CS 6240: Advance Compiler Design

Assignment 2

Name: Nilesh Shah, CS19MTECH11021
Name: Gagandeep Goyal, CS19MTECH01003

This document is generated by LaTeX

Contents

1 Introduction

We are using our Semantic Analysis code for creating Inheritance graph and basic class structure.After creating those we are traversing the class hierarchy in BFS fashion and generating the LLVM IR as we visit each node.

2 CodeGen

2.1 Constant prints

Static IR is printed corresponding to Functions of IO, Object, String, Header info and main function.

2.2 Class Declaration

Classes are declared according to Inheritance Graph. Firstly an object class is declared. The classes like Main, IO have object class as parent while classes like B if it inherits from A declares A as its parent. Integer appear as i32 , Bool appear as i1 and string appear as i8*.

2.3 Static strings

Strings which are required to be printed as IO are declared globally in the IR. For this we also need to cast these strings to i8.

2.4 Constructor

Constructor is defined for every class. It initializes the values for all the declarations not initialized in that class. The main program execution starts from the main function which calls the main class constructor. Initializations like 0 for int , "" for string and false for bool is default.

2.5 Methods

Methods definition is printed with return type of its own class. We traverse the method body using our ExpressionTraversal class.

2.6 Divide by Zero

The Divide by zero exception is handled by checking the denominator for 0 value and then exiting with printing global variable - DivByZeroException.

2.7 Dispatch void

When there is a function call from a null object, it leads to an static dispatch error. This exception is handled using if condition and therefore exiting with printing global variable - DispatchToVoid

2.8 Expressions

2.8.1 Arithmetic

For basic operators like Addition, Multiplication, Subtraction and Division directly the corresponding operators Addition, Multiplication, Subtraction, Division are used. Divide by zero case is explicitly handled as explained above.

2.8.2 Logical Operators

For Logical operators like Equal to, less than, greater than, less than equal, greater than equal, directly the corresponding operators like icmp eq, icmp slt, icmp sgt, icmp sle, icmp sge are used.

2.8.3 Void

We used icmp eq operator with null for checking void type.

2.8.4 Object

If your expression is of object class then it can appear at two places. Either it can appear as a attribute of a class else formal of a method. Formals can be directly loaded whereas for attributes we allocate a memory and provide it with that pointer information and the loading into variable follows ahead.

2.8.5 Condition

We create 3 blocks for handling if-else condition i.e if.then, if.else, if.end. Each block handles its own path execution.

2.8.6 Loop

We create 3 blocks for handling loops i.e loop_cond, loop_body, loop_end. Each block handles its own path execution.

2.8.7 Block

Block expressions are directly written to IR.

2.8.8 Static Dispatch

Firstly we check whether the object exist or not. If it exists then we build the parameter list. After that if the parameter list consists of the attribute then we insert a getElementPointer instruction and therefore call the function.

3 Class Structure

3.1 Codegen.java

We use a BFS traversal on all our classes because untill and unless the parent code does not prints , there is no use of printing child code. We perform the BFS traversal twice. During the first BFS traversal, the class attributes are declared. In the second BFS traversal we print the IR corresponding to all the methods of that classes. Finally we print IR for main method, methods of string classes and global strings thereafter.

3.2 ExpressionTraversal.java

In this file we print Constructors for each class. The next step that follows is the printing the methods of that class. For each method we Emit the IR for each expression residing in it.

3.3 PrintBaseClasses.java

This file prints IR for methods of base class like IO, Object, String.

3.4 TypeParse.java

This files does type conversion like class name to defined IR types and vice versa as well i.e defined IR types to class name.

4 Test Cases

4.1 Test-2

This test case covers the concept of method generation of a class.

4.2 Test-2

This test case covers the concept of generating IR during use of Normal arithmetic operations.

4.3 Test-3

This test case covers the concept of Loop iterations in the code.

4.4 Test-4

This test case covers the concept of static dispatch.

4.5 Test-5

This test case covers the concept of Asssignment operations.