Ben Mishkanian

Project Report Draft 1

Introduction

This paper investigates the differences in bug-finding effectiveness and efficiency between different static analysis tools for C. We look at various static analyzers, including Cppcheck, CPAChecker, clang static analyzer, and Frama-C. These four tools are free for non-commercial use. The tools use different bug-finding algorithms, and as a result they different in the number, type, and quality of bugs they report. Cppcheck checks for bounds violations, memory leaks, null pointer dereferences, uninitialized variables, invalid STL usage, exception safety, unsafe function usage, and dead code. CPAChecker performs formal verification by checking if an error state is reachable. clang static analyzer can check for division by zero, null pointer dereferences, uninitialized variables, dead code, insecure API usage, and other platform-specific errors. Frama-C is a suite of tools that allows supplying and proving functional specifications.

Motivating Example

Consider the following code snippet:

```
int main(void) {
  int *p;
  *p = 1;
}
```

This should clearly generate a null pointer dereference warning from the static analyzers, but the results vary. Given the default options, clang static analyzer fails to find any problem. CPAChecker can not find the bug unless a specific specification is provided, as it can only validate adherence to a specification, rather than finding arbitrary bugs. It would be interesting to see what types of bugs are detected by the different tools, and how the tool's configuration options affects the result.

Description of technical approach

I will create some simple programs with sample bugs of various types and run these static analyzers on those programs. When a tool fails to find a bug that it should have found (according to the tool's feature set description), I will try changing the tool's parameters to improve the performance, and take note of what changes were required. This will provide some qualitative information on what type of bugs the analyzers can detect, and what settings need to be enabled for the bugs to be detected. Then, to check performance, I will run the tools on a large open source project to see how long it takes to complete execution. The target projects will be taken from Github. I will run the tools from whatever environment and operating system are recommended by the authors, as long as I can conveniently obtain that environment.

Related work

In class, we learned about KLEE, an automated symbolic execution system that generates and executes test cases to find bugs. A similar tool is CUTE (now known as CREST), which is a concolic testing tool used to achieve high code coverage. Although these tools can be effective in finding bugs, they require manually instrumenting the source code, so it is infeasible for me to run a user study on their performance. Furthermore, CREST does not generate bug reports; it only reports the code coverage it achieved. There are also plenty of other static analysis tools for C, but many of them are commercial software. In this paper, I will focus on the four free static analyzers mentioned above, because they are readily available and generate bug reports without having to instrument the code.

Evaluation Methodology

One way to evaluate the tools is to analyze the types of bugs it can find. Since CPAChecker and Frama-C use specifications to define failure conditions, they may not be directly comparable to Cppcheck and clang static analyzer. However, I can try to find specifications that correspond to the bug classes detectable by those two tools. I will run the tools on sample programs that contain a clear case of one bug class, and see which tools detect each class of bug.

Another evaluation dimension is the number of bugs found. I will run the tools on a few different open source projects to see how many bugs are found in each project by each tool. In this way, I can discover if a tool consistently finds more bugs than another tool. I can also do some case studies to see what types of bugs are commonly caught by all tools, what types are rarely caught, and how useful the bug reports are. If possible, I would like to do some qualitative analysis on the bugs reported, to see if the most salient reports correspond to important bugs, or if they are false positives, or duplicates of other bugs.

One other measure of the usefulness of these tools is the extent of the modifications and customizations required to make the tool find bugs. In some cases, a tool that finds a few bugs “out of the box” is preferable to a tool that finds no bugs with default settings and many bugs with major customizations.

Finally, I will evaluate the runtime performance of the tools by running them on a large open source and measuring the amount of time required for each tool to finish running. I then take the number and quality of bug reported into account to give a rough measurement of how efficient the tools are in finding real distinct bugs.