Cooperative
Program Analysis

Bor-Yuh Evan Chang
University of Colorado Boulder

October 3, 2013
A program analysis story ...
Software is everywhere and varying more and more
Software is everywhere and varying more and more
Software is everywhere and varying more and more
Software is everywhere and varying more and more
Software is everywhere and varying more and more
Software is everywhere and varying more and more

Software is getting more and more complex
Software is everywhere and varying more and more

Software is getting more and more complex
1980s: Bug in Therac-25 kills
1980s: Bug in Therac-25 kills

2000s: Conficker worm costs $9.1 billion in damages
1980s: Bug in Therac-25 kills

2000s: Conficker worm costs $9.1 billion in damages

Today: “Don’t buy this app, it crashes.”
How does program analysis save the day?

Program Analysis for Formal Verification

Systematically examine the program to "simulate" running it on "all inputs"
Program Analysis for Formal Verification

Systematically examine the program to "simulate" running it on "all inputs"
How does program analysis save the day?

Program Analysis for Formal Verification

Systematically examine the program to “simulate” running it on “all inputs”
How does program analysis save the day?

Program Analysis for Formal Verification

Systematically examine the program to “simulate” running it on “all inputs”
Systematically examine the program to “simulate” running it on “all inputs”
Systematically examine the program to “simulate” running it on “all inputs”
Program Analysis for Formal Verification

Undecidability necessitates the possibility of **false alarms**. We hope not too many.
Uncooperative Program Analysis?
Oh Verifier, help me prove my program has no bugs
Oh Verifier, help me prove my program has no bugs

On line 142, there may be a bug
Uncooperative Program Analysis?

Oh Verifier, help me prove my program has no bugs

On line 142, there may be a bug

Isn’t it obvious this can’t happen!?!?
Oh
Verifier, help me prove my program has no bugs

On line 142, there may be a bug

Isn't it obvious this can't happen!??!

And noisily repeated over and over!
Uncooperative Program Analysis?

Oh Verifier, help me prove my program has no bugs

On line 142, there may be a bug

Isn’t it obvious this can’t happen!?!?

And noisily repeated over and over!

The well-known false alarm problem
Uncooperative Program Analysis?

Oh Verifier, help me prove my program has no bugs!

On line 142, there may be a bug. Isn't it obvious this can't happen!!?!?

The well-known false alarm problem.
"[M]ore than a 30% [false alarm rate] easily causes problems. True bugs get lost in the false. A vicious cycle starts where low trust causes complex [true] bugs to be labeled false [alarms], leading to yet lower trust."

“A stupid false [alarm] implies the tool is stupid.”

The traditional approach to the false alarm problem focuses on improving the verifier.
The traditional approach to the false alarm problem focuses on improving the verifier.

Redesign the verifier with more magic to hopefully reduce the number of false alarms
The traditional approach to the false alarm problem focuses on improving the verifier.

Redesign the verifier with more magic to hopefully reduce the number of false alarms.

But it can never be perfect (undecidability).
The traditional approach to the false alarm problem focuses on improving the verifier.

Redesign the verifier with more magic to hopefully reduce the number of false alarms.

But it can never be perfect (undecidability).

Also not a sufficient “excuse”
Agenda: The cooperative approach addresses the whole bug mitigation process.
Agenda: The cooperative approach addresses the whole bug mitigation process.
Agenda: The cooperative approach addresses the whole bug mitigation process.
Agenda: The cooperative approach addresses the whole bug mitigation process.

Verifier

Proof of no bug

Alarm Report

Program

Verifer

Thresher: Assisting Triage by Refutation Analysis
[Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]

Manual Triaging

PhD Advisee
Agenda: The cooperative approach addresses the whole bug mitigation process.
Agenda: The cooperative approach addresses the whole bug mitigation process.

**Enforcement**

Windows: Measuring Bug Avoidance

[Coughlin+ ISSTA’12, NSF EAGER]

**Program-ming**

**Verifier**

Thresher: Assisting Triage by Refutation Analysis

[Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]

**Manual Triaging**

PhD Advisee
Agenda: The cooperative approach addresses the whole bug mitigation process.

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Verifier

Thresher: Assisting Triage by Refutation Analysis
[Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]

Verifier

Manual Triaging

Spec-ification

Program
Agenda: The cooperative approach addresses the whole bug mitigation process.

**Enforcement**
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

**Program-**

**Fissile Types:**
Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

**Specification**

**Veriﬁer**

**Thresher:** Assisting Triage by Refutation Analysis
[Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]

**Alarm Report**

proof of no bug
Agenda: The cooperative approach addresses the whole bug mitigation process.

