A Brief Introduction to Static Analysis

Sam Blackshear

March 13, 2012

Outline

- **A theoretical problem and how to ignore it**
- An example static analysis
- What is static analysis used for?
- Commercial successes
- Free static analysis tools you should use
- Current research in static analysis

An Interesting Problem

- We've written a program P and we want to know...
- Does P satisfy property of interest ϕ (for example, P does not dereference a null pointer, all casts in P are safe, ...)
- Manually verifying that P satisfies ϕ may be tedious or impractical if P is large or complex
- Would be great to write a program (or *static analysis* tool) that can inspect the source code of P and determine if ϕ holds!

An Inconvenient Truth

We cannot write such a program; the problem is undecidable in general!

- Rice's Theorem (paraphrase): For any nontrivial program property ϕ, no general automated method can determine whether ϕ holds for P.
- Where nontrivial means we can there exists both a program that has property ϕ and one that does not.

So should we give up on writing that program?

A Way Out

Only if not getting an exact answer bothers us (and it shouldn't).

Key insight: we can abstract the behaviors of the program into a decidable *overapproximation* or *underapproximation*, then attempt to prove the property in the abstract version of the program

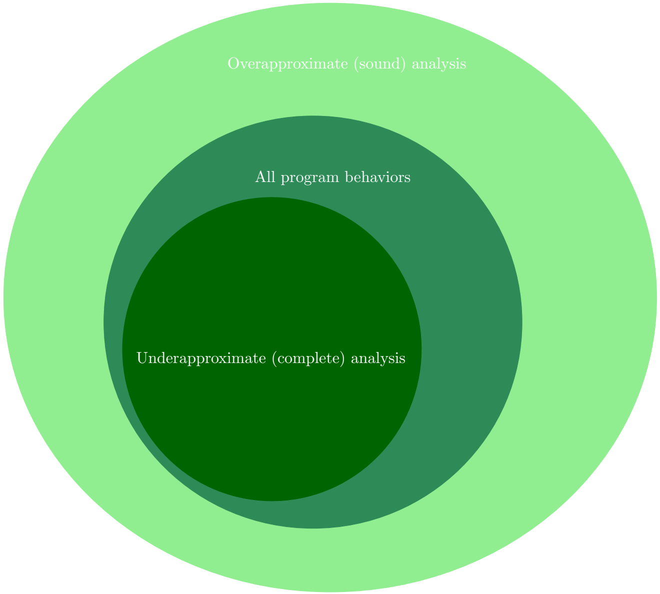
This way, we can have our decidability, but can also make very strong guarantees about the program

Analysis Choices

- A *sound* static analysis overapproximates the behaviors of the program. A sound static analyzer is guaranteed to identify all violations of our property ϕ, but may also report some "false alarms", or violations of ϕ that cannot actually occur.
- A *complete* static analysis underapproximates the behaviors of the program. Any violation of our property ϕ reported by a complete static analyzer corresponds to an actual violation of ϕ, but there is no guarantee that all actual violations of ϕ will be reported

Note that when a sound static analyzer reports no errors, our program is guaranteed not to violate ϕ! This is a powerful guarantee. As a result, most static analysis tools choose to be sound rather than complete.

Visualizing Soundness and Completeness


Overapproximate (sound) analysis
All program behaviors
Underapproximate (complete) analysis

Outline

- *A theoretical problem and how to ignore it*
- **An example static analysis**
- What is static analysis used for?
- Commercial successes
- Free static analysis tools you should use
- Current research in static analysis

An Example Static Analysis : Sign Analysis

Classic example: sign analysis!
Abstract *concrete domain* of integer values into *abstract domain* of signs (-, **0**, **+**) $\cup$ $\{\top,\bot\}$
Step 1: Define abstraction function `abstract()` over integer literals:
-`abstract`(i) = - if $i < 0$
-`abstract`(i) = **0** if $i == 0$
-`abstract`(0) = **+** if $i > 0$

Sign Analysis (continued)

Define *transfer functions* over abstract domain that show how to evaluate expressions in the abstract domain:

