**CS6109 COMPILER DESIGN LAB**

**B.E CSE V Q - BATCH**

**LAB NO: 18** **DATE: 08.01.2022**

**TEAM MEMBERS:** **TEAM NO: 16**

| S.No. | REG. NO. | NAME |
|---|---|---|
| 1 | 2019103503 | Ajitesh M |
| 2 | 2019103527 | Ishwarya Rani M |
| 3 | 2019103548 | Navvya L |

**Project Title: GoGrammer – Natural Language Parsing using SLR Parser with grammar suggestions.**

| | |
|---|---|
| **Abstract, Deliverables – Input & Output, Problem Statement & Objective and Modules (I/O) (2)** | |
| **Literature Survey (2)** | |
| **Block Diagram (3)** | |
| **Dataset Description (3)** | |
| **Result Implementation (5)** | |
| **Total (15)** | |
| **Signature** | |

## ABSTRACT:

Usage of good grammar and correctly spelled words helps you to write and communicate clearly. Whether you are working on an article, essay, or email, presenting your ideas with clear and correct language makes a good impression on your readers. Often while typing, one makes a lot of grammatical and spelling mistakes.

This writing aid checks for grammatical correctness for a given piece of text like essay, article, mail etc. It offers spell and grammar check and provides suggestions for correcting it. It also provides analysis of the script such as word count, spell checks etc. This writing aid can be used to verify the grammatical correctness of any text such as essays, articles, letters etc.

In this project we use SLR (1) parser for checking grammatical correctness and semantic analysis using NLP to provide suggestions to any incorrect sentence.

## DELIVERABLES:

**Input:** File containing English text

**Output:** Report file comprising word count, accuracy percentage, spelling mistakes and grammatically incorrect sentences with their corrections.

# 1. INTRODUCTION:

In daily life the language used for communication can be termed as Natural Language (NL) and it evolves from generation to generation. NL is the most powerful tool that humans possess for conveying information. At the core of Natural Language Processing (NLP) task there is an important issue of Natural Language Understanding (NLU). NLP is computer manipulation of NL.

Syntactic analysis is a method of assigning a syntactic structure or a parse tree, to a given natural language sentence. Syntactic processing requires natural language grammar rules, lexicon and parsing technique. Parsing is a process to determine how an input string might be derived using productions (rewrite rules) of a given grammar. It can be used to check whether or not an input string belongs to a given language. When a statement written in a language is input, it is parsed by a parser to check whether or not it is syntactically correct.

There are many different types of parsers for context free grammars, in this project we use SLR (1) parser. This writing aid checks for grammatical correctness for a given piece of text like essay, article, mail etc. Semantic analysis with NLP is used to provide suggestions to any incorrect sentence. It offers spell and grammar check and provides suggestions for correcting it. It also provides analysis of the script such as word count, spell checks etc.

# 2. PROBLEM STATEMENT:

With the help of a SLR parser, check for grammatical correctness of an English sentence. Also provide suggestions for incorrect statement using NLP Algorithms.

**Input:** Dynamically typed piece of text like essay, article, mail etc

**Output:** Check for its grammatical correctness, if not correct the text is highlighted and suggestions are provided for its correction. Analysis of the script such as word count, sentiment score etc of the text is also provided.

# 3. OBJECTIVE:

To build a writing aid that checks for grammatical correctness of any dynamically typed piece of text.

## 4. LITERATURE SURVEY (Base Paper)

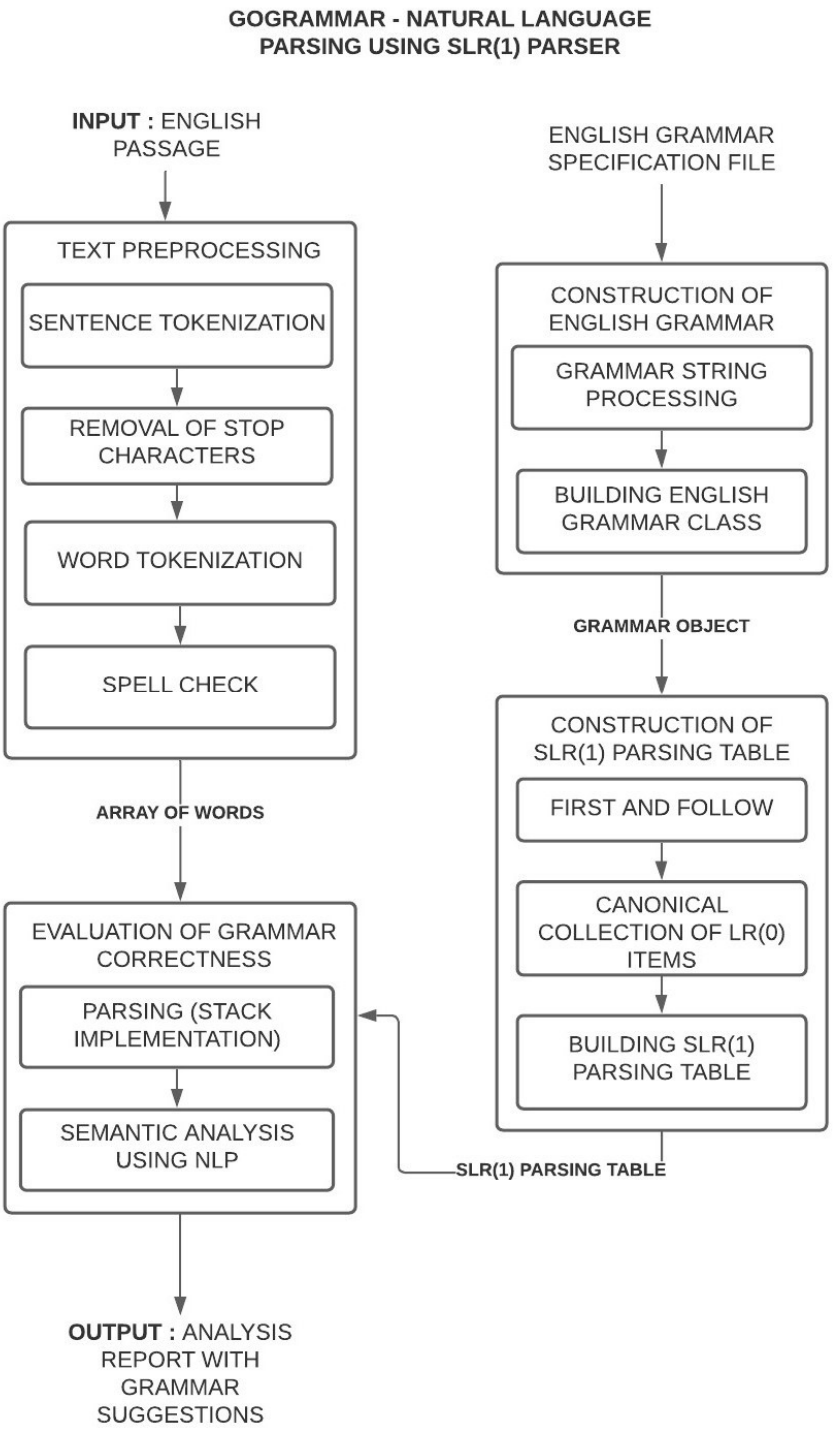
The language used for communication in daily life is termed as Natural Language (NL) and it evolves from generation to generation. NL is the most powerful tool that humans possess for conveying information. At the core of Natural Language Processing (NLP) task there is an important issue of Natural Language Understanding (NLU). NLP is computer manipulation of NL.

A fuzzy parser is a form of syntax analyzer that performs analysis of a complete source input is. The Bottom up LR (left to right) syntax analysis method is a useful and versatile technique for parsing deterministic Fuzzy context-free languages. This Fuzzy Simple LR parser (FSLR) for parsing English sentences uses Fuzzy Context Free Grammar (FCFG). This paper discusses a procedure for constructing an LR parse table from Fuzzy context free grammar and using this table the input sentence is tested for syntactic correctness. Here Fuzzy context free grammar is designed for parsing Natural language. Here FSLR is implemented in C Programming Language. Considering English language sentence as an input, permutations are generated and for the generated permutations Fuzzy Simple LR algorithm is applied. Finally Fuzzy max-min technique is applied to get the degree of fuzzyness. This research work involves the design of fuzzy parsing algorithms and implementation to provide the results. This fuzzy parser gives degree of fuzzyness and syntactic correctness for partially parsed.

Fuzzy Context-Free Grammar (FCFG) is a straightforward extension of context-free grammar, and has been introduced to express uncertainty, ambiguity, and vagueness in natural language fragments. Here production rules are written by considering noun phrase (NP), verb phrase (VP), preposition (PP), adjective (ADJP) and fuzzy membership values are assigned for each rule. Membership values are assigned randomly to each of these rules, finally the values for the entire set of rules for each phrase sums up to 1.

