

A Nepali Computational Grammar Analyzer

Bal Krishna Bal

Madan Puraskar Pustakalaya
Lalitpur, PatanDhoka, Nepal
bal@mpp.org.np

Prajwal Rupakheti

Madan Puraskar Pustakalaya
Lalitpur, PatanDhoka, Nepal
prajwalrupakheti@gmail.com

Abstract

The document reports on the Nepali Computational Grammar Analyzer, a Natural Language Processing(NLP) tool, giving an overview on it's system architecture and work flow, followed by a detailed description accompanied by illustrative screenshots on how to use the application. The document also briefly discusses on the formalism followed behind the implementation and reports current performance results.

1 Introduction

The Nepali Computational Grammar Analyzer(NCGA) is a tool that analyzes the correctness or rather well-formedness of a given Nepali input sentence. Analyzing the sentence of a language involves a careful examination of the sentence both from a syntactic and semantic perspectives of language analysis. The NCGA tool which is currently in a research prototypical stage, however, employs just the syntactic criteria. The tool in itself is an integrated form of the intermediate Natural Language Processing (NLP) tools like the Word Aligner, PostPosition Tokenizer, Parts of Speech tagger, Chunker and finally the Parser. Nepali, being a free word order language, we have implemented the parser using the Dependency Grammar Formalism[4] and in particular the Paninian Dependency Grammar framework[1,2,3,5]. Among the intermediate NLP tools, we have implemented all of them on our own except the POS Tagger, for which we have statistically trained the TnT tagger(<http://www.coli.uni-saarland.de/> for Nepali with a manually annotated tagged training data of about 80,000 words. The Chunker is a rule-based one which is based on some linguistic rules developed as a result of the manual analysis

of a text corpus(1 M words) in Nepali. The overall integrated system, i.e. the NCGA follows a pipe line architecture, whereby the output of one of the modules serves as an input to the following module. We present in the figure below(Figure 1), the high level system architecture of the NCGA.

2 System Architecture of the NCGA, An Overview

The NCGA basically consists of three modules, namely, the Parts of Speech(POS) tagger, Chunker and the Parser. However, some smaller sub-modules like the Word Aligner and the Postposition tokenizer also have some role prior to the POS tagging process. About the work flow, a given input Nepali sentence is first fed to the Word Aligner module that aligns the words from the sentence into one word per line format. Next, the aligned words are subjected to postposition tokenization, i.e., breaking the word into root words and suffixes. In Nepali, postpositions usually come together with mostly nouns and sometimes with other word categories in the form of suffixes.

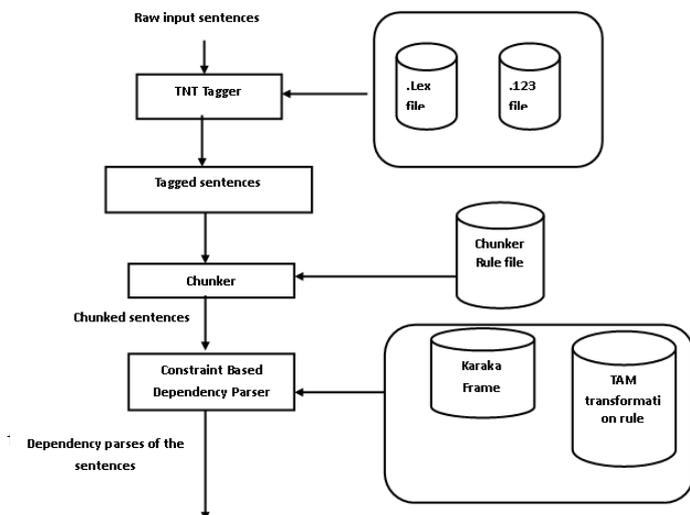


Figure 1: System Architecture of the NCGA

Once, words are broken down respectively into the root words and suffixes, they are then passed to the POS tagger. The tagged words returned by the tagger are then fed to the chunker, which groups the tagged words into word groups or chunks on the basis of the chunker rules contained in the chunker rule database. The chunk rules as mentioned above have been developed as a result of the manual analysis of a Nepali text corpus. In our case, these chunks relay modifier-modified relationships. Finally, the chunked text is passed to the Parser module, which is a very crucial module in the overall system. Since, we have followed the Paninian Dependency Grammar framework, to suit with the free word order nature of Nepali, we make abundant use of the knowledge on the Karaka and the Transformation rules as well as the vibaktis.¹ The information is managed in easy to process XML file formats. Depending upon the knowledge available in the Karaka and Transformation files, the information is further processed by the sub-modules, namely, the Equation generator and the Equation solver. We have applied the constraint based integer programming[3] for these sub-modules. The final result is/are parsed result(s) of the given input sentence.

