CPSC 323

Parser Documentation

Mario Pinzon
Adam Secrest

1. **Problem Statement**
   Given the output of the Lexical Analyzer, our goal is to create a Syntax Analyzer, or parser, that defines and evaluates the grammar of Rat25s. We need to:
   - Rewrite the Rat25S grammar to remove left recursion and apply left factoring
   - Use the Lexer to tokenize a given input
   - Implement a top-down parsing approach
   - Output tokens, lexemes, and parse tree evaluated by the program.
   - Include error handling to report meaningful syntax errors in the input

2. **How to use your program**

Compile: g++ main.cpp Parser.cpp Lexer.cpp -o parser
   or: g++ -std=c++17 main.cpp Parser.cpp Lexer.cpp -o program

Run with: ./program input.txt

with input.txt being the Rat25S code

3. **Design of your program**
   The major components of our program are:
   Lexer – tokenizes input and feeds it to the parser
   Parser – Analyzes the stream top-down
   Main – Handles the input / output to each file

   Key data structures:
   CurrentToken – A pointer to the current token being evaluated on the stack

   Std::stack<std::variant<std::string, TokenType>> parserStack – Stack to track total / remaining Tokens and expressions to be evaluated

Bool Match() - Checks that the current / next token type is the same as what is expected and throws an error otherwise.

The algorithm used is Predictive Parsing, utilizing a stack-based FSM to predict the possible next values of tokens and then evaluate whether or not the next token is a legal / expected value.

Left recursion will need to have been removed from the rules before coding, which it is.

Error detection occurs whenever there is an unexpected token value or a certain token is expected but not present.

4. **Any Limitation**
   None

5. **Any shortcomings**
   None