## GRAMMAR

## AUTOCORRECTOR

**PROJECT REPORT**

***Submitted by***

**Yanamadala Sri Harshitha and K Nireesha**

*Under the guidance of*

### Dr.Vijayabaskar

*in partial fulfillment*

## *for the completion of course*

***CSA1363-Theory of computations***



**SIMATS ENGINEERING**

## **THANDALAM**

### BONAFIDE CERTIFICATE

This project report titled **“ GRAMMAR CORRECTOR** ” is the bonafide work of **“Yanamadala Sri Harshitha(192210230)”** who carried out the project work under my supervision as a batch. Certified further, that to the best of my knowledge the work reported herein does not form any other project report.

**Date : Project supervisor Head of the Department**

# ABSTRACT

The "Grammar Corrector" project is a to develop a system that can parse sentences based on a given Context-Free Grammar (CFG) and suggest corrections if the sentences do not conform to the grammar.

Grammar auto-correctors are intelligent systems that use advanced language processing to identify and fix grammatical errors in written text. These powerful tools leverage cutting-edge computational linguistics to enhance the clarity and correctness of our communication.

This project explores the fundamental concepts and techniques behind grammar auto-correctors, a powerful tool that helps writers improve their writing by automatically detecting and correcting grammatical errors.

Grammar auto-corrector leverage natural language processing and machine learning to identify and correct grammatical errors in written text. These powerful tools enhance writing quality and promote clear communication.

**Keywords**:- Grammar,Natural Language Processing,Tokenization,Parsing.

**INTRODUCTION:**

* Any Grammar can be represented by 4 tuples--(V,T,P,S)
* A CFG consists of a set of production rules that describe all possible strings in a given formal language. Each rule describes how symbols in the language can be replaced with other symbols.

- Components:

- Non-terminals(V): Symbols used to help define the grammar rules.

-Terminals(T): The actual symbols of the language (e.g., characters or words).

- Production Rules(P): Rules that define how non-terminals can be replaced by terminals or other non-terminals.

- Start Symbol(S): A special non-terminal from which production starts

**Algorithms for Grammar Auto-Correction:**

****ERROR DETECTION:**Combining lexical analysis, parsing, and pattern matching to identify grammatical errors in text.**

**ERROR CORRECTION:Generating and evaluating potential corrections using language models, edit distance, and machine learning techniques.**

****CONTEXTUAL AWARENESS**:Leveraging semantic and discourse information to provide more accurate and contextually appropriate corrections.**

****INTERACTIVE FEEDBACK**:Enabling users to provide feedback and refinements to improve the auto-correction system over time.**

**Real-World Application:**While the current implementation serves as a proof of concept, real-world applications would require a more robust approach:

* **Academic and Professional Use:** In theoretical computer science or related fields, a more sophisticated tool could assist in writing and validating formal notations and proofs.
* **Educational Tools:** Such a system could be integrated into educational tools to help students and researchers learn and apply correct notation.

## METHODOLOGY

### ****1.Define the Language****

### Identify the language to autocorrect.

### 2. ****Define the Grammar****

You need to define a CFG that captures the rules for the notation you want to support. For theoretical computer science notation, this might involve defining rules for various constructs used in the field.

### 3. ****Tokenization****

Break down the input sentences into tokens that can be analyzed according to the grammar rules.

### 3. ****Parsing****

Implement or use a parsing algorithm to check if the input sentence adheres to the defined CFG.

### 4. ****Error Detection****

Identify whether the sentence confirms to the grammar rules.Compare parsing input with valid strins generated by the CFG.And Identify the errors.

### 5. ****Error Correction****

Suggest corrections or modifications to make the sentence conform to the grammar.This include modifying production rules to suggest most likely corrections.

## IMPLEMENTATION:

**Expand the Grammar:**

-Define more complex grammar rules if needed.

**Improve Error Correction:**

-Develop algorithms to suggest specific corrections, such as word substitutions or reordering based on common patterns.

**Integrate Machine Learning:**

**-** For more sophisticated grammar correction, consider using machine learning models trained on annotated data.

**User Interface:**

-Create a user-friendly interface for input and correction suggestions.