3 Nepali Language and Grammar: Dependency Grammar Formalism

As already mentioned in the previous section, the Nepali Language follows a free word order. This means that the order of chunks within a sentence can be in arbitrary positions and hence it would be really tedious to develop phrase structure grammar rules as in the case of fixed word order languages like English do not necessarily work for Nepali. The Dependency Grammar Formalism, on the other hand, which we have adopted for Nepali, makes use of the Karaka relations. For Nepali, we have identified altogether six different karaka relations, namely, Karta-K1, Karma-K2, Karan-K3, Sampradaan-K4, Apadaan-K5, Others-KX. The assumption is that if we can establish valid karaka relations between the chunks of the sentence and the verb, then the given input sentence is valid. For example, in the Nepali sentence Ram le bhaat khaayo (meaning Ram

¹In essence, karakas simply represent relationships between verbs and other constituents of a sentence whereas vibaktis functionally map with postpositions.

ate rice), there is a K1 relation between the verb khaayo and the noun chunk Ram le, and similarly K2 relation between the verb khaayo and the noun chunk bhaat.

4 Installing and Using the NCGA: How to?

This section gives a brief walkthrough on installing and using the NCGA.

4.1 Operating System and Dependencies

The NCGA has command line interface and is executed in Linux shell environment. At the moment NCGA is Linux specific and can run only on the Linux machine. The Linux machine needs to have set jdk 1.6 or above with its environmental variable set up.

4.2 Installing the NCGA

The installation process is pretty simple as copy and paste. We only need to copy our NCGA folder anywhere in our Linux machine.

4.3 Configuring the NCGA

In order to configure NCGA we need to have a brief understanding of NGC folder hierarchy. The NCGA contains various folder and files as shown in the snapshot below (Figure 2).

The bin directory contains all the binary and executable files necessary for running NCGA. While the log directory contains the entire log generated by NCGA in due course of the analysis, the res directory consists of all the resources and data required by NCGA to operate. The run.sh is the script to run NCGA. Final output file, which is a completely parsed analysis of a given input sentence is stored in the parser.log file within the

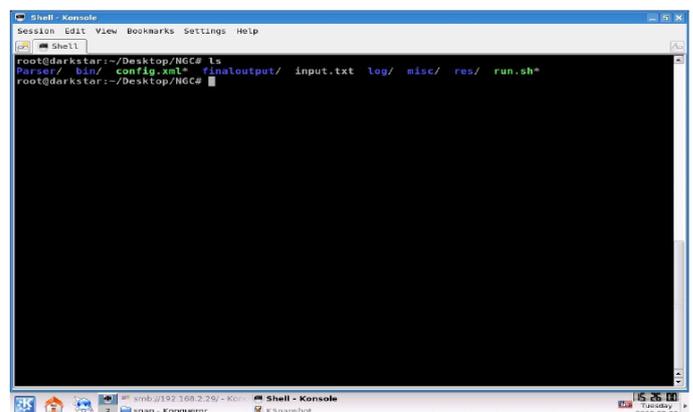


Figure 2: The NGC folder and its subdirectories

log sub directory. The config.xml is the configuration file to describe the location of various data file. In order to configure NCGA, all we need to do is to provide the right data in different tags of config.xml. Following is the view of the config.xml. (Figure 3)

For ease of readability, we present the contents of the config.xml file below:

```
nepali@nepalinux:NGC$ cat config.xml
<?xml version="1.0" encoding="UTF-8" ?>
<ConfigParser>
<seedFile>../res/Parser/seedLex.txt
</seedFile>
<postPosFile> ../res/Parser/postSuffix.txt
</postPosFile>
<primaryRootFrame>
../res/Parser/rootFrame.xml</primaryRootFrame>
<generalFrameA> ../res/Parser/typeA.xml
</generalFrameA>
<generalFrameB> ../res/Parser/typeB.xml
</generalFrameB>
<transFormer> </transFormer>
<outputFileTrans> ../res/output1.txt
</outputFileTrans>
<ruleFile1> </ruleFile1>
<ruleFile9> </ruleFile9>
<inputFile> ../Parser/res/input.txt </inputFile>
<outputFile> ../log/parser.log</outputFile>
<eqlog> ../log/eqsolver.log</eqlog>
<parserlog> ../log/parser.log </parserlog>
</ConfigParser>
```

4.4 Running the NCGA

To run NCGA, we just need to run run.sh file with config and input file as command line argument as follows:

