Abstract Debuggers

Exploring Program Behaviors using Static Analysis Results

Karoliine Holter[1], Juhan Oskar Hennoste[1], Patrick Lam[2], Simmo Saan[1], Vesal Vojdani[1]

[1]University of Tartu, [2]University of Waterloo

Onward!
October 24, 2024

Static analyzers are not usable

Why Don't Software Developers Use Static Analysis Tools to Find Bugs?

Brittany Johnson, Yoonki Song, and Emerson Murphy-Hill
North Carolina State University
Raleigh, NC, U.S.A.
bijohnso,ysong2@ncsu.edu,emerson@csc.ncsu.edu

Robert Bowdidge
Google
Mountain View, CA, U.S.A.
bowdidge@google.com

What Developers Want and Need from Program Analysis: An Empirical Study

Maria Christakis Christian Bird
Microsoft Research, Redmond, USA
{mchri, cbird}@microsoft.com

A Large-Scale Study of Usability Criteria Addressed by Static Analysis Tools

Marcus Nachtigall
Heinz Nixdorf Institute, Paderborn University
Germany
marcus.nachtigall@uni-paderborn.de

Michael Schlichtig
Heinz Nixdorf Institute, Paderborn University
Germany
michael.schlichtig@uni-paderborn.de

Eric Bodden
Heinz Nixdorf Institute, Paderborn University & Fraunhofer IEM
Germany
eric.bodden@uni-paderborn.de

When the analysis results in warnings

C example.c > ...

```
    show cfg
8   int f(ThreadAction action) {
9       int cache = 0;
10      switch (action) {
11          case CACHE:
12              printf("Store in local cache!\n");
13              cache = 42;
14          case PUBLISH:
15              printf("Publish work!\n");
16              global = 42;
17      }
18  }
19
    show cfg
20  void *t(void
21      if (pthr
22          f(CA
23      } else {
24          f(PU
```

```
[Race] Group: Memory location global (race with conf. 110)
write with thread:[main, t@example.c:32:5-32:38] (conf. 110)  (exp: & global) GobPie

example.c(16, 13): write with [lock:{mutex}, thread:[main, t@example.c:32:5-32:38]] (conf. 110)
(exp: & global)

example.c(16, 13): write with thread:[main, t@example.c:33:5-33:38] (conf. 110) (exp: & global)

example.c(16, 13): write with [lock:{mutex}, thread:[main, t@example.c:33:5-33:38]] (conf. 110)
(exp: & global)
```

Analysis reasoning (intermediate results) hidden

program

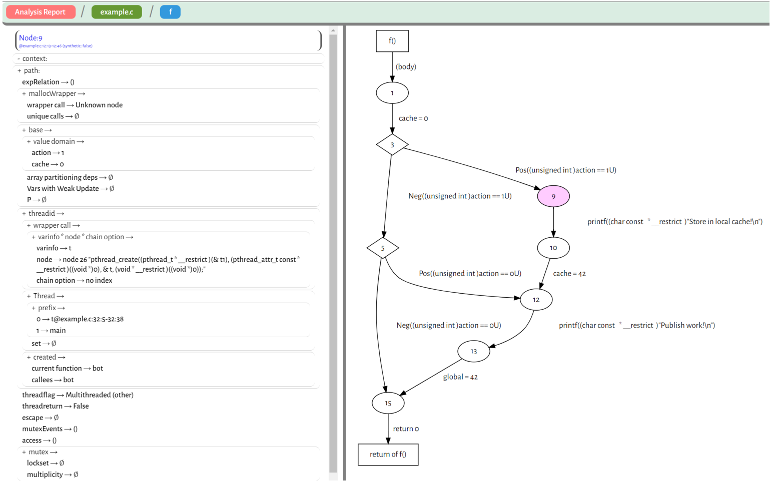
↓

???

