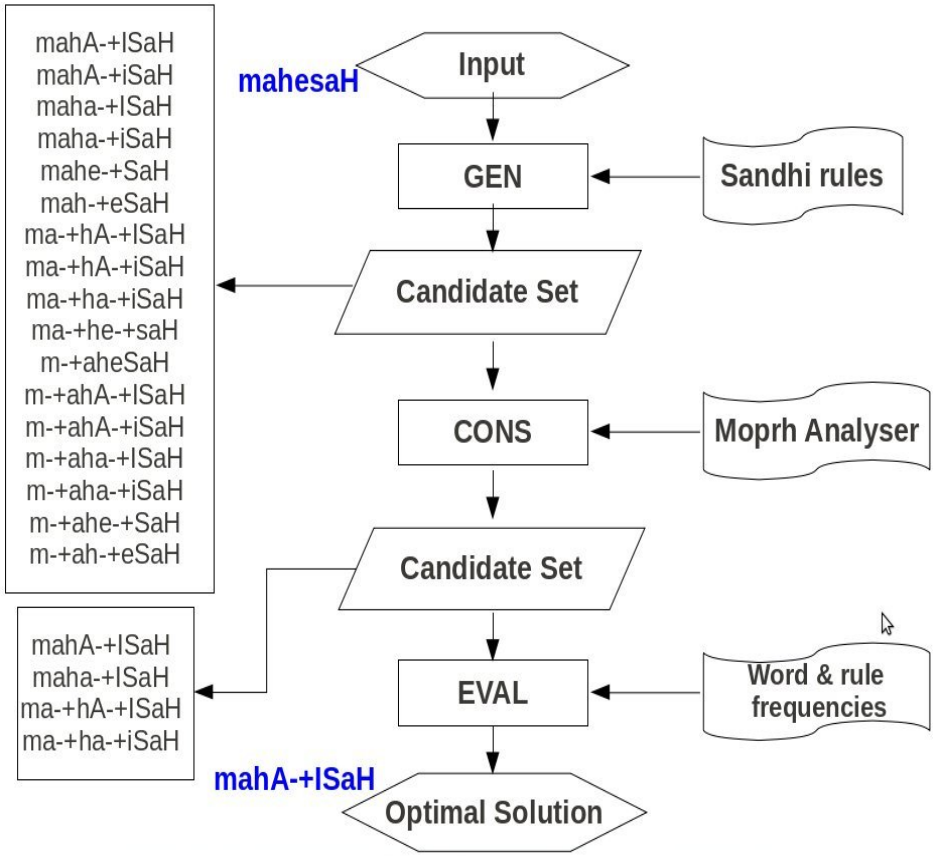
Flow-chart represetation of Compound Segmentation



The basic outline of the algorithm is:

1. Recursively break a word at every possible position applying a sandhi rule and generate all possible candidates for the input. (17 segments)
2. Pass the constituents of all the candidates through the morph analyser.
3. Declare the candidate as a valid candidate, if all its constituents are recognised by the morphological analyser.(4 solutions)

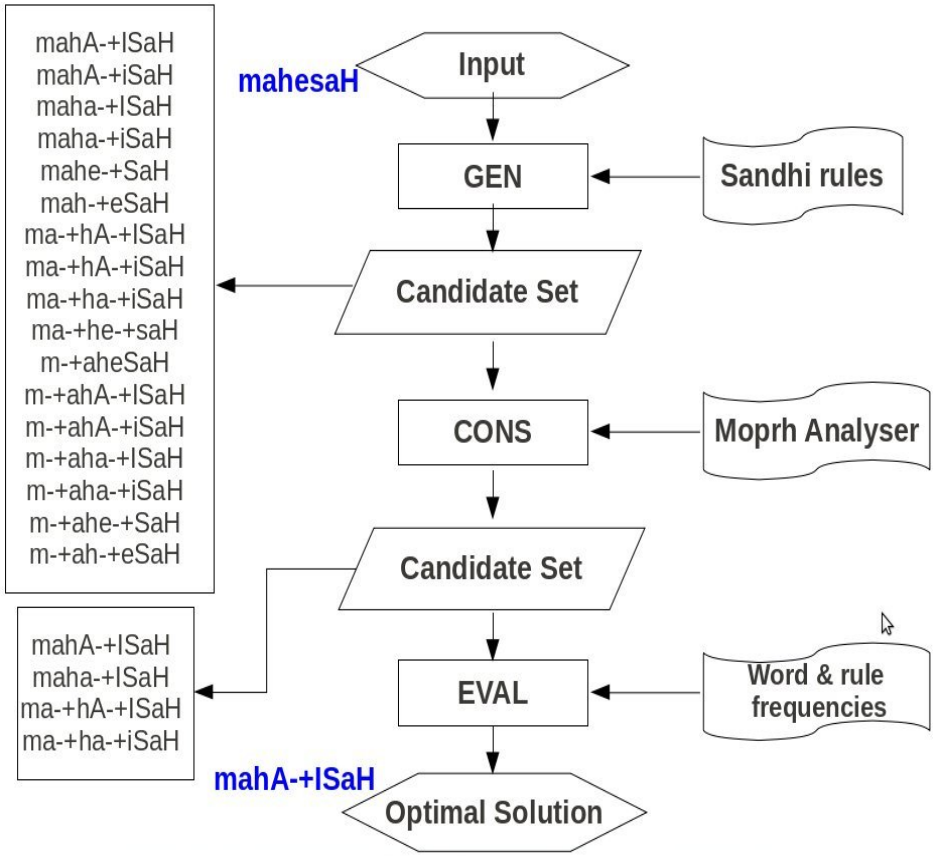
Flow-chart represetation of Compound Segmentation