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Programming

Fissile Types:
Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Verifier

Test Input

Program

Spec-ification

Test Output

proof of no bug

Alarm Report

Thresher: Assisting Triage by Refutation Analysis
[Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]

Manual Triaging

PhD Advisee
Agenda: The cooperative approach addresses the whole bug mitigation process.

- **Enforcement**
  - Windows: Measuring Bug Avoidance
    - [Coughlin+ ISSTA’12, NSF EAGER]

- **Fissile Types**
  - Checking Almost Everywhere Invariants
    - [Coughlin+ POPL’14, NSF SHF]

- **Main**
  - Manual Triaging

- **Runner**

- **Verifier**
  - Thresher: Assisting Triage by Refutation Analysis
    - [Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]
  - Static Incrementalization of Data Structure Checks
    - [NSF CAREER]

- **Test Input**
  - Test Output

- **Spec-ification**
  - Program

- **Program**

- **Proof of no bug**

- **Alarm Report**

PhD Advisee
Agenda: The cooperative approach addresses the whole bug mitigation process.

**Enforcement**

*Windows: Measuring Bug Avoidance*  
*Coughlin+ ISSTA’12, NSF EAGER*

**Programming**

**Fissile Types:**  
*Checking Almost Everywhere Invariants*  
*Coughlin+ POPL’14, NSF SHF*

**Verifier**

*Thresher: Assisting Triage by Refutation Analysis*  
*Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER*

**Runner**

**Manual Triaging**

**Test Input**

**Spec-iﬁcation**

**Program**

**Test Output**

**Alarm Report**

Proof of no bug

PhD Advisee
Agenda: The cooperative approach addresses the whole bug mitigation process.

**Fissile Types:** Checking Almost Everywhere Invariants  
[Coughlin+ POPL’14, NSF SHF]

**This Talk**

**Enforcement**
Windows: Measuring Bug Avoidance  
[Coughlin+ ISSTA’12, NSF EAGER]

**Program-**
ming

**Test Input**

**Spec-**
ification

**Program**

**Runner**

**Verifier**

**Alarm**

**Report**

**Test Output**

**Alarm Report**

**Thresher:** Assisting Triage by Refutation Analysis  
[Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]

**Static Incrementalization of Data Structure Checks**  
[NSF CAREER]

**Enforcement**
Windows: Measuring Bug Avoidance  
[Coughlin+ ISSTA’12, NSF EAGER]

**Fissile Types:** Checking Almost Everywhere Invariants  
[Coughlin+ POPL’14, NSF SHF]

**This Talk**

**Thresher:** Assisting Triage by Refutation Analysis  
[Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]
Agenda: The cooperative approach addresses the whole bug mitigation process.

**Thresher: Assisting Triage by Refutation Analysis**

[Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]

**Fissile Types: Checking Almost Everywhere Invariants**

[Coughlin+ POPL’14, NSF SHF]

**Enforcement Windows: Measuring Bug Avoidance**

[Coughlin+ ISSTA’12, NSF EAGER]

**Program**

**Specification**

**Test Input**

**Runner**

**Verifier**

**Test Output**

**Alarm Report**

**Proof of no bug**

**This Talk**

**Static Incrementalization of Data Structure Checks**

[NSF CAREER]

**Static Incrementalization of Data Structure Checks**

[NSF CAREER]
Agenda: The cooperative approach addresses the whole bug mitigation process.

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

This Talk

Thresher: Assisting Triage by Refutation Analysis
[Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]
**Thresher: Precise Refutations for Heap Reachability**

- Assist in triage of queries about heap relations
  - Idea: Assume alarms false, prove them so automatically
  - Filters out $\sim 90\%$ of false alarms to expose true bugs
  - Going from $\sim 450$ hours of manual work to $\sim 30$ hours
  - Application: Find memory leaks and eliminate crashes in Android
This Talk: Highlights

**Thresher:** Precise Refutations for Heap Reachability

Assist in triage of queries about heap relations
- Idea: Assume alarms false, prove them so automatically
- Filters out \( \sim 90\% \) of false alarms to expose true bugs
- Going from \( \sim 450 \) hours of manual work to \( \sim 30 \) hours
- Application: Find memory leaks and eliminate crashes in Android

