Greenlee 560 Summary 9

Elliot Greenlee

April 13, 2017

1 Introduction

This paper covers work done in automatically finding errors in static source code. Previous to this, the team attempted to find errors by manually specifying system rules to be checked. After their improvement to automatic belief finding, the program is finding x10 to x100 more rules. There is also similar work being done in finding errors in the Daikon and Eraser projects; however, both of these use dynamic analyses rather than static.

2 Problems

What problems does this paper try to solve? Why are such problems important?

It is difficult and time consuming for programmers to learn the details of a new system. This compounds further in attempting to discover bugs in the code that runs the system. It is important that this time cost be removed for two closely related reasons. Bugs must be found in order to prevent security breaches and other errors from occurring, and programmers must frequently spend time searching for and removing these bugs.

3 Assumptions

What are assumptions made by this paper? Are they verifiable? Are there logical holes in such assumptions?

The writers assume that the biggest difficulty in discovering bugs is understanding the system in order to know what rules to check. For each system, this rule set must be re-discovered and applied to finding bugs. There are other difficulties in discovering bugs when applied to the manual approach; however, it would seem that they are correct in that finding rules is the main challenge with automatic approaches. Creating templates to apply to code must still be done manually even in the automated case, which shows that indeed other problems must be solved even after rule sets can be automated.

The writers also assume that their system is properly finding errors in the analyzed systems. However, they acknowledge this assumption in the paper,

and have corrected for it by sending bugs found to the main developers. Many of those sent bugs resulted in major and immediate patches.

4 Solutions

What are the major solutions of this paper? Do you think the solutions in this paper will work for the problem? Do you think the paper evaluates the solutions in a convincing manner? List two limitations of the solutions proposed in this paper, and outline your method to fix them.

This paper details two methods for automatically finding bugs in a system without directly coding rules the system must obey. These solutions are evaluated very thoroughly through multiple implementations and case studies, and have already returned good results with finding kernel errors that were immediately patched.

4.1 Internal Consistency

In this approach, rules of absolute certainty are compared against each other. When two contradict, each is flagged as an error to be reported. This implementation, when given the correct templates to search for, works very well. It has a challenge with analyzing all possible paths statically, but experimentally has held up well.

4.2 Statistical Analyses

When rules are not certain and more noise is present, this approach compares large samples of similar rules in order to determine the rules with the fewest errors. It assumes that these have the least false positives, since programmer is assumed to be mostly correct. Results are then organized from least to most errors, with the lower sections becoming more and more noisy with false positives.

4.3 Limitations

This code is still system specific in some aspects. Currently program templates must be implemented for each case, rather than discovered. Additionally, some variable names and functions have specific properties in certain domains, which again must be programmed manually. Solving this issue using machine learning seems like the most likely approach, and is what the team intends to do in future work.

Currently the system is not supplemented with a dynamic component. The writers acknowledge that this is a limitation they have that would be beneficial to correct. Dynamic analyses could allow the most common use cases to be explored, which could help to predict errors and aid in the ranking process.