The basic outline of the algorithm is:

1. Recursively break a word at every possible position applying a sandhi rule and generate all possible candidates for the input. (17 segments)
2. Pass the constituents of all the candidates through the morph analyser.
3. Declare the candidate as a valid candidate, if all its constituents are recognised by the morphological analyser.(4 solutions)
4. Assign weights to the accepted candidates and sort them based on the weights.

Flow-chart represetation of Compound Segmentation



The basic outline of the algorithm is:

1. Recursively break a word at every possible position applying a sandhi rule and generate all possible candidates for the input. (17 segments)
2. Pass the constituents of all the candidates through the morph analyser.
3. Declare the candidate as a valid candidate, if all its constituents are recognised by the morphological analyser.(4 solutions)
4. Assign weights to the accepted candidates and sort them based on the weights.
5. The optimal solution will be the one with the highest weight.

Compound Segmenter

- The current morphological analyser can recognise around 140 million words.

Compound Segmenter

- The current morphological analyser can recognise around 140 million words.
- Corpus = Children stories, dramas, purāṇas, Āyurveda texts.
- 100K words of a parallel corpus of sandhied and unsandhied text had 25K parallel instances of sandhied and unsandhied text were extracted.

Compound Segmenter

- The current morphological analyser can recognise around 140 million words.
- Corpus = Children stories, dramas, purāṇas, Āyurveda texts.
- 100K words of a parallel corpus of sandhied and unsandhied text had 25K parallel instances of sandhied and unsandhied text were extracted.
    - Almost 92.5% of the times, the first segmentation is correct. And in almost 99.1% of the cases, the correct split was among the top 3 possible splits.

Compound Segmenter

- The current morphological analyser can recognise around 140 million words.
- Corpus = Children stories, dramas, purāṇas, Āyurveda texts.
- 100K words of a parallel corpus of sandhied and unsandhied text had 25K parallel instances of sandhied and unsandhied text were extracted.
  - Almost 92.5% of the times, the first segmentation is correct. And in almost 99.1% of the cases, the correct split was among the top 3 possible splits.
  - The precision was about 92.46% (measured in terms of the number of words for which the first answer is correct w.r.t. the total words for which correct segmentation was obtained).

Compound Segmenter

- The current morphological analyser can recognise around 140 million words.
- Corpus = Children stories, dramas, purāṇas, Āyurveda texts.
- 100K words of a parallel corpus of sandhied and unsandhied text had 25K parallel instances of sandhied and unsandhied text were extracted.
  - Almost 92.5% of the times, the first segmentation is correct. And in almost 99.1% of the cases, the correct split was among the top 3 possible splits.
  - The precision was about 92.46% (measured in terms of the number of words for which the first answer is correct w.r.t. the total words for which correct segmentation was obtained).

Rank wise distribution

Rank	% of words
1	92.4635
2	5.0492
3	1.6235
4	0.2979
5	0.1936
>5	0.3723

Compound Processor

- Generates unnecessary splits
- Almost 90% of them are discarded at constraint stage

Compound processor

1. Segmentation (Samāsapadacchedaḥ)
2. **Constituency Parsing (Sāmarthyaniradhāraṇam)**
3. Type Identification (Samāsabhedanirdhāraṇam)
4. Paraphrase generation (Vigrahavākyanirmāṇam)

Constituency Parser

- Constituency parser takes an output of the segmenter and produces a binary tree showing the syntactic composition of the compound corresponding to each of the possible segmentations.

Constituency Parser

- Constituency parser takes an output of the segmenter and produces a binary tree showing the syntactic composition of the compound corresponding to each of the possible segmentations.
- Each of these compositions show the possible ways various segments can be grouped.