**Fissile Types:** Checking Reflection with Almost Everywhere Invariants

Strengthen type checking with symbolic analysis
- Interactive checking speeds: making IDE integration possible
- Application: Prevent “MethodNotFound” errors in Objective-C (MacOS/iOS)
Thresher: Precise Refutations for Heap Reachability

- Assist in triage of queries about heap relations
  - Idea: Assume alarms false, prove them so automatically
  - Filters out \(\sim 90\%\) of false alarms to expose true bugs
  - Going from \(\sim 450\) hours of manual work to \(\sim 30\) hours
  - Application: Find memory leaks and eliminate crashes in Android

Fissile Types: Checking Reflection with Almost Everywhere Invariants

- Strengthen type checking with symbolic analysis
  - Interactive checking speeds: making IDE integration possible
  - Application: Prevent “MethodNotFound” errors in Objective-C (MacOS/iOS)
Thresher: Precise Refutations for Heap Reachability
What are heap reachability queries?
What are heap reachability queries?

Can an object ever be reached from another object via pointer dereferences?
What are heap reachability queries?

Can an object ever be reached from another object via pointer dereferences?

Example
How is this useful? We identify memory leaks that cause your app to crash!
How is this useful? We identify memory leaks that cause your app to crash!

Android: Crash on rotation, horizontal to vertical
Crash is detected after rotating phone in Gmail Sync now view a

[important bug]cordova 1.9 crash on rotation android
5 posts by 2 authors

App crashes when rotating Samsung phone

Android Terminal Emulator

Project Home Downloads Wiki Issues Source

New issue Search Open issues for

★ Issue 20: Crashes when rotating phone horizontally
1 person starred this issue and may be notified of changes.
How is this useful? We identify memory leaks that cause your app to crash!

How can you have memory leaks with a garbage collected run-time?
Android memory leaks underly rotation-based crashes.

Activity objects encapsulate the UI.
Android memory leaks underly rotation-based crashes.

Activity objects encapsulate the UI.
Android memory leaks underly rotation-based crashes.

Activity objects encapsulate the UI
Android memory leaks underly rotation-based crashes.

Activity objects encapsulate the UI
Android memory leaks underly rotation-based crashes.

Activity objects encapsulate the UI

Android OS

a_static_field

program heap

of type Activity

of type Activity

Android OS
Android memory leaks underly rotation-based crashes.

Activity objects encapsulate the UI

Android OS

of type Activity

Program heap

of type Activity

a_static_field

I can't collect this dead Activity!
Android memory leaks underly rotation-based crashes.

Activity objects encapsulate the UI

I'm full of garbage!

I can't collect this dead Activity!
Android memory leaks underly rotation-based crashes.

Bug: Holding reference to "old" Activity
Android memory leaks underly rotation-based crashes.

Bug: Holding reference to “old” Activity

“an Activity leak”

Activity objects encapsulate the UI

a_static_field

program heap

of type Activity

of type Activity

Android OS

I'm full of garbage!

I can’t collect this dead Activity!

I can't collect this dead Activity!
The expert recommendation ...
The expert recommendation ...
The expert recommendation ...

“Do not keep long-lived references to a context-activity”
The expert recommendation ...

“Do not keep long-lived references to a context-activity”

I don’t know how I created a long-lived reference to an Activity!
The expert recommendation ...

“Do not keep long-lived references to a context-activity”

I don’t know how I created a long-lived reference to an Activity!

Often: A misunderstanding of a library causes the library to keep the Activity reference.
The state of practice in debugging Activity leaks ...
1. Run the app
The state of practice in debugging Activity leaks ...

1. Run the app
2. Watch the heap usage
The state of practice in debugging Activity leaks ...

1. Run the app
2. Watch the heap usage
3. Dump the heap. Dig around and hope to find the culprit
The state of practice in debugging Activity leaks ...

Suppose we’re lucky and find a possible culprit. Now what?

- Where in the code is this object allocated?
- What about the object that references it?
- Where is the reference created?
- Is this reference needed?
- For what periods?

