Info-F403 – Introduction to language theory and compiling

Genial Imperative Language for Learning and the Enlightenment of Students Part-III Report

Herma ELEZI

Gilles GEERAERTS
Mrudula BALACHANDER
Mathieu SASSOLAS

2024

Contents

1 Introduction

This document provides a detailed report on the implementation of Part 3 of the INFO-F403 project. The primary objective of Part 3 was to generate LLVM Intermediate Representation (LLVM IR) from a custom GLS programming language and to verify its execution using test cases.

2 Objectives

The main objectives of Part 3 were:

- Extend the GLS compiler to generate LLVM IR for given GLS programs.
- Implement functionality to handle conditional statements, loops, and basic arithmetic operations in GLS.
- Test the correctness of LLVM IR generation using predefined test cases.

3 Implementation

3.1 Key Classes and Methods

The following classes and methods were central to the implementation:

- `Main.java`: The entry point for the compiler.
- `Parser.java`: Parses the GLS source code into an Abstract Syntax Tree (AST).
- `LLVMGenerator.java`: Converts the AST into LLVM IR.
- `LexicalAnalyzer.flex`: Handles tokenization of the GLS source code.

3.2 LLVM Generation

The `LLVMGenerator.java` class traverses the AST and generates LLVM IR instructions. Key features include:

- Support for integer arithmetic operations (e.g., addition, subtraction).
- Conditional branching using LLVM's `icmp` and `br` instructions.
- Input and output handling using LLVM's `scanf` and `printf`.

3.3 Test Cases

Several test cases were created to validate the correctness of the compiler. These included:

- `if_test.gls`: Verifies conditional statements.
- `operation.gls`: Tests arithmetic operations.
- `loop.gls`: Checks loop handling.

4 Results

4.1 Example LLVM IR Output

For the `if_test.gls` file, the generated LLVM IR is as follows:

Listing 4.1: Generated LLVM IR for if_test.gls

```
@.strP = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1

... (full LLVM IR output here) ...
```

4.2 Test Execution

The generated LLVM IR was tested using the LLVM interpreter `lli`. The results were:

- Input: 3, Output: `0`
- Input: 7, Output: `10`

5 Challenges and Solutions

5.0.1 Challenges

- Handling edge cases in GLS parsing.
- Generating correct LLVM IR for nested conditionals and loops.

5.1 Solutions

- Enhanced error handling in `Parser.java`.
- Incremental testing and debugging for LLVM IR generation.

6 Conclusion

Part 3 of the INFO-F403 project successfully extended the GLS compiler to generate LLVM IR. The output was verified using multiple test cases and matched the expected results.