Constituency Parser

- Constituency parser takes an output of the segmenter and produces a binary tree showing the syntactic composition of the compound corresponding to each of the possible segmentations.
- Each of these compositions show the possible ways various segments can be grouped.
- Let a-b-c be the segmentation of a compound. Since a compound is binary, the three components a-b-c may be parsed in two ways as -

Constituency Parser

- Constituency parser takes an output of the segmenter and produces a binary tree showing the syntactic composition of the compound corresponding to each of the possible segmentations.
- Each of these compositions show the possible ways various segments can be grouped.
- Let a-b-c be the segmentation of a compound. Since a compound is binary, the three components a-b-c may be parsed in two ways as -

First way to parse

<a– <b-c>>

Second way to parse

<<a-b> –c>

Constituency Parser

Only one of the ways of grouping may be correct in a given context as -

Constituency Parser

Only one of the ways of grouping may be correct in a given context as -

Example

eka - priya - darśanaḥ
one - dear - appearance
(**Gloss:** one who is dear to all.)

Constituency Parser

Only one of the ways of grouping may be correct in a given context as -

Example

eka - priya - darśanaḥ
one - dear - appearance
(**Gloss:** one who is dear to all.)

First way to parse

< eka - < priya - darśanaḥ >>

Constituency Parser

Only one of the ways of grouping may be correct in a given context as -

Example

eka - priya - darśanaḥ
one - dear - appearance
(**Gloss:** one who is dear to all.)

First way to parse

< eka - < priya - darśanaḥ >>

Second way to parse

<< eka - priya>- darśanaḥ >

Constituency Parser

Only one of the ways of grouping may be correct in a given context as -

Example

eka - priya - darśanaḥ
one - dear - appearance
(**Gloss:** one who is dear to all.)

First way to parse

< eka - < priya - darśanaḥ >>

Second way to parse

<< eka - priya>- darśanaḥ >

First one is correct.

Constituency Parser

Only one of the ways of grouping may be correct in a given context as -

Example

eka - priya - darśanaḥ
one - dear - appearance
(**Gloss:** one who is dear to all.)

Example

tapas-svādhyāya-niratam
penance - self study - constantly engaged
(**Gloss:** constantly engaged in penance and self-study.)

First way to parse

< eka - < priya - darśanaḥ >>

Second way to parse

<< eka - priya>- darśanaḥ >

First one is correct.

Constituency Parser

Only one of the ways of grouping may be correct in a given context as -

Example

eka - priya - darśanaḥ
one - dear - appearance
(**Gloss:** one who is dear to all.)

First way to parse

< eka - < priya - darśanaḥ >>

Second way to parse

<< eka - priya>- darśanaḥ >

First one is correct.

Example

tapas-svādhyāya-niratam
penance - self study - constantly engaged
(**Gloss:** constantly engaged in penance and self-study.)

First way to parse

< tapas-<svādhyāya- niratam >>

Constituency Parser

Only one of the ways of grouping may be correct in a given context as -

Example

eka - priya - darśanaḥ
one - dear - appearance
(**Gloss:** one who is dear to all.)

First way to parse

< eka - < priya - darśanaḥ >>

Second way to parse

<< eka - priya>- darśanaḥ >

First one is correct.

Example

tapas-svādhyāya-niratam
penance - self study - constantly engaged
(**Gloss:** constantly engaged in penance and self-study.)

First way to parse

< tapas-<svādhyāya- niratam >>

Second way to parse

<< tapas-svādhyāya >- niratam >

Constituency Parser

Only one of the ways of grouping may be correct in a given context as -

Example

eka - priya - darśanaḥ
one - dear - appearance
(**Gloss:** one who is dear to all.)

First way to parse

< eka - < priya - darśanaḥ >>

Second way to parse

<< eka - priya>- darśanaḥ >

First one is correct.

Example

tapas-svādhyāya-niratam
penance - self study - constantly engaged
(**Gloss:** constantly engaged in penance and self-study.)

First way to parse

< tapas-<svādhyāya- niratam >>

Second way to parse

<< tapas-svādhyāya >- niratam >

Second one is correct.

Constituency Parser

With 3 components, only these two parses are possible. But as the number of constituents increase, the number of possible ways the constituents can be grouped grows very fast.

Constituency Parser

With 3 components, only these two parses are possible. But as the number of constituents increase, the number of possible ways the constituents can be grouped grows very fast.

For instance

The compound is

a-b-c-d

Constituency Parser

With 3 components, only these two parses are possible. But as the number of constituents increase, the number of possible ways the constituents can be grouped grows very fast.

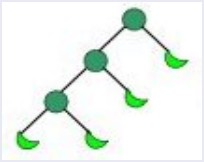
For instance

The compound is

a-b-c-d

First possible way to be grouped

<<<a-b>-c>-d>



Constituency Parser

With 3 components, only these two parses are possible. But as the number of constituents increase, the number of possible ways the constituents can be grouped grows very fast.

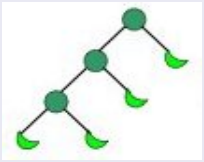
For instance

The compound is

a-b-c-d

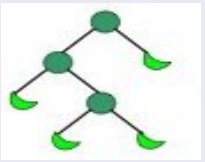
First possible way to be grouped

<<<a-b>-c>-d>



Second possible way to be grouped

<<a-<b-c>>-d>



Constituency Parser

With 3 components, only these two parses are possible. But as the number of constituents increase, the number of possible ways the constituents can be grouped grows very fast.