↓

analysis result

Raw intermediate results often cluttered and inconvenient to inspect



Motivation

Problem: Static analyzers are not usable

Goal: Improve the explainability and usability of static analysis

Proposal

Shoehorning static analysis results into a robust and well-established interface: a debugger

Concept demo

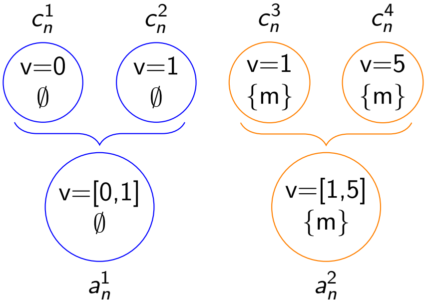
Context

- Abstractions
- Debuggers
- Abstract debuggers

Abstraction of program states

Static analyzers fundamentally compute an over-approximation of the set of reachable states of a program:

- Having a set of reachable program states C_n that correspond to some program location $n \in N$,
- We have a set of abstract states $\mathcal{A}_n$, which correspond to sets of concrete states of the same program location.



Traditional debugger

Debugger

A tool that allows step-by-step execution of a program.

- Live reverse debugger[1]:
  - reverse debugger (record-and-play)
  - + exploration of alternative paths.

Stepping operations play a crucial role

Step into, step over, step out, step back.

[1] A. Savidis and V. Tsiatsianas. 2021. Implementation of Live Reverse Debugging in LLDB.

Abstract debugger

Abstract debugger

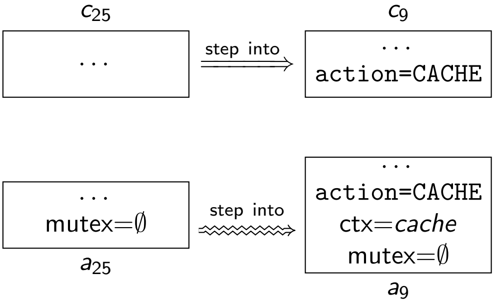
An abstraction of the debugger's button presses, i.e, an abstract-interpretation-like formalization of the stepping operations

$$5 \quad + \quad 5$$
$$[5, 5] \quad +^\sharp \quad [5, 10]$$

$$\cdots c \xRightarrow{\text{into}} \cdots cc'$$
$$a \overset{\text{into}}{\rightsquigarrow} a'$$

```
int f(ThreadAction action) {
  int cache = 0;
  ...
}
```

```
void *t(void *arg) {
  if (pthread_mutex_trylock(&mutex)) {
    f(CACHE);
  } else {
    f(PUBLISH);
    pthread_mutex_unlock(&mutex);
  }
}
```



Formalization of operational semantics

$$\textsc{Step into}\quad \frac{c \xrightarrow{e} c' \qquad e \in B \cup {\downarrow}F \cup F{\uparrow}}{\cdots c \xRightarrow{\text{into}} \cdots cc'}$$

$$\textsc{Step over (basic, return)}\quad \frac{c \xrightarrow{e} c' \qquad e \in B \cup F{\uparrow}}{\cdots c \xRightarrow{\text{over}} \cdots cc'}$$

$$\textsc{Step over (entry)}\quad \frac{c \xrightarrow{{\downarrow}f} c_1 \xrightarrow{\pi'}{}^{\star} c_2 \xrightarrow{f{\uparrow}} c' \qquad \pi = c_1 \pi' c_2}{\cdots c \xRightarrow{\text{over}} \cdots c \pi c'}$$

Figure 1: Selection of concrete operational semantics of a *live reverse* debugger.

$$\textsc{Step into}\quad \frac{a \overset{e}{\rightsquigarrow} a' \qquad e \in B \cup {\downarrow}F \cup F{\uparrow}}{a \overset{\text{into}}{\rightsquigarrow} a'}$$

$$\textsc{Step over}\quad \frac{a \overset{e}{\rightsquigarrow} a' \qquad e \in B \cup F{\uparrow} \cup F}{a \overset{\text{over}}{\rightsquigarrow} a'}$$

Figure 2: Selection of operational semantics of the abstract debugger.

