Static Analysis of C++ Projects with CodeSonar

John Plaice, Senior Scientist, GrammaTech
`jplaice@grammatech.com`

25 July 2017, Meetup C++ de Montréal

Abstract

Static program analysis consists of the analysis of a computer program without actually executing it. Common problems detected using static analysis include taint analysis, null pointer detection, and buffer overruns. This talk introduced the topic of static analysis, and gave a demo using CodeSonar, the flagship product of GrammaTech. This short paper summarizes the talk.

1 What is Static Analysis?

Static analysis infers information about software behavior based on a static representation of the software, without actually running it. It is not the same as simulation, nor does it require sample input or test cases. Static analysis usually is split into a two-phase process:

- Extract semantic information from source code;
- Use information to discover defects or other properties of interest.

2 Applications of Static Analysis

- Stronger type checking;
- Checking conformance with coding guidelines;
- Software metrics;
- Software architecture analysis;
- ***Finding bugs!***

3 Benefits of Static Analysis

Static analysis examines far more execution paths than conventional testing will ever do. Furthermore, static analysis can be applied automatically, right from the beginning of the development cycle. In general, static analysis

- Catches problems that test suites miss;
- Catches bugs early, when they are less expensive to fix;
- Pinpoints defects automatically, improving productivity;
- Shows if code quality is improving over time;
- Improves software quality (and security);
- Reduces time to market.

4 Why Use Advanced Static Analysis?

- Conventional testing is only as good as the test cases.
- The number of paths in a program far exceeds the number exercised by conventional testing.
- Most paths go untested, even in the most stringent approaches to testing.
- Tools like GrammaTech CodeSonar work directly on the code.
- They find interesting and serious bugs.
- Heuristics are used to avoid uninteresting results.
- Easy to apply in the development cycle.
- Much more scalable than formal verification systems using model checking.

5 Not All Flaws Are Equal

Easy-to-find flaws

- Typically structural or syntactic issues;
- Usually unambiguous;
- Often unlikely to cause problems in the field;
- May also be easily revealed by testing.

Hard-to-find flaws

- Typically semantic;
- Require advanced analysis to find;
- Prone to false positives;
- Usually expensive to fix in the field;
- Hard to find with conventional testing.

6 What Types of Bugs Can Be Identified

Static-analysis tools can catch

Language misuse: e.g., buffer overruns and null-pointer dereferences;

Library misuse: e.g., calling `accept()` on a socket in wrong state;

Suspicious code: e.g., dead code, redundant conditions.

CodeSonar also provides an API to add custom checkers

- e.g., do not call `foo()` from within an interrupt handler.

Static-analysis tools cannot catch

- Bugs that have to do with the functionality of an application
  e.g., "This algorithm calculates the wrong answer."

7 Some CodeSonar Checks

- Buffer Overrun
- Null-pointer Dereference
- Divide by Zero
- Uninitialized Variable
- Free Non-heap Variable
- Use after Free
- Double Free/Close
- Free Null Pointer
- Format String Vulnerability
- Race Conditions
- Unreachable Code
- Memory/Resource Leak
- Return Pointer to Local
- Mismatched Array New/Delete
- Invalid Parameter
- Missing Return Statement
- Dangerous Cast
- Inconsistent Form
- User-defined Checks
- ***Many More...***

8 Static Analysis Metrics

Given

$$\begin{aligned} TP &= \text{number of true positives} \\ FP &= \text{number of true negatives} \\ TN &= \text{number of false positives} \\ FN &= \text{number of false negatives} \end{aligned}$$

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \end{aligned}$$

Perfect Recall: No false negatives.

Perfect Precision: No false positives.

Ideal Performance: Runs fast, uses little memory.

9 The Core Tradeoff

Perfect Recall is easy (no false negatives)

- "All points in your program have bugs," or
- "This tool only finds the easy-to-find bugs."

Perfect Precision is easy (no false positives)

- "There are no bugs in your program," or
- "This tool only reports the obvious bugs."

Ideal Performance is easy

- Either one of the above, or guess randomly.

Core tradeoff

- Precision vs Recall vs Performance simultaneously for serious flaws.

10 Symbolic Execution

Uses techniques from model checking and abstract interpretation.

- Using the control-flow graph (CFG):
  - "Simulate" executions of a function.
  - Use abstract values instead of concrete values:
    - Domain of abstract values are typically equations describing relations between variables: "$x < 0$", "$i = 2 \times j$."
  - Explore multiple paths.
  - Use "summaries" at call sites.
- Look for anomalies:
  - e.g., at "`x = *p`", "`p == NULL`" $\rightarrow$ Null pointer dereference.
- Use heuristics to suppress false positives.

11 Path Exploration

All serious path-sensitive analyses limit path exploration.

- Need to keep time and space requirements roughly linear in the size of the program;
- Limit on number of paths or time spent explored per procedure;
- Some paths not considered:
  - `setjmp/longjmp`, Indirect function calls, Context switches, Recursion;
  - Current solutions are coarse and partial.

12 Demo

I gave a demo of the CodeSonar tool, finding flaws *in vivo* in the latest GNUchess program. A lot of discussion took place.

13 When to Deploy?

Recommendation: Deploy early in development cycle

- As soon as code is compilable.
- The cheapest bug is the one you find earliest.
- Avoids last-minute avalanche of problems.
- Get programmers "writing for the tool":
  - Encourages good style, and makes it easier for the tool.
- Get programmers extending the tool:
  - When a bug does surface: "How could I have written the code so that the Static Analysis tool would have found this?"

Workflow?

- Do a partial analysis before committing changes:
  - Not as accurate as a full analysis, but faster.
- Do a complete analysis triggered by changes:
  - New items can be correlated with the changes in a just-in-time fashion.

14 For more info...

Please have a look at our Web site: `http://www.grammatech.com`