- $\mathbf{+} + \mathbf{+} = \mathbf{+}$
- $\mathbf{-} + \mathbf{-} = \mathbf{-}$
- $\mathbf{0} + \mathbf{0} = \mathbf{0}$
- $\mathbf{0} + \mathbf{+} = \mathbf{+}$
- $\mathbf{0} + \mathbf{-} = \mathbf{-}$

  $\vdots$
- $\mathbf{+} + \mathbf{-} = \top$ (analysis notation for "unknown")
- $\{\mathbf{+}, \mathbf{-}, \mathbf{0}, \top\} + \top = \top$
- $\{\mathbf{+}, \mathbf{-}, \mathbf{0}, \top\} \,/\, \mathbf{0} = \bot$ (analysis notation for "undefined")

  $\vdots$

An Example Static Analysis (continued)

Now, every integer value and expression has been abstracted into a $\{\mathbf{+}, \text{-}, \mathbf{0}, \top, \bot\}$. An exceedingly silly analysis, but even so can be used to:

- Check for division by zero (as simple as looking for occurrences of $\bot$)
- Optimize: store $\mathbf{+}$ variables as unsigned integers or $\mathbf{0}$'s as `false` boolean literals
- See if (say) a banking program erroneously allows negative account values (see if `balance` variable is - or $\top$)
- More?

Outline

- *A theoretical problem and how to ignore it*
- *An example static analysis*
- **What is static analysis used for?**
- Commercial successes
- Free static analysis tools you should use
- Current research in static analysis

What Are Static Analysis Tools Used For?

To name a few:

- Compilers (type checking, optimization)
- Bugfinding
- Formal Verification

Compilers - Type Checking

Most common and practically useful example of static analysis

- Statically ensure that arithmetic operations are computable (prevent adding an integer to a boolean in C++, for example)
- Guarantee that functions are called with the correct number/type of arguments
- Real-world static analysis success story: “undefined variable analysis” to ensure that an undefined variable is never read
  - Common source of nondeterminism in C; causes nasty bugs
  - Analysis built in to Java compiler! Statically guarantees that an undefined variable will never be read

```
Object o; o.foo();
```

Compiler: “Variable o might not have been initialized”!

Compilers - Optimization

Examples:

- Machine-aware optimizations: convert `x*2` into `x + x`
- Loop-invariant code motion: move code out of a loop

  ```
  int a = 7, b = 6, sum, z, i;
  for (i = 0; i < 25; i++)
    z = a + b;
    sum = sum + z + i;
  ```

  Lift `z = a + b` out of the loop
- Function inlining - save overhead of procedure call by inserting code for procedure (related: loop unrolling)
- Many more!

Bugfinding

Big picture: identify illegal or undesirable language behaviors, see if program can trigger them

- Null pointer dereference analysis (C, C++, Java ...)
- Buffer overflow analysis: can the program write past the bounds of a buffer? (C, C++, Objective-C)
- Cast safety analysis: can a cast from one type to another fail?
- Taint analysis: can a program leak secret data, or use untrusted input in an insecure way? (web application privacy, SQL injection, ...)
- Memory leak analysis: is `malloc()` called without `free()`? (C, C++) Is a heap location that is never read again reachable from the GC roots? (Java)
- Race condition checking: Can threads interleave in such a way that threads t_1 and t_2 simultaneously access variable x, where at least one access is a write?

Formal Verification

Given a rigorous complete or partial `specification`, prove that no possible behavior of the program violates the specification

- Assertion checking - user writes `assert()` statements that fail at runtime if assertion evaluates to false. We can use static analysis to prove that an assertion can never fail
- Given a specification for an algorithm and a formal semantics for the language the program is written in, can prove that the *implementation* (not just the algorithm!) is correct. Note: giving specification is sometimes harder than checking it! Example: Specification for sorting. How would you define? Takes input ℓ, 0-indexed array of integers, returns ℓ', 0-indexed array of integers, where:

Formal Verification

Given a rigorous complete or partial `specification`, prove that no possible behavior of the program violates the specification

- Assertion checking - user writes `assert()` statements that fail at runtime if assertion evaluates to false. We can use static analysis to prove that an assertion can never fail
- Given a specification for an algorithm and a formal semantics for the language the program is written in, can prove that the *implementation* (not just the algorithm!) is correct. Note: giving specification is sometimes harder than checking it! Example: Specification for sorting. How would you define? Takes input ℓ, 0-indexed array of integers, returns ℓ', 0-indexed array of integers, where:
  (1) for i in [0, length(ℓ') - 1) , $\ell'[i] \leq \ell'[i+1]$ (obvious)

