DAYANANDA SAGAR COLLEGE OF ARTS SCIENCE AND COMMERCE

Shavige Malleshwara Hills, 1st Stage, Kumaraswamy Layout, Bengaluru, Karnataka.

Department of Computer Application-MCA(BU)

A project report on

TOC-Based NLP System Using RNT and ANT

Submitted To:	**Submitted By:**
Ms. Akshatha	Sudeep S(P03CJ24S126092)
Assistant Professor	Sushma S((P03CJ24S126093)
DSCASC	Naveen Kumar(P03CJ24S126069)
	Pranav Gopal Bhat((P03CJ24S126075)

CONTENTS

CHAPTERS	PARTICULARS	PAGE NO.
1	Problem statement	1
2	Introduction	2
3	TOC and NLP in AI	3
4	RNT and ANT Representations	4
5	Implementation and Example	5-6
6	Conclusion	7

1.PROBLEM STATEMENT

Explain the application of TOC in AI. Elaborate the discussion on RNT and ANT implementation and develop a simplified Natural Language Processing (NLP) system that processes a sentence input, extracts its syntactic structure, and generates two key linguistic representations: Reverse Normal Form (RNT) and Abstract Normal Tree (ANT).

2.INTRODUCTION

This project explores the application of **Theory of Computation (TOC)** in the field of **Artificial Intelligence (AI)**, particularly in **Natural Language Processing (NLP)**. TOC provides the theoretical foundation for understanding how machines can process and interpret human language using formal grammars and automata. NLP enables computers to understand, analyze, and generate human language. By combining TOC with NLP, we can create systems that process sentences and extract their syntactic structure. The project focuses on two key representations: **Reverse Normal Form (RNT)** and **Abstract Normal Tree (ANT)**. RNT gives a reversed syntactic view of a sentence, while ANT visualizes sentence structure like a tree. A simplified C program is used to tokenize a sentence and simulate these forms. This integration demonstrates how theoretical models enhance language understanding in AI systems. Such approaches are useful in real-world applications like chatbots, translators, and virtual assistants.

Features of TOC in AI

1. **Formal Language Foundation**
   TOC provides the foundation for designing formal languages, which are essential in understanding and generating human languages in AI.
2. **Automata Models**
   Automata like Finite State Machines (FSMs) and Pushdown Automata (PDAs) help model various AI behaviors, such as language processing and decision-making.
3. **Grammar-Based Parsing**
   Context-Free Grammars (CFGs), derived from TOC, are used in AI to parse sentences and analyze their syntactic structure in NLP applications.
4. **Complexity Analysis**
   TOC helps AI developers analyze the computational complexity of algorithms, ensuring efficient performance in large-scale applications
5. **Pattern Recognition**
   Regular expressions and automata from TOC are used to detect patterns in text and speech, which is fundamental in NLP and speech recognition.

3.TOC AND NLP IN AI

Theory of Computation (TOC) and **Natural Language Processing (NLP)** are two fundamental pillars that together enable machines to understand, interpret, and generate human language in a structured way. While TOC focuses on the *formal rules and models* of computation, NLP applies those rules to analyze and process natural language data.

TOC provides the **formal grammar structures** such as Regular Grammars and Context-Free Grammars (CFGs), which are essential in parsing and analysing the syntax of language. These grammars help in breaking down a sentence into meaningful parts (subject, verb, object, etc.), making it easier for a computer to understand the structure of a language.

On the other hand, NLP uses these computational models from TOC to extract meaning from text, perform sentiment analysis, translate languages, and even generate human-like responses. NLP systems use **tokenization**, **parsing**, and **part-of-speech tagging**—all of which are built upon TOC concepts like **Finite State Machines (FSMs)** and **Pushdown Automata (PDAs)**.

The collaboration of TOC and NLP enables AI systems like **chatbots, voice assistants, translation tools**, and **grammar checkers** to understand and process human language accurately. By integrating TOC into NLP, we make language processing **more reliable, systematic, and efficient**, providing the mathematical foundation for building intelligent systems.

4. RNT AND ANT REPRESENTATIONS

In the domain of **Natural Language Processing (NLP)**, understanding the structure of a sentence is essential for accurate interpretation and processing. To achieve this, linguistic representations such as **Reverse Normal Form (RNT)** and **Abstract Normal Tree (ANT)** are used. These representations help machines analyze the syntax and relationships within a sentence in a structured and logical manner.

Reverse Normal Form (RNT)

Reverse Normal Form (RNT) is a linear representation of a sentence where the **sequence of words or tokens is reversed** while retaining the original parts of speech or grammatical tags. It is mainly used to analyze sentence structures in reverse to detect patterns, dependencies, or context that may not be visible in the standard order.

Example:

Original Sentence:

John eats an apple

POS Tags:

[('John', NNP), ('eats', VBZ), ('an', DT), ('apple', NN)]

RNT:

[('apple', NN), ('an', DT), ('eats', VBZ), ('John', NNP)]

Abstract Normal Tree (ANT)

Abstract Normal Tree (ANT) is a **tree-like hierarchical structure** that represents the **syntactic organization** of a sentence based on grammar rules. Each node in the tree represents a grammatical component such as Noun Phrase (NP), Verb Phrase (VP), or specific words (like nouns, verbs, etc.).