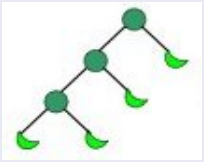
For instance

The compound is

a-b-c-d

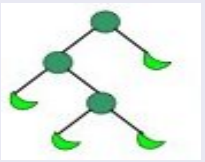
First possible way to be grouped

<<<a-b>-c>-d>



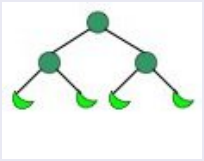
Second possible way to be grouped

<<a-<b-c>>-d>



Third possible way to be grouped

<<a-b>-<c-d>>



Constituency Parser

With 3 components, only these two parses are possible. But as the number of constituents increase, the number of possible ways the constituents can be grouped grows very fast.

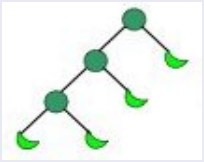
For instance

The compound is

a-b-c-d

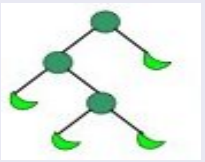
First possible way to be grouped

<<<a-b>-c>-d>



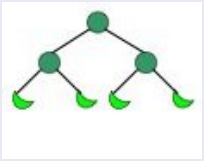
Second possible way to be grouped

<<a-<b-c>>-d>



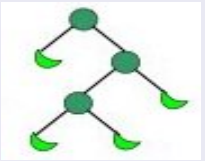
Third possible way to be grouped

<<a-b>-<c-d>>



Fourth possible way to be grouped

<a-<<b-c>-d>>



Constituency Parser

With 3 components, only these two parses are possible. But as the number of constituents increase, the number of possible ways the constituents can be grouped grows very fast.

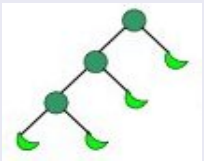
For instance

The compound is

a-b-c-d

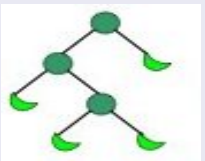
First possible way to be grouped

<<<a-b>-c>-d>



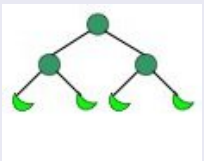
Second possible way to be grouped

<<a-<b-c>>-d>



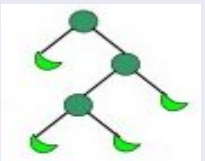
Third possible way to be grouped

<<a-b>-<c-d>>



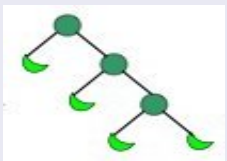
Fourth possible way to be grouped

<a-<<b-c>-d>>



Fifth possible way to be grouped

<a-<b-<c-d>>>



Constituency Parser

- The constituency parsing is similar to the problem of completely parenthesizing *n+1* factors in all possible ways.
- The total possible ways of parsing a compound with $n+1$ constituents is equal to a *Catalan number* C_n, where for $n \geq 0$,

$$C_n = \frac{(2n)!}{(n+1)!n!}$$

(Huet, 2009)

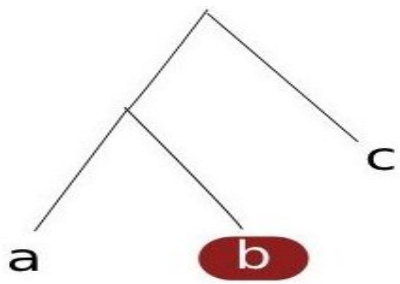
- First few Catalan Numbers: 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796

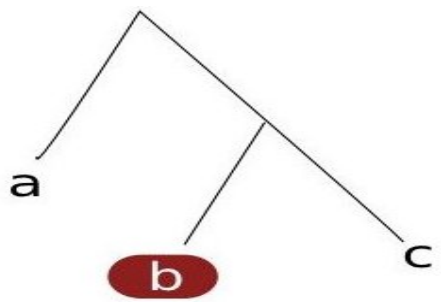
Developing the constituency parser

- The correctness of parse is governed by the semantics.
- Two approaches to handle semantic compatibility:-
  - Use semantically rich lexicon and follow rule based approach.
  - Use manually parsed compounds and statistical methods.

