Incremental Detection of Parameterized Code Clones

Torsten Görg
University of Stuttgart
Universitaetsstr. 38, 70569 Stuttgart, Germany
torsten.goerg@informatik.uni-stuttgart.de

Abstract: *This paper presents a new approach to incremental code clone detection that is based on a special clone representation model. The algorithm detects parameterized clones with parameters of arbitrary size. It can be used for cross-system and cross-revision clone detection.*

1 Introduction

Many approaches to code clone detection have been developed during the last two decades [1]. Most of them calculate a set of clone pairs for a given source code. To detect clones incremantally is a more advanced problem. Incremental clone detection tries to determine an updated set of clones with minimal effort when source code modules are added or modified [7]. Usually a clone detection algorithm derives some data from the source code and transforms it in further processing steps. E.g., the tree-based algorithm of Baxter et al. calculates a hash map to cluster AST subtrees and uses these clusters to find clone pairs by comparing each subtree with all subtrees in the same cluster [3]. The derived intermediate data has to be recalculated for the whole source code on each subsequent clone detection pass if it is just held transiently for one pass. The whole set of clones has to be recalculated because to search for new clones in the modified or added code separately misses clone pair relationships between old and new code. Our clone detector solves this problem by using a special clone representation model.
Different kinds of clones are distinguished in the clone literature [1]. Our approach is able to detect parameterized code clones incrementally. Parameterized clones are a generalization of gapped clones with arbitrary gaps of any size. A parameterized clone is characterized by an upper and a lower horizontal cut in an AST [4].

2 Clone Representation Model

Our clone detection approach is based on the data model to represent code clones provided in [2]. This model consists of a set of clone groups. Each clone group is characterized by a prototype which is a subtree copied from an AST. This prototype implicitly specifies the upper and lower horizontal AST cuts of the grouped parameterized clones. A code fragment is matched against the prototype of a clone group to check whether it is an adequate clone belonging to that group.
The clone representation model also expresses relationships between the clone groups. A clone group A is called a supergroup of a clone group B iff the prototype of A is a subtree of the prototype of B. Iff A is a supergroup of B then B is called a subgroup of A. This construction guarantees that a supergroup represents the commonalities of its subgroups. The supergroup relation forms a DAG (directed acyclic graph). Optionally, a clone group can hold references to the instances in a given AST.
A further design idea of the model is to encompass not only code fragments that are clones but also fragments that appear just once. In this way the model is equivalent to the original AST.

3 Incremental Detection Approach

The clone detection algorithm suggested in [2] creates a clone representation model for a given AST by successively processing all AST nodes and adding subtrees that are spanned by the processed nodes to the model as clone fragments. For inserting a clone fragment the most appropriate clone group is found following a principle called stepwise flooding (see Fig. 1). The search starts with the smallest clone groups whose prototypes contain a node that matches the currently processed AST node. If the whole prototype can be flooded with coinciding AST nodes the search proceeds with the subgroups.

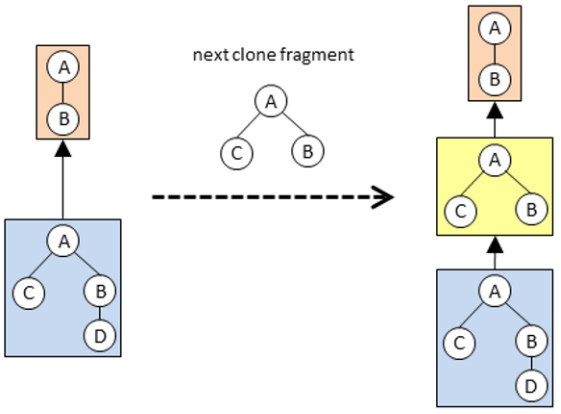

Fig. 1. Example of an insertion of a clone fragment into the clone group hierarchy with stepwise flooding

A new clone group is created and linked into the clone group hierarchy whenever currently uncovered commonalities are detected. Thus the clone group hierarchy is constructed incrementally by adding further clone groups on demand. This incremental fashion of the internal process of the algorithm is exploited to realize an incremental code clone detector.

