Scalable and precise abstractions of programs for trustworthy software

Matthew Might  
University of Utah

David Van Horn  
University of Maryland

March 9, 2015

Abstract

This document gives an overview of the Scalable and precise abstractions of programs for trustworthy software project which was supported by the DARPA Information Innovation Office, Automated Program Analysis for Cybersecurity program.

1 Executive summary

Motivation. Applications deployed on mobile devices play a critical role in the fabric of national cyberinfrastructure. They carry sensitive data and have capabilities with significant social and financial effect. Yet while it is paramount that such software is trustworthy, these applications pose challenges beyond the reach of current practice for low-cost, high-assurance verification and analysis. These programs are large, modular, and interactive. They communicate with distributed services that are fundamentally unavailable for analysis. They are written in expressive high-level, higher-order programming languages, for which traditional “Fortran-style” approaches to analysis simply do not apply.

Completed work. We investigated a systematic and scalable approach to the fully automatic analysis and verification of applications deployed on mobile devices.

Technical keystone. To make program analysis for cybersecurity economically feasible, scalability and precision must both improve by orders of magnitude. We conjectured scalability and precision—often seen as competing concerns—are inseparable instances of the same problem. Based on recent work [14], we hypothesized that to significantly improve scalability, precision must be increased to such a degree that swaths of false-positive analytic state-space are eliminated.

Thrusts. With the goal of substantially elevating precision (and speed) in static analysis, we proposed the following novel, transformative techniques:

1. A method for the systematic abstraction of object-oriented languages. Van Horn and Might’s systematic abstraction [13, 5, 18] exposes the full inventory of parameters to tune the precision of a static analysis. These parameters induce an analytic framework that spans a
continuum from the null analysis up to the concrete semantics, with intermediate points that include classical data-flow analysis; rich abstract interpretations; and symbolic execution.

2. A family of locally non-monotonic techniques for abstract transfer functions. We propose “locally non-monotonic transfer functions,” and argue that they are essential to reducing false positives. Locally non-monotonic analyses can revoke judgments made across transition between states, thereby avoiding the inevitable merging that creates false positives. Global monotonicity still guarantees termination.

3. A mobile contract infrastructure for Java, with corresponding higher-order generalizations of relational abstract domains. Contracts are executable specifications that sit at the boundary between software components. Contracts are a run-time enforcement mechanism, but our approach will leverage contracts as symbolic values for compile-time symbolic execution. By carrying out symbolic execution with contracts, we can verify rich behavioral properties with minimal false-positives. A higher-order generalization of relational abstract domains—entangled abstract domains—will enable us to conduct such symbolic execution.
Objective: Advanced static analysis techniques for secure mobile Java applications.

**Technical Keystone**

Exploit local non-monotonicity to disrupt the time-precision cost curve and to gain access to this “paradoxical” region:

Goal: Order-of-magnitude improvements to both time and precision in automated security analysis:

Cleaving false positives

**Unique Aspects**

- Seeks to exploit transformative hypothesis linking speed to precision
- Engages well-meshed team with extensive high-impact collaboration.
- Utilizes static software security contracts to enforce system security.
- Leverages systematic abstraction method discovered by the team.
- Exploits powerful, nontraditional small-step analytic framework.
- Ports “secret weapons” from functional analysis to object-orientation.
- Keeps GPU-based acceleration open as option for improving speed.
- Seeks to handle full, unrestricted Java, including reflective techniques.
- Works inherently with parallel code due to use of small-step method.

**Innovations**

- Adapts and deploys abstract garbage collection to objects in Java.
- Devises an array of locally non-monotonic methods for precision.
- Resolves null-field initialization problem with anodized addresses.
- Develops and deploys “entangled” abstract domains for objects.
- Extends pushdown return-flow analysis to object-oriented languages.
- Enables optimal polyvariance-learning to autobalance time/precision.
- Investigates opportunistic widening to cleave unnecessary precision.
- Applies symbolic analysis to enforce security contracts statically.