Basic Intuition

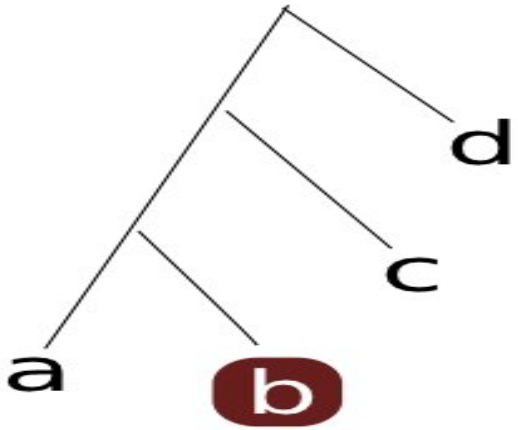
Whether a component is initial(pūrvapada) or final(uttarapada)

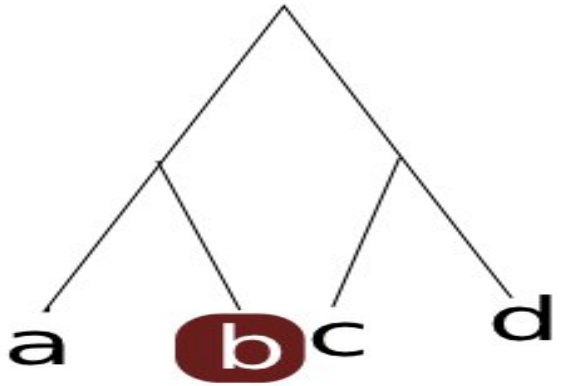




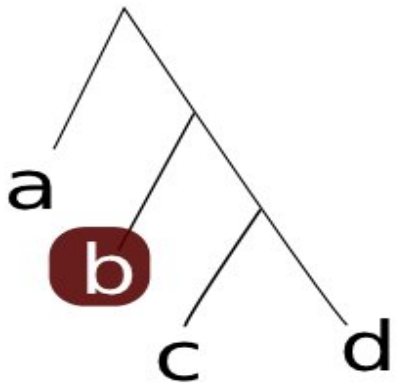
In case of 4 components

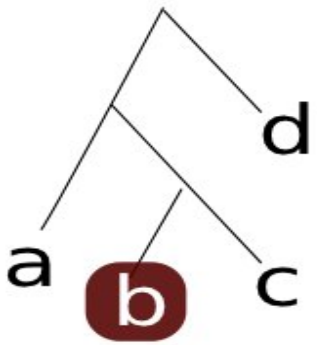
Final components :-

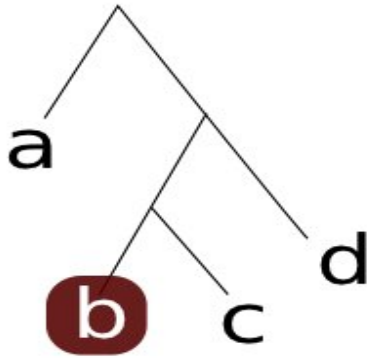




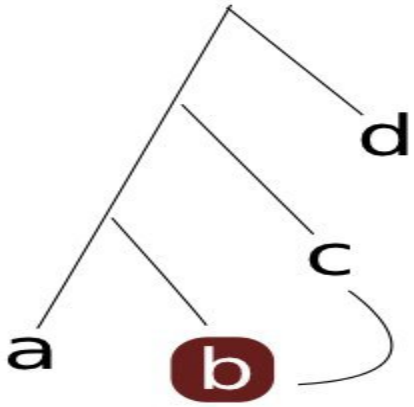
Initial components:-

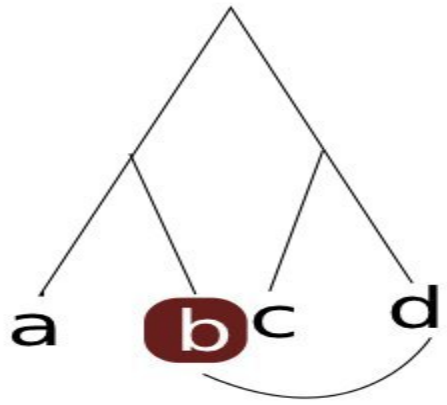


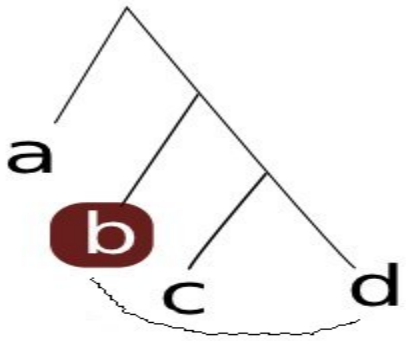


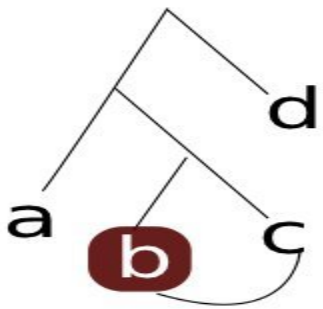


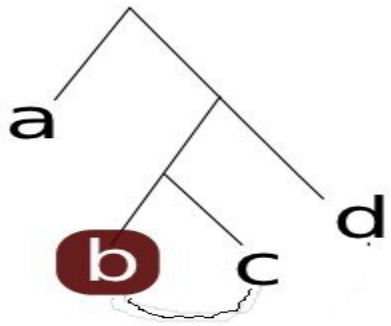
Q. What is the other component?