Example:

For the sentence: John eats an apple

```
S
├── NP: John
└── VP
    ├── V: eats
    └── NP: an apple
```

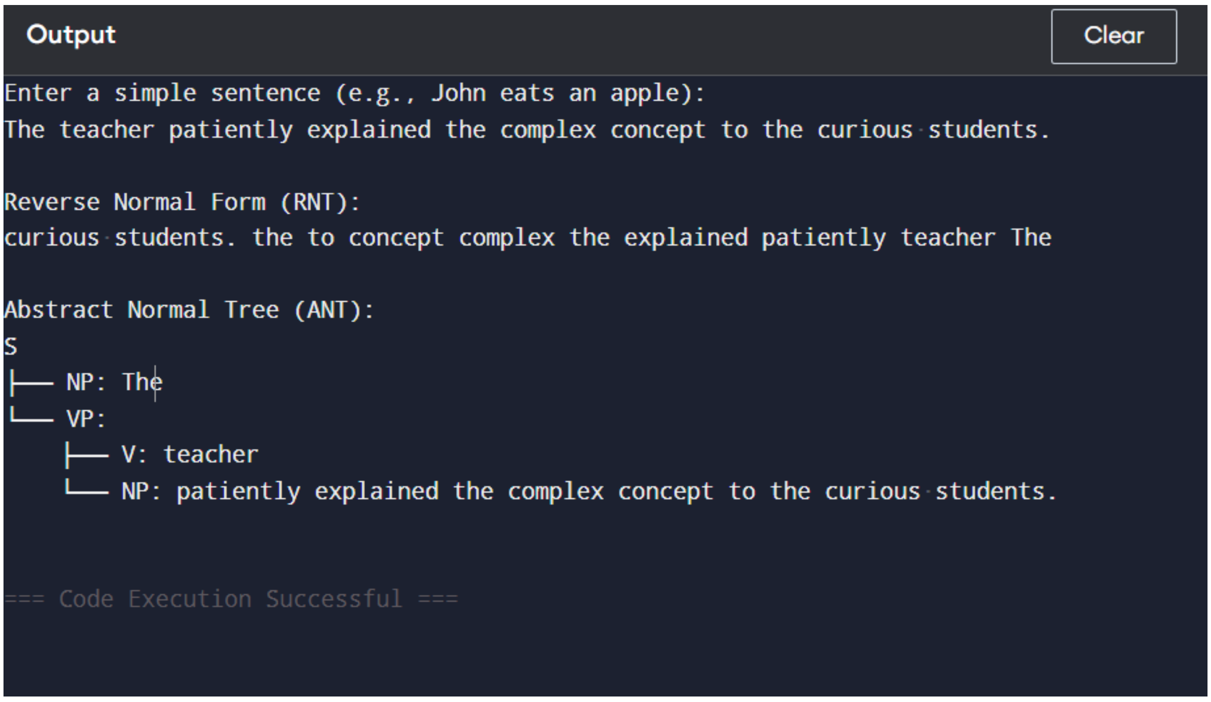
5. IMPLEMENTATION AND EXAMPLE

C Program for NLP System (RNT and ANT)

```
#include <stdio.h>
#include <string.h>

#define MAX_WORDS 50
#define MAX_LEN 100

// Function to tokenize the sentence into words
int tokenize(char *sentence, char words[MAX_WORDS][MAX_LEN]) {
    int count = 0;
    char *token = strtok(sentence, " ");
    while (token != NULL) {
        strcpy(words[count++], token);
        token = strtok(NULL, " ");
    }
    return count;
}

// Function to print Reverse Normal Form (RNT)
void printRNT(char words[MAX_WORDS][MAX_LEN], int word_count) {
    printf("\nReverse Normal Form (RNT):\n");
    for (int i = word_count - 1; i >= 0; i--) {
        printf("%s ", words[i]);
    }
    printf("\n");
}

// Function to simulate Abstract Normal Tree (ANT)
void printANT(char words[MAX_WORDS][MAX_LEN], int word_count) {
    printf("\nAbstract Normal Tree (ANT):\n");
    printf("S\n");
    printf(" ├── NP: %s\n", words[0]); // Subject
    printf(" └── VP:\n");
    printf("     ├── V: %s\n", words[1]); // Verb
    printf("     └── NP: ");
    for (int i = 2; i < word_count; i++) {
        printf("%s ", words[i]); // Object
    }
    printf("\n");
}

int main() {
    char sentence[MAX_LEN];
    char words[MAX_WORDS][MAX_LEN];
    int word_count;
    printf("Enter a simple sentence :\n");
    fgets(sentence, MAX_LEN, stdin);
    sentence[strcspn(sentence, "\n")] = 0; // Remove newline

    word_count = tokenize(sentence, words);
    printRNT(words, word_count);
    printANT(words, word_count);

    return 0;
}
```

OUTPUT:

```
Enter a simple sentence :
John eats an apple

Reverse Normal Form (RNT):
apple an eats John

Abstract Normal Tree (ANT):
S
├── NP: John
└── VP:
    ├── V: eats
    └── NP: an apple
```



```
Enter a simple sentence (e.g., John eats an apple):
The teacher patiently explained the complex concept to the curious students.

Reverse Normal Form (RNT):
curious students. the to concept complex the explained patiently teacher The

Abstract Normal Tree (ANT):
S
├── NP: The
└── VP:
    ├── V: teacher
    └── NP: patiently explained the complex concept to the curious students.


=== Code Execution Successful ===
```

6. CONCLUSION

In this project, we explored the integration of the **Theory of Computation (TOC)** with **Natural Language Processing (NLP)** to analyse sentence structures using concepts like **Reverse Normal Form (RNT)** and **Abstract Normal Tree (ANT)**. Through the development of a simplified NLP system in C, we demonstrated how basic computational principles can be applied to understand and represent linguistic data.

The system tokenized input sentences, applied syntactic rules, and produced structured representations that are crucial for deeper language processing tasks. This not only enhanced our understanding of sentence syntax but also showed how computational logic plays a vital role in AI-based language systems.

Such projects lay the foundation for building intelligent language models, chatbots, grammar checkers, and translation systems. Overall, this exercise emphasized the importance of merging theoretical concepts with real-world applications in AI.