English language sentence is given as an input and permutations are generated for the input. Here FCFG is designed. Using FCFG Closure of item sets, FIRST and FOLLOW sets are generated, and then these are used for constructing ACTION and GOTO tables. Further these tables are used for checking the syntactic correctness for the generated permutations. Fuzzy Max-Min technique is applied to get the degree of fuzzyness for the parsed input sentences. Here completely parsed sentences show the maximum fuzzy membership value as 1 and partially parsed sentences shows the respective degree of fuzzyness.

# 5. BLOCK DIAGRAM



# 6. MODULES

1. Module 1 – Construction of English grammar
2. Module 2 – Construction of SLR (1) Parsing Table
3. Module 3 – Text Preprocessing
4. Module 4 – Evaluation of Grammar Correctness

### ❖ Module 1: Construction of English grammar

**Input**: English Grammar specification file

**Output**: Grammar Object

The input Grammar specification file is parsed and is stored in the appropriate data structures in the grammar Object.

### ❖ Module 2: Construction of SLR (1) Parsing Table

**Input**: English grammar object

**Output**: SLR (1) Parsing table

Augmented English Grammar and First and follow sets for each non terminal is constructed. Then the canonical collection of LR (0) items are built and is used for the construction of the SLR (1) parser table.

### ❖ Module 3: Text Preprocessing

**Input**: English text

**Output**: Array of words

Given English text segmented into sentences and stop characters are removed. Each sentence is further tokenized into array of words. Then spell check is performed and incorrect words are replaced with correct spelling.

### ❖ Module 4: Evaluation of Grammar Correctness

**Input**: Array of words (with correct spelling) and SLR (1) Parsing table

**Output**: Analysis report with grammar suggestions

Stack implementation for the input sentence (array of words) is performed to evaluate the grammar correctness. If the sentence is grammatically incorrect, semantic analysis using NLP is performed to provide suggestions.

## 7. DATASET DESCRIPTION

The dataset we have considered is a sample data related to general english tenses sentences. It consists of a total of 70 sentences. There are a total of 420 words which cover almost all the kind of tenses.

Sample data :

She were writing a letter to Ishwarya.

My teacher teaches well.

She went to school.

Raj is climbs a mountian.

Raj plays fotball well.

Ajitesh cook delisious food.

Navvya is a toper.

Ishwarya is a kabbadi player.

Navvya will sings a song in the show.

These people is from Canada.

Ref : https://www.english-practice.at/b1/grammar/tenses/b1-tenses-index.htm

# 8. RESULTS DISCUSSION

## Installing required packages and libraries:

Code snippet 1-

```
pip install pyspellchecker
```

Output snippet 1-

```
Collecting pyspellchecker
  Downloading pyspellchecker-0.6.2-py3-none-any.whl (2.7 MB)
     |████████████████████████████████| 2.7 MB 5.4 MB/s
Installing collected packages: pyspellchecker
Successfully installed pyspellchecker-0.6.2
```

Code snippet 2-

```
pip install gingerit
```

Output snippet 2–

```
Collecting gingerit
  Downloading gingerit-0.8.2-py3-none-any.whl (3.3 kB)
Collecting requests<3.0.0,>=2.25.1
  Downloading requests-2.26.0-py2.py3-none-any.whl (62 kB)
     |████████████████████████████████| 62 kB 679 kB/s
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.25.1->gingerit) (1.24.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.25.1->gingerit) (2.0.8)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.25.1->gingerit) (2021.10.8)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.25.1->gingerit) (2.10)
Installing collected packages: requests, gingerit
  Attempting uninstall: requests
    Found existing installation: requests 2.23.0
    Uninstalling requests-2.23.0:
      Successfully uninstalled requests-2.23.0
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency
google-colab 1.0.0 requires requests~=2.23.0, but you have requests 2.26.0 which is incompatible.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatible.
Successfully installed gingerit-0.8.2 requests-2.26.0
```

Code snippet 3-

```
from gingerit.gingerit import GingerIt
from spellchecker import SpellChecker
```

# MODULE 1 – Construction of English grammar

Code snippet 4 - Opening text file and reading production rules from it.

```
[ ]  # READING GRAMMAR SPECIFICATIONS

     inputFile = open('grammar.txt', 'r')
     inputText = inputFile.read()
     grammarRules = [ rule.strip() for rule in inputText.split("\n")]
     grammarRules
```

Output snippet 4 - The production rules read is displayed

```
['S -> NP_SG PAST S1 | NP_SG PAST_PERFECT S1 | NP_SG PAST_CONT_SG S1 | NP_SG PAST_PERFECT_CONT S1 | NP_SG PRES_SG S1 | NP_SG PRES_PERFECT_SG S1 | NP_SG PRES_CONT_SG S1 |
 'S1 -> NP | ADV',
 'NP -> NP_SG | NP_PL | PP | NP PP',
 'NP_SG ->  PRONOUN_SG | PRONOUN_SG NOM_SG| PROPERNOUN | NOM_SG | DET_SG NOM_SG | DET_BOTH NOM_SG | ADV NP_SG | NP_SG CONJ NP',
 'NOM_SG -> NOUN_SG | ADJ NOUN_SG | NOM_SG NOUN_SG',
 'NP_PL -> PRONOUN_PL | PRONOUN_PL NOM_PL| NOM_PL | DET_PL NOM_PL | DET_BOTH NOM_PL | ADV NP_PL | NP_PL CONJ NP',
 'NOM_PL -> NOUN_PL | ADJ NOUN_PL | NOM_PL NOUN_PL',
 'PP -> PREPOS NP_SG | PREPOS NP_PL',
 'PAST -> V2 | ADV V2',
 'PAST_PERFECT -> had V3 | had ADV V3',
 'PAST_CONT_SG -> was GERUNDV | was ADV GERUNDV',
 'PAST_CONT_PL -> were GERUNDV | were ADV GERUNDV',
 'PAST_PERFECT_CONT -> had been GERUNDV | had ADV been GERUNDV',
 'PRES_SG -> V1_SG | ADV V1_SG',
 'PRES_PL -> V1_PL | ADV V1_PL',
 'PRES_PERFECT_SG -> has V3 | has ADV V3',
 'PRES_PERFECT_PL -> have V3 | have ADV V3',
 'PRES_CONT_SG -> is GERUNDV | is ADV GERUNDV',
 'PRES_CONT_PL -> are GERUNDV | are ADV GERUNDV',
 'PRES_PERFECT_CONT_SG -> has been GERUNDV | has ADV been GERUNDV | has been | has ADV been',
 'PRES_PERFECT_CONT_PL -> have been GERUNDV | have ADV been GERUNDV',
 'FUTURE -> will V1_PL | will ADV V1_PL',
 'FUTURE_PERFECT -> will have V3 | will have ADV V3',
 'FUTURE_CONT -> will be GERUNDV | will ADV be GERUNDV',
 'FUTURE_PERFECT_CONT -> will have been GERUNDV | will have ADV been GERUNDV',
 'PRONOUN_SG -> my | she | he | it | him | i | her | his',
 'PRONOUN_PL -> they | us | you | we | their',
 'RELPR_SG -> which',
 'RELPR_PL -> whom',
 'PROPERNOUN -> ishwarya | joseph | india | germany | ajitesh | navvya | ritheesh | switzerland | paris | wednesday | january | may | turkey | easter | november | raj |
 'NOUN_SG -> sentence | me | kabaddi | friend | patient | parcel | player | moment | morning | grandfather | outlet | evening | programming | building | thinking | mom |
 'NOUN_PL -> roses | mountains | goats | films | times | emails | people | children |  specialists | girls | parents | jacksons | cows | boys | books | years | portraits
 'DET_SG -> a | an | this | that',

'DET_SG -> a | an | this | that',
'DET_PL -> these | all | many | no',
'DET_BOTH -> the',
'CONJ -> and | but | because | while',
'ADJ -> red | delicious | first | few | two | four | three | fun | since | sweet | nice | severe | terrible | hungry | new | whole | last | every | critical',
'ADV -> well | gracefully | before | next | very | when | not | whereabouts | constantly | lately | yesterday | ago | tomorrow | early | still | twice | always | alre
'PREPOS -> at | in | of | for | by | from | far | on | around | as | to | down',
'V1_SG -> climbs | teaches | plays | cooks | is | likes | be | know | am | draws | works | understands | invloves | sings',
'V1_PL -> teach | are | sing | give | think | rain | turn | go | see | do | take | study | speak | drink | cook',
'GERUNDV -> teaching | running | writing | opening | burning | feeling | enjoying | living | spending | ringing | repairing | visiting | having | climbing | getting |
'V2 -> taught | were | was | went | knew | would | started | received | gave | caught | married | understood | flew | drink | read | made | tired | saw | did',
'V3 -> seen | written | given | done | gone | made | flown | painted | fed | met | finished | acted | watched']
```

