Project 3: Gold Lexer and Parser

Profesor: Silvia Takahashi

Course: Lenguajes y Maquinas (Ingles)

Section: 1

Daniela Uribe 201923291
Juan Felipe Patiño 201922857

21 Noviembre 2022

Abstract:

Using the language Gold create a program that does syntactical analysis for the Robot. It contains a lexer and parser. The lexer is a finite state transducer while the parser is a pushdown automaton.

Key Words: *Gold, Lexer, Finite State Transducer, Parser, Pushdown Automata*

1. Introduction:

The purpose was to utilize the programming language of Gold to create a finite state transducer lexer and an accompanying pushdown automaton parser. It receives through console the complete code to revise, and it goes through the lexer first and then through the parser, for it to be accepted or rejected. It follows the robot rules which have previously been used in other projects, and in this case doesn't implement any actions, but simply checks the validity. It is worth noting that this lexer-parser does not verify that functions, variables or parameters and declared beforehand but merely that their syntax is correct.

2. Materials and method

a. Lexer

For the first part we created a lexer by modifying the initialize method. As a finite state transducer its purpose was to tokenize all the reserved words of the Robot language, while also identifying invalid cases if they don't fit into any of the cases. The input was al the characters from 'a' to 'z' in both lower and uppercase, along with the Arabic numbers and the space and line break. The keywords were all possible functions, the directional inputs, and the keywords form the language. Also, the all-bracket types, semicolons, commas, equals and colon, were left the same as valid inputs. For the keywords each one was tokenized to a different letter of the alphabet, however some were grouped together. In the case of jump, drop, grab, get, free and pop all were replaced by the same character as their inputs are the same and their possible uses are also interchangeable so for a syntactical analysis it doesn't matter which one it is. The other case of grouping came from directional arguments, where the four cardinal directions were grouped together, left and right were another group and front and back were also grouped together. Finally, all numbers were replaced by the octothorpe, while any string that wasn't tokenized before was replaced by an asterisk, which represents user created functions, or variables, or parameters.

b. Parser

For the second part of the parser the choice was made to mix many states with the stack, to create a robust parser. For that purpose, we created a healthy number of states, with transitions based on the tokenization from the lexer and using the stack sometimes. The stack served a couple of purposes, the first and most important was to encapsulate the whole automaton with pushing and popping the '$' symbol to ensure everything was done correctly. Similarly for the main structure of states and transitions it was used to identify when a part of the code is the body of a specific control structure, or if the code is all part of the final block of instructions that would be executed. The other use was with controlling commas, to allow an infinite number of parameters within a function. As for the overarching idea of states, we created the key states from 0 to 7 for the main steps of initializing, variables, procedures, end of procedures, start of instructions, end of instructions, and end of the whole transducer. This main overarching idea can be seen in figure 1.0.

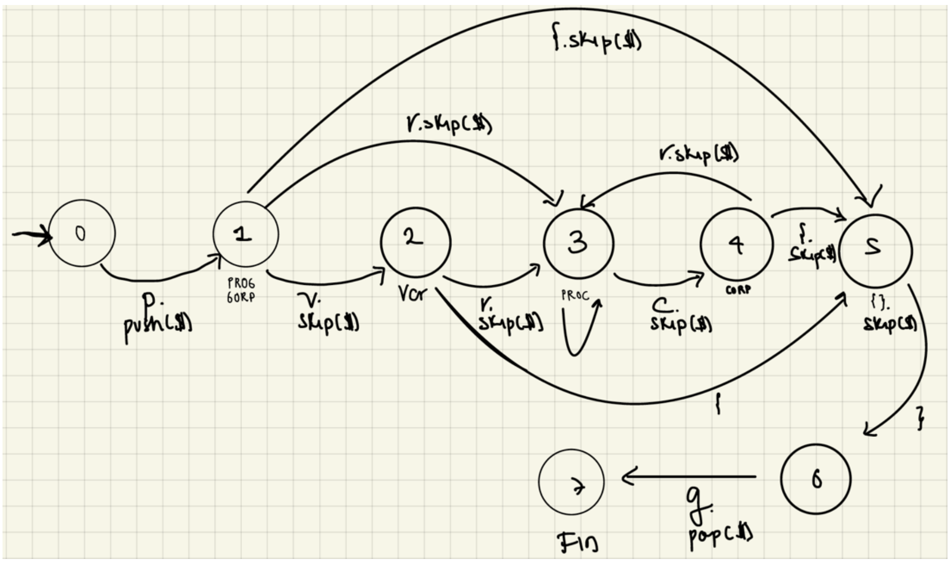

Figure 1.0.

For the minutia we have the figure 2.0 detailing the states involved in reading variables and accepting them.

