CPSC 323 Documentation

1. Problem Statement

The main goal of this project is to construct a parser that can recognize the syntax of a given programming language based on the rules defined by the context-free grammar. This parser should be able to efficiently and accurately parse the input source code and produce an error-free parse tree. The project will require a thorough understanding of parsing algorithms, lexical analysis, syntax analysis, and compiler construction concepts. The successful completion of this project will demonstrate the ability to develop a reliable and efficient parser for a programming language, which is a crucial component of any compiler or interpreter.

2. How to Use the Program

Dependencies and Environment

- OS: Mac OS || Linux
- Compiler: LLVM Clang

Installing

- Download repository here

How to run the program

- Open the command line of your OS
- Change into downloaded repository directory
- Execute the folliwng command:
  - ./ run_parser <input_file>
- **NOTE**: the <input_file> MUST be one of the following:
  - input_01
  - input_02
  - input_03

3. Design of your program

The main goal of this program is to implement a table-driven predictive parser for the given context-free grammar (CFG) using a 3D vector table, two unordered maps, and a stack. The program takes an input string, performs lexical analysis using a previously implemented lexer, and then passes the resulting list of Tokens to the parser. The parser then uses the table, maps, and stack to parse the input string and determine if it is syntactically valid according to the CFG.

The first step in the program design is to initialize the 3D vector table. This table isused to store the productions of the CFG in a format that can be easily accessed by the parser

during parsing. The table has three dimensions: rows, columns, and a third dimension that holds the right-hand side of each production as a string separated by terminals. The table is manually populated from the information given for the project.

The next step is to initialize two unordered maps: one mapping each terminal symbol to its corresponding column index in the table, and another mapping each non-terminal symbol to its corresponding row index in the table. These maps will be used by the parser to quickly access the appropriate cells in the table during parsing.

The parser itself uses a stack to keep track of the symbols that it is currently processing. The stack is initially populated with the start symbol of the CFG. During parsing, the program pops symbols from the top of the stack and performs one of two actions depending on whether the symbol is a non-terminal or a terminal. If the symbol is a non-terminal, the program looks up the corresponding row in the table using the non-terminal to row index map, and the corresponding column using the terminal to column index map based on the next token. If the table cell contains a production, the program pushes the symbols from the production RHS onto the stack in reverse order. If the table cell is empty, the parser should report a syntax error. If the symbol is a terminal, the program compares it with the next token. If they match, the program pops the token and continues parsing. If they do not match, the parser should report a syntax error.

Overall, the design of the program follows a clear and logical progression from initializing the data structures to parsing the input string. The use of a table, maps, and stack allows for efficient and accurate parsing of the input string according to the rules of the CFG.

4. Any Limitation

None.

5. Any shortcomings

None.