Code snippet 5 - Grammar class is defined. The production rules read from grammar.txt file is split into a tuple containing three elements and is stored in the appropriate data structures in the grammar Object. Object of class grammar is created.

```
class Grammar :
  def __init__(self, grammarStr) :
    self.input = grammarStr
    self.symbols = set()
    self.terminals = set()
    self.nonterminals = set()
    self.startSymbol = None
    self.grammar = {}

    for production in grammarStr :
      # The partition() method searches for a specified string, and splits the string into a tuple containing three elements.
      head, _, bodies = production.partition(' -> ')

      # The nonterminal of the first production is considered to be the start symbol
      if self.startSymbol is None :
        self.startSymbol = head

      self.grammar.setdefault(head, set())
      self.nonterminals.add(head)

      bodies = {tuple(body.strip().split(" ")) for body in ' '.join(bodies.split(" ")).split('|') }

      for body in bodies :
        self.grammar[head].add(body)
        for symbol in body :
          if symbol.isupper() :
            self.nonterminals.add(symbol)
          else :
            self.terminals.add(symbol)
    self.symbols = self.nonterminals | self.terminals

  def __str__ (self) :
    print("Start Symbol : " , self.startSymbol)
    print("Non-terminals : " , self.nonterminals)
    print("Terminals : " , self.terminals)
    print("Productions : ")
    for head in self.grammar :
      for body in self.grammar[head] :
        print(head, "->", body)
    return ""
```

```
[ ]  G = Grammar(grammarRules)
     print(G)
```

Output snippet 5 - The grammar object is displayed.

```
Start Symbol :  S
Non-terminals :  {'PAST_CONT_PL', 'NOUN_SG', 'CONJ', 'NOUN_PL', 'PRONOUN_SG', 'S1', 'PP', 'NP_PL', 'PRES_CONT_SG', 'RELPR_SG', 'DET_SG', 'V1_I
Terminals :  {'first', 'listening', 'knew', 'building', 'year', 'turn', 'past', 'twice', 'navvya', 'football', 'meeting', 'turkey', 'wednesda
Productions :
S -> ('NP_PL', 'PRES_CONT_PL', 'S1')
S -> ('NP_PL', 'FUTURE', 'S1')
S -> ('NP_PL', 'FUTURE_PERFECT', 'S1')
S -> ('NP_PL', 'FUTURE_PERFECT_CONT', 'S1')
S -> ('NP_SG', 'PAST_PERFECT_CONT', 'S1')
S -> ('NP_SG', 'FUTURE_PERFECT', 'S1')
S -> ('NP_PL', 'PRES_PERFECT_PL', 'S1')
S -> ('NP_SG', 'PAST', 'S1')
S -> ('NP_SG', 'FUTURE', 'S1')
S -> ('NP_SG', 'PRES_CONT_SG', 'S1')
S -> ('NP_PL', 'FUTURE_CONT', 'S1')
S -> ('NP_PL', 'PRES_PERFECT_CONT_PL', 'S1')
S -> ('NP_SG', 'FUTURE_PERFECT_CONT', 'S1')
S -> ('NP_SG', 'PRES_PERFECT_SG', 'S1')
S -> ('NP_PL', 'PAST_PERFECT', 'S1')
S -> ('NP_SG', 'PRES_PERFECT_CONT_SG', 'S1')
S -> ('NP_SG', 'PAST_PERFECT', 'S1')
S -> ('NP_PL', 'PAST', 'S1')
S -> ('NP_PL', 'PAST_PERFECT_CONT', 'S1')
S -> ('NP_PL', 'PRES_PL', 'S1')
```

```
S -> ('NP_PL', 'PRES_PL', 'S1')
S -> ('NP_SG', 'PAST_CONT_SG', 'S1')
S -> ('NP_SG', 'FUTURE_CONT', 'S1')
S -> ('NP_SG', 'PRES_SG', 'S1')
S -> ('S', 'ADV')
S -> ('NP_PL', 'PAST_CONT_PL', 'S1')
S1 -> ('NP',)
S1 -> ('ADV',)
NP -> ('NP_PL',)
NP -> ('NP_SG',)
NP -> ('NP', 'PP')
NP -> ('PP',)
NP_SG -> ('PRONOUN_SG',)
NP_SG -> ('NP_SG', 'CONJ', 'NP')
NP_SG -> ('ADV', 'NP_SG')
NP_SG -> ('NOM_SG',)
NP_SG -> ('PROPERNOUN',)
NP_SG -> ('DET_BOTH', 'NOM_SG')
NP_SG -> ('PRONOUN_SG', 'NOM_SG')
NP_SG -> ('DET_SG', 'NOM_SG')
NOM_SG -> ('NOUN_SG',)
NOM_SG -> ('NOM_SG', 'NOUN_SG')
NOM_SG -> ('ADJ', 'NOUN_SG')
NP_PL -> ('DET_PL', 'NOM_PL')
NP_PL -> ('ADV', 'NP_PL')
NP_PL -> ('PRONOUN_PL',)
NP_PL -> ('PRONOUN_PL', 'NOM_PL')
NP_PL -> ('NOM_PL',)
NP_PL -> ('DET_BOTH', 'NOM_PL')
NP_PL -> ('NP_PL', 'CONJ', 'NP')

NOUN_SG -> ('accountant',)
NOUN_SG -> ('letter',)
NOUN_SG -> ('peacock',)
NOUN_SG -> ('friend',)
NOUN_SG -> ('morning',)
NOUN_SG -> ('baby',)
NOUN_SG -> ('food',)
NOUN_SG -> ('pilot',)
NOUN_SG -> ('street',)
NOUN_SG -> ('show',)
NOUN_SG -> ('album',)
NOUN_SG -> ('teacher',)
NOUN_SG -> ('call',)
NOUN_SG -> ('neighbourhood',)
NOUN_SG -> ('milk',)
NOUN_SG -> ('basketball',)
NOUN_SG -> ('picture',)
NOUN_SG -> ('car',)
NOUN_SG -> ('airplane',)
NOUN_SG -> ('mountain',)
NOUN_SG -> ('weather',)
NOUN_SG -> ('building',)
NOUN_SG -> ('dad',)
NOUN_SG -> ('year',)
NOUN_SG -> ('assignment',)
NOUN_SG -> ('movie',)
NOUN_SG -> ('grandfather',)
NOUN_PL -> ('children',)
NOUN_PL -> ('ages',)
NOUN_PL -> ('years',)
NOUN_PL -> ('goats',)
NOUN_PL -> ('films',)
NOUN_PL -> ('friends',)
```

## MODULE 2 – Construction of SLR (1) Parsing Table

Code snippet 6 -

Generating first and follow sets. Initialising the data structures to store the first and follow values.

```
# Generating first and follow sets

def first_follow(G) :
  def union(set1, set2):
    set1len = len(set1)
    set1 |= set2
    return set1len != len(set1)

  # initialising the data structures to store and first and follow values
  first = { symbol: set() for symbol in G.symbols }
  follow = { symbol: set() for symbol in G.symbols }

  for symbol in G.symbols :
    if symbol in G.terminals :
      first[symbol].add(symbol)

  follow[G.startSymbol].add('$')

  while True :
    updated = False
    for head, bodies in G.grammar.items() :
      for body in bodies :
        for symbol in body :
          if symbol != '^':
            updated |= union(first[head], first[symbol] - set('^'))
            if '^' not in first[symbol] :
              break
            else :
              updated |= union(first[head], set('^'))
        aux = follow[head]
        for symbol in reversed(body) :
          if symbol == '^':
            continue
          if symbol in follow:
            updated |= union(follow[symbol], aux - set('^'))
          if '^' in first[symbol] :
            aux = aux | first[symbol]
          else :
            aux = first[symbol]
    if not updated:
      return first, follow
```

Constrution of LR(0) items and constrution of cananical collection of sets of LR(0) items

```
# Construction of LR(0) items

# Computation of set of items constructed from I
# I - set of items for G
def CLOSURE(aug_G, I):
  J = I

  while True :
    item_len = len(J)
    for head, bodies in J.copy().items() :
      for body in bodies.copy():
        if '.' in body[:-1]:
          dot_pos = body.index('.')
          symbol_after_dot = body[dot_pos + 1]
          if symbol_after_dot in aug_G.nonterminals :
            for G_body in aug_G.grammar[symbol_after_dot]:
              if G_body == tuple('^'):
                J.setdefault(symbol_after_dot, set()).add(set('.'))
```

```
                else :
                  J.setdefault(symbol_after_dot, set()).add(('.',) + G_body)
        if item_len == len(J):
          return J

    # GOTO(I, X) defined to be the closure of the set of all items [A -> aX.b] such that [A -> a.Xb] is in I
    # I - set of items
    # X - grammar symbol
    def GOTO(aug_G, I, X):
      goto = {}

      for head, bodies in I.items() :
        for body in bodies :
          if '.' in body[:-1]:
            dot_pos = body.index('.')
            if body[dot_pos + 1] == X :
              replaced_dot_body = body[ : dot_pos] + (X, '.') + body[dot_pos + 2: ]
              for C_head, C_bodies in CLOSURE(aug_G, {head: {replaced_dot_body}}).items():
                goto.setdefault(C_head, set()).update(C_bodies)

      return goto
```

```
    # Construction of canonical collection of sets of LR(0) items for G'
    def items(aug_G):
      C = [CLOSURE(aug_G, {aug_G.startSymbol : {('.', aug_G.startSymbol[:-1])}})]

      while True:
        item_len = len(C)
        for I in C.copy() :
          for X in aug_G.symbols:
            goto = GOTO(aug_G, I, X)
            if (len(goto) > 0) and goto not in C:
              C.append(goto)
        if item_len == len(C) :
          return C
```