3. Dump the heap. Dig around and hope to find the culprit.

### Table:

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
<th>Total Size</th>
<th>Smallest</th>
<th>Largest</th>
<th>Median</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>free</td>
<td>1,772</td>
<td>107,312 KB</td>
<td>16 B</td>
<td>48,297 KB</td>
<td>24 B</td>
<td>62 B</td>
</tr>
<tr>
<td>data object</td>
<td>405,28</td>
<td>1,225 MB</td>
<td>16 B</td>
<td>1,047 KB</td>
<td>32 B</td>
<td>31 B</td>
</tr>
<tr>
<td>class object</td>
<td>2,187</td>
<td>373,234 KB</td>
<td>168 B</td>
<td>34,125 KB</td>
<td>1,68 B</td>
<td>295 B</td>
</tr>
<tr>
<td>1-byte array (byte)</td>
<td>2,247</td>
<td>3,651 MB</td>
<td>24 B</td>
<td>1,500 MB</td>
<td>48 B</td>
<td>66 B</td>
</tr>
<tr>
<td>2-byte array (short)</td>
<td>10,373</td>
<td>677,352 KB</td>
<td>24 B</td>
<td>28,023 KB</td>
<td>48 B</td>
<td>66 B</td>
</tr>
<tr>
<td>4-byte array (int)</td>
<td>3,663</td>
<td>276,612 KB</td>
<td>24 B</td>
<td>16,023 KB</td>
<td>40 B</td>
<td>77 B</td>
</tr>
<tr>
<td>8-byte array (long)</td>
<td>283</td>
<td>14,875 KB</td>
<td>24 B</td>
<td>4,000 KB</td>
<td>32 B</td>
<td>51 B</td>
</tr>
<tr>
<td>non-Java object</td>
<td>92</td>
<td>14,219 KB</td>
<td>16 B</td>
<td>8,023 KB</td>
<td>32 B</td>
<td>159 B</td>
</tr>
</tbody>
</table>
The state of practice in debugging Activity leaks ...

Suppose we’re lucky and find a possible culprit. Now what?

- Where in the code is this object allocated?
- What about the object that references it?
- Where is the reference created?
- Is this reference needed?
- For what periods?

“One of the most dreaded bugs in Android is a memory leak. They are nasty because one piece of code causes an issue and in some other piece of code, your application crashes.” – http://therockncoder.blogspot.com/2012/09/fixing-android-memory-leak.html
Can an object ever be reached from another object via pointer dereferences?

Example:

Is there a program execution where at some time

```
  a_static_field
  ...
  ...
  ...
```
Answering “Is there an Activity leak?” with program analysis ...

Can an object ever be reached from another object via pointer dereferences?

Example

Is there a program execution where at some time

\[ \text{a\_static\_field} \]

Can be answered with a points-to analysis

\[ \text{of type Activity} \]
Answering “Is there an Activity leak?” with program analysis ...

Can an object ever be reached from another object via pointer dereferences?

Example

Is there a program execution where at some time

\[ \text{a_static_field} \]

\[ \text{of type Activity} \]

Can be answered with a points-to analysis

Hidden Truth

with approximation
Can an object ever be reached from another object via pointer dereferences?

Example

Is there a program execution where at some time

a_static_field

of type Activity

Can be answered with a points-to analysis

with approximation

Some pointer relations may be false

Hidden Truth
But with the cooperative approach ...
But with the cooperative approach ...
Thresher addresses alarm triage in a particularly challenging domain.

- Manual Triaging
- Program
- Verifier
- Alarm Report
- ✔ proof of no bug
- ✘
Thresher addresses alarm triage in a particularly challenging domain.
Thresher addresses alarm triage in a particularly challenging domain.

Known: Precise points-to analysis challenging
Thresher addresses alarm triage in a particularly challenging domain.

Known: Precise points-to analysis challenging
Thresher addresses alarm triage in a particularly challenging domain.

Known: Precise points-to analysis challenging
Thresher addresses alarm triage in a particularly challenging domain.

Hind. “Pointer Analysis: Haven’t We Solved This Problem Yet?”

Known: Precise points-to analysis challenging
Thresher addresses alarm **triage** in a particularly challenging domain.

**Known**: Precise points-to analysis challenging

Hind. “Pointer Analysis: Haven’t We Solved This Problem Yet?”
- 75 papers, 9 PhD theses
Thresher addresses alarm triage in a particularly challenging domain.

Verifier

Program

Points-To Analyzer

Points-To Facts

Manual

proof of no bug

✔

✘

Hind. “Pointer Analysis: Haven’t We Solved This Problem Yet?” (2001)

- 75 papers, 9 PhD theses

Known: Precise points-to analysis challenging
Thresher addresses alarm **triage** in a particularly challenging domain.

