Researching Static Analyzers: Cppcheck and Clang

JOSH PHILLIPS

UNIVERSITY OF CINCINNATI

Introduction

- Goal: Create a resource for identifying the best static analyzer to use for any given C++ project
- Research various static analyzers and compare their soundness, completeness, cost, and effectiveness

Background

- Basic knowledge of writing and compiling C++ programs

Preliminary Results

- Limited knowledge
  - Project was mainly research for this reason

Merit

- Inform myself and others about two popular static analyzers, Cppcheck and Clang
- Compare Cppcheck and Clang based on:
  - Soundness
  - Completeness
  - Cost
  - Effectiveness

Cppcheck Background

- Unique code analysis to detect bugs and focuses on detecting:
  - undefined behavior
  - dangerous coding constructs
- Very few false alarms
- Unsound flow sensitive analysis
  - this is claimed as a positive because other analyzers will more commonly use path sensitive analysis

Cppcheck Features

- Automatic variable checking
- Bounds checking
- Classes checking
- Exception Safety checking
- Memory leaks
- Resource leaks
- Invalid usage of STL
- Dead code elimination
- Stylistic and performance errors

Cppcheck Undefined Behavior

- Dead pointers
- Division by zero
- Integer overflows
- Invalid bit shift operands
- Invalid conversions
- Invalid usage of STL
- Memory management
- Null pointer dereferences
- Out of bounds checking
- Uninitialized variables
- Writing const data

Soundness of Cppcheck

- Cppcheck uses unsound flow sensitive analysis
  - In theory this would not be as good as having path sensitive analysis
  - In practice it means Cppcheck can detect bugs other static analyzers cannot
  - Data flow analysis is bi-directional
    - Could detect the following while other analyzers may not

```
void foo(int x)
{
   int buf[10];
   buf[x] = 0; // <- ERROR
   if (x == 1000) {}
}
```

Cost and Effectiveness of Cppcheck

- Cost of analysis will be proportional to the size of the program
- Effectiveness is unsound and somewhat complete
  - May miss errors other tools do not detect
  - Few false alarms meaning it will not report spurious errors

Clang Background

- The Clang Static Analyzer is a source code analysis tool that finds bugs in C, C++, and Objective-C programs
- Can be run from the command line in tandem with compiling a program
- Analyzes source code to find bugs
- Identifies bugs that are traditionally found during run time

Clang – a work in progress

- Open source project currently working on improving precision and scope of Clang

Soundness and Completeness of Clang

- Reports many false positives so Clang is incomplete

- This is a major drawback of static analyzers and again Clang is still a work in progress

Cost and Effectiveness of Clang

- Cost is can be very expensive with Clang:
  - Finding some bugs may grow the search time exponentially
  - If you are running Clang in tandem with the compiler than this can result in much longer compile times than expected
- Effectiveness of Clang is both unsound and incomplete:
  - Clang may miss errors making it unsound
  - Clang may report spurious errors making it incomplete

Results and Takeaways

- Generally the cost of static analyzers are proportional to their size
- Static analyzers are also usually incomplete as they may report spurious errors
- From different static analyzer to analyzer the types of bugs they will detect and their levels of Soundness and Completeness can change dramatically
- I recommend using more than one static analyzer at a time as they will often find different bugs when searching on the same program.

Limitations and Further Research

- With more time I would’ve liked to further the research by gathering data on the number and types of bugs found by these analyzers on various C++ programs
- Further research would allow me to compare different static analyzers in an empirical way
- I would also like to look into other analyzers in further research

Sources:

Clang Static Analyzer (llvm.org)

http://cppcheck.sourceforge.net/