## Constrution of SLR(1) parsing table

```
    # Construction of SLR(1) Parsing table

    class SLRParser:
      def __init__(self, G):
        self.aug_G = Grammar([f"{G.startSymbol}' -> {G.startSymbol}"] + G.input)
        self.max_aug_G_len = len(max(self.aug_G.grammar, key=len))

        self.G_indexed = []
        for head, bodies in self.aug_G.grammar.items():
          for body in bodies:
            self.G_indexed.append([head, body])

        self.first, self.follow = first_follow(self.aug_G)

        self.C = items(self.aug_G)

        self.action = list(self.aug_G.terminals) + ['$']
        self.goto = []
        for nonterminal in self.aug_G.nonterminals :
          if nonterminal != self.aug_G.startSymbol :
            self.goto.append(nonterminal)
```

```
    self.parse_table_symbols = self.action + self.goto
    self.parse_table = self.construct_table()

  def construct_table(self):
    # Step 1 - Construct C
    # Step 2 - parsing actions for state i are determined
    parse_table = {r: {c: '' for c in self.parse_table_symbols} for r in range(len(self.C))}

    for i, I in enumerate(self.C) :
      for head, bodies in I.items() :
        for body in bodies :
          if '.' in body[:-1] : # Case 2a
            dot_pos = body.index('.')
            symbol_after_dot = body[dot_pos + 1]

            if symbol_after_dot in self.aug_G.terminals :
              s = f's{self.C.index(GOTO(self.aug_G, I, symbol_after_dot))}'
              if s not in parse_table[i][symbol_after_dot]:
                if 'r' in parse_table[i][symbol_after_dot] :
                  parse_table[i][symbol_after_dot] += '/'
                parse_table[i][symbol_after_dot] += s
          elif body[-1] == '.' and head != self.aug_G.startSymbol : # Case 2b
            for j, (G_head, G_body) in enumerate(self.G_indexed) :
              if G_head == head and (G_body == body[:-1] or G_body == ('^',) and body == ('.',)):
                for f in self.follow[head] :
                  if parse_table[i][f] :
                    parse_table[i][f] += '/'
                  parse_table[i][f] += f'r{j}'

                  break

          else : # Case 2c
            parse_table[i]['$'] = 'acc'

      # Step 3 - Goto transistions for all nonterminals are constructed
      for A in self.aug_G.nonterminals :
        j = GOTO(self.aug_G, I, A)
        if j in self.C :
          parse_table[i][A] = self.C.index(j)

    return parse_table

  def __str__ (self) :
    def fprint(text, variable) :
      print(f'{text:>12}: {", ".join(variable)}')
    def print_line() :
      print(f'+{("-" * width + "+") * (len(list(self.aug_G.symbols) + ["$"]))}')
    def symbols_width(symbols) :
      return (width + 1) * len(symbols) - 1

    print("Augmented Grammar : ")
    for i, (head, body) in enumerate(self.G_indexed):
      print(f'{i:>{len(str(len(self.G_indexed) - 1))}}: {head:>{self.max_aug_G_len}} -> {" ".join(body)}')
    print()
    fprint('TERMINALS', self.aug_G.terminals)
    fprint('NONTERMINALS', self.aug_G.nonterminals)
    fprint('SYMBOLS', self.aug_G.symbols)

    print('\nFIRST:')
    for head in self.aug_G.grammar:
      print(f'{head:>{self.max_aug_G_len}} = {{ {", ".join(self.first[head])} }}')

    print('\nFOLLOW:')
    for head in self.aug_G.grammar:
      print(f'{head:>{self.max_aug_G_len}} = {{ {", ".join(self.follow[head])} }}')
```

```
        width = max(len(c) for c in {'ACTION'} | self.aug_G.symbols) + 2
        for r in range(len(self.C)):
          max_len = max(len(str(c)) for c in self.parse_table[r].values())

          if width < max_len + 2:
            width = max_len + 2

        print('\nPARSING TABLE:')
        print(f'+{"-" * width}+{"-" * symbols_width(self.action)}+{"-" * symbols_width(self.goto)}+')
        print(f'|{"":{width}}|{"ACTION":^{symbols_width(self.action)}}|{"GOTO":^{symbols_width(self.goto)}}|')
        print(f'|{"STATE":^{width}}+{("-" * width + "+") * len(self.parse_table_symbols)}')
        print(f'|{"":^{width}}|', end=' ')

        for symbol in self.parse_table_symbols:
          print(f'{symbol:^{width - 1}}|', end=' ')

        print()
        print_line()
```

```
        for symbol in self.parse_table_symbols:
          print(f'{symbol:^{width - 1}}|', end=' ')

        print()
        print_line()

        for r in range(len(self.C)):
          print(f'|{r:^{width}}|', end=' ')

          for c in self.parse_table_symbols:
            print(f'{self.parse_table[r][c]:^{width - 1}}|', end=' ')

          print()

        print_line()
        print()
        return ""
```

```
slr = SLRParser(G)
print(slr)
```