Preserving soundness

Theorem (Soundness)

Let $c_0 \Rightarrow c_1 \Rightarrow \cdots \Rightarrow c_n$ be a debugging session in the concrete world. Then, there exists a corresponding debugging session $a_0 \rightsquigarrow a_1 \rightsquigarrow \cdots \rightsquigarrow a_n$ in the abstract world such that $c_i \in \gamma(a_i)$ for $0 \leq i \leq n$.

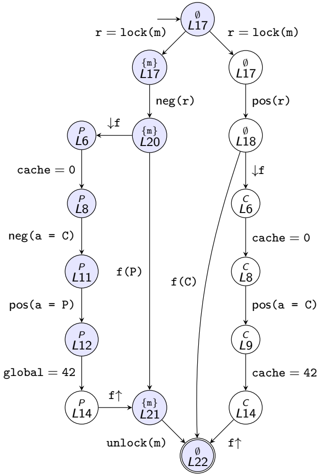
Proof.

By demo. □

Proof by demo

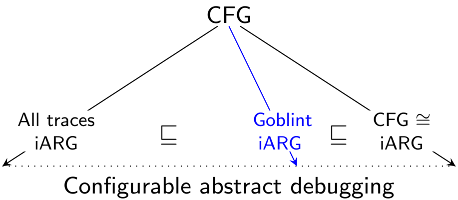
Looking under the hood of an abstract debugger

- Uses static analysis results
- Results are represented as an interprocedural Abstract Reachability Graph (iARG) of a program.
- iARG models the control flow relation between the abstract representations of the concrete states.
- Navigation of the iARG mimics the step-based program execution in a traditional debugger



Configurable abstract debugging

- A general framework
- We hypothesize that any configurable program analysis[2] that can result in the iARG can result in sound abstract debugging.
- Underlying analysis method dictates precision and cost of the results, and abstract debugger's functionality.



[2] D. Beyer, T. A. Henzinger, and G. Théoduloz. 2007. Configurable software verification: concretizing the convergence of model checking and program analysis.

Benefits

Integrating static analysis results with the debugging process yields the following benefits

Users:

- Enables interactive exploration of what a static analysis tool has calculated
- Allows the use of formal verification tools in an already familiar environment
- Incorporates the benefits of static analysis into debugging, such as simplifying the challenge of identifying inputs that trigger bugs

Static analysis tool developers:

- Simplifies engineering efforts for user interface development with enabling the use of inherent IDE features through Debug Adapter Protocol

Conclusion

- A step in the direction of usable and explainable static analyzers
- Interactively exploring static analysis results in a well-established interface
- A general framework of configurable abstract debugging

Onward!

- Static analysis usability is far from solved
- Stringification of abstract domains
- Explaining verification results
- The evolution of new analysis techniques and the usability of these methods must progress together
- Sound static analysis tools into masses!

Abstract Debuggers

Exploring Program Behaviors using Static Analysis Results

Karoliine Holter[1], Juhan Oskar Hennoste[1], Patrick Lam[2], Simmo Saan[1], Vesal Vojdani[1]

[1]University of Tartu, [2]University of Waterloo

Onward!
October 24, 2024

$$\textsc{Balance (basic)}\quad \frac{c \xrightarrow{b} c' \qquad b \in B \qquad \pi = \epsilon}{c \xrightarrow{\pi}{}^{\star} c'}$$

$$\textsc{Balance (function)}\quad \frac{c \xrightarrow{\downarrow f} c_1 \xrightarrow{\pi'}{}^{\star} c_2 \xrightarrow{f\uparrow} c' \qquad \pi = c_1 \pi' c_2}{c \xrightarrow{\pi}{}^{\star} c'}$$

$$\textsc{Balance (append)}\quad \frac{c \xrightarrow{\pi_1}{}^{\star} c_1 \xrightarrow{\pi_2}{}^{\star} c' \qquad \pi = \pi_1 c_1 \pi_2}{c \xrightarrow{\pi}{}^{\star} c'}$$