**Impact**

- Seeks to exploit transformative hypothesis linking speed to precision.
- Engages well-meshed team with extensive high-impact collaboration.
- Utilizes static software security contracts to enforce system security.
- Leverages systematic abstraction method discovered by the team.
- Exploits powerful, nontraditional small-step analytic framework.
- Ports “secret weapons” from functional analysis to object-orientation.
- Keeps GPU-based acceleration open as option for improving speed.
- Seeks to handle full, unrestricted Java, including reflective techniques.
- Works inherently with parallel code due to use of small-step method.

**Technical Keystone**

![Graph showing the relationship between time and precision](image)

locally non-monotonic critical point

**Unique Aspects**

- Seeks to exploit transformative hypothesis linking speed to precision
- Engages well-meshed team with extensive high-impact collaboration.
- Utilizes static software security contracts to enforce system security.
- Leverages systematic abstraction method discovered by the team.
- Exploits powerful, nontraditional small-step analytic framework.
- Ports “secret weapons” from functional analysis to object-orientation.
- Keeps GPU-based acceleration open as option for improving speed.
- Seeks to handle full, unrestricted Java, including reflective techniques.
- Works inherently with parallel code due to use of small-step method.

**Innovations**

- Adapts and deploys abstract garbage collection to objects in Java.
- Devises an array of locally non-monotonic methods for precision.
- Resolves null-field initialization problem with anodized addresses.
- Develops and deploys “entangled” abstract domains for objects.
- Extends pushdown return-flow analysis to object-oriented languages.
- Enables optimal polyvariance-learning to autobalance time/precision.
- Investigates opportunistic widening to cleave unnecessary precision.
- Applies symbolic analysis to enforce security contracts statically.

**Impact**

Goal: Order-of-magnitude improvements to both time and precision in automated security analysis:

Cleaving false positives

Adapts and deploys abstract garbage collection to objects in Java.

Devises an array of locally non-monotonic methods for precision.

Resolves null-field initialization problem with anodized addresses.

Develops and deploys “entangled” abstract domains for objects.

Extends pushdown return-flow analysis to object-oriented languages.

Enables optimal polyvariance-learning to autobalance time/precision.

Investigates opportunistic widening to cleave unnecessary precision.

Applies symbolic analysis to enforce security contracts statically.
2 Goals and Impact

Our primary goal is to enable sound, secure, automatic program analysis for the elimination of security vulnerabilities in mobile applications written in high-level programming languages.

Toward this goal, we aim for simultaneous orders-of-magnitude improvements over state-of-the-art in the scalability and precision of static analysis. The differentiating keystone of our technical strategy is to assault scalability through increased precision. Conventional wisdom in the field holds that precision comes only at the cost of additional analysis time:

The argument goes that increasing context-sensitivity—allocating more abstract variants of concrete addresses to boost polyvariance—will increase the size of the abstract state-space, and therefore, the worst case (and, in practice, the actual) analysis time.

Though linked, context-sensitivity is not the same as precision.

The leveragable hypothesis of our proposed effort is that cracks in the conventional wisdom on static analysis can and will be pried open. Early cracks such as Wright and Jagannathan’s work on polymorphic analysis showed analysis time can fall even as precision (and worst-case complexity) increase \[20\]. The message of that work was clear: it is not the number of contexts that matter—it is when they’re allocated (and just as critically, when they are not). Co-PI Van Horn extended this understanding by proving that Shivers’ venerable context-sensitive $k$-CFA \[16\] occupies a point in the design space that runs in exponential time, yet can only learn a polynomial number of true facts about a program \[2, 17\]. In other words, $k$-CFA is hard and imprecise; it spends its time computing junk. A more recent crack—PI Might’s work on abstract garbage collection—makes more efficient use of any finite set of contexts under any allocation strategy, yielding simultaneous order-of-magnitude improvements in both time and precision \[14\].

Extrapolating from these points, we argue that there is a region, in which locally non-monotonic methods are employed, where more precision cleaves so much false positive state-space and spurious rediscovery that analysis time falls:
Locally non-monotonic methods, of which abstract garbage collection is an instance, allow contractionary change to the abstract store under transition, while global monotonicity guarantees termination. Under this “best of both words” scenario, an analyzer can soundly revoke prior judgments, rather than enforcing them as eternal invariants. Revoking judgements diminishes the merging that generates false positives.

3 Publications

This project has yielded several publications [19, 5, 15, 11, 9, 10, 12, 7, 8, 6, 4, 11].

References