Output snippet 6- First and follow and parsing table is displayed

```
Augmented Grammar :
  0:                    S' -> S
  1:                     S -> NP_SG PRES_CONT_SG S1
  2:                     S -> NP_PL FUTURE S1
  3:                     S -> NP_SG FUTURE_PERFECT_CONT S1
  4:                     S -> NP_SG FUTURE_CONT S1
  5:                     S -> NP_SG PAST_CONT_SG S1
  6:                     S -> NP_PL PRES_PERFECT_PL S1
  7:                     S -> NP_PL PAST_CONT_PL S1
  8:                     S -> NP_SG PAST_PERFECT S1
  9:                     S -> NP_SG PRES_PERFECT_CONT_SG S1
 10:                     S -> NP_SG PRES_SG S1
 11:                     S -> NP_PL PRES_PERFECT_CONT_PL S1
 12:                     S -> NP_PL PAST_PERFECT_CONT S1
 13:                     S -> NP_SG PAST_PERFECT_CONT S1
 14:                     S -> NP_PL PAST_PERFECT S1
 15:                     S -> S ADV
 16:                     S -> NP_PL FUTURE_PERFECT_CONT S1
 17:                     S -> NP_PL PRES_PL S1
 18:                     S -> NP_PL PRES_CONT_PL S1
 19:                     S -> NP_SG PRES_PERFECT_SG S1
 20:                     S -> NP_PL FUTURE_CONT S1
 21:                     S -> NP_SG FUTURE S1
 22:                     S -> NP_SG PAST S1
 23:                     S -> NP_PL PAST S1
 24:                     S -> NP_PL FUTURE_PERFECT S1
 25:                     S -> NP_SG FUTURE_PERFECT S1
 26:                    S1 -> ADV
 27:                    S1 -> NP
 28:                    NP -> NP_SG
 29:                    NP -> PP
```

```
   TERMINALS: yesterday, jacksons, their, peacocks, me, homework, roses, food, an, grandfather, dad, repairing, whereabouts, ritheesh, climbing, sings, snowing, ringing
NONTERMINALS: PAST_PERFECT_CONT, NOUN_PL, RELPR_SG, DET_BOTH, V3, NOM_PL, PREPOS, S, NOM_SG, DET_PL, PRES_PERFECT_PL, NP_SG, NOUN_SG, PRES_CONT_SG, PRES_PERFECT_CONT_PL
     SYMBOLS: yesterday, NOUN_PL, their, me, DET_BOTH, roses, food, an, grandfather, whereabouts, sings, NOM_PL, snowing, far, tea, received, were, people, PRES_CONT_SG

FIRST:
                 S' = { yesterday, their, me, roses, food, an, grandfather, whereabouts, tea, people, player, brother, letter, easter, teacher, she, november, parents,
                  S = { yesterday, their, me, roses, food, an, grandfather, whereabouts, tea, people, player, brother, letter, easter, teacher, she, november, parents,
                 S1 = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter, easter, teacher, she, november, par
                 NP = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter, easter, teacher, she, november, par
              NP_SG = { yesterday, me, homework, food, an, grandfather, dad, whereabouts, ritheesh, gracefully, fire, tea, sentence, wife, few, player, brother, nice,
             NOM_SG = { me, homework, food, grandfather, dad, fire, tea, sentence, wife, few, player, brother, nice, letter, teacher, sweet, morning, week, parcel, ter
              NP_PL = { yesterday, jacksons, their, peacocks, still, roses, whereabouts, whole, gracefully, next, few, people, children, portraits, nice, we, parents,
             NOM_PL = { jacksons, peacocks, roses, whole, few, people, children, portraits, nice, parents, sweet, films, goats, mountains, terrible, years, ages, delic
                 PP = { in, as, at, around, by, for, from, down, to, far, of, on }
               PAST = { yesterday, made, constantly, still, twice, understood, whereabouts, was, gave, lately, ago, drink, started, already, gracefully, received, saw,
       PAST_PERFECT = { had }
       PAST_CONT_SG = { was }
       PAST_CONT_PL = { were }
  PAST_PERFECT_CONT = { had }
            PRES_SG = { yesterday, constantly, still, twice, whereabouts, understands, draws, lately, sings, ago, already, gracefully, cooks, when, next, works, always
            PRES_PL = { yesterday, think, speak, constantly, go, still, twice, whereabouts, turn, lately, ago, drink, already, teach, gracefully, when, next, always, n
    PRES_PERFECT_SG = { has }
    PRES_PERFECT_PL = { have }

FOLLOW:
                 S' = { $ }
                  S = { yesterday, constantly, still, twice, whereabouts, lately, ago, already, gracefully, when, next, always, not,
                 S1 = { yesterday, constantly, still, twice, whereabouts, lately, ago, already, gracefully, when, next, always, not,
                 NP = { yesterday, whereabouts, sings, gracefully, far, received, has, were, of, meet, flew, am, today, while, went,
              NP_SG = { yesterday, whereabouts, sings, gracefully, far, received, has, were, of, meet, flew, am, today, while, went,
             NOM_SG = { yesterday, me, homework, food, grandfather, dad, whereabouts, sings, gracefully, far, fire, tea, sentence, r
              NP_PL = { yesterday, whereabouts, sings, gracefully, far, received, has, were, of, meet, flew, am, today, while, went,
             NOM_PL = { yesterday, jacksons, peacocks, roses, whereabouts, sings, gracefully, far, received, has, were, of, people,
                 PP = { yesterday, whereabouts, sings, gracefully, far, received, has, were, of, meet, flew, am, today, while, went,
               PAST = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
       PAST_PERFECT = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
       PAST_CONT_SG = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
       PAST_CONT_PL = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
  PAST_PERFECT_CONT = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
            PRES_SG = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
            PRES_PL = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
    PRES_PERFECT_SG = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
    PRES_PERFECT_PL = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
       PRES_CONT_SG = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
       PRES_CONT_PL = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
PRES_PERFECT_CONT_SG = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
PRES_PERFECT_CONT_PL = { yesterday, their, me, roses, food, an, grandfather, whereabouts, far, tea, people, player, brother, letter,
```

PARSING TABLE:

| STATE | yesterday | jacksons | their | peacocks | me | homework | roses |
|---|---|---|---|---|---|---|---|
| 0 | s1 | s101 | s3 | s102 | s4 | s103 | s6 |
| 1 | r253 | r253 | r253 | r253 | r253 | r253 | r253 |
| 2 | r52 | r52 |  | r52 |  |  | r52 |
| 3 | r102 | r102 |  | r102 |  |  | r102 |
| 4 | r132 |  |  |  | r132 | r132 |  |
| 5 |  | s101 |  | s102 | s4 | s103 | s6 |
| 6 | r194 | r194 |  | r194 |  |  | r194 |
| 7 | r175 |  |  |  | r175 | r175 |  |
| 8 |  |  |  |  | r216 | r216 |  |
| 9 | r138 |  |  |  | r138 | r138 |  |
| 10 | r259 | r259 | r259 | r259 | r259 | r259 | r259 |
| 11 | r45 | s101 |  | s102 |  |  | s6 |
| 12 | r146 |  |  |  | r146 | r146 |  |
| 13 | r213 | r213 |  | r213 |  |  | r213 |
| 14 | r191 |  |  |  | r191 | r191 |  |
| 15 | r135 |  |  |  | r135 | r135 |  |
| 16 | r127 |  |  |  | r127 | r127 |  |
| 17 | r118 |  |  |  |  |  |  |
| 18 | r129 |  |  |  | r129 | r129 |  |
| 19 | r94 |  |  |  | r94 | r94 |  |
| 20 | r119 |  |  |  |  |  |  |
| 21 | r195 | r195 |  | r195 |  |  | r195 |

## MODULE 3 – Text Preprocessing

Code snippet 7 - English paragraph is read from a text file and is stored as a string.

```
# READING INPUT FROM FILE

inputFile = open('input.txt', 'r')
inputText = inputFile.read()
text = inputText.replace('\n', ' ')
inputFile.close()
text
```

Output snippet 7 - The text in the file is displayed.

```
'She were writing a letter to Ishwarya. My teacher teaches well. She went to school. Raj is climbs a mountian. Raj plays fotball well. Mom cook delisious food. Navvya is a topper. Ishwarya is a kabbadi player. Ishwarya will sings a song in the show. These people is from Canada.'
```

Code snippet 8 – Creation of output file

```
# OUTPUT REPORT

outputLines = []
outputLines.append("ANALYSIS REPORT : \n\n")
outputLines.append("-------------------------------------------------\n\n")
```

Code snippet 9 - The English passage is split into sentences. And the count of sentence is also calculated.

```
# SENTENCE TOKENIZATION

stop_Char=["!",".","?",";"]

sentences=[]

sentence=""
for ch in text :
  if ch not in stop_Char:
    sentence += ch
  else :
    sentence = sentence.strip()
    sentences.append(sentence)
    sentence=""

sentenceCount = len(sentences)
print("Sentence count : " + str(sentenceCount))

for sentence in sentences :
  print(sentence)
```

Output snippet 9 - Sentence count is displayed. The sentences stored in the array is displayed.

```
Sentence count : 10
She were writing a letter to Ishwarya
My teacher teaches well
She went to school
Raj is climbs a mountian
Raj plays fotball well
Mom cook delisious food
Navvya is a topper
Ishwarya is a kabbadi player
Ishwarya will sings a song in the show
These people is from Canada
```

Code snippet 10 - Each Sentence is read. Then each word in that sentence is read. And stop character such as !,&,@ etc is removed.

```
# REMOVAL OF STOP CHARACTERS

reqSplChars = ["@", "&", "(", ")", "-"]
splChars = ["#", "%", "&", "^", "$", "*", "`", "+"]
punctuation = [",", ".", "!"]

exclude = splChars + punctuation
cleanSentences=[]
for sentence in sentences :
  cleantext=""
  for ch in sentence :
    if ch not in exclude:
      cleantext += ch
  cleanSentences.append(cleantext)

for cleanSentence in cleanSentences :
  print(cleanSentence)
```

Output snippet 10 - Sentence count is displayed. Each sentence after removing the stop characters is stored in the array and is displayed.

```
She were writing a letter to Ishwarya
My teacher teaches well
She went to school
Raj is climbs a mountian
Raj plays fotball well
Mom cook delisious food
Navvya is a topper
Ishwarya is a kabbadi player
Ishwarya will sings a song in the show
These people is from Canada
```

Code snippet 11 - Every sentence is tokenized into words. The number of words per sentence and the total number of words in that passage is counted.

```
# WORD TOKENIZATION

wordsArray = []
wordsCount = []
for cleanText in cleanSentences :
  words = []
  word = ""
  i = 0
  while(i < len(cleanText)):
    if(cleanText[i] == " "):
      i += 1
    else:
      word = ""
      while(i < len(cleanText) and cleanText[i] != " "):
        word += cleanText[i]
        i += 1
      if(len(word) > 0):
        words.append(word)
  wordsArray.append(words)
  wordsCount.append(len(words))

totalWordCount = sum(wordsCount)

for i in range(sentenceCount) :
  print(wordsArray[i])
  print("Word count : " + str(wordsCount[i]))
  print()
print("Total word count : " + str(totalWordCount))
```

Output snippet 11 - The tokens are displayed. The number of words per sentence and the total number of words in that passage is displayed.

```
['She', 'were', 'writing', 'a', 'letter', 'to', 'Ishwarya']
Word count : 7

['My', 'teacher', 'teaches', 'well']
Word count : 4

['She', 'went', 'to', 'school']
Word count : 4

['Raj', 'is', 'climbs', 'a', 'mountian']
Word count : 5

['Raj', 'plays', 'fotball', 'well']
Word count : 4

['Mom', 'cook', 'delisious', 'food']
Word count : 4

['Navvya', 'is', 'a', 'topper']
Word count : 4

['Ishwarya', 'is', 'a', 'kabbadi', 'player']
Word count : 5

['Ishwarya', 'will', 'sings', 'a', 'song', 'in', 'the', 'show']
Word count : 8

['These', 'people', 'is', 'from', 'Canada']
Word count : 5

Total word count : 50
```