Formal Verification

Given a rigorous complete or partial `specification`, prove that no possible behavior of the program violates the specification

- Assertion checking - user writes `assert()` statements that fail at runtime if assertion evaluates to false. We can use static analysis to prove that an assertion can never fail
- Given a specification for an algorithm and a formal semantics for the language the program is written in, can prove that the *implementation* (not just the algorithm!) is correct. Note: giving specification is sometimes harder than checking it! Example: Specification for sorting. How would you define? Takes input ℓ, 0-indexed array of integers, returns ℓ', 0-indexed array of integers, where:
  (1) for i in [0, length(ℓ') - 1) , $\ell'[i] \leq \ell'[i+1]$ (obvious)
  (2) ℓ' is a permutation of ℓ (subtle!)

Outline

- *A theoretical problem and how to ignore it*
- *An example static analysis*
- *What is static analysis used for?*
- **Commercial successes**
- Free static analysis tools you should use
- Current research in static analysis

Commercial Successes

- Astree
- Coverity
- Microsoft Static Driver Verifier
- Java Pathfinder
- Microsoft Visual C/C++ Static Analyzer

Astrée

Goal: Prove absence of undefined behavior and runtime errors in C (null pointer dereference, integer, overflow, divide by zero, buffer overflow, . . .)

- Developed by INRIA (France), commercial sponsorship by Airbus (aircraft manufacturer)
- Astrée proved absence of errors for 132,000 lines of flight control software in only 50 minutes!
- Has also been used to verify absence of runtime errors in docking software used for the International Space Station
- Over 20 publications on techniques developed/used

More information at `http://www.astree.ens.fr/`, `http://www.absint.com/astree/index.htm`

Coverity

Goal: Catch bugs in C, C++, Java, and C# code during development

- Commercial spinoff of Stanford static analysis tools
- Analysis is neither sound nor complete; emphasis is on finding bugs that are *easy to explain to the user:* "Explaining errors is often more difficult than finding them. A misunderstood explanation means the error is ignored or, worse, transmuted into a false positive."
- Used by over 1100 companies!
- Very interesting CACM article on challenges of commercializing a research tool *A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World*.

`http://www.coverity.com`

Microsoft Static Driver Verifier (SLAM)

Goal: Find bugs in Windows device drivers written in C

- Major effort in defining what constitutes a bug; requires developing rigorous model for operating system and specifying what behaviors are unacceptable
- Uses *model checking*: intelligently enumerating and exploring state space of program to ensure that no error states are reachable from the initial state
- Tool is sound and quite precise (5% false positive rate)
- Works in a few minutes for typical drivers; up to 20 minutes for complex ones

`msdn.microsoft.com/en-us/windows/hardware/gg487506`

Java PathFinder

Goal: find bugs in mission-critical NASA code

- NASA-developed model checker that runs on Java bytecode
- Specialty is detecting concurrency errors such as race conditions; also handles uncaught exceptions
- Now free and open source! Can configure JPF to check properties you are interested in by adding plugins
- Drawback: can't handle native Java libraries, must use models instead
- Sound, but not very precise

`http://babelfish.arc.nasa.gov/trac/jpf`

Microsoft Visual C/C++ Static Analyzer

- Invoke using `/analyze` flag when compiling
- Emphasis on finding security-critical errors such as buffer overruns and `printf` format string vulnerabilities
- Neither sound nor complete; lots of heuristics, but finds lots of bugs
- Developer legend John Carmack (of Quake fame) is a big fan: `http://altdevblogaday.com/2011/12/24/static-code-analysis/`

`http://msdn.microsoft.com/en-us/library/ms182025.aspx`

Outline

- *A theoretical problem and how to ignore it*
- *An example static analysis*
- *What is static analysis used for?*
- *Commercial successes*
- **Free static analysis tools you should use**
- Current research in static analysis

Some (Good) Free and Open Source Static Analysis Tools

- Clang static analyzer
- FindBugs
- WALA
- VELLVM

Clang Static Analyzer

- Part of llvm compiler infrastructure; works only on C and Objective-C programs
- Over 30 checks built into default analyzer
- Built for debugging iOS apps, so includes extensive functionality for finding memory problems
- Neither sound nor complete
- Can suppress false positives reported by tool by adding annotations to code
- Very snappy IDE Integration with Xcode
- Support for C++ coming soon!

