Maintenance Support Tools for JAVA Programs: CCFinder and JAAT

Toshihiro KAMIYA†+
kamiya@ics.es.osaka-u.ac.jp

Fumiaki OHATA†
oohata@ics.es.osaka-u.ac.jp

Kazuhiro KONDOU†
kondou@ics.es.osaka-u.ac.jp

Shinji KUSUMOTO†
kusumoto@ics.es.osaka-u.ac.jp

Katsuro INOUE† ‡
inoue@ics.es.osaka-u.ac.jp

† Graduate School of Engineering Science,
Osaka University
1-3 Machikaneyama, Toyonaka,
Osaka 560-8531, Japan
+81 6 6850 6571

+Intelligent Cooperation and Control Group,
TOREST, JST
‡ Graduate School of Information Science,
Nara Institute of Science and Technology

ABSTRACT
This paper describes a maintenance support tools, CCFinder and JAAT, for JAVA programs. CCFinder identifies code clones in JAVA program. JAAT executes alias analysis for JAVA program.

1 Introduction

Maintaining (fixing reported errors, changing the software to some new environment etc.) large and complex software systems requires a lot of effort since it is very difficult to understand the whole of the software and identify the parts that should be modified. It is difficult to find up-to-date figures for the relative effort devoted to these different types of maintenance[7]. So, it is necessary to develop the supporting method to each of the activity.

Program slicing has been proposed to efficiently localize faults in the program[6]. By definition, slicing is a technique which extracts all statements that affect some set of variables in the program. The set of all extracted statements is called a slice. In order to extract the slice, alias analysis is one of the most important problems. When an expression refers to a memory location which is referred to by another expression, there is an alias relation between those expressions. Alias analysis, i.e. extraction method of such relations is an essential technique to efficiently extracting slice from object-oriented programs. However, the method has not been established.

On the other hands, it is pointed that code clones makes the source files very hard to modify consistently modify. A code clone is one of a code portions in source files that is identical or similar to each another. Clones are introduced because of various reasons such as reusing code by 'cut-and-paste' or intentionally repeating a code portion for performance enhancement[3]. For example, assume that a software system has several clone subsystems created by duplication with slight modification. When a fault is found in one subsystem, the engineer has to carefully modify all other subsystems. For a large and complex system, there are many engineers who take care of each subsystem, and modification becomes very difficult. Various clone detection tools have been proposed and implemented [2][3] and a number of algorithms for finding clones have been used for them, such as line-by-line matching for an abstracted source program, and similarity detection for metrics values of function bodies. However, we consider there are several problems to be solved. For example, our pilot experiment has revealed that certain types of clones seem difficult to be rewritten as a shared code even if they are found as clones.

This paper describes the maintenance support tools, CCFinder and JAAT, for JAVA programs. CCFinder efficiently identifies code clones and JAAT executes the alias analysis for large scale JAVA programs.

2 Code clone detection tool: CCFinder

We have devised a clone detection algorithm and implemented a tool named CCFinder. The underlying concepts for designing it was as follows.

(1) CCFinder should have industrial-size strength, and be applicable to million-line size system within affordable computation time.

(2) The language dependent part of CCFinder should be limited to small parts, and the tool has to be easily adaptable to many other languages.

(3) CCFinder should detect clones of practical interest clones. Not only syntactically same portions, but also similar portions which are considered to be actual clones have to be effectively extracted.

We have used a simplified suffix-tree[4] to find clones practically and effectively. Various optimization techniques were also built into the tool. The tool was initially developed for

C and C++, and then successfully extended to Java by two person days. The tool transforms a source code with transformation rules so that portions of interest (but syntactically not exactly identical structures) can be detected and uninteresting portions (even when they structurally similar) are not detected. The uninteresting clone portions do not contribute to reduction of total size of code since they are hard to be merged into single portions. It performs an abstraction of a token sequence called parameter-replacement before executing the token-by-token matching algorithm. This parameter-replacement is a pre-process of parameterized-match[2], and is very effective for clones with name substitution. Also, token-by-token matching algorithms are able to find clones with modified line structures, which cannot be detected by line-by-line algorithm. Token-by-token matching is much more expensive than line-by-line matching in computing complexity. However, we propose several optimization techniques especially for the token-by-token matching algorithm, which enables the algorithm practically useful for large software.