Code snippet 12 - Every word is checked for its spelling. Misspelled words and its respective best match and possible matches for correct spelling are found. The number of misspelled words in each sentence is counted and each misspelled word is replaced with its corrected spelling.

```
# Spell Check
spell = SpellChecker()
correctedWordsArray = []
misspelledCount = []
misspelledResult = []
for words in wordsArray :
  count = 0
  misspelled = spell.unknown(words)

  if(len(misspelled) > 0):
    print(words)
    for word in misspelled:
        if(word not in properNouns) :
          count += 1
          print("Word : " + word)
          print("Best match : " + str(spell.correction(word)))
          print("Possible matches : " + str(spell.candidates(word)))
          misspelledResult.append("Misspelled word : {}\nBest match : {}\nPossible matches : {}\n\n".format(word, str(spell.correction(word)), str(spell.candidates(word))))
    print("Misspelled word count : " + str(count))
    misspelledCount.append(count)
    print()

    # Correcting the misspelled words
    correctedWords = []
    for word in words :
      if word in misspelled :
        correctedWords.append((spell.correction(word)).lower())
      else :
        correctedWords.append(word.lower())
    correctedWordsArray.append(correctedWords)
  else :
    correctedWords = [word.lower() for word in words]
    correctedWordsArray.append(correctedWords)
    print(words)
    print("Misspelled word count : " + str(len(misspelled)))
    print()
```

Output snippet 12 - The misspelled words and its respective best match and possible matches for correct spelling is displayed. The number of misspelled words in each sentence is displayed.

```
['She', 'were', 'writing', 'a', 'letter', 'to', 'Ishwarya']
Misspelled word count : 0

['My', 'teacher', 'teaches', 'well']
Misspelled word count : 0

['She', 'went', 'to', 'school']
Misspelled word count : 0

['Raj', 'is', 'climbs', 'a', 'mountian']
Word : mountian
Best match : mountain
Possible matches : {'mountain'}
Misspelled word count : 1

['Raj', 'plays', 'fotball', 'well']
Word : fotball
Best match : football
Possible matches : {'football', 'ootball', 'fooball'}
Misspelled word count : 1

['Mom', 'cook', 'delisious', 'food']
Word : delisious
Best match : delicious
Possible matches : {'delicious', 'delirious'}
Misspelled word count : 1

['Navvya', 'is', 'a', 'topper']
Misspelled word count : 0

['Ishwarya', 'is', 'a', 'kabbadi', 'player']
Word : kabbadi
Best match : kabaddi
Possible matches : {'kabaddi'}
Misspelled word count : 1

['Ishwarya', 'will', 'sings', 'a', 'song', 'in', 'the', 'show']
Misspelled word count : 0

['These', 'people', 'is', 'from', 'Canada']
Misspelled word count : 0
```

Code snippet 13 - The total number of misspelled words is counted. The array of words with corrected spelling is stored. The final pre-processed text is ready.

```
totalMispelledCount = sum(misspelledCount)
print("Total misspelled words : " + str(totalMispelledCount))

print("Array of words with correct spelling : ")
for words in correctedWordsArray :
  print(words)
```

Output snippet 13 - The total number of misspelled words is displayed. The array of words with corrected spelling is displayed.

```
Total misspelled words : 4
Array of words with correct spelling :
['she', 'were', 'writing', 'a', 'letter', 'to', 'ishwarya']
['my', 'teacher', 'teaches', 'well']
['she', 'went', 'to', 'school']
['raj', 'is', 'climbs', 'a', 'mountain']
['raj', 'plays', 'football', 'well']
['mom', 'cook', 'delicious', 'food']
['navvya', 'is', 'a', 'topper']
['ishwarya', 'is', 'a', 'kabaddi', 'player']
['ishwarya', 'will', 'sings', 'a', 'song', 'in', 'the', 'show']
['these', 'people', 'is', 'from', 'canada']
```

Code snippet 14 - Writing text processing analysis to output file

```
outputLines.append("Sentence Count : {}\n".format(sentenceCount))
outputLines.append("Word Count : {}\n".format(totalWordCount))
outputLines.append("Misspelled Count : {}\n\n".format(totalMispelledCount))
outputLines.append("---------------------------------------------\n\n")
for result in misspelledResult :
  outputLines.append(result)
outputLines.append("---------------------------------------------\n\n")
```

# Module 4 – Evaluation of Grammar Correctness

Code snippet 15 - Stack implementation algorithm

```
# Stack implementaion of input strings

def LR_parser(slr, w):
  isAccepted = False
  buffer = [x for x in w]
  buffer.append('$')
  pointer = 0
  a = buffer[pointer]
  stack = ['0']
  symbols = ['']
  results = {'step': [''], 'stack': ['STACK'] + stack, 'symbols': ['SYMBOLS'] + symbols, 'input': ['INPUT'], 'action': ['ACTION']}

  step = 0
  while True:
    s = int(stack[-1])
    step += 1
    results['step'].append(f'({step})')
    results['input'].append(' '.join(buffer[pointer:]))

    if a not in slr.parse_table[s]:
      results['action'].append(f'ERROR: unrecognized symbol {a}')
      isAccepted = False
      break
    elif not slr.parse_table[s][a]:
      results['action'].append('ERROR: input cannot be parsed by given grammar')
      isAccepted = False
      break

    elif '/' in slr.parse_table[s][a]:
      action = 'reduce' if slr.parse_table[s][a].count('r') > 1 else 'shift'
      results['action'].append(f'ERROR: {action}-reduce conflict at state {s}, symbol {a}')
      isAccepted = False
      break

    elif slr.parse_table[s][a].startswith('s'):
      results['action'].append('shift')
      stack.append(slr.parse_table[s][a][1:])
      symbols.append(a)
      results['stack'].append(' '.join(stack))
      results['symbols'].append(' '.join(symbols))
      pointer += 1
      a = buffer[pointer]

    elif slr.parse_table[s][a].startswith('r'):
      head, body = slr.G_indexed[int(slr.parse_table[s][a][1:])]
      results['action'].append(f'reduce by {head} -> {" ".join(body)}')
      if body != ('^',):
        stack = stack[:-len(body)]
        symbols = symbols[:-len(body)]

      stack.append(str(slr.parse_table[int(stack[-1])][head]))
      symbols.append(head)
      results['stack'].append(' '.join(stack))
      results['symbols'].append(' '.join(symbols))

    elif slr.parse_table[s][a] == 'acc':
      results['action'].append('accept')
      isAccepted = True
      break
```

```
    # Printing results
    def print_line():
      print(f'{"".join(["+" + ("-" * (max_len + 2)) for max_len in max_lens.values()])}+')

    max_lens = {key: max(len(value) for value in results[key]) for key in results}
    justs = {'step': '>', 'stack': '', 'symbols': '', 'input': '>', 'action': ''}

    print_line()
    print(''.join([f'| {history[0]:^{max_len}} ' for history, max_len in zip(results.values(), max_lens.values())]) + '|')
    print_line()
    for i, step in enumerate(results['step'][:-1], 1):
      print(''.join([f'| {history[i]:{just}{max_len}} ' for history, just, max_len in zip(results.values(), justs.values(), max_lens.values())]) + '|')

    print_line()

    return isAccepted
```

```
incorrectSentenceCount = 0
incorrectSentenceResult = []
for input_text in correctedWordsArray :
  isAccepted = LR_parser(slr, input_text)
  # Performing semantic analysis  using nlp for grammar corrections
  if(not isAccepted) :
    print("Correction : ")
    correction = GingerIt().parse(" ".join(input_text))['result']
    print(correction)
    incorrectSentenceCount += 1
    incorrectSentenceResult.append("Incorrect sentence : {}\nCorrection : {}\n\n".format(" ".join(input_text), correction))
```

