Sudhanshu Chawhan
CS16BTECH11037

Irfan Ali
CS16BTECH11019

Compilers2

Assignment : Code Generation

Overview

In the previous assignments, we have successfully built a parser and semantic analyser for COOL language. We can generate an AST for an error free COOL code. We will now generate the working LLVM IR codes for COOL codes using the generated ASTs.

Code Design

The code is split in multiple subfiles for better readability and elegant design pattern. Following is the directory structure of the code.

Directory Structure

There are mainly 6 files :-

InheritanceGraph.java - Inheritance graph for the AST created after parsing.

Visitor.java - Implements visit methods for different AST nodes such as AST.method, AST.attr, AST.program, etc. All these methods together print all the necessary IR codes.

VisitorUtils.java - Implements all the helper functions required in Visitor.java file.

IRInstructions.java - Implements all the necessary IR instructions required throughout the code.

GlobalData.java - Contains variables and methods that are used globally in all the files.

Default.java - Implements methods that print IR code for default data.

Working

The code starts with visit(program) method in Visitor.java that first creates the inheritance graph and then prints all the default strings and methods and string constants used in the IR. Then, for all the classes the constructors are printed. Then visit is called on the methods that prints the required IR code.

- Inheritance graph is defined by calling it's constructor on program node. A strConsToRegister HashMap is stored in GlobalData that maps string constant to it's register. This map is filled in visit(program) and then used In GEP instructions.

- For printing the structs for all classes, the inheritance graph is traversed in DFS fashion and structs are added for each visited class. Meanwhile, we update the attrIndexMap that keeps track of the index of a particular attribute in the struct of a class.

- For Constructors, the constructor for root class is directly added and then for all other classes, similar to structs, DFS traversal is performed on inheritance graph. For each class, Firstly, the parent constructor is called and then each attribute is printed. If an attribute is unassigned then default values are pushed to that variable otherwise proper casting is done if incase return type is inconsistent with attribute type.

- For printing methods for all classes, similar approach is performed i.e. DFS traversal. For each class, we print all it's methods and then move recursively to it's children and print their methods. Printing of method is performed in visit(method) method in visitor.java. Simply put, we first print the necessary IR code for method formals like allocating memory, then we print the body expression and then finally we cast the return register if types are inconsistent.

Printing Default methods

These methods are hardcoded after understanding LLVM IR codes of C programs consisting these methods.

These methods are hardcoded and simply printed to the IR. The variables are merged in the printing process to make the code working.

Complications faced while programming

The first big step was understanding the working, design and guidelines of LLVM IR. We were not experienced at working with assembly codes, so it was a little hard to get through. The next step was planning code generation. Code design and implementation details proved out to be more hectic than we imagined.

After extensive coding, the first compilation spitted tons of error messages. Most of the errors were trivial while some were minor design faults.

Test Cases

Necessary and sufficient test cases are added for evaluating the proper working. All the test cases are error free.