We have applied CCFinder on million-lines codes from JDK, Qt, Linux, and FreeBSD, to evaluate its effectiveness quantitatively and qualitatively. The similarity of Linux and FreeBSD, as well as nature of JDK, has also been explored. The tool has detected clones that are small in size by themselves but many lines can be removed by rewriting them using a shared routine.

3 Alias analysis tool: JAAT

Alias analysis was first proposed for traditional procedural languages such as C and Pascal[5]. Recently prevalent object-oriented(OO) languages such as C++ and JAVA have used new concepts such as *class, inheritance, dynamic binding* and *polymorphism*. Alias analysis methods have been proposed for OO programs[8]; however, those do not consider reuse of analysis results previously obtained, and so we were unable to expect efficient alias analysis for OO programs in which reusability is essential.

In order to decrease re-computation costs, modularize the alias analysis results and compute inner alias relations and outer alias relations, we propose the following two step procedure:

1. Alias flow graph(AFG) construction
2. Alias computation using AFG.

An *alias flow graph(AFG)* is an undirected graph, from which we can extract FS alias relations. In this section, we will describe the AFG based alias analysis algorithm for reference-type expressions, such as reference variables, class instance creation expressions and method invocations in JAVA ; however, we can easily adapt these algorithms to computing aliases generated by pointer variables of other languages, such as C and C++.

We have implemented the proposed method as a JAVA alias analysis tool called JAAT. Using this system, we have executed various programs and obtained several metrics values.

This system consists of two subsystems, the analysis subsystem(JAVA Alias Analysis Libraries) and the user interface(UI) subsystem(JAVA Alias Representation Tool).

The analysis subsystem was written in C++, and consists of three libraries, libANTLR(lexical and syntax analyses)[1], libjavamm(semantic analysis) and libAFG(alias analysis). Using these libraries, we can generate the syntax tree, the semantic tree, AFGs, and MFGs for each source program.

The UI subsystem was also written in C++, and used Gtk−− tool kit. This subsystem has two main functions; editing programs and visualizing aliases with *Text Window* and *Alias Tree Window*. Once the user indicates an expression as an alias criterion, the UI subsystem sends the query to the Analysis subsystem, and displays analysis results on the Text Window and the Alias Tree Window.

REFERENCES

[1] `http://www.ANTLR.org/`, “ANTLR Website.”

[2] B. S. Baker, ”On finding Duplication and Near-Duplication in Large Software System”, Proc. Second IEEE Working Conf. on Reverse Eng., pp. 86-95, Jul. 1995

[3] I. D. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier, ”Clone Detection Using Abstract Syntax Trees”, Proc. of ICSM98, pp. 368-377, Nov. 1998, Bethesda, Maryland, USA.

[4] D. Gusfield, Algorithms on Strings, Trees, and Sequences, pp. 89-180, Cambridge University Press 1997.

[5] M. Hind, and A. Pioli, “An empirical comparison of interprocedural pointer alias analysis,” in *IBM Research Report*, #21058, 1997.

[6] M. Weiser, “Program slicing,” in *Proc. of the 5th ICSE*, pp. 439-449, 1981, San Diego, California, USA.

[7] I. Sommerville: *Software Engineering,* Addison-Wesley, 1992.

[8] P. Tonella, G. Antoniol, R. Fiutem, and E. Merlo, “Flow insensitive C++ pointers and polymorphism analysis and its application to slicing,” in *Proc. of the 19th ICSE*, pp. 433-443, 1997, Boston, Massachusetts, USA.

[1] ANTLR[1] is a lexer and parser generator.