Output snippet 15- For incorrect sentence, the stack implementation terminates with error and with the help of semantic analysis correction is provided. For correct sentence, the stack implementation terminates with accept.

```
+-----+-----------+-------------+----------------------------------------+-------------------------------------------------+
|     |   STACK   |   SYMBOLS   |                 INPUT                  |                     ACTION                      |
+-----+-----------+-------------+----------------------------------------+-------------------------------------------------+
| (1) | 0         |             | she were writing a letter to ishwarya $ | shift                                           |
| (2) | 0 19      |  she        |     were writing a letter to ishwarya $ | reduce by PRONOUN_SG -> she                     |
| (3) | 0 100     |  PRONOUN_SG |     were writing a letter to ishwarya $ | reduce by NP_SG -> PRONOUN_SG                   |
| (4) | 0 127     |  NP_SG      |     were writing a letter to ishwarya $ | shift                                           |
| (5) | 0 127 260 |  NP_SG were |          writing a letter to ishwarya $ | ERROR: input cannot be parsed by given grammar |
+-----+-----------+-------------+----------------------------------------+-------------------------------------------------+
Correction :
she was writing a letter to ishwarya
+------+---------------+---------------------+---------------------------+-----------------------------------------+
|      |     STACK     |       SYMBOLS       |           INPUT           |                ACTION                   |
+------+---------------+---------------------+---------------------------+-----------------------------------------+
|  (1) | 0             |                     | my teacher teaches well $ | shift                                   |
|  (2) | 0 163         |  my                 |    teacher teaches well $ | reduce by PRONOUN_SG -> my              |
|  (3) | 0 100         |  PRONOUN_SG         |    teacher teaches well $ | shift                                   |
|  (4) | 0 100 18      |  PRONOUN_SG teacher |           teaches well $ | reduce by NOUN_SG -> teacher            |
|  (5) | 0 100 60      |  PRONOUN_SG NOUN_SG |           teaches well $ | reduce by NOM_SG -> NOUN_SG             |
|  (6) | 0 100 255     |  PRONOUN_SG NOM_SG  |           teaches well $ | reduce by NP_SG -> PRONOUN_SG NOM_SG    |
|  (7) | 0 127         |  NP_SG              |           teaches well $ | shift                                   |
|  (8) | 0 127 267     |  NP_SG teaches      |                  well $ | reduce by V1_SG -> teaches              |
|  (9) | 0 127 263     |  NP_SG V1_SG        |                  well $ | reduce by PRES_SG -> V1_SG              |
| (10) | 0 127 278     |  NP_SG PRES_SG      |                  well $ | shift                                   |
| (11) | 0 127 278 137 |  NP_SG PRES_SG well |                       $ | reduce by ADV -> well                   |
| (12) | 0 127 278 346 |  NP_SG PRES_SG ADV  |                       $ | reduce by S1 -> ADV                     |
| (13) | 0 127 278 397 |  NP_SG PRES_SG S1   |                       $ | reduce by S -> NP_SG PRES_SG S1         |
| (14) | 0 83          |  S                  |                       $ | accept                                  |
+------+---------------+---------------------+---------------------------+-----------------------------------------+
```

| | STACK | SYMBOLS | INPUT | ACTION |
|---|---|---|---|---|
| (1) | 0 | | she went to school $ | shift |
| (2) | 0 19 | she | went to school $ | reduce by PRONOUN_SG -> she |
| (3) | 0 100 | PRONOUN_SG | went to school $ | reduce by NP_SG -> PRONOUN_SG |
| (4) | 0 127 | NP_SG | went to school $ | shift |
| (5) | 0 127 201 | NP_SG went | to school $ | reduce by V2 -> went |
| (6) | 0 127 249 | NP_SG V2 | to school $ | reduce by PAST -> V2 |
| (7) | 0 127 281 | NP_SG PAST | to school $ | shift |
| (8) | 0 127 281 332 | NP_SG PAST to | school $ | reduce by PREPOS -> to |
| (9) | 0 127 281 329 | NP_SG PAST PREPOS | school $ | shift |
| (10) | 0 127 281 329 44 | NP_SG PAST PREPOS school | $ | reduce by NOUN_SG -> school |
| (11) | 0 127 281 329 60 | NP_SG PAST PREPOS NOUN_SG | $ | reduce by NOM_SG -> NOUN_SG |
| (12) | 0 127 281 329 106 | NP_SG PAST PREPOS NOM_SG | $ | reduce by NP_SG -> NOM_SG |
| (13) | 0 127 281 329 412 | NP_SG PAST PREPOS NP_SG | $ | reduce by PP -> PREPOS NP_SG |
| (14) | 0 127 281 337 | NP_SG PAST PP | $ | reduce by NP -> PP |
| (15) | 0 127 281 331 | NP_SG PAST NP | $ | reduce by S1 -> NP |
| (16) | 0 127 281 399 | NP_SG PAST S1 | $ | reduce by S -> NP_SG PAST S1 |
| (17) | 0 83 | S | $ | accept |

| | STACK | SYMBOLS | INPUT | ACTION |
|---|---|---|---|---|
| (1) | 0 | | raj is climbs a mountain $ | shift |
| (2) | 0 124 | raj | is climbs a mountain $ | reduce by PROPERNOUN -> raj |
| (3) | 0 50 | PROPERNOUN | is climbs a mountain $ | reduce by NP_SG -> PROPERNOUN |
| (4) | 0 127 | NP_SG | is climbs a mountain $ | shift |
| (5) | 0 127 282 | NP_SG is | climbs a mountain $ | ERROR: input cannot be parsed by given grammar |

Correction :
raj climbs a mountain

| | STACK | SYMBOLS | INPUT | ACTION |
|---|---|---|---|---|
| (1) | 0 | | raj plays football well $ | shift |
| (2) | 0 124 | raj | plays football well $ | reduce by PROPERNOUN -> raj |
| (3) | 0 50 | PROPERNOUN | plays football well $ | reduce by NP_SG -> PROPERNOUN |
| (4) | 0 127 | NP_SG | plays football well $ | shift |
| (5) | 0 127 269 | NP_SG plays | football well $ | reduce by V1_SG -> plays |
| (6) | 0 127 263 | NP_SG V1_SG | football well $ | reduce by PRES_SG -> V1_SG |
| (7) | 0 127 278 | NP_SG PRES_SG | football well $ | shift |
| (8) | 0 127 278 32 | NP_SG PRES_SG football | well $ | reduce by NOUN_SG -> football |
| (9) | 0 127 278 60 | NP_SG PRES_SG NOUN_SG | well $ | reduce by NOM_SG -> NOUN_SG |
| (10) | 0 127 278 106 | NP_SG PRES_SG NOM_SG | well $ | reduce by NP_SG -> NOM_SG |
| (11) | 0 127 278 341 | NP_SG PRES_SG NP_SG | well $ | reduce by NP -> NP_SG |
| (12) | 0 127 278 331 | NP_SG PRES_SG NP | well $ | reduce by S1 -> NP |
| (13) | 0 127 278 397 | NP_SG PRES_SG S1 | well $ | reduce by S -> NP_SG PRES_SG S1 |
| (14) | 0 83 | S | well $ | shift |
| (15) | 0 83 137 | S well | $ | reduce by ADV -> well |
| (16) | 0 83 253 | S ADV | $ | reduce by S -> S ADV |
| (17) | 0 83 | S | $ | accept |

| | STACK | SYMBOLS | INPUT | ACTION |
|---|---|---|---|---|
| (1) | 0 | | mom cook delicious food $ | shift |
| (2) | 0 146 | mom | cook delicious food $ | reduce by NOUN_SG -> mom |
| (3) | 0 60 | NOUN_SG | cook delicious food $ | reduce by NOM_SG -> NOUN_SG |
| (4) | 0 106 | NOM_SG | cook delicious food $ | reduce by NP_SG -> NOM_SG |
| (5) | 0 127 | NP_SG | cook delicious food $ | ERROR: input cannot be parsed by given grammar |

