Global Journal of Management and Business Studies.
ISSN 2248-9878 Volume 3, Number 10 (2013), pp. 1135-1142

http://www.ripublication.com/gjmbs.htm

Sanskrit as a Programming Language and Natural Language Processing

Shashank Saxena and Raghav Agrawal

C.S C.S, IIET IIET.

Abstract

In this paper represents the work toward developing a dependency parser for Sanskrit language and also represents the efforts in developing a NLU(Natural Language Understanding) and NLP(Natural Language Processing) systems. Here, we use ashtadhayayi (a book of Sanskrit grammar) to implement this idea. We use this concept because the Sanskrit is an unambiguous language. In this paper, we are presenting our work towards building a dependency parser for Sanskrit language that uses deterministic finite automata(DFA) for morphological analysis and 'utsarga apavaada' approach for relation analysis. The importance of astadhayayi is it provide a grammatical framework which is general enough to analyze other language as well therefore it is uses for language analysis.

Keyword: Panani Ashtadhayayi, Vibhakti, Karaka, NLP, Sandhi.

1. Introduction

Parsing is the process of analyzing a string of symbols either in natural language or computer languages according to the rule of formal grammar. Determine the functions of words in the input sentence. Getting an efficient and unambiguous parse of natural languages has been a subject of wide interest in the field of artificial intelligence over past 50 years. Most of the research have been done for English sentences but English has ambiguous grammar so we need a strong and unambiguous grammar which is provided by maharishi Panini in the form of astadhayayi. Briggs(Briggs, 1985) demonstrated in his article the silent feature of Sanskrit language that can make it serve as an artificial language. The computational grammar described here takes the concept of vibhakti and karaka relations from Panini framework and uses them to get an

efficient parse for Sanskrit Text.Vibhakti guides for making sentence in Sanskrit and there are seven kinds of vibhakti. Vibhakti also provides information on respective karaka. These seven vibhkti's are :

- Prathama - Nominative
- Dvitiya - Accusative
- Tritiya - Instrumental
- Chaturthi - Dative
- PA.Nchami - Ablative
- Shhashhthi - Possessive
- saptami - Locative
- Sambodhana - Denominative

Karaka approach helps in generating grammatical relationship of nouns and pronouns to other words in a sentence. The grammar is written in 'utsarga apavaada' approach i.e. rules are arranged in several layers each layer forming the exception of previous one

2. A Standard Method for Analyzing Sanskrit Text

For every word in a given sentence, machine/computer is supposed to identify the word in following structure.
<Word> <base> <form> <relation>

A. Word

Given a sentence, the parser identifies a singular word and processes it using the guidelines laid out in this section. If it is a compound word, then the compound word is breakdown in to two part for e.g. vidhyalaya = vidhya + alaya

B. Base

The base is the original, uninflected form of the word. For Simple words: The computer Activates the DFA on the ISCII code (ISCII,1999) of the Sanskrit text. For compound words: The Computer shows the nesting of internal and external samas using nested parentheses. Undo sandhi changes be-tween the component words.

C. Form

It contains the information about the words like verbs or action to be performed

1. For undeclined words, just write u in this col-umn.
2. For nouns, write first m, f or n to indicate the gender, followed by a number for the case (1 through 7, or 8 for vocative), and s, d or p to indicate singular, dual or plural.
3. For adjectives and pronouns, write first a, followed by the indications, as for nouns, of gender (skipping this for pronouns unmarked for gender), case and number

4. For verbs, in one column indicate the class ( ) and voice.

D. Relation
As we read from the above, this attribute gives the relationship between the different words coming in a sentence.

3. Rulebase for Sanskrit

3.1 Samjna Sutra

It assigns attributes to the input string thereby creating an environment for certain sutras to get triggered

3.2 Adhikara Sutras

It assign necessary condition to the sutras for getting triggered (χ)

3.3 Paribhasha Sutras

It takes decision and help us in resolving a conflicts and deadlock conditions . It also provides a meta language for interpreting other sutras

The input for our system is the *karaka* level analysis of the nominal stem and the output is the final form after traversing through the whole *Astadhyayi*

4. Algorithm for Sanskrit Parser

The parser takes as input a Sanskrit sentence and using the Sanskrit Rule base from the DFA Analyzer, analyzes each word of the sentence and returns the base form of each word along with their attributes. This information is analyzed to get relations among the words in the sentence using If Then rules and then output a complete dependency parse. The parser incorporates Panini framework of dependency structure. Due to rich case endings of Sanskrit words, we are using morphological analyzer. To demonstrate the Morphological Analyzer that we have designed for subsequent Sanskrit sentence parsing, the following resources are built:

1) Nominal rule database (contains entries for nouns and pronouns declensions)
2) Verb rule database (contains entries for 10 classes of verbs)
3) Particle database (contains word entries)

Now using these resources, the morphological analyzer, which parses the complete sentences of the text is designed.

