David Nguyen

Professor Anthony Le

CPSC 323-04

27 September 2019

Lexical Analyzer Outline

1. Problem Statement

The goal is to build a lexical analyzer (lexer) from scratch. First, we implement an FSM by determining the regular expression. Based on the regular expression, we can make an NFSM by drawing out the NFA diagram using Thompson's Principle. Since we use Thompson's Principle for the layout, there are epsilon that signals an absence of input and spontaneous transitions to other states. Now we can create an NFA table with epsilons. The epsilons will be used to determine the DFA. Then, we will use the e-closure from each state in the NFA to build a DFA. The e-closure consists of the current state and the set of epsilons. Next, the first e-closure will be the first subset to be used to determine other subsets for the input characters (Subset Method). Since we converted the NFA to a DFA, the DFA table and diagram does not contain any more epsilons. The subsets in the DFA table can be relabeled to a new state so that it can be used as a reference to draw a DFA diagram. Also, the DFA table is a reference for a lookup table for state traversal. The DFA diagram shows the state traversal process and how a token can be determined. The lexer function can look up the column in the state table to determine its state. The lexer function act as a token parser. When the lexer function traverses on a final state, it returns a token for that lexeme.

2. How to use your program

CLIENTS

The lexer.rar must be downloaded first on a Windows/Linux operating system. Then the user must extract the folder to any destination. Next, the user must navigate towards the Debug folder and click on DavidNguyen_lexer_analyzer.exe to execute the program. The executable file is in this address \lexer\DavidNguyen_lexer_analyzer\Debug. The program will run its task and create or modify the lexer_output.txt file. The output file will show all the parsed tokens and lexemes.

SOURCE CODE

After downloading and extracting the lexer.rar file, navigate towards the source code folder at this address \lexer\DavidNguyen_lexer_analyzer\DavidNguyen_lexer_analyzer. All of the source code to compile and build an executable file should be on here. It is recommended to use Visual Studio to open the visual studio file (.vcxproj) that contains all the source code. On Visual Studio, to start my program, you can either set up breakpoints to debug my code (F5) or start without debugging (CTRL + F5). The program will compile, execute, and write to an output file.

3. Design of your program

The program consists of three sources files (Lexer_analysis header, implementation, and the main file). Four test text files are used to parse lexemes into tokens.

Lexer_analysis_h.h:

I used OOP methodologies to make it convenient for the client to access the functions. The client can access the readFile() and the writeFile() function. However, only within the *lexer_analysis* class can access the lexer(), insert_token(), and char_to_col() functions. There are class data structures and members like a 2-D lookup table for DFA states, a list of token and lexeme pairs, a sorted lookup table for keywords, and the state variable. I used a type alias to “nickname” the pair type name pair<string, string>. The state variable will be used to keep track of what state it is currently on.

Lexer_analysis.cpp:

This file is an implementation file where I defined the functions that I declared in the header file. The createline() function is a helper function for writeFile() function. The createline() function creates a line of hyphens in the output stream. The readFile() function reads the input file and parse the input file until the end of the file. The writeFile() function writes the parsed tokens to an output file. The char_to_col() function acts as a helper function for the lexer(). After encountering a char from a line of string, the char will be evaluated to see if any of the conditions match a column in a state table, then it will return the matched column (integer). The lexer() function parse the line of string using the char_to_col() function to traverse the state table. The time complexity for the lexer() function is O(N). The lexer() function will use an insert_token() function to insert the token and lexeme into a vector. The insert_token() function is a helper function to insert tokens into a vector. For the keyword search in the insert_token(), I was able to optimize my linear search from O(N) to O(log(N)) binary search.

main.cpp:

The main file is used as a way for the client instantiate a Lexer_analysis object and to call the object functions. The object allows the client to retrieve tokens and lexemes from the output file.

4. Any Limitations

I was unable to create a proper comment parser for C++ language due to lack of knowledge.

5. Any shortcomings

None