Correction :
mom cooks delicious food

```
+------+------------------+-----------------------------+---------------------+--------------------------------+
|      |       STACK      |           SYMBOLS           |        INPUT        |             ACTION             |
+------+------------------+-----------------------------+---------------------+--------------------------------+
|  (1) | 0                |                             | navvya is a topper $ | shift                          |
|  (2) | 0 89             |  navvya                     |        is a topper $ | reduce by PROPERNOUN -> navvya |
|  (3) | 0 50             |  PROPERNOUN                 |        is a topper $ | reduce by NP_SG -> PROPERNOUN  |
|  (4) | 0 127            |  NP_SG                      |        is a topper $ | shift                          |
|  (5) | 0 127 282        |  NP_SG is                   |           a topper $ | reduce by V1_SG -> is          |
|  (6) | 0 127 263        |  NP_SG V1_SG                |           a topper $ | reduce by PRES_SG -> V1_SG     |
|  (7) | 0 127 278        |  NP_SG PRES_SG              |           a topper $ | shift                          |
|  (8) | 0 127 278 125    |  NP_SG PRES_SG a            |             topper $ | reduce by DET_SG -> a          |
|  (9) | 0 127 278 114    |  NP_SG PRES_SG DET_SG       |             topper $ | shift                          |
| (10) | 0 127 278 114 172 |  NP_SG PRES_SG DET_SG topper |                    $ | reduce by NOUN_SG -> topper    |
| (11) | 0 127 278 114 60 |  NP_SG PRES_SG DET_SG NOUN_SG |                    $ | reduce by NOM_SG -> NOUN_SG    |
| (12) | 0 127 278 114 258 |  NP_SG PRES_SG DET_SG NOM_SG |                    $ | reduce by NP_SG -> DET_SG NOM_SG |
| (13) | 0 127 278 341    |  NP_SG PRES_SG NP_SG        |                    $ | reduce by NP -> NP_SG          |
| (14) | 0 127 278 331    |  NP_SG PRES_SG NP           |                    $ | reduce by S1 -> NP             |
| (15) | 0 127 278 397    |  NP_SG PRES_SG S1           |                    $ | reduce by S -> NP_SG PRES_SG S1 |
| (16) | 0 83             |  S                          |                    $ | accept                         |
+------+------------------+-----------------------------+---------------------+--------------------------------+
```

```
+------+---------------------+-----------------------------+-----------------------------+--------------------------------+
|      |        STACK        |           SYMBOLS           |            INPUT            |             ACTION             |
+------+---------------------+-----------------------------+-----------------------------+--------------------------------+
|  (1) | 0                   |                             | ishwarya is a kabaddi player $ | shift                          |
|  (2) | 0 168               |  ishwarya                   |          is a kabaddi player $ | reduce by PROPERNOUN -> ishwarya |
|  (3) | 0 50                |  PROPERNOUN                 |          is a kabaddi player $ | reduce by NP_SG -> PROPERNOUN  |
|  (4) | 0 127               |  NP_SG                      |          is a kabaddi player $ | shift                          |
|  (5) | 0 127 282           |  NP_SG is                   |             a kabaddi player $ | reduce by V1_SG -> is          |
|  (6) | 0 127 263           |  NP_SG V1_SG                |             a kabaddi player $ | reduce by PRES_SG -> V1_SG     |
|  (7) | 0 127 278           |  NP_SG PRES_SG              |             a kabaddi player $ | shift                          |
|  (8) | 0 127 278 125       |  NP_SG PRES_SG a            |               kabaddi player $ | reduce by DET_SG -> a          |
|  (9) | 0 127 278 114       |  NP_SG PRES_SG DET_SG       |               kabaddi player $ | shift                          |
| (10) | 0 127 278 114 178   |  NP_SG PRES_SG DET_SG kabaddi |                      player $ | reduce by NOUN_SG -> kabaddi   |
| (11) | 0 127 278 114 60    |  NP_SG PRES_SG DET_SG NOUN_SG |                      player $ | reduce by NOM_SG -> NOUN_SG    |
| (12) | 0 127 278 114 258   |  NP_SG PRES_SG DET_SG NOM_SG |                      player $ | shift                          |
| (13) | 0 127 278 114 258 14 |  NP_SG PRES_SG DET_SG NOM_SG player |                             $ | reduce by NOUN_SG -> player    |
| (14) | 0 127 278 114 258 256 |  NP_SG PRES_SG DET_SG NOM_SG NOUN_SG |                             $ | reduce by NOM_SG -> NOM_SG NOUN_SG |
| (15) | 0 127 278 114 258   |  NP_SG PRES_SG DET_SG NOM_SG |                             $ | reduce by NP_SG -> DET_SG NOM_SG |
| (16) | 0 127 278 341       |  NP_SG PRES_SG NP_SG        |                             $ | reduce by NP -> NP_SG          |
| (17) | 0 127 278 331       |  NP_SG PRES_SG NP           |                             $ | reduce by S1 -> NP             |
| (18) | 0 127 278 397       |  NP_SG PRES_SG S1           |                             $ | reduce by S -> NP_SG PRES_SG S1 |
| (19) | 0 83                |  S                          |                             $ | accept                         |
+------+---------------------+-----------------------------+-----------------------------+--------------------------------+
```

```
+-----+----------+------------+-----------------------------------+-----------------------------------------------+
|     |  STACK   |  SYMBOLS   |               INPUT               |                    ACTION                     |
+-----+----------+------------+-----------------------------------+-----------------------------------------------+
| (1) | 0        |            | ishwarya will sings a song in the show $ | shift                                         |
| (2) | 0 168    |  ishwarya  |         will sings a song in the show $ | reduce by PROPERNOUN -> ishwarya              |
| (3) | 0 50     |  PROPERNOUN |         will sings a song in the show $ | reduce by NP_SG -> PROPERNOUN                 |
| (4) | 0 127    |  NP_SG     |         will sings a song in the show $ | shift                                         |
| (5) | 0 127 241 |  NP_SG will |              sings a song in the show $ | ERROR: input cannot be parsed by given grammar |
+-----+----------+------------+-----------------------------------+-----------------------------------------------+
Correction :
Ishwarya will sing a song in the show
```

```
+-----+-----------+-----------------+-------------------------------+-----------------------------------------------+
|     |   STACK   |     SYMBOLS     |             INPUT             |                    ACTION                     |
+-----+-----------+-----------------+-------------------------------+-----------------------------------------------+
| (1) | 0         |                 | these people is from canada $ | shift                                         |
| (2) | 0 40      |  these          |       people is from canada $ | reduce by DET_PL -> these                     |
| (3) | 0 107     |  DET_PL         |       people is from canada $ | shift                                         |
| (4) | 0 107 13  |  DET_PL people  |              is from canada $ | reduce by NOUN_PL -> people                   |
| (5) | 0 107 2   |  DET_PL NOUN_PL |              is from canada $ | reduce by NOM_PL -> NOUN_PL                   |
| (6) | 0 107 257 |  DET_PL NOM_PL  |              is from canada $ | reduce by NP_PL -> DET_PL NOM_PL              |
| (7) | 0 66      |  NP_PL          |              is from canada $ | ERROR: input cannot be parsed by given grammar |
+-----+-----------+-----------------+-------------------------------+-----------------------------------------------+
Correction :
these people are from Canada
```

## Code snippet 16 – Writing results to output file

```
outputLines.append("Incorrect sentences count : {}\n\n".format(incorrectSentenceCount))
outputLines.append("-----------------------------------------------\n\n")
for result in incorrectSentenceResult :
  outputLines.append(result)
outputLines.append("-----------------------------------------------\n\n")
```

```
# WRITING RESULT TO THE OUTPUT FILE

outputFile = open("output.txt", "w")
for line in outputLines:
  outputFile.write(line)
outputFile.close()
```

INPUT.TXT

input.txt

```
She were writing a letter to Ishwarya.
My teacher teaches well.
She went to school.
Raj is climbs a mountian.
Raj plays fotball well.
Mom cook delisious food.
Navvya is a topper.
Ishwarya is a kabbadi player.
Ishwarya will sings a song in the show.
These people is from Canada.
```

OUTPUT.TXT

output.txt

```
ANALYSIS REPORT :

-----------------------------------------------

Sentence Count : 10
Word Count : 50
Misspelled Count : 4

-----------------------------------------------

Misspelled word : mountian
Best match : mountain
Possible matches : {'mountain'}

Misspelled word : fotball
Best match : football
Possible matches : {'football', 'ootball', 'fooball'}

Misspelled word : delisious
Best match : delicious
Possible matches : {'delicious', 'delirious'}

Misspelled word : kabbadi
Best match : kabaddi
Possible matches : {'kabaddi'}

-----------------------------------------------

Incorrect sentences count : 5

-----------------------------------------------

Incorrect sentence : she were writing a letter to ishwarya
Correction : she was writing a letter to ishwarya

Incorrect sentence : raj is climbs a mountain
Correction : raj climbs a mountain

Incorrect sentence : mom cook delicious food
Correction : mom cooks delicious food

Incorrect sentence : ishwarya will sings a song in the show
Correction : Ishwarya will sing a song in the show

Incorrect sentence : these people is from canada
Correction : these people are from Canada

-----------------------------------------------
```

## 9. CONCLUSION

In this project we have used the parsing technique called SLR (1) for parsing natural language. SLR (1) is implemented in 'Python' Programming Language. Considering File containing English text as an input, Stack implementation for the input sentence is performed to evaluate the grammar correctness. If the sentence is grammatically incorrect, semantic analysis using NLP is performed to provide suggestions. Report file comprising word count, accuracy percentage, spelling mistakes and grammatically incorrect sentences with their corrections is produced for the output. Experimental results have been presented here.

## REFERENCES


[1] Chapter 4 of "Compilers: Principles, Techniques, & Tools" 2nd Edition by Ravi Sethi, Alfred V. Aho, Monica S. Lam, D. Jeffrey Ulman

[2] Suvarna G Kanakaraddi, V Ramaswamy, Natural Language Parsing using Fuzzy Simple LR (FSLR) Parser, 978-1-4799-2572-8/14 2014 IEEE

[3] Chapter 12 of "An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition", by Daniel Jurafsky and James H. Martin

[4] https://leverageedu.com/blog/tenses-rule/

[5] https://towardsdatascience.com/introduction-to-natural-language-processing-for-text-df845750fb63