**Verifier**

- **Points-To Analyzer**
- **Points-To Facts**

- **Manual**

**Known: Precise points-to analysis challenging**

**Hind. “Pointer Analysis: Haven’t We Solved This Problem Yet?” (2001)**
- 75 papers, 9 PhD theses

**Dagstuhl 13162: Pointer Analysis (2013)**
Thresher addresses alarm triage in a particularly challenging domain.

Verifier

Program → Points-To Analyzer → Points-To Facts

Manual

Hind. “Pointer Analysis: Haven’t We Solved This Problem Yet?” (2001)
- 75 papers, 9 PhD theses

Dagstuhl 13162: Pointer Analysis (2013)

Known: Precise points-to analysis challenging enough?

(impossible?)
Manual triage is particularly hard for heap reachability reports.
Manual triage is particularly hard for heap reachability reports.
Manual triage is particularly hard for heap reachability reports.
Manual triage is particularly hard for heap reachability reports.
Manual triage is particularly hard for heap reachability reports.
Manual triage is particularly hard for heap reachability reports.
Manual triage is particularly hard for heap reachability reports.
Manual triage is particularly hard for heap reachability reports.

Get abstract heap path + maybe allocation sites

Guesstimate: >1 to 2 hours per alarm to triage “well”
Examining manual triage ...
Examining manual triage ...
What does the user need to do with an alarm? He starts at, say, line 142 and traces back to see if a bug is possible given what’s happening.
Examining manual triage ...

What does the user need to do with an alarm? He starts at, say, line 142 and traces back to see if a bug is possible given what’s happening.

We can do this with analysis!
Examining manual triage ...

What does the user need to do with an alarm? He starts at, say, line 142 and traces back to see if a bug is possible given what’s happening.

We can do this with analysis!
What does the user need to do with an alarm? He starts at, say, line 142 and traces back to see if a bug is possible given what’s happening.

We can do this with analysis!

If we filter most false alarms, the user can triage more quickly and get to true bugs earlier (without frustration).
Automate the tasks that developers hate doing
But not the things they love
Automate the tasks that developers hate doing
But not the things they love
Automate the tasks that developers hate doing
But not the things they love

require human creativity and insight
Automate the tasks that developers hate doing
But not the things they love
Automate the tasks that developers hate doing
But not the things they love

my graduate students

Automate the tasks that developers hate doing
But not the things they love
Thresher filters out false alarms by refuting them one-by-one.
Thresher filters out false alarms by refuting them one-by-one.

Verifier

Points-To Analyzer (off-the-shelf)

Points-To Facts

Manual Triaging

Filter with Thresher

Program

Manual Triaging

Filter with Thresher

Proof of no bug

Leak Alarms
Thresher filters out false alarms by refuting them one-by-one.

Verifier

Points-To Analyzer (off-the-shelf)

Points-To Facts

Manual Triaging

Filter with Thresher

Program

Manual Triaging

Filter with Thresher

Leak Alarms

✔ proof of no bug

✘
Thresher filters out false alarms by refuting them one-by-one.
Thresher filters out false alarms by refuting them one-by-one.

Idea 1: Refute points-to on-demand with second “uber-precise” filter analysis
Thresher filters out false alarms by refuting them one-by-one.

Idea 1: Refute points-to on-demand with second "uber-precise" filter analysis

Verifier

Program

Points-To Analyzer (off-the-shelf)

Points-To Facts

Manual Triaging

Filter with Thresher

1

proof of no bug

Leak Alarms

* -sensitive
Leak Alarms

Verifier

Points-To Analyzer (off-the-shelf)

Points-To Facts

Program

Manual Triaging

Filter with Thresher

Leak Alarms

✔ proof of no bug

Manual Triaging

Program

Idea 1: Refute points-to on-demand with second "uber-precise" filter analysis
Thresher filters out false alarms by refuting them one-by-one.

Idea 1: Refute points-to on-demand with second "uber-precise" filter analysis

Idea 2: Leverage the facts from the first analysis in the filter analysis to scale
Refutation analysis is “Proof by Contradiction” with the “But Why?” game
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

There **may** be an execution where at some time

\[ o \]

\[ o' \]

\[ \cdots \]

of type \( T \).
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

There may be an execution where at some time

\[ o \]

\[ o' \]

of type \( T \).

A. Why does object \( o \) possibly point to \( o' \)?
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

A. Why does object $o$ possibly point to $o'$?

B. Because statement $s$ may execute to make $o$ point to $o'$

There \textbf{may} be an execution where at some time

\begin{align*}
   &o \\
   \downarrow \\
   &o' \\
   \vdots \\
   \downarrow \\
   \text{of type T.}
\end{align*}
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

There **may** be an execution where at some time

![Diagram](image)

of type T.

