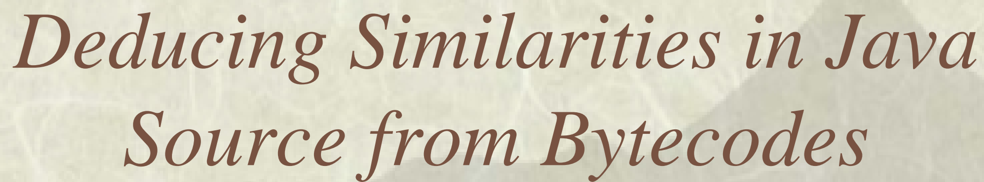

# Deducing Similarities in Java Source from Bytecodes

(Appeared in the 1998 USENIX Technical Conference.)

Brenda S. Baker

Bell Laboratories, Lucent Technologies

Udi Manber

University of Arizona

Presented by: Huang Ji

# Introduction

On network computers, Java bytecode is a common way to execute programs and it always arrive through the net transparently. This bring the problems of security, update, portability, and so on.This paper gives ways of solving one of the basic problems: find similarity of the source code among a set of bytecode files.

We adapt three tools – Siff, Dup and Diff and by combining these three tools we can do more effectively at finding the similar files while keeping false positives low.

These tools have several applications including program management, plagiarism detection, program reuse and reengineering, security, compression and so on.

# *Outline*

1. The definition of three tools.
   - Siff : is a program to find the similarities in text files. It is designed to find pairs that contain a significant number of common blocks("fingerprints") from a large number of text files.
   - Dup : search sets of source files to look for sufficiently long sections that match except for systematic transformations of names, such as variable names, and can deal efficiently with a few million lines of source code. This kind of match called p-match.
   - Diff : is used for finding commonality between files. It use dynamic programming to identify lines-by-lines changes(insertion and deletion) between files.
2. The combination of the three tools.

# *Outline(continue)*

The combination of Siff and Dup for searching large numbers of files make it more broaden and stringent.

files found by Siff files found by Dup

# *Outline(continue)*

The combination of Dup and Diff for more detailed analysis of files.

3. Adapting the tools to Java bytecode file(focus on)

The adaptation of Siff and Diff do not working directly on bytecode files as well as the disassembled bytecode files. So we need to encode the disassembled bytecode files first. Dup has already incorporated with this encode technique but additional preprocessing can improve its performance.

4. Experiments

- Experiment 1 with the random changes to one Java program.

For a range of source similarity between 70% to 79% and after trails of 59 times, average source similarity is 75.9% and we got bytecode similarity of 71.9% with max difference of 7%.

We can see that average source code similarity and bytecode similarity match very well.

# *Outline(continue)*

-Experiment 2 with a large set of bytecode files(2056 files).

We test using Siff only, Dup only, both of Dup and Siff and find that very few false positives by combine of Dup and Siff(18 out of 2 million pairs at most and in this experiment, 9 instances were reported but all are valid instance of similarity). While out of 43 pairs reported from different collections with at least 50% of similarity, Siff reported several that are not valid instance of similarity and Dup reported 1 that are not valid from the pairs with at least 200 lines in common.

So by combining the Dup and Siff, we can get the result more stringently.

# How to adapting the tools

1. Review of the Java class files(bytecode files)
   - Constant pool and Method table.
2. Disassemble of Java class files.
   - first need to keep track of position of parsing in the hierarchy table and structure.
   - The disassembled code contains opcodes and arguments as follows:

```
o182    #invokevirtual
c106
o153    #ifeq
j+17
o025    #aload
v4
o180    #getfield
```

# How to adapting the tools(continue)

- Siff and Diff still fail at this stage

  Why?

  Slight changes to the Java file may result in the change of indices into the constant table or local variable table.
- Dup can run if it is provided with appropriate lexical analyzer but often need preprocessing.

3. Preprocessing for Dup

- Change the jump offsets in the disassembled code into a “goto label” form and Dup will compute the offsets itself for the labels. Thus, this process evokes mismatch and enables dup to find longer p-matches.

4. Preprocessing for Siff and Diff .

- Absolute values of table indices may change, but positional relationship will maintain.

# *How to adapting the tools(continue)*

- Use the same offset encoding as dup. Siff and Diff are run on the offset-encoded files.
- Rules:

  1.Replace each index into the constant pool and local variable table with an offset independently.(from absolute value to relative position)

  2. The first occurrence of index is encode as 0, thereafter each use is encoded as negative of the number of line.

  So the above example can be encoded as following:

| | | |
|---|---|---|
| o182 | #invokevirtual | o 182 |
| c106 | | c -26 |
| o153 | #ifeq | o 153 |
| j+17 | | j+17 |
| o025 | #aload | o025 |
| v4 | | v -10 |
| o180 | #getfield | o 180 |
| c253 | | c -10 |
| o025 | #aload | o025 |
| v5 | | v -10 |

Left-hand is the bytecode before encode, right-hand is one after encode

We can see that encoding decreases reliance on the absolute value and preserve the indices into the instruction to see they are same or not.

# *Conclusions and Questions*

## Conclusions

1. This report is perfectly good for it comprehensively illustrated the theory of finding the similarity through the bytecode and give the experiment result to validate it. They made a practical software product but it is a bit hard to understand because it involves materials extensively and intensively.

2.By cooperating the encoding technique and combining the three tools, we can keep the false negative very low and get perfect result.

## Questions

1. I'm not quite sure of my understanding of the disassemble bytecode after calculating offsets is right, any suggest will be grateful.

2. A number of "obfuscators" has been used to deter the decompilation, can technique of finding similarity still be effective? Why?