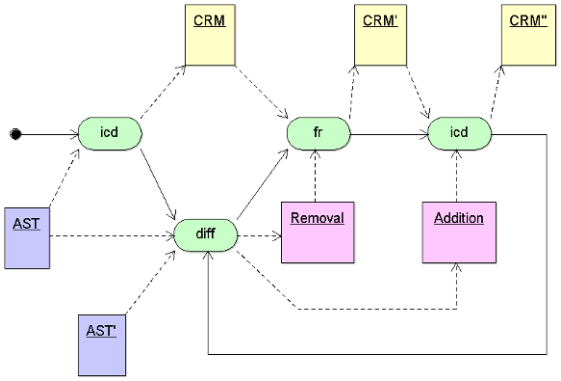

Fig. 2. The incremental clone detection process

The incremental process is depicted in Fig. 2 as an UML activity diagram. After the creation of an initial clone representation model (CRM) with our incremental clone detector (icd) a modified syntax tree (AST') is handled incrementally. Its differences to the previous AST are calculated as additions and removals of subtrees (diff). Modifications are replaced by a removal of the original subtree followed by an addition of the modified subtree.
A fragment remover (fr) processes the required removals, producing CRM'. To insert the additions icd is called again. CRM'' is the updated result model. For each new version of the AST the process is repeated starting with the diff step.
The fragment remover implements an algorithm inverse to the clone detection algorithm in icd. The AST nodes of the removed code parts are processed one by one and the footprint of each node is eliminated in the clone representation model. This potentially includes the deletion of clone groups.
Our implementation is based on the reengineering framework Bauhaus [5]. This implies that the clone detection process can be applied on C and C++ code as Bauhaus provides an appropriate frontend.

4 Related Work

Another approach to incremental tree-based clone detection is presented in [6]. Similar to our algorithm, it finds clone pairs by pairwise comparisons of code fragments that are clustered into buckets. Several heuristics are used, i.e., vector distances are calculated as similarity values and compared with a distance threshold to detect clone pairs. It is not guaranteed that an incremental calculation provides the same clone groups as a non-incremental calculation. Our approach provides always the exact results at the cost of higher calculation effort.
A token-based incremental algorithm is realized in the clone detection tool iClones [7]. It calculates generalized suffix trees to detect clones. Similar to our algorithm modifications are handled as additions and removals, but restricted to file granularity.

5 Conclusion

Together with the inverse algorithm for fragment removal cross-revision clone detection is realized. This allows updating a clone representation model in order to reflect all code clones in modified source code.
The incremental clone detection process makes it possible to successively feed the ASTs of further systems into an existing clone representation model. The algorithm guarantees to detect all parameterized clones without any loss of information even if they are spread over multiple systems. Such cross-system clone detections provide code idioms that are relevant for more than one system.

References

[1] Chanchal Kumar Roy and James R. Cordy, "A survey on software clone detection research," technical report, Queen's University, Canada, 2007.

[2] Torsten Görg, "Mining of Source Code Concepts and Idioms," unpublished, available on http://www.iste.uni-stuttgart.de/ps/goerg.html.

[3] Ira Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna, and Lorraine Bier, "Clone Detection Using Abstract Syntax Trees," in Proceedings of the 14th International Conference on Software Maintenance (ICSM'98), pp. 368-377, Bethesda, Maryland, November 1998.

[4] Torsten Görg, "A Model-Based Approach to Type-3 Clone Elimination," in Proceedings of the 14. Workshop Software-Reengineering (WSR 2012) of the Gesellschaft für Informatik (GI) special interest group Software-Reengineering, pp. 21-22, Bad-Honnef, May 2012.

[5] Aoun Raza, Gunther Vogel, and Erhard Plödereder, "Bauhaus – A Tool Suite for Program Analysis and Reverse Engineering," in Proceedings of Ada Europe 2006, LNCS 4006, pp. 71-82.

[6] Tung Thang Nguyen, Hoan Anh Nguyen, Jafar M. Al-Kofahi, Nam H. Pham, and Tien N. Nguyen, „Scalable and Incremental Clone Detection for Evolving Software," in Proceedings of IEEE International Conference on Software Maintenance (ICSM 2009), pp. 491-494, September 2009

[7] Nils Göde and Rainer Koschke, "Incremental Clone Detection," in Proceedings of the 2009 European Conference on Software Maintenance and Reengineering (CSMR '09), pp. 219–228, 2009