A. Why does object $o$ possibly point to $o'$?

B. Because statement $s$ may execute to make $o$ point to $o'$

A. Why does statement $s$ cause $o$ to point to $o'$?
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

There **may** be an execution where at some time

\[
\begin{array}{c}
o \\
\downarrow \\
o' \\
\downarrow \\
\end{array}
\]

of type $T$.  

A. Why does object $o$ possibly point to $o'$?

B. Because statement $s$ may execute to make $o$ point to $o'$

A. Why does statement $s$ cause $o$ to point to $o'$?

B. Because before statement $s$, the program state could satisfy formula $\varphi$
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

There **may** be an execution where at some time

\[ o \]

\[ \downarrow \]

\[ o' \]

\[ \vdots \]

\[ \downarrow \]

of type $T$.

A. Why does object $o$ possibly point to $o'$?

B. Because statement $s$ may execute to make $o$ point to $o'$

A. Why does statement $s$ cause $o$ to point to $o'$?

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

There **may** be an execution where at some time

\[ o \]

\[ \downarrow \]

\[ o' \]

\[ \vdots \]

\[ \downarrow \]

of type T.

A. Why does object \( o \) possibly point to \( o' \)?

B. Because statement \( s \) may execute to make \( o \) point to \( o' \)

A. Why does statement \( s \) cause \( o \) to point to \( o' \)?

B. Because before statement \( s \), the program state could satisfy formula \( \varphi \)

A. Why can the state before statement \( s \) satisfy \( \varphi \)?

B. Because before the previous statement \( s' \), the state could satisfy formula \( \varphi' \)
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

A. Why does object $o$ possibly point to $o'$?

B. Because statement $s$ may execute to make $o$ point to $o'$

A. Why does statement $s$ cause $o$ to point to $o'$?

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

There may be an execution where at some time:

A. Why does object $o$ possibly point to $o'$?

B. Because statement $s$ may execute to make $o$ point to $o'$

A. Why does statement $s$ cause $o$ to point to $o'$?

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

Theorem: If B can’t give an answer, contradiction. The alarm is false. It’s been refuted. (A wins)
Leverage first analysis by designing specialized constraint forms

B. Because before statement \( s \), the program state could satisfy formula \( \varphi \)

A. Why can the state before statement \( s \) satisfy \( \varphi \)?

B. Because before the previous statement \( s' \), the state could satisfy formula \( \varphi' \)
Leverage first analysis by designing specialized constraint forms

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$
Leverage first analysis by designing specialized constraint forms

B. Because before statement \( s \), the program state could satisfy formula \( \varphi \)

A. Why can the state before statement \( s \) satisfy \( \varphi \)?

B. Because before the previous statement \( s' \), the state could satisfy formula \( \varphi' \)
Leverage first analysis by designing specialized constraint forms

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

set of possible states
B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

- If empty, then refuted (A wins)

set of possible states

$\varphi$
Leverage first analysis by designing specialized constraint forms

B. Because before statement \( s \), the program state could satisfy formula \( \varphi \)

A. Why can the state before statement \( s \) satisfy \( \varphi \)?

if empty, then refuted (A wins)

set of possible states

\[ \varphi \]

\[ pre_s \]

\[ \varphi' \]
Leverage first analysis by designing specialized constraint forms

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

If empty, then refuted (A wins)

Set of possible states

$\varphi$

$\varphi'$

$\text{pre}_{s'}$

Points-To Facts
B. Because before statement \( s \), the program state could satisfy formula \( \varphi \)

A. Why can the state before statement \( s \) satisfy \( \varphi \)?

B. Because before the previous statement \( s' \), the state could satisfy formula \( \varphi' \).

if empty, then refuted (A wins)
Leverage first analysis by designing specialized constraint forms

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

if empty, then refuted (A wins)

set of possible states

$\varphi \rightarrow \varphi'$
Leverage first analysis by designing specialized constraint forms

B. Because before statement \( s \), the program state could satisfy formula \( \varphi \)

A. Why can the state before statement \( s \) satisfy \( \varphi \)?

B. Because before the previous statement \( s' \), the state could satisfy formula \( \varphi' \)
Leverage first analysis by designing **specialized constraint forms**

B. Because before statement \( s \), the program state could satisfy formula \( \varphi \)

A. Why can the state before statement \( s \) satisfy \( \varphi \)?

B. Because before the previous statement \( s' \), the state could satisfy formula \( \varphi' \)
B. Because before statement \( s \), the program state could satisfy formula \( \varphi \)

