Compilers

Group 12
Stuart Dilts
Joe Whitney

February 8, 2017

1 Introduction

When writing a computer program, many developers and scientists want to think about what the program is going accomplish, and focus on the algorithms that are deployed, not how the computer's state machine is actually performing each and every action. In order to achieve this abstraction, a compiler is needed to translate a programmer's ideas into the low level assembly code of the system. Compilers have other benefits as well, including simple code optimization and code portability. For our senior capstone project, we are implementing a compiler for the Tiny Language. To write a compiler, one needs to be able to apply various aspects of computer science including language parsing, discrete math, and finite automata, all while following good software engineering principles.

2 Background

Java and ANTLR are being used to build the compiler. ANTLR is generating both the scanner and the parser modules of the compiler. Both of these were chosen to speed up development time, as we are very familiar with Java, and ANTLR comes strongly recommended.

3 Methods and Discussion

3.1 Scanner

The purpose of a scanner is to take raw input (usually in the form of code) and *tokenize* it. Tokenization is the process of converting an input stream into a series of tokens, i.e. literals, keywords, identifiers, etc. Regular expressions are used in defining tokens, and subsequently they are central to the tokenization process.

As mentioned above, we used ANTLR in Java. We chose to use a scanner generator as it was recommended, and made the overall process of crating a scanner much simpler. The only real difficulties we faced were coming up with correct regular expressions for recognizing tokens; however, had we coded our own scanner we surely would have faced many more difficulties.

3.2 Parser

3.3 Semantic routines

3.4 Full-fledged compiler

4 Conclusion and Future Work