**Testing and Validation:**

-Test the system with various inputs to ensure accuracy and robustness.

### Additional Considerations:

**Grammar Complexity:** The CFG provided is quite simple. Real-world grammars can be significantly more complex, requiring more advanced parsing techniques.

**Performance:** For large-scale or real-time applications, ensure the parsing and correction algorithms are optimized for performance.

## CODE:

## #include <stdio.h>

## #include <string.h>

## #include <stdbool.h>

## // Define the maximum length of the sentence and number of tokens

## #define MAX\_SENTENCE\_LENGTH 100

## #define MAX\_TOKENS 20

## // Grammar rules

## const char\* valid\_tokens[] = {"a", "the", "machine", "algorithm", "state", "system", "in", "with", "is", "uses"};

## const int valid\_token\_count = sizeof(valid\_tokens) / sizeof(valid\_tokens[0]);

## // Function to tokenize the sentence

## void tokenize(char \*sentence, char tokens[MAX\_TOKENS][MAX\_SENTENCE\_LENGTH], int \*token\_count) {

## char \*token = strtok(sentence, " ");

## \*token\_count = 0;

## 

## while (token != NULL && \*token\_count < MAX\_TOKENS) {

## strcpy(tokens[\*token\_count], token);

## (\*token\_count)++;

## token = strtok(NULL, " ");

## }

## }

## // Function to check if a token is valid

## bool is\_valid\_token(const char \*token) {

## for (int i = 0; i < valid\_token\_count; i++) {

## if (strcmp(token, valid\_tokens[i]) == 0) {

## return true;

## }

## }

## return false;

## }

## // Function to parse the sentence (dummy implementation for illustration)

## bool parse\_sentence(char tokens[MAX\_TOKENS][MAX\_SENTENCE\_LENGTH], int token\_count) {

## // This is a very simplified parsing check

## if (token\_count < 2) {

## return false;

## }

## // Basic structure validation (not complete CFG parsing)

## if (strcmp(tokens[0], "the") == 0 || strcmp(tokens[0], "a") == 0) {

## if (strcmp(tokens[1], "machine") == 0 || strcmp(tokens[1], "algorithm") == 0 ||

## strcmp(tokens[1], "state") == 0 || strcmp(tokens[1], "system") == 0) {

## return true;

## }

## }

## return false;

## }

## // Function to correct the sentence (simple correction for invalid sentences)

## void correct\_sentence(char \*sentence) {

## char tokens[MAX\_TOKENS][MAX\_SENTENCE\_LENGTH];

## int token\_count = 0;

## tokenize(sentence, tokens, &token\_count);

## if (parse\_sentence(tokens, token\_count)) {

## printf("Original sentence: %s\n", sentence);

## printf("Corrected sentence: %s\n", sentence); // No correction needed if valid

## } else {

## printf("Original sentence: %s\n", sentence);

## printf("Corrected sentence: Syntax error detected. Please check the sentence structure.\n");

## }

## }

## int main() {

## char input\_sentence[MAX\_SENTENCE\_LENGTH] = "the machine is with algorithm";

## correct\_sentence(input\_sentence);

## return 0;

## }

## RESULTS AND DISCUSSION:

* **Improved Communication:**

**-Grammar auto-correctors enhance the clarity and professionalism of written communication.**

* **Time-Saving:**

**-Automated grammar correction saves users time and effort, allowing them to focus on content.**

* ****Personalized Experience:****

**-Adaptive grammar correction systems can provide a more tailored and effective user experience.**

* **Grammar auto-corrector will continue to evolve, leveraging advancements in NLP and machine learning and The ease of use and integration of the grammar auto-corrector into the user's workflow .**
* **Finally ensuring grammar auto-correction can handle diverse and evolving language constructs without significant performance degradation.**

**CONCLUSION:**

The project successfully demonstrated the basic concept of a grammar auto-corrector using CFG and parsing techniques. However, to be fully functional and useful, the system requires enhancements in grammar coverage, error correction capabilities, and user interaction. Future work should focus on addressing these limitations to build a more advanced and practical tool for grammar correction in theoretical computer science notation.

**REFERENCES:**

窗体顶端

窗体底端