Github: https://github.com/cs-ubbcluj-ro/lab-work-computer-science-2024-iuliamariagroza/tree/main/5-Parser

Documentation for Parsing Algorithm Implementation

Task Requirements

Statement

1. **Inputs**:
    - g1.txt and seq.txt
    - g2.txt and PIF.out (result from Lab 3)
2. **Outputs**:
    - out1.txt: Parsing results for g1.txt with seq.txt
    - out2.txt: Parsing results for g2.txt with PIF.out
    - Detailed error messages in case of syntax conflicts or errors, specifying the location when possible.

Components and Files

Source Code

1. **Symbol.py**:
    - Defines the Symbol class used to represent nodes in the parse tree.
    - Attributes:
        - value: The symbol's value (terminal or non-terminal).
        - father: Parent node in the parse tree.
        - sibling: Left sibling in the parse tree.
        - production: The production number used to derive this symbol.
2. **Grammar.py**:
    - Manages grammar definitions and provides helper functions to parse and validate grammar files.
    - Key Methods:
        - load_grammar(file_path): Reads grammar definitions from a file.
        - productions_for(non_terminal): Fetches production rules for a given non-terminal.
        - has_additional_production(non_terminal, production_number): Checks for alternative productions.
        - specific_production(non_terminal, production_number): Retrieves a specific production rule.

3. **Parser.py**:
    - Implements the descendant recursive parsing strategy.
    - Key Features:
        - States (q, b, f, e) manage normal, backtracking, success, and error scenarios.
        - Handles operations like **expand**, **advance**, **back**, and **anotherTry**.
        - Generates parsing results and handles syntax conflicts/errors.
4. **PrintParser.py**:
    - Generates a human-readable parse tree from the parsing process.
    - Outputs the tree to a file in a tabular format using the tabulate library.
5. **main.py**:
    - Entry point for the program.
    - Initializes grammar, parser, and outputs results to files.

Input Files

1. **g1.txt**: Defines a grammar with non-terminals, terminals, start symbol, and production rules.
2. **g2.txt**: Contains a more complex grammar for structured programming constructs.
3. **seq.txt**: A sequence of terminals (a, b) to parse using g1.txt.
4. **PIF.out**: A tokenized sequence generated in a previous lab, used for parsing with g2.txt.

Outputs

1. **Parsing Results**:
    - out1.txt: Contains parsing results for g1.txt and seq.txt.
    - out2.txt: Contains parsing results for g2.txt and PIF.out.
2. **Parse Tree**:
    - Generated as tree.txt, showing the hierarchical structure of the parsed input.
3. **Error Messages**:
    - Detailed messages are included in the output files if conflicts or syntax errors occur. These include:
        - **Location of the error**: Index in the input sequence.
        - **Description of the issue**: Conflict or unexpected token.

Example Workflow

Input (g1.txt):

N = S, A

E = a, b

S = S

P =

S -> a$A

A -> a$A | b

Sequence (seq.txt):

a

a

b

Output (out1.txt):

Sequence ['a', 'a', 'b'] is accepted!

Parse tree generated in tree.txt

Parse Tree (tree.txt):

| Index | Value | Parent | Left Sibling |

|-------|-------|--------|--------------|

| 0 | S | -1 | -1 |

| 1 | a | 0 | -1 |

| 2 | A | 0 | 1 |

| 3 | a | 2 | -1 |

| 4 | A | 2 | 3 |

| 5 | b | 4 | -1 |

Error Handling

1. **Conflict Detection**:
    - When a non-terminal has multiple valid productions but the input does not match any, the parser outputs:
    - Conflict at index [position]: Multiple productions possible for [non-terminal].
2. **Syntax Errors**:
    - If the input cannot be derived from the grammar:
    - Syntax error at index [position]: Unexpected token [token].