4. 1 Morphological Analysis

In this step, the Sanskrit sentence is taken as in put in Devanagari format and converted into ISCII format. Each word is then analyzed using the DFA. Following along any path from start to final of this DFA tree returns us the root word of the word that we

wish to analyze, along with its attributes. While evaluating the Sanskrit words in the sentence, we have followed these steps for computation:

1) First, a left-right parsing to separate out the words in the sentence is done.
2) Second, each word is checked against the Sanskrit rules base represented by the DFA trees in the following precedence order: Each word is checked first against the avavya database, next in pronoun, then verb and lastly in the noun tree.

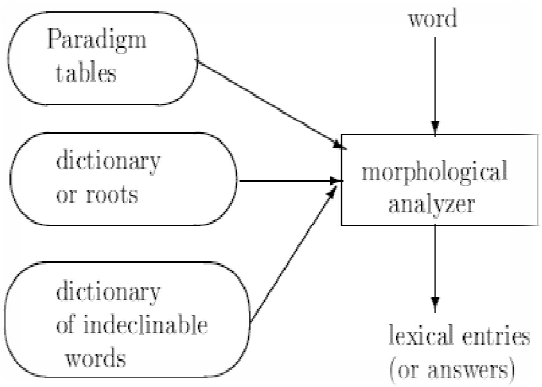

Figure 1: Morphological analyzer input-output.

A.1 Forming Paradigm Table
Algorithm: Forming paradigm table.
Purpose: To form paradigm table from word forms table for a root.
Input: Root r, Word forms table WFT (with Labels for rows and columns)
Output: Paradigm table PT.
Algorithm:
(1) Create an empty table PT of the same Dimensionally ,size and labels as the Word forms table WFT.
(2) For every entry w in WFT ,do If w=r
Then store "(0, Φ)" in the corresponding position in PT.
else begin let i be the position of the first characters in w and r which are different store(size(r)-i+1,suffix(i,w)) at the corresponding position in PT.
(3) Return PT.
End algorithm

A.2 Generating a word form
Algorithm: Generating a word form
Purpose: To generate a word form given a root and desired feature values.
Input: Root r, feature values FV

Uses: paradigm tables, dictionary of roots DR,
Dictionary of indeclinable words DI

Output: word w
Algorithm:

1) If root r belongs to DI
   Then return (word stored in DI for r irrespective of FV)
2) Let p=paradigm type of r as obtained from DR
3) Let PT=paradigm table for p
4) Let (n, s)=entry in PT for feature values FV
5) W := r minus n characters at the end
6) W:= w plus suffix s

End algorithm

Root	*Type*	*Gender*
laDakaa	(n,laDakaa)	m
kapaDaa	(n,laDakaa)	m
bhaaSaa	(n,bhaaSaa)	f
roTii	(n,laDakii)	f
laDakii	(n,laDakii)	f

Figure: Dictionary of roots.

A.3 Morphological analysis using
Paradigms

Word forms table for bhaaSaa

	Case	
Number	*direct*	*oblique*
Singular	bhaaSaa	bhaaSaa
Plural	bhaaSaaeM	bhaaSaaoM

Paradigm table for bhaaSaa

	Case	
Number	*direct*	*oblique*
Singular	(0, ϕ)	(0, ϕ)
Plural	(0, eM)	(0, oM)

Algorithm: morphological analysis using
Paradigm tables.

Purpose: To identify root and grammatical
Features of a given word.
Input: A word w
Output: A set of lexical entries L(where each
lexical entry stands for a root and its
grammatical features)
Uses: Paradigm tables, Dictionary of roots DR
, Dictionary of indeclinable words DI.
Algorithm:

1) L := empty set
2) If w is in DI with entry b then add b to L.
3) For i := 0 to length of w do

Let s= suffix of length I in w
for each paradigm table p
for each entry b(consisting of a pair)
in p do
if s = suffix in entry b then
begin

- r = suffix in entry b then
- j = number of characters to be deleted shown in b
- proposed-root = (w-suffix s) + suffix of r consisting of j characters
- if (proposed-root is in DR) and (the root has paradigm p) then construct a lexical a entry 1 by combining (a) feature given in DR with the proposed-root ,and (b) feature associated with e.
- Add 1 to the set L. End of begin

End for every entry in p
End for every paradigm
End for every i

If L is empty
then return "unknown word w" else return (L)

End algorithm

B. Relation Analysis

Processing of natural language for extraction of the meaning is a challenge in the field of artificial intelligence. Research work in this area is being carried out in most of the Indian and foreign languages by analyzing the grammatical aspect of these languages. Sanskrit, a language that possesses a definite rule-based structure given by Panini, has a great potential in the field of semantic extraction. Hence, Sanskrit and computational linguistic are strongly associated. As given in the grammar of Sanskrit language, its case endings are strong identifiers of the respective word in the sentence. To extract the semantic from the language, dependencies amongst the words of a sentence are developed, and the semantic role of words is identified (e.g., agent, object, etc.). In this

work, an algorithm has been developed for creating a dependency-based structure for the sentence in Sanskrit by analyzing the features given by the Part of Speech (POS) tags. Dependency Tags (DTs) are used to relate the verb with other words in a sentence. POS tags give the syntactic information and DT gives the semantic information. Mapping between the two is established in the proposed algorithm and its analysis is done. Sanskrit, being an order-free language, imposes a great challenge for the development of dependency-based structure for the sentence

With the root words and the information for each

word derived for the previous step, we shall

now compute the relations among the words in the sentence in order to generate the parse tree. Using these relation values we can determine the structure of each of the sentences and thus derive the

semantic analyzer. The Sanskrit language has dependency grammar. Hence the karaka based approach is used to obtain a dependency parse tree. Karaka approach helps in generating grammatical relationship of nouns and pronouns to other words in a sentence. There are reasons for going for dependency parse.