Statistical details of the corpus

Total size in words	150K
Total compounds	30K
Compounds with 2 components	21,384
Compounds with 3 components	6,809
Compounds with 4 components	1,321
Compounds with 5 components	319
Compounds with more than 5 components	133

Statistical details of corpus

Table: Compounds with 3 components

Pattern no.	Patterns	No. of cases	cases (in %)
1	<a-<b-c>>	466	6.8
2	<<a-b>-c>	6,343	93.2

Table: Compounds with 4 components

Pattern no.	Patterns	No. of cases	cases (in %)
1	<a-<b-<c-d>>>	5	0.3
2	<a-<<b-c>-d>>	33	2.3
3	<<a-<b-c>>-d>	127	9.7
4	<<a-b>-<c-d>>	324	24.7
5	<<<a-b>-c>-d>	832	63.0

- 5-fold testing

Results

The average performance of this experiment:

Table: Performance of compounds with 3 components

Patterns	Precision (in %)	Recall (in %)	F-measure
<a-<b-c>>	57	45	0.503
<<a-b>-c>	95.8	97.27	0.965

The average performance is 93.66% which is very close to the baseline, and at the same time the F-measure for the frequent pattern is also close to 1.

Results

Performance for 4 components is:-

Patterns	Precision (in %)	Recall (in %)	F-measure
<<<a-b>-c>-d>	72	87	0.788
<<a-<b-c>>-d>	80	3	0.57
<<a-b>-<c-d>>	79	40	0.53
<a-<b-<c-d>>>	50	20	0.28
<a-<<b-c>-d>>	-	0	-

The average performance for 4 components is 65.4% which is again just above the base line performance.

Results

Examples with more than 4 component compounds being very small in number, their precision and recall are not measured. The overall performance for 5 test data is given below.

Sr no.	Total compounds	Correct Parse	Wrong Parse	% success
1	1,738	1,493	245	85.9
2	1,734	1,503	231	86.6
3	1,729	1,497	232	86.5
4	1,759	1,532	227	87.0
5	1,737	1,500	237	86.3

The average success rate is 86.5%.

Compound processor

1. Segmentation (Samāsapadacchedaḥ)
2. Constituency Parsing (Sāmarthyaniradhāraṇam)
3. **Type Identification (Samāsabhedanirdhāraṇam)**
4. Paraphrase generation (Vigrahavākyanirmāṇam)

Type-Identification

- The Type Identifier takes an output of the constituency parser and assigns an appropriate operator (tag) to each non-leaf node.
- Tag specifies the relation between the components. This relation is semantic in nature and is not expressed by any morpheme.
- We first apply the relevant rules from Pāṇini and if the rules fail to identify the tag, we use some simple statistical methods.

The aphorisms dealing with compounds can be broadly classified into two types

1. aphorisms that deal with the semantic content to decide the type of a compound.
2. aphorisms that deal with the process of compound formation involving
   - deciding the word order
   - deletion of vibhakti
   - assigning an accent

- The second type of aphorisms are thus useful from generation point of view. They deal with the morpholgy and phonology.
- The first type of aphorisms provide semantic clues for deciding the type of a compound.

Some examples of Implemented aphorisms with conditions and extra semanitcs information

		Condtions			
S.No.	Sūtra Reffr.	Initial word	Final word	Extra-Info	Type
1	2-1-6	All the prefixs or indeclinables like प्र, परा, अप, यथा etc.	neuter gender	-	A1
2	2-1-8	यावत्	Neuter gender	-	A1
3	2-1-9	सु, प्रति	Neuter gender	-	A1
12	2-1-19	The numerals like एक, द्वि	वंश्यवाची words	वंश्यवाची list	A6
13	2-1-20	The numerals like एक, द्वि	नदीवाची words	नदीवाची list	A6
14	2-1-24	-	श्रित, अतीत, पतित, गत, अत्यस्त, प्राप्त, आपन्न	-	T2

List of implemented aphorisms

1. "अव्ययं विभक्ति-समीप-समृद्धि-व्यृद्ध्यर्थाभावात्यया-सम्प्रति-शब्दप्रादुर्भाव.
2. "यावदवधारणे" (2-1-8)
3. "सुप्प्रतिना मात्रार्थे" (2-1-9)
4. "अक्षशलाकासंख्याः परिणा" (2-1-10)
5. "अपपरिबहिरञ्चवः पञ्चम्या" (2-1-12)
6. "आङ् मर्यादाऽभिविध्योः" (2-1-13)
7. "लक्षणेनाऽभिप्रती आभिमुख्ये" (2-1-14)
8. "अनुर्यत्समया" (2-1-15)
9. "यस्य चायामः" (2-1-16)
10. "तिष्ठद्गुप्रभृतीनि च" (2-1-17)
11. "पारे मध्ये षष्ठ्या वा" (2-1-18)
12. "संख्या वंश्येन" (2-1-19)
13. "नदीभिश्च" (2-1-20)
14. "द्वितीया श्रितातीतपतितगतात्यस्तप्राप्तापन्नैः" (2-1-24)
15. "स्वयं क्तेन" (2-1-25)
16. "खट्वा क्षेपे" (2-1-26)

The conditions stated by Pāṇini fall under the following categories.

