CSc 620: Krakatoa: Decompilation in Java

Introduction

This paper presents a technique for decompiling Java byte code into Java source. The decompiler, Krakatoa exploits the type information and well-behaved property of the Java class files to recover the Java source code from the class file. The paper focuses on two problems:

- recovering source level expressions
- synthesizing high level constructs from goto-like primitives

The decompilation process consists of 5 stages:

- Expression recovery which involves recovering expressions using stack-simulation technique
- sequencer which orders the nodes of the CFG for input to the Ramshaw’s algorithm which removes goto-like primitives
- Goto Eliminator which removes the goto statements
- code simplifier which restructures the Java AST into a simpler one
- Final Java Printer which traverses the Java AST and emits the Java source code

Expression Recovery

Symbolic execution of the byte code creates the corresponding Java source expressions. Symbolic execution simulates the JVM evaluation stack with strings that represent the source-level expression being computed. The algorithm recovers the expressions one basic block at a time. Symbolic execution of conditional operators poses some problems as these basic blocks do not begin with empty stack.
Instruction Ordering

Ramshaw’s algorithm

After recovering expressions, we get a source code with conditional and unconditional gotos. Ramshaw’s extended algorithm removes these gotos by replacing each forward goto with a break and each backward goto with a continue statement. Then, the algorithm inserts a loop that ends just before the target of each break statement and similarly inserts a loop that starts just before the target of each continue statement. Each inserted loop ends with a break statement to make sure that the control falls out of the loop. Ramshaw’s algorithm requires as inputs the control flow graph and an instruction ordering which is encoded into the CFG by means of augmenting edges. These edges occur between every pair of neighboring instructions. He proves that if the resulting augmented graph is reducible, then a structurally equivalent program can be created without gotos.

Creating Reducible Augmented Flow Graphs

This requires that no augmenting edge enters a loop anywhere other than at its header. This can be prevented by making the header first and then contiguously reordering the loop instructions. This would ensure that the only augmenting edge entering loop will enter at the loop header.

A loop with no nested loops is ordered by removing the back edges and then doing a topological sort. A loop with nested loops is ordered by compressing the inner loops into single nodes and then ordering both the loop and the surrounding graph. After ordering both, we expand the compressed node. If the flow of control has changed due to re-ordering then the algorithm adds new branches to restore the original flow control.

Code Transformation

The output of Ramshaw’s algorithm is complex but yet legal Java source code. The decompiler then proceeds to extract high-level constructs like if-then-else statements. It uses program point analysis to track the control flow of the program. A program point is a syntactic point in the program. Every
statement has a program point before and after it. A program point has two properties:

- *reachability* A program point is unreachable if and only if it is preceded by unconditional transfer statements like return, break etc.

- *Equivalence Class* Two program points are said to be equivalent iff the future computation of the program is the same from both the points.

**AST Rewrite Rules**

The decompiler now performs a series of AST rewriting transformations to recover as many of the natural program constructs as it can. The rewriting rules fall under 3 categories:

- *if-then-else rewriting rules* in which there are rules to eliminate unnecessary else statements and create if-then-else construct.

- *Loop Rewriting Rules* consist of rules to delete unnecessary breaks and continues and others.


**Comments**

Krakatoa is very effective in reproducing readable Java source code Java bytes. Some limitations are that one could introduce irreducible control flow into the program through bogus conditional jumps and then Ramshaw’s algorithm would fail. Also as stated earlier, it doesn’t handle conditional-expression operator. If the system is presented with a high-level language construct that it does not know, then it might leave unnecessary breaks and continues in the program but still produces legal Java code.