Keeping DRY: A Proposal for a Comprehensive Measure of Similarity Within a Computer Program

J. Hassler Thurston

1 Introduction

I would like to extend Charlie Lehner's and Dan Hassin's Javascript code generation project (https://github.com/RocHack/jschain) to compute a measure of how much a programmer follows the "DRY" principle. "DRY", or "Don't Repeat Yourself", is a programming ideal and standard that urges programmers to avoid duplicating code or structures of code in order to make programming more efficient and readable. By designing and implementing such a system, we can not only come up with a novel method of evaluating DRY, but we can also encourage amateur programmers to use such a program to improve their coding style.

2 Research

Much of the work already done on program similarity involve evaluating the similarity of two different programs for plagiarism detection. Joy, Joy, and Luck [1] note that much of student plagiarism involves changing a program structurally (e.g. replacing a for loop with a while loop) and lexically (e.g. rewording comments). They then propose an "incremental comparison method" for detecting plagiarism, whereby they examine multiple modifications of two programs for similarity. Whale [2] takes a slightly different approach, whereby he counts the number of occurrences of certain attributes of programs (such as the number of unique operators and operands), and combines this metric with a similarity measure over optimized versions of code at runtime. Both approaches involve detecting a multitude of pre-defined lexical and structural differences between two programs.

3 Proposal

Although much of the current research on program similarity centers around plagiarism detection, similar techniques can nonetheless be used to evaluate

similarity *within* a program. I propose to analyze blocks of a program for lexical and structural similarities similar to [2]. For example, given two methods of a Java program, I will compare/contrast them by first counting the number of occurrences of certain attributes of these two methods, and then analyze the methods for structural similarity. I can then combine the results of these two metrics to produce a real number between 0 and 1, that will measure the amount of similarity between those two methods. If I then were to calculate this similarity between all methods in a Java program, and calculate measures of similarity between other attributes of the Java code, such as expressions and conditionals, I would be able to compute a measure of internal similarity for the program itself. This would also take the form of a real number between 0 and 1 (0 being completely "dry"), and would constitute the "dryness" metric.

It is important to note that this metric can be extended to other programming languages besides Java, although I propose to solely focus on Java due to its enormous popularity among amateur programmers. As an extension, I will also try to implement a system that gives an amateur programmer suggestions on how to improve their code to be more "dry". To do this, my program would first point out to the programmer which attributes of their existing program are not dry, and then suggest different ways of structuring their program that would decrease the dryness metric. Due to the undecidability of detecting whether two programs produce the same output, however, I suspect my program will not be able to give a programmer concrete recommendations of how to modify their existing program.

4 Conclusion

Evaluating the dryness of a program is currently considered to be an unsolved problem, even though there has been a lot of research on a similar aspect of program similarity – that is, examining two different programs for plagiarism detection. My proposal however involves computing a measure of similarity *within* a Java program, by computing a similarity metric between parts of a Java program. I then would ultimately like to give an amateur programmer recommendations on how to improve their code to be more dry, which I hope will help advance their understanding of concepts related to code duplication and structural efficiency.

References

[1] M. Joy, M. Joy, and M. Luck. Plagiarism in programming assignments. *IEEE Transactions on Education*, 42(2):129–133, 1999. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=762946.

[2] G. WHALE. Identification of program similarity in large populations. *COMPUTER JOURNAL*, 33(2):140–146, 1990. http://gateway.webofknowledge.com/gateway/Gateway.cgi?GWVersion=2\&SrcAuth=SerialsSolutions\&SrcApp=Summon\&KeyUT=A1990CZ98300007\&DestLinkType=FullRecord\&DestApp=WOS.