1. A restricted list of words is provided.
2. A restriction in terms of special inflectional suffix / derivational suffix / category is mentioned.
3. A restriction is stated in terms of special technical terms, which are theory internal.
4. A restriction in terms of semantic relations between the components is mentioned.
5. Semantic property of the component is stated as a condition.

Clues for important types of relations.

- Sanskrit WordNet
- Amarakośa

(i) viśeṣaṇa-viśeṣya-bhāva

(ii) upamāna-upameya-bhāva

(iii) avayaya-avayavī-bhāva

(iv) instrument-action relation

Semantic properties

(a) a number
(b) a river
(c) a family name
(d) a direction
(e) an abusing word
(f) a praising word
(g) a 4-legged animal
(h) a color word
(i) a class (jāti)
(j) an adjective

Sr. No.	Tag	Manually tagged	Tagged by m/c	Precision	Recall	F-measure
1	Tn	3801	3764	99.73	98.76	99.24
2	T4	1069	1033	88.96	85.96	87.43
3	A1	1004	1245	76.86	95.31	85.09
4	Tk	57	55	83.63	80.7	82.13
5	A7	13	19	68.42	100	81.24
6	K1	7894	2754	85.14	29.7	44.03
7	T2	244	188	42.02	28.11	33.68
8	T1	132	67	43.28	21.96	29.13
9	T5	359	105	48.57	14.2	22.97
10	T3	2509	301	21.26	2.55	4.55
11	T7	1328	30	23.33	0.52	1.01

Statistical approaches

1. Manually tagged corpus of approximately 600K words contains 80,155 compounds.

Statistical approaches

1. Manually tagged corpus of approximately 600K words contains 80,155 compounds.
2. These texts were tagged based on 55 compound tags.

Compound Name	**Name of Tags**	**Total**
Avyayībhāvaḥ	A[1-7]	7
Tatpuruṣaḥ	T[1-7] |T[nbgmpk] |Td[stu]|U	17
Karmadhārayaḥ	K[1-7]|Km	8
Bahuvrihiḥ	Bs[2-7] |Bs[dgpsu]|Bsmn |B[bv] |Bv[psSU]	18
Dvandvaḥ	D[is] |[ESd]	5

Statistical approaches

1. Manually tagged corpus of approximately 600K words contains 80,155 compounds.
2. These texts were tagged based on 55 compound tags.

Compound Name	Name of Tags	Total
Avyayībhāvaḥ	A[1-7]	7
Tatpuruṣaḥ	T[1-7] |T[nbgmpk] |Td[stu]|U	17
Karmadhārayaḥ	K[1-7]|Km	8
Bahuvrihiḥ	Bs[2-7] |Bs[dgpsu]|Bsmn |B[bv] |Bv[psSU]	18
Dvandvaḥ	D[is] |[ESd]	5

3. Compounds are thus tagged in context and thus contain only one tag.(Training Corpus)

Statistical approaches

1. Manually tagged corpus of approximately 600K words contains 80,155 compounds.
2. These texts were tagged based on 55 compound tags.

Compound Name	Name of Tags	Total
Avyayībhāvaḥ	A[1-7]	7
Tatpuruṣaḥ	T[1-7] |T[nbgmpk] |Td[stu]|U	17
Karmadhārayaḥ	K[1-7]|Km	8
Bahuvrihiḥ	Bs[2-7] |Bs[dgpsu]|Bsmn |B[bv] |Bv[psSU]	18
Dvandvaḥ	D[is] |[ESd]	5

3. Compounds are thus tagged in context and thus contain only one tag.(Training Corpus)
4. 400 tagged compounds from totally different texts. (Testing)

Some features of the manually tagged data

1. Around 25% of the compounds were repeated.
2. 14,203 types of left word and 24,391 types of right word.

Tag	% of words
T6	29.12
K1	12.05
Bs6	6.97
T3	5.64
Tn	4.50
Di	3.75
U	3.64

Table 2: Distribution of Fine-grain-Tags

Tag	% of words
T	50.51
K	19.01
B	12.53
D	4.70
U	3.66
A	0.94
S	0.78

Table 3: Distribution of Coarse-Tags

Probability

From the manually tagged data, we define the following probabilities

(a) P(T/w_1-w_2) = probability that the compound w_1-w_2 has a tag T.

$$= \frac{\#(w_1 - w_2 \textit{ of type } T)}{\#(w_1 - w_2)}$$

(b) P(T/w_1-) = probability that a compound with w_1 as the initial component has tag T.

$$= \frac{\#\textit{of words with } w_1 \textit{ as initial component with tag } T}{\#\textit{of words with } w_1 \textit{ as an initial component}}$$