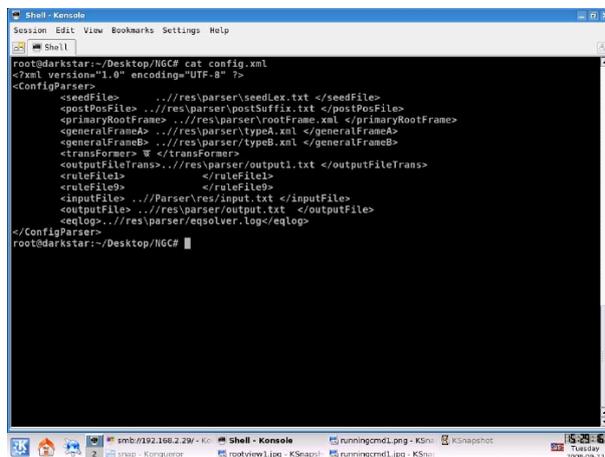


Figure 3: View of the file config.xml

```
nepali@nepalinux:NGC$ ./run.sh input1.txt config.xml
```

On the execution of the above command, the following message is displayed on the screen.

```
Parsing started...
Word alignment accomplished.
Word breaking and tokenization accomplished.
POS tagging accomplished.
Sentence chunking accomplished.
config.xml
Parsing ended...
```

Note: Look at the parser.log file inside the log directory for a complete parse analysis of the given sentence in the input file.

4.5 Viewing the overall output of the NCGA

In order to view the overall output of the NCGA, we need to change the directory to the log file and open the parser.log file.

4.6 Test cases for the NCGA

As test cases for the NCGA, we have developed five simple Nepali declarative sentences representing one or more of the Karaka relations. These test sentences have been stored one sentence each per file in respectively five input files. Below, we present the test sentences along with their corresponding correct Karaka relations analysis.

Sentence1: Ram le bhaat khaayo
(meaning Ram ate rice).

Analysis of sentence1: K1 relation exists between khaayo and Ram le, K2 relation exists between khaayo and bhaat.

Sentence2: Ram le chakku le aap kaaty
(meaning Ram cut the mango with a knife).

Analysis of sentence2: K1 relation exists between kaaty and Ram le, K2 relation between kaaty and aap, K3 relation exists between kaaty and chakku le.

Sentence3: Ram ghara jaanchha
(meaning Ram goes home).

Analysis of sentence3: K1 relation exists between jaanchha and Ram, K2 relation exists between jaanchha and ghara.

Sentence4: Ram le Shyam laai pustak

diyo (meaning Ram gave Shyam a book).

Analysis of sentence4:K1 relation exists between diyo and Ram le, K2 relation exists between diyo and pustak, K4 relation exists between diyo and Shyam lai.

Sentence5:Ram ghar maa padhchha (meaning Ram studies at home).

Analysis of sentence5: K1 relation exists between padchha and Ram, KX relation between padchha and ghar maa.

Experiments have shown that the NCGA correctly handles each category of the above mentioned simple declarative sentences. The work performance of the NCGA is expected to become higher as the intermediate modules like the POS tagger, the Chunker undergo more enhancements. Besides, the Karaka and the Transformation frames that contain crucial information on the Karaka relations also would need to be improved for better performance of the NCGA.

Acknowledgments

We are grateful to the International Development and Research Center (IDRC) for the support provided to this work under the PAN Localization Project. Our thanks also goes to the host institution, Madan Puraskar Pustakalaya, Nepal for the persistent support and platform provided for the research. We would like to duly thank Mr. Laxmi Prasad Khatiwada for providing the linguistic help for the work. Our sincere thanks, similarly go to our intern students from Kathmandu University, Mr. Niraj Pokharel, Ms. Dipti Sharma and Ms. Srijana Pokharel for their contributions to the work.

References

- A. Bharati, V. Chaitanya, and R. Sangal, Natural Language Processing - A Paninian Perspective. New Delhi: Easter Economy Edition ed. Kantipur: Prentice Hall, 1995.
- A. Bharati and R. Sangal, "Parsing free word order languages in the Paninian framework," in Proceedings of the 31'st Annual Meeting on Association For Computational Linguistics (Columbus, Ohio, June 22, 1993). Annual Meeting of the ACL., Morristown, NJ, 1993, pp. 105-111.
- A. Bharati, R. Sangal, and T. Reddy, "A Constraint Based Parser Using Integer Program-

ming," in Proceedings of the ICON-2002, Mumbai, 2002, pp. 121-127.

J. Nivre. <http://w3.msi.vxu.se>. [Online]. <http://w3.msi.vxu.se/nivre/papers/05133.pdf>

M. Pederson, D. Eades, S. Amin, and L. Prakash, "Relative clauses in Hindi and Arabic: A paninian dependency grammar analysis." in Proceedings of the Twentieth International Conference on Computational Linguistics., Geneva, 2004, pp. 17-24.