Github Link: https://github.com/edwardinio18/LFTC/tree/main/Labs/Lab7Final

DOCUMENTATION PARSER

Grammar:

- self.N = []: set of non-terminals
- self.E = []: set of terminals
- self.S = "": starting symbol
- self.P = {}: productions

1. *__process_line(line: str, delimiter=' '):*
- This is a private static method used to process a line from a file by splitting it into elements using a specified delimiter.
2. *read_from_file(self, file_name: str):*
- Reads context-free grammar information from a file and populates the object's attributes with the parsed data.
3. *check_cfg(self):*
- Checks if the provided context-free grammar is valid. It ensures that it has a starting symbol, all non-terminals are declared in N, and all symbols used in productions are either in N or E.
4. *get_non_terminals(self):*
- Returns the set of non-terminals (N) in the context-free grammar.
5. *get_terminals(self):*
- Returns the set of terminals (E) in the context-free grammar.
6. *get_start_symbol(self):*
Returns the starting symbol (S) of the context-free grammar.
7. *get_productions(self):*
- Returns all the production rules (P) in the context-free grammar.
8. *get_productions_for_non_terminal(self, nt):*
- Returns the production rules associated with a specific non-terminal (nt) in the context-free grammar.

Parser:

The Parser class is designed to perform parsing of input sequences using a recursive descent parsing strategy based on a given context-free grammar. Recursive descent parsing is a top-down parsing technique where the parsing process starts at the highest-level grammar rule and recursively explores the production rules to match the input sequence.

1. *__init__(self, grammar, sequence_file, out_file):*
- Initializes the Parser object with the provided grammar, sequence file, and output file. It sets up the initial state, input, and working stack for parsing.
2. *read_sequence(self, seq_file):*
- Reads the input sequence from a file and prepares it for parsing. It also creates or clears the output file for logging.
3. *get_situation(self):*

- Logs the current parsing situation, including state, index, working stack, and input, to the output file and prints it to the console.

4. expand(self):
- Implements the "expand" parsing action, which expands a non-terminal in the input using a production rule from the grammar.
- (q,i, α, Aβ) ⊢ (q,i, $\alpha A_1 \gamma_1 \beta$)

5. *advance(self):*
- Implements the "advance" parsing action, which advances the parsing index when a terminal symbol in the input matches the sequence.
- (q,i, α, $a_i\beta$) ⊢ (q,i+1, αa_i, β)

6. *momentary_insuccess(self):*
- Implements the "momentary insuccess" parsing action, indicating a temporary parsing failure.
- (q,i, α, $a_i\beta$) ⊢ (b,i, α, $a_i\beta$)

7. *back(self):*
- Implements the "back" parsing action, which backtracks by undoing the last parsing action.
- (b,i, αa, β) ⊢ (b,i-1, α, aβ)

8. *success(self):*
- Implements the "success" parsing action, indicating successful parsing of the input sequence.
- (q,n+1, α, ε) ⊢ (f,n+1, α, ε)

9. *another_try(self):*
- Implements the "another try" parsing action, which attempts to use an alternative production for a non-terminal or backtracks if no alternatives are available.
- (b,i,$\alpha A_j, \gamma_j \beta$)⊢ (q,i,$\alpha A_{j+1}, \gamma_{j+1} \beta$), if∃A→$\gamma_{j+1}$
  (b,i, α, A β), otherwise with the exception
  (e,i, α, β), if i=1, A =S, **ERROR**

10. *error(self):*
- Implements the "error" parsing action, indicating an error state when no more input can be processed.
- (e,i, α, β)

11. *run(self):*
- Runs the parsing process until a final state is reached (either success or error). It repeatedly calls parsing actions based on the current state.

ParserOutput:

The "father-sibling" table is a data structure used in the context of a recursive descent parser to represent the parsing tree's hierarchical structure. This table helps maintain relationships between nodes in the parsing tree, allowing the parser to traverse and construct the tree as it recognizes the input language.

1. *__init__(self, grammar, sequence_file, out_file):*

- Initializes the ParserOutput object with the provided grammar, sequence file, and output file. It prepares for generating and displaying the parsing tree.

2. *read_sequence(self, seq_file):*

- Reads the input sequence from a file and prepares it for creating the parsing tree.

3. *create_parsing_tree(self, working):*

- Generates a parsing tree based on the working stack used during parsing. This function creates nodes and establishes relationships between them to represent the parsing tree structure.

4. *get_len_depth(self, index, working):*

- Calculates the length and depth of a subtree rooted at the given index in the parsing tree. This is used to determine the offset for sibling nodes.

5. *write_parsing_tree(self, state, working):*

- Writes the parsing tree information to the output file and prints it to the console. It includes the index, value, father node, and sibling node information for each node in the parsing tree.