Jacob Nguyen, James Nguyen, and Michael Lim

CPSC 323-05: Compilers and Languages

Negin Ashrafi

5/7/2025

Python Lexer Program

This program is a top-down predictive parser, it validates strings based on the given context-free grammar using an LL(1) parsing table. It simulates the parsing process by taking in and reading input strings from left to right and using a stack to track grammar rule applications. The parser checks whether the given input strings are accepted by the grammar and also provides detailed tracing of each step, including stack changes and rule applications. The time complexity of the parser is O(n) because n is the length of the input string, since each character is processed only once during matching or rule application. Space complexity is also O(n) because the stack and input buffer both grow linearly with input size.

Code Discussion

In lines 4 - 10 created a nested dictionary titled *parse_table* where each non-terminal symbol maps to dictionaries of terminals each pointing to a production rule. Lines 12 - 13 created the *terminal* and *non_terminal* arrays. Lines 15 - 77 defines the *predictive_parser* function that takes in an input string. Line 20 - 23 is an edge case that checks if the input string ends with '$', if it does, prints an error message and returns false. Lines 26 -28 initialize the stack starting with '$' and 'E', sets the *input_buffer* to *input_string* as a list, and sets *pointer* to 0. Line 32 creates a while loop that will run until the stack is empty. Lines 34 and 36 look at the last value in the list, and set *current_input* to *input_buffer*[*pointer*]. Lines 41 - 45 check if the *top* matches *current_input* if it does, removes the top element from stack and increments the pointer. In lines 46 -48, if top doesn't equal *current_input* prints an error message and returns false. Lines 49 - 58 checks if top is in an array of non terminals by initializing *production* to the rule based on current input, if it fails returns false. Lines 59 - 62 return false if there is no parse table entry for given values. Lines 65 - 67 check if stack is empty and input is not fully processed return false. Lines 70 - 77 check the stack if its empty return true. If the parsing is finished but input not fully consumed return false. In lines 82 - 98 creates a main function that tests input strings. Line 88 loops through the test strings and assigns *is_accepted* to the value of the function *predictive_parser* given the input string. Lines 92 - 95 display output, checks if *is_accepted* is true prints accept statement, else prints error statement.