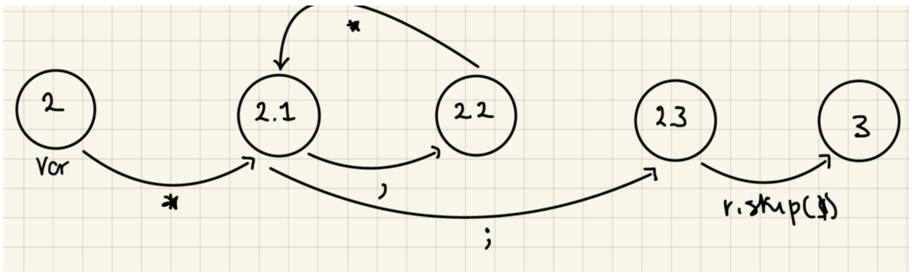

Figure 2.0.

Meanwhile figure 3.0 is the largest part as it interprets instructions. This may be inside a procedure block, or a control structure, or the final instructions, or any mix of these, including nested control structures. The states from 3 to 3.4 receive the function name and parameters of a user created functions. It is worth noting that state 3.4 serves as the start of instructions and is the hub of new outgoing lines of code. Meanwhile 3.5 to 3.10 accepts assignments or invoking other user created methods. From 3.11 to 3.15 we have the control structure declarations with their validation of conditions. Then, 3.16 to 3.18 serves for dealing with end of instructions blocks and redirecting to the appropriate state, whether it be within a control structure, or if its procedure, or the final instructions. In this last case we also deal with reaching the final state upon an empty stack.

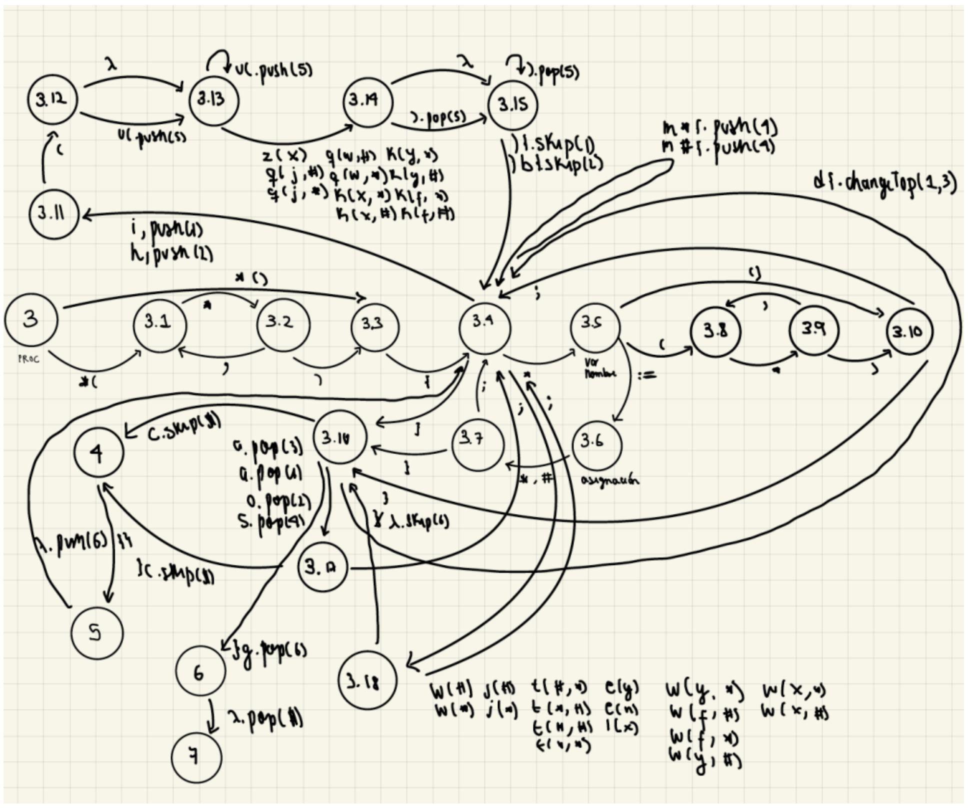

Figure 3.0.

3. Results

The results were successful as the tests cases given have worked perfectly, alongside other user-created inputs. It is worth highlighting that thanks to previously developed projects it was possible to verify and corroborate both projects as if they reached the same conclusion then they were both correct.

4. Discussion

a. Difficulties

The main difficulty to highlight was the approach to the parser. At the start we considered less states and a more dynamic stack, but this proved to be highly complicated and could lead to hard to solve problems in fringe loop cases. Because of this, the choice was made to use states with a stack to handle limitless input cases and to reutilize parts of the states. The whole block of states focused on interpreting instructions would need to have been created in the instance of each control structures instructions, and procedure instructions, and final instructions, however thanks to the stack we could use the same set of transitions and at the end pop the identifying aspect as to return to a base state. This was we used the stack to get the most out of a robust code identifier by reutilizing it constantly. However, this also proved the challenge of creating said robust set of states, which is why the figures pictured earlier in the report was the starting point and the key aspect. After developing this we used to excel to efficiently write the lines of transitions in gold, after which with some troubleshooting and a couple of changes the parser was done. All of this thanks to developing beforehand the diagram of states to be implemented and finding identifying issues before writing a single line of code.