Descabouda Scala Decompiler

Submitted in fulfillment of the requirements of the mini project of

Programming Language Translation Course

Submitted by

Abdallah Elerian	2158
Anwar Mohamed	2491
Yasmine El-Habashi	2083

Department of Computer and Communications Engineering
FACULTY OF ENGINEERING - ALEXANDRIA UNIVERSITY
Alexandria, Egypt

Fall Semester 2016

Contents

Chapter 1

Problem Definition

You are required to translate a scala bytecode into a meaningful format to aid in reverse engineering.

Chapter 2

Introduction

2.1 Decompiler Definition

To begin, let's start by defining what it means by *a decompiler*:

A decompiler is a computer program that takes an executable file as input, and attempts to create a high level source file which can be recompiled successfully. It is therefore the opposite of a compiler, which takes a source file and makes an executable. Decompilers are usually unable to perfectly reconstruct the original source code, and as such, will frequently produce obfuscated code.

2.2 Decompiler Design

Decompilers can be thought of as composed of a series of phases each of which contributes specific aspects of the overall decompilation process.

2.2.1 Loader

The first decompilation phase loads and parses the input machine code or intermediate language program's binary file format. It should be able to discover basic facts about the input program, such as the jvm version and the entry point. In many cases, it should be able to find the main function of a scala program, which is the start of the user written code.

2.2.2 Disassembly

The next logical phase is the disassembly of machine code instructions into a machine independent intermediate representation (IR).

2.2.3 Idioms

Idiomatic machine code sequences are sequences of code whose combined semantics is not immediately apparent from the instructions' individual semantics. Either as part of the disassembly phase, or as part of later analyses, these idiomatic sequences need to be translated into known equivalent IR.

2.2.4 Program analysis

Various program analyses can be applied to the IR. In particular, expression propagation combines the semantics of several instructions into more complex expressions.

2.2.5 Data flow analysis

The places where register contents are defined and used must be traced using data flow analysis. The same analysis can be applied to locations that are used for temporaries and local data. A different name can then be formed for each such connected set of value definitions and uses. It is possible that the same local variable location was used for more than one variable in different parts of the original program.

2.2.6 Type analysis

A good machine code decompiler will perform type analysis. Here, the way registers or memory locations are used result in constraints on the possible type of the location.

2.2.7 Structuring

The penultimate decompilation phase involves structuring of the IR into higher level constructs such as while loops and if/then/else conditional statements.

2.2.8 Code generation

The final phase is the generation of the high level code in the back end of the decompiler. Just as a compiler may have several back ends for generating machine code for different architectures, a decompiler may have several back ends for generating high level code in different high level languages.

Chapter 3

Work Done

3.1 Disassembler Implementation

3.1.1 Basic Blocks

```
ClassFile {
   u4           magic;
   u2           minor_version;
   u2           major_version;
   u2           constant_pool_count;
   cp_info      constant_pool[constant_pool_count-1];
   u2           access_flags;
   u2           this_class;
   u2           super_class;
   u2           interfaces_count;
   u2           interfaces[interfaces_count];
   u2           fields_count;
   field_info   fields[fields_count];
   u2           methods_count;
   method_info  methods[methods_count];
}

field_info {
   u2           access_flags;
   u2           name_index;
   u2           descriptor_index;
   u2           attributes_count;
   attribute_info attributes[attributes_count];
}

method_info {
   u2           access_flags;
   u2           name_index;
   u2           descriptor_index;
   u2           attributes_count;
   attribute_info attributes[attributes_count];
}
```

3.1.2 Decoders

Decoders are used to decode the jvm bytecode into an intermediate form. Currently theses decoders are implemented:

- Attributes Decoder
- Class Decoder
- Codes Decoder
- Constants Decoder
- Fields Decoder
- Methods Decoder
- Interfaces Decoder

3.1.3 Jasmin Printer

Jasmin Printer is used to print the java disassembly codes, constant pool, fields, methods of the disassembled class.

3.2 Decompiler Implementation

3.2.1 Generators

Generators are used to transform the IR structures into a high-level java code. Currently theses generators are implemented:

- Attribute Generator
- Class Generator
- Field Generator
- Method Generator

3.2.2 Java Printer

Java Printer is used to print the java high-level codes, class definition, fields, methods of the decompiled class.

References

[1] Decompiler: From Wikipedia, `https://en.wikipedia.org/wiki/Decompiler`

[2] The class File Format, `https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-4.html`