CloneDiff — Semantic Differencing of Clones

Yinxing Xue, Zhenchang Xing and Stan Jarzabek
School of Computing
National University of Singapore
{yinxing,xingzc,stan}@comp.nus.edu.sg

ABSTRACT

Clone detection provides a scalable and efficient way to detect similar codes, while program differencing is a powerful and effective way to analyze similar codes. CloneDiff, a Program Dependence Graphs (PDGs) differencing tool, complements clone detection with program differencing for the purpose of characterizing clones. It captures semantic information of clones from PDGs, and uses graph matching techniques to compute a precise characterization of clones in terms of a category of semantic differences.

Categories and Subject Descriptors
D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement

General Terms: Design, Languages.

Keywords: Clones analysis, semantic differencing.

1. Motivation

To scale to large software systems, clone detection methods either ignore program semantics (e.g., token-based techniques) or use “reduced” representation that approximates program semantics [5]. The drawback of the reduced representations is that they throw away the important program semantic information. Consequently, clone detection methods on their own can offer only limited explanation of the nature of differences among the reported code. However, knowing the details of differences among clones is important in post-detection clone analysis to perform a concrete maintenance task of code that affects these clones.

Tree and graph differencing techniques have been applied for the detection of clones. CloneDr [1] compares abstract syntax tree (AST) of similar code fragments (with same hash index) to determine clones. PDG-based detection tools use sub-graph isomorphism to detect similar code fragments. As tree or graph differencing is computationally expensive, these techniques may not scale to large systems.

To efficiently and effectively analyze software clones, we propose to complement clone detection with program differencing for the purpose of characterizing clones. We present an approach to ease post-detection clone analysis by analyzing semantic information implied by contexts in which clones occur.

The expected merits of our approach are to extricate the user of clone detection tools from risky and time-consuming manual analysis of clones to determine appropriate maintenance action on clones. No matter the manual comparison means or the syntactic differencing tools, such as AST differencing [8], are still sensitive to the arbitrary syntactic decisions a programmer made, such as the string name of a variable, the ordering of program statements and interleaving of unrelated statements, which often appear in clones. This sensitivity affects the quality of post-detection clone analysis.

Suppose the clone detection tool reports the pair of gapped clones, shown in Figure 1, understanding the semantic difference between the clones contained in method `readProxyDesc(boolean)` and `readNonProxyDesc(boolean)` in class `java.io.ObjectInputStream` is interesting and helpful for maintenance of this pair of clones. Apart from the existing gap of several different statements, this pair of gapped clone has some inconvenient and minor changes in used constant variables and the accessed function call.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Conference’11, Month 1–2, 2011, City, State, Country.
Copyright 2011 ACM 1-58113-000-0/00/0010…$10.00.
program statements. Given a pair/class of clones detected by Clone Analyzer (actually not limited to Clone Analyzer, any clone detection tool will work), we use Wala [10], a static analysis library for Java byte-code, to generate the PDG of the method.

For the optimization and abstraction techniques working for cloning domain, CloneDiff configures GenericDiff [7] to compare those generated PDGs from the clone instances. GenericDiff [7] is a general framework for model comparison. Given two input models, GenericDiff casts the problem of comparing two models as the problem of recognizing the Maximum Common Subgraph of two Typed Attributed Graphs (TAGs). Given two PDGs, PDG1 and PDG2, GenericDiff parses the input PDGs into typed attributed graphs, TAG1 and TAG2, consisting of graph nodes whose type attribute represents the type of the corresponding SSA statements [2], and consisting of graph edges whose type attribute represents either control or data dependence.

To better produce the readable and accurate clone analysis report, we define the node/property change for nodes in PDGs [6]. Based on the different types of node/property change, we summarize the patterns of semantic differences of comparison results include differential properties, additional branches, partially matched branches, additional operations, and unmatched operations [6]. According to the above defined types of semantic differences, our tool finally reports the differing result of Figure 1 in various highlighting colors of comparison editor shown in Figure 2 as follows: differential properties at line 1, additional operations for the clone gap between line 6 and line 7, additional branches for the clone gap between line 6 and line 7, and unmatched operations at line 9. The reported semantic differences are rational, since at line 1 the two constant variables TC_PROXYCLASSDESC and TC_CLASSDESC refer to the different constant integer values. Furthermore, between line 6 and line 7 in readProxyDesc(boolean), there exists a clone gap, namely a for loop statement, which indicates an additional branches in the result. In contrast, there is an additional operations between line 6 and line 7 in readNonProxyDesc(boolean), which is due to the clone gap—some additional assignments. At line 9, the two methods have similar but semantically different function calls—namely initProxy() and initNonProxy(), which indicates an unmatched operations.

REFERENCES

Figure 2. The interface of CloneDiff plugin