(c) P(T/-w_2) = probability that a compound with w_2 as the final component has tag T.

$$= \frac{\#(\textit{of words with } w_2 \textit{ as initial component with tag } T)}{\#\textit{of words with } w_2 \textit{ as an initial component}}$$

Algorithm

Given w_1-w_2

(i) If P(T/w_1-w_2) exists, we choose the max P(T_i/w_1-w_2)

(ii) else if P(T_i/w_1-) and P(T_i/-w_2) exist, we choose maximum P(T_i/w_1-)P(T_i/-w_2)

(iii) else if only P(T_i/w_1-) exists, we choose maximum P(T_i/w_1-)

(iv) if only P(T_i/-w_1) exists, we choose maximum P(T_i/-w_1)

(v) and finally if both w_1- and -w_2 do not occur in the manually tagged corpus, we assign a tag with maximum probability.

Performance Evaluation

1. The test data of 400 words are tagged 'in context'. While our compound tagger does not see the context, and thus suggests more than one possible tag, and ranks them.
2. The tool always produces tags with weights associated with them. Hence, the coverage is 100%. The precision is evaluated based on the ranks of the correct tags. Table-4 shows results of 400 words with a coarse as well as fine grained tagset.

	with 55 tags		with 8 tags	
Rank	no of words	% of words	no of words	% of words
1	250	62.0	307	76.1
2	59	14.6	56	13.8
3	28	6.9	24	5.9

Table 4: Precision of Type Identifier.

3. Thus, if we consider only the first rank, the precision with 8 tags is 76.1% and with 55 tags, it is 62.0%. The precision increases substantially to 95.8% and 83.5% respectively if we take first three ranks into account.

Compound processor

1. Segmentation (Samāsapadacchedaḥ)
2. Constituency Parsing (Sāmarthyaniradhāraṇam)
3. Type Identification (Samāsabhedanirdhāraṇam)
4. **Paraphrase generation (Vigrahavākyanirmāṇam)**

Paraphrase Generation

- A paraphrase generator takes a well formulated tagged compound as an input and produces its paraphrase as an output.
- A semantically analysed compound has the following syntax.

```
compound: '<' component '-' component '>' tag
component: word | compound
tag: A[1-7]
    | Bs[2-7] | Bs[dgpsu] | Bsm[gn] | Bv[sSU] | B[bv]
    | D[is]   | K[1-5]    | Km      | T[1-7]  | T[bgmnpk]
    | Tds      | [ESd] | U[1-5,7]
word: [a-zA-Z]+
```

Sanskrit compounds tagging syntax

- We define a compound formed with the leaf nodes of a binary tree as a simple compound. Compounds with at least one component as a compound (i.e. a non-leaf node) are termed as nested compounds.

Examples

Example-1 Input : <Daśaratha-putraḥ>T6
Output : Daśarathasya putraḥ = Daśarathaputraḥ
The general rule for generating the paraphrase of a T6 type compound is
$<W_1$ - $W_2>$T6 $=>$ $W_1'\{6\}$ - $W_2'\{1\}$ = $(W_1'$ + $W_2')\{1\}$

Example-2

Input: <pīta-ambaraḥ>Bs6
Output: pītam ambaram yasya saḥ = pītāmbaraḥ
The general rule for generating the paraphrase is
$<W_1$ - $W_2>$Bs6 $=>$ $W_1'\{g\}\{1\}$, $W_2'\{g\}\{1\}$ yat {g'}{6} tat{g'}{1}
= $(W_1'$ + $W_2')\{g'\}\{1\}$

Paraphrase generation of simple compounds

The paraphrase generation involves three major steps. In the first step we analyse the components and in the second step we generate the required word forms and construct the paraphrase.

Step-A

Input: $<W_1$ - $W_2>T_n$

a. Analyse W_1
b. Analyse W_2

Step-B Apply the paraphrase rule and generate the paraphrase.

Step-C Finally, if the compound word is not in the prathamā-vibhakti, then appropriate pronominal phrase is also generated as
tat{g}{vibh}{num}
W_2{g}{vibh}{num}
Where g, is the gender, vibh and num are the vibhakti and number of the *ifc* - samāsa-uttarapada.

Problem cases

- Aluk compounds.
- Madhyama-pada-lopi compounds.
- Special cases from Gaṇapāṭa etc.
- Upapada-compounds.
- The requirement of a special morph.
- The requirement of Sandhi module.
- Special treatment of Dvandva compounds.

Evaluation

We tested 200 simple compounds and 100 nested compounds. 89% simple compounds and 80% nested compounds paraphrases were correct.

Conclusions

- Model for modern Indian Languages
- The components of compound together convey a unique meaning which is over and above its components meanings. So had the components of a Sanskrit be written seperately, they are very close to a Multi Word Expression (MWE). So the problem of constituency parsing of Sanskrit compounds then is directly relevent for determining the association of words within a MWE with each other. The insights gained in handling compounds will be directly available for handling MWEs of Indian languages.