$$\textsc{Step into}\quad \frac{c \xrightarrow{e} c' \qquad e \in B \cup {\downarrow}F \cup F{\uparrow}}{\cdots c \xRightarrow{\text{into}} \cdots c c'}$$

$$\textsc{Step over (basic, return)}\quad \frac{c \xrightarrow{e} c' \qquad e \in B \cup F{\uparrow}}{\cdots c \xRightarrow{\text{over}} \cdots c c'}$$

$$\textsc{Step over (entry)}\quad \frac{c \xrightarrow{\downarrow f} c_1 \xrightarrow{\pi'}{}^{\star} c_2 \xrightarrow{f\uparrow} c' \qquad \pi = c_1 \pi' c_2}{\cdots c \xRightarrow{\text{over}} \cdots c \pi c'}$$

$$\textsc{Step out (basic, entry)}\quad \frac{c \xrightarrow{\pi'}{}^{\star} c_1 \xrightarrow{f\uparrow} c' \qquad \pi = \pi' c_1}{\cdots c \xRightarrow{\text{out}} \cdots c \pi c'}$$

$$\textsc{Step out (return)}\quad \frac{c \xrightarrow{f\uparrow} c'}{\cdots c \xRightarrow{\text{out}} \cdots c c'}$$

$$\textsc{Step back (basic, entry)}\quad \frac{c \xrightarrow{e} c' \qquad e \in B \cup {\downarrow}F}{\cdots c c' \xRightarrow{\text{back}} \cdots c}$$

$$\textsc{Step back (return)}\quad \frac{c \xrightarrow{\downarrow f} c_1 \xrightarrow{\pi'}{}^{\star} c_2 \xrightarrow{f\uparrow} c' \qquad \pi = c_1 \pi' c_2}{\cdots c \pi c' \xRightarrow{\text{back}} \cdots c}$$

Figure 3: Operational semantics of a *live reverse* debugger.

INTRAPROCEDURAL PATH (EMPTY)

$$\frac{}{a \rightsquigarrow^{\star} a}$$

INTRAPROCEDURAL PATH (BASIC)

$$\frac{a \overset{e}{\rightsquigarrow} a'' \rightsquigarrow^{\star} a' \qquad e \in B \cup F}{a \rightsquigarrow^{\star} a'}$$

STEP INTO

$$\frac{a \overset{e}{\rightsquigarrow} a' \qquad e \in {\downarrow}F \cup F{\uparrow} \cup B}{a \overset{\text{into}}{\rightsquigarrow} a'}$$

STEP OVER

$$\frac{a \overset{e}{\rightsquigarrow} a' \qquad e \in F \cup F{\uparrow} \cup B}{a \overset{\text{over}}{\rightsquigarrow} a'}$$

STEP OUT

$$\frac{a \rightsquigarrow^{\star} a'' \overset{f\uparrow}{\rightsquigarrow} a'}{a \overset{\text{out}}{\rightsquigarrow} a'}$$

STEP BACK

$$\frac{a \overset{e}{\rightsquigarrow} a' \qquad e \in {\downarrow}F \cup F \cup B}{a' \overset{\text{back}}{\rightsquigarrow} a}$$

Figure 4: Operational semantics of an *abstract debugger*.

$$\frac{c \xrightarrow{e} c' \qquad e \in B \cup {\downarrow}F \cup F{\uparrow}}{\cdots cc' \xRightarrow{\text{back into}} \cdots c}\ \textsc{Step back into (basic, entry, return)}$$

$$\frac{c \xrightarrow{\downarrow f} c'}{\cdots cc' \xRightarrow{\text{back out}} \cdots c}\ \textsc{Step back out (entry)}$$

$$\frac{c \xrightarrow{\downarrow f} c_1 \xrightarrow{\pi'}{}^{\star} c' \qquad \pi = c_1\pi'}{\cdots c\pi c' \xRightarrow{\text{back out}} \cdots c}\ \textsc{Step back out (basic, return)}$$

Figure 5: Alternative abstract operational semantics for step back.