A. Why can the state before statement \( s \) satisfy \( \varphi \)?

B. Because before the previous statement \( s' \), the state could satisfy formula \( \varphi' \)

Technical Contribution: Specialized constraint forms
Leverage first analysis by designing **specialized constraint forms**

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

**Technical Contribution:** Specialized constraint forms
Leverage first analysis by designing specialized constraint forms

B. Because before statement $s$, the program state could satisfy formula $\varphi$

A. Why can the state before statement $s$ satisfy $\varphi$?

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

Technical Contribution: Specialized constraint forms
Leverage first analysis by designing specialized constraint forms

A. Why can the state before statement $s$ satisfy formula $\varphi$?

B. Because before statement $s$, the program state could satisfy formula $\varphi$

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$

Technical Contribution: Specialized constraint forms
Leverage first analysis by designing specialized constraint forms

A. Why can the state before statement $s$ satisfy formula $\varphi$?

B. Because before statement $s$, the program state could satisfy formula $\varphi$.

B. Because before the previous statement $s'$, the state could satisfy formula $\varphi'$.

Specialized constraint forms makes finding refutations feasible

Technical Contribution: Specialized constraint forms
Summary: Thresher assists the user with alarm triaging by effectively filtering out many false alarms.

Idea 1: Refute points-to on-demand with second “uber-precise” filter analysis

Idea 2: Leverage the facts from the first analysis in the filter analysis to scale
Is Thresher effective at filtering?

Thresher analyzes Java VM bytecode

7 Android app benchmarks
2,000 to 40,000 source lines of code
+ 880,000 sources lines of Android framework code

Off-the-shelf, state-of-the-art points-to analysis from WALA
## Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
<th>Thresher Time (s)</th>
<th>False Alarm %</th>
<th>Filtered %</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>95</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>1068</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>46</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
<td>18</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
<td>374</td>
<td>18</td>
<td>90</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>72K</strong></td>
<td><strong>311</strong></td>
<td><strong>172</strong></td>
<td><strong>115</strong></td>
<td><strong>1602</strong></td>
<td><strong>17</strong></td>
<td><strong>88</strong></td>
</tr>
</tbody>
</table>
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
<th>Thresher Time (s)</th>
<th>False Alarm %</th>
<th>Filtered %</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>95</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>1068</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>46</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
<td>18</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
<td>374</td>
<td>18</td>
<td>90</td>
</tr>
<tr>
<td>Total</td>
<td>72K</td>
<td>311</td>
<td>172</td>
<td>115</td>
<td>1602</td>
<td>17</td>
<td>88</td>
</tr>
</tbody>
</table>

staticfield-
Activity pairs
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
<th>Thresher Time (s)</th>
<th>False Alarm %</th>
<th>Filtered %</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>95</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>1068</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>46</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>aMetro</td>
<td></td>
<td>54</td>
<td>18</td>
<td>36</td>
<td>18</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>K9Mail</td>
<td></td>
<td>208</td>
<td>130</td>
<td>64</td>
<td>374</td>
<td>18</td>
<td>90</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>311</strong></td>
<td><strong>172</strong></td>
<td><strong>115</strong></td>
<td><strong>1602</strong></td>
<td><strong>17</strong></td>
<td><strong>88</strong></td>
</tr>
</tbody>
</table>

- triage “well” at $\sim 1-2$ hours per alarm
- staticfield-Activity pairs
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>72K</strong></td>
<td><strong>311</strong></td>
<td><strong>172</strong></td>
</tr>
</tbody>
</table>
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
</tr>
<tr>
<td>Total</td>
<td>72K</td>
<td>311</td>
<td>172</td>
<td>115</td>
</tr>
</tbody>
</table>

staticfield-Activity pairs

Filtered

Manual
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td></td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td></td>
<td></td>
<td>64</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>72K</strong></td>
<td><strong>311</strong></td>
<td><strong>172</strong></td>
<td><strong>115</strong></td>
</tr>
</tbody>
</table>

