Steps of Compilation According to Niklaus Wirth

Introduction

Compiling turns source code, readable by humans into sets of instruction that computers understand and process. The process involved in the parsing and compilation of a language is usually defined by structured procedures that consist of Niklaus Wirth's Compiler Construction (2005) This process guarantees sequentially good syntax and can be applied to implement its own parser or compiler. Knowing the nature of these phases and CFG's role in it, tend to show how high-level programming language is transformed into hardware through means of a compiler.

Lexical Analysis

In the case of Wirth's model, the first step is lexical analysis, which takes a program's raw text and converts it into a sequence of tokens. Tokens are the smallest yet meaningful and important elements of the language (e.g., they could be identifiers, keywords, or symbols). This stage filters out unnecessary characters like whitespace and comments, allowing the compiler to focus on the core structure of the code (Wirth, 2005).

Syntax Analysis

Then the compiler does syntax analysis (or parsing) to see if that order of tokens conforms to the grammar rules of the language. Here, enters the concept of a Context Free Grammar. A CFG is a collection of production rules that specify how legal sentences in a programming language can be built up from tokens (Aho et al., 2007). The language is used to specify hierarchical structures such as loops, conditionals and expressions. The grammar is used as input by the parser to build

an AST -- a syntax tree, which illustrates the logical operations and process sequence of the program. If the source code is not in accordance with its grammar, then the compiler encounters syntax errors.

Semantic Analysis

After the syntax tree is built, semantic analysis checks for meaning in the program. This step makes sure that the program developing logical relationships holds, like allowing one to verify if variables are declared before use and if types are matched during assignment operations, etc. Wirth (2005) described that the use of a symbol table is important here to keep track of variable types, scopes and functions.

Intermediate Code Generation

The syntax tree is then transformed into a machine-independent intermediate representation (IR) by the compiler. This simplifies optimization and code generation at a later stage, for the compiler to have instructions in hand for performing logic operations perfectly without bothering about any hardware constraints.

Optimization

Optimization further polishes the intermediate code to make it run more efficiently. Methods include eliminating duplicate calculations and simplifying loops. While optimization is regarded as optional according to Wirth (2005), it can be crucial for applications which are performance sensitive.

Code Generation and Assembly

The optimized intermediate code is then translated by the compiler into target machine code or assembly language. This stage comprises instruction selection, register assignment and binary output generation. The machine code generated at this stage can be run by the computer's processor, without an interpreter or a compiler.

Role of Context Free Grammar

CFG is the basis of both lexical and syntax analysis. It provides the back-bone structure that a compiler can use to tell whether code is valid and what the syntax tree should be. With no CFG, the compiler could not recursively parse high-level languages into logical components translating machine codes accurately will be impossible.

Conclusion

Wirth's six stages of compilation—lexical, syntax, and semantic analysis; generation of intermediate code; optimization; and final code output, form a clear road map from source code to executable program. C F Grammar is fundamental for parsing and doing syntax checking, which allows a compiler to accurately translate human-readable instructions into the specific operations needed to be executed on a machine.

References

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compilers: Principles, techniques, and tools* (2nd ed.). Addison-Wesley.

https://mrce.in/ebooks/Compilers%20Principles,%20Techniques,%20&%20Tools%202nd%20Ed.pdf

Wirth, N. (2005). *Compiler construction*. ETH Zürich. https://people.inf.ethz.ch/wirth/CompilerConstruction/CompilerConstruction1.pdf

Word Count: *615*