Building a Compilator on Java for a mini-C Programming Language

J. Agustín Barrachina
IEEE Student Member
École Polytechnique
Université Saclay-Paris

Philémon Poux
École Polytechnique
Université Saclay-Paris

***Abstract*—In this project, a compilator was created to generate a x86-64 assembler code from a C fragment called mini-C. This is a 100% C-compatible fragment, in the sense that any Mini C program is also a C program.**

I. Introduction

"Optimizing compilers are so difficult to get right that we dare say that no optimizing compiler is error-free! Thus, the most important objective in writing a compiler is that it is correct" [1]

Some basic knowledge on C programming language is [2].

A. Structure of a Compiler

II. Lexical Analyzer

A lexical analyzer (*Lexer*) is the first front-end step in compilers, matching keywords, comments, operators, etc, and generating a token stream for parsers called *lexemes*. The Lexer reads input from the programming language to compile (mini c in our case) and matches it against regular expressions and runs a corresponding action if such an expression is matched.

To make the Lexer are going to use:

- Regular Expresions: To describe the lexemes
- Finite Automata: To recongnize the expresions

A. Regular Expresions

The concept of regular expression arose in the 1950's when the American mathematician Stephen Cole Kleene formalized the description of a regular language.

A regular expression is a sequence of characters that define a search pattern. In other words, there are a conjunction of letters and digits that follow a certain rule.

Let us define *letter* as any letter in the Latin alphabet and *digit* any number [0-9]. Then we can define rules as follow:

$$0|[1-9](< digit > *|[]) \tag{1}$$

Last equation 1 is a declaration of a decimal digit. The "—" is a logic or, it means, either the digit is 0 or it will be another thing. If it is not only 0, the number cannot start by 0 in C syntax, so it must start with a digit different from 0, which is range from 1 to 9 (encoded as [0-9]). Secondly, this digit can be followed by either nothing (represented by: []) or by any digit for as many digits are they must be. The format ¡rule¿* means the repetition of a rule for as many times as necessary, or no repetition at all.

B. Finite Automata

A *finite automata* is basically a binary graph which just say "yes" or "no" by means of a *recognizer* to each possible string.

There are two different classes of automatas:

1) *Nondeterministic Finite Automata* (NFA)
2) *Deterministic Finite Automata* (DFA)

The first class (NFA) have no restrictions on the labels of their edges. A symbol can label several edges out of the same state. The DFA on the other hand have for each state and symbol exactly one edge with that symbol leaving that state.

C. Implementation

For the lexical analyzer, a flex library was used [3]. A .flex file was created and then, by means of jflex, converted to the final java class.

Jflex lexers are based on a DFA automata. For more information about jflex library please refer to [4].

III. Syntax Analyzer

The *syntax analyzer's* job is to detect error of syntax and display the error that corresponds.

IV. Semantic Analyzer

"Well typed programs do not go wrong"

A *Semantic Analyzer* uses the syntax tree and the information in the symbol table to check the source program for semantic consistency with the language definition.

An important part of the semantic analysis is the type checking where it gathers type information and checks that each operator has matching operands.

An example of the type checking will be to make sure the index which whom an array is accessed is an integer and not any other incompatible type.

In a equation like $8.0+4$, the type checking will make sure to convert the integer "4" into a floating point before making the operation.

V. Code Generation

VI. Conclusion

Acknowledgment

References

[1] A. V. A. M. S. L. R. S. J. D. Ullman, *Compilers. Principles, Techiniques & Tools*, 2nd ed. Addison Wesley, 2006.

[2] B. K. . D. Richie, *The C Programming Language*, 2nd ed. Prentice Hall, mar 1988.

[3] J. Team. (2015, mar) Jflex - the fast scanner generator for java. [Online]. Available: http://jflex.de/

[4] G. K. S. R. R. Dcamps, *JFlex User's Manual*, apr 2015.