1. Sanskrit is a highly word-order free language It means that you can take a Sanskrit sentence, jumble its words the way you wish and there is good probability that the resulting sentence would still mean the same as the original one.
2. Once the karaka relations are obtained, it is very easy to get the actual relation of the words in the sentence.

5. Conclusion

The significant aspect of our approach is that we do not try to get the full semantics immediately, rather it is extracted in stages depending on when it is most appropriate to do so. The results we have got are quite encouraging and we hope to analyze any Sanskrit text unambiguously. we have successfully demonstrated the parsing of a Sanskrit Corpus employing techniques designed and developed in our previous section. Our analysis of the Sanskrit sentences in the form of morphological analysis and relation analysis is based on sentences as shown in the paragraphs in previous section. We can use the Fuzzy logic and Fuzzy reasoning to deal with the uncertainty information in Panini's Sanskrit Grammar to make it convenient for further computer processing. We also assert that Vaakkriti would be a preliminary contribution to the realm of NLP. Adding to the major work that have been done already. Vaakkriti is an attempt to enhance the existing work. We would extend the current system and develop a fully-fledged parser. Through this we can make a successful parser and semantic analyser and it will also help in generating an NLP.

References

[1] Natural language processing: A paninian perspective By Akshar Bharati, Vineet Chaitanya , Rajeev Sangal

[2] Hopcroft, John E., Motwani, Rajeev, Ullman, Jeffrey D. 2002. Introduction to Automata Theory, Languages and Computation. 2nd Ed, Pearson Education Pvt. Ltd., 2002.

[3] Briggs, Rick. 1985. Knowledge Representation in Sanskrit and artificial Intelligence, pp 33-39. The AI Magazine.

[4] Analysis of Sanskrit text: parsing and semantic relations by Pawan Goyal, vipul Arora, Laxmidhar Behera.

[5] Sanskrit pathan paathan ki anubhut saraltam vidhi Part 1 and Part 2 by Shri Pt. Bhramdutt Jigyasu , Yudhistir Mimansak.

[6] vyakaran parimal by Dr. Ravindra Dergan and Shri Kamlesh Dergan, sultan Chandra and sons (pvt).ltd.

[7] Panini grammar in computer science by Parul Saxena, Kuldeep Pandey and Vinay Saxena

[8] Introduction to Panini Grammar

[9] A case for Sanskrit as a computer programming language by P. Ramanujum

[10] Panini grammar and computer science serve by Saroj Bhatt and Subhash Kak annals of the Bhandarkar Orientel research institute vol. 72, 1993, PP.79-94

[11] Computer simulation of astadhyayi : by Pawan goyal , Himanshu Singh , Amba Kulkerni and Lakshmidhar Behera

[12] Principles of compiler design by Alfread v. Aho, Jeffrey D. Ullman

[13] http://uttishthabharata.wordpress.com/2011/04/20/sanskrit/

[14] http://pustak.org/bs/home.php?bookid=4883&act=continue&index=10&booktype=free#10.

[15] http://www.facweb.iitkgp.ernet.in/~sudeshna/courses/nlp07/

[16] Artificial Intelligence , Elain Rich and Kevin Knight,' 2nd Edition , Tata McGrawHill , 1991

[17] Cognitive science learning resource http://www.comp.leeds.ac.uk/ugadmit/cogsci/knowled

[18] Briggs, Ricks. 1985. Knowledge Representation in Sanskrit and Artificial Intelligence, pp 33-39. The AI Magzine.

[19] A Higher Sanskrit Grammar. 4th Ed, Motilal Banarasidass Publishers Pvt. Ltd.

[20] Huet. 2006. Shallow syntax analysis in Sanskrit guided by semantic nets constraints. International Workshop on Research Issues in Digital Libraries, Kolkata

[21] Bharti, Akshar, Vineet, Chaitanya and Rajeev Sangal, Tree adjoining grammar and paninian grammar, Technical report TRCS-94-219,Dept of CSE, IIT Kanpur, March 1994b

[22] Sharma Ram Nath, 1987, The Astadhayayi of panani, Volume I, Munshiram Manoharlal Publishers Pvt. Ltd, New Delhi

[23] Sharma Ram Nath, 1990, The Astadhayayi of panani, Volume II, Munshiram Manoharlal Publishers Pvt. Ltd, New Delhi