`http://clang-analyzer.llvm.org/`

FindBugs

- University of Maryland research tool by David Hovemeyer and Bill Pugh (of SkipList fame)
- Rather than using rigorous formal methods, focus is on heuristics; in particular, identifying common "bug patterns"
- Looks for 45 bug patterns such as "equal objects must have equal hashcodes" and "wait not in loop"
- Bugs are given a "severity rating" from 1 - 20
- No soundness or completeness guarantee, but proven to be very useful in practice
- Integration with numerous IDEs including Eclipse and NetBeans

`http://findbugs.sourceforge.net/`

SAFECode + Vellvm

- Research tool from UPenn and UIUC
- Automatically add memory safety to existing C source code: no buffer overflows, dangling pointers, e.t.c
- All you have to do is recompile your source using llvm-clang with the SAFECode flag
- Runtime overhead only 10-40%; small price to avoid getting hacked!
- Implementation of compiler transformation proven correct with proof assistant (Coq). Very strong guarantee!

`http://sva.cs.illinois.edu/downloads.html`

Outline

- *A theoretical problem and how to ignore it*
- *An example static analysis*
- *What is static analysis used for?*
- *Commercial successes*
- *Free static analysis tools you should use*
- **Current research in static analysis**

Current Research in Static Analysis: a Taste

Too many topics to enumerate here, but will mention a few:

- Concurrency - the state spaces of large concurrent programs are much too large to explore exhaustively. How can we make guarantees about the correctness of concurrent programs while only exploring a fraction of this space?
- JavaScript and other dynamic languages - how can we deal with dynamic behavior like reflection and dynamic code evaluation with `eval()`?
- Checking deeper properties - most static analyses prove properties that must be true of all programs in their target language. How can we automatically infer and check properties that correspond more closely to program correctness?

Further Reading

- Foundations of Static Analysis - *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.* Patrick Cousot and Radhia Cousot. POPL 1977.
- Astrée - *A Static Analyzer for Large Safety-Critical Software.* Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. PLDI 2003.
- Coverity - *Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code*. Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, and Benjamin Chelf. SOSP 2001.

Further Reading (continued)

- Microsoft Static Driver Verifier/SLAM - *Automatic Predicate Abstraction of C Programs*. Thomas Ball, Rupak Majumdar, Todd D. Millstein, Sriram K. Rajamani, PLDI 2001.
- Java PathFinder - *Test Input Generation with Java PathFinder*. W. Visser, C. Pasareanu, S. Khurshid. ISSTA 2004.
- FindBugs - *Finding Bugs is Easy*. David Hovemeyer and William Pugh. OOPSLA 2004.
- SAFECode - *Formalizing the LLVM Intermediate Representation for Verified Program Transformation.* Jianzhou Zhao, Santosh Nagarakatte, Milo M K Martin and Steve Zdancewic. POPL 2012

Further Reading (still continued)

- Analysis of concurrent programs - *Reducing Concurrent Analysis Under a Context Bound to Sequential Analysis*. Akash Lal and Thomas Reps. Formal Methods in System Design (FMSD) 2009.
- Static analysis of JavaScript - *The Essence of JavaScript*. Arjun Guha, Claudiu Saftoiu, Shriram Krishnamurthi. ECOOP 2010.
- Specification inference - *Probabilistic, Modular and Scalable Inference of Typestate Specifications*, Nels E. Beckman and Aditya V. Nori. PLDI 2011.

Executive Summary

- Static analysis is infeasible in theoretical terms, but quite feasible in practice
- Static analysis has been used in industry for important applications such as finding bugs in aircraft software
- There are free and open source static analysis tools that can help you find common problems in your code
- You should use them!
- Current research in static analysis is attacking interesting problems that will continue to push the boundaries of automated reasonng