*Filtering* staticfield-Activity pairs triage “well” at 10–15 minutes per.
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>72K</td>
<td><strong>311</strong></td>
<td><strong>172</strong></td>
<td><strong>115</strong></td>
</tr>
</tbody>
</table>
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
<th>Thresher Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>95</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>1068</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>46</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
<td>18</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
<td>374</td>
</tr>
<tr>
<td>Total</td>
<td>72K</td>
<td>311</td>
<td>172</td>
<td>115</td>
<td>1602</td>
</tr>
</tbody>
</table>
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
<th>Thresher Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>95</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>1068</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>46</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
<td>18</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
<td>374</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>72K</strong></td>
<td><strong>311</strong></td>
<td><strong>172</strong></td>
<td><strong>115</strong></td>
<td><strong>1602</strong></td>
</tr>
</tbody>
</table>
## Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
<th>Thresher Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>95</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>1068</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>46</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
<td>18</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
<td>374</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>72K</strong></td>
<td><strong>311</strong></td>
<td><strong>172</strong></td>
<td><strong>115</strong></td>
<td><strong>1602</strong></td>
</tr>
</tbody>
</table>
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
<th>Thresher Time (s)</th>
<th>False Alarm %</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>95</td>
<td>0</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>1068</td>
<td>100</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
<td>374</td>
<td>18</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>72K</td>
<td>311</td>
<td>172</td>
<td>115</td>
<td>1602</td>
<td><strong>17</strong></td>
</tr>
</tbody>
</table>

% after filtering
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
<th>Thresher Time (s)</th>
<th>False Alarm %</th>
<th>Filtered %</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>95</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>1068</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>46</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
<td>18</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
<td>374</td>
<td>18</td>
<td>90</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>72K</strong></td>
<td><strong>311</strong></td>
<td><strong>172</strong></td>
<td><strong>115</strong></td>
<td><strong>1602</strong></td>
<td><strong>17</strong></td>
<td><strong>88</strong></td>
</tr>
</tbody>
</table>

% after filtering
Is Thresher effective at filtering?

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Points-To Alarms</th>
<th>Thresher Refuted</th>
<th>True Bugs</th>
<th>Thresher Time (s)</th>
<th>False Alarm %</th>
<th>Filtered %</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>95</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>1068</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>46</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
<td>18</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
<td>374</td>
<td>18</td>
<td>90</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>72K</strong></td>
<td><strong>311</strong></td>
<td><strong>172</strong></td>
<td><strong>115</strong></td>
<td><strong>1602</strong></td>
<td><strong>17</strong></td>
<td><strong>88</strong></td>
</tr>
</tbody>
</table>

False alarms down to 17% from 63% (points-to analysis only)
Thresher filters 88% of false alarms from points-to analysis
False alarms down to 17% from 63% (points-to analysis only)
Thresher filters 88% of false alarms from points-to analysis

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Alarms</th>
<th>Related</th>
<th>Bugs</th>
<th>Time (s)</th>
<th>False Alarm %</th>
<th>Thresher Refuted</th>
<th>Thresher Filtered %</th>
</tr>
</thead>
<tbody>
<tr>
<td>PulsePoint</td>
<td>unknown</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>95</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>StandupTimer</td>
<td>2K</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>1068</td>
<td>100</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>DroidLife</td>
<td>3K</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SMSPopUp</td>
<td>7K</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>46</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>aMetro</td>
<td>20K</td>
<td>54</td>
<td>18</td>
<td>36</td>
<td>18</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>K9Mail</td>
<td>40K</td>
<td>208</td>
<td>130</td>
<td>64</td>
<td>374</td>
<td>18</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>72K</td>
<td>311</td>
<td>172</td>
<td>115</td>
<td>1602</td>
<td>17</td>
<td>88</td>
<td></td>
</tr>
</tbody>
</table>
Android OS
... in the process of finding leaks in apps
class HashMap {
    static Object[] EMPTY = new Object[2]; ...
    HashMap() { this.tbl = EMPTY; capacity initially empty }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) { this.tbl = EMPTY; }
        else { this.tbl = new Object[at least m.size()]; }  
        copy from m
    }
}
Find the Android’s HashMap bug ...

class HashMap {
    static Object[] EMPTY = new Object[2]; ...

    HashMap() {
        this.tbl = EMPTY; capacity initially empty
    }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) {
            this.tbl = EMPTY;
        } else {
            this.tbl = new Object[at least m.size()];
            copy from m
        }
    }
}
Find the Android’s HashMap bug ...

class HashMap {
    static Object[] EMPTY = new Object[2]; ...
    HashMap() { \texttt{this.tbl = EMPTY}; \textit{capacity initially empty} }
    void put(Object key, Object val) {
        if (need capacity) {
            \texttt{this.tbl = new Object[more capacity];}
            \textit{copy from old table}
        }
        \texttt{this.tbl[bucket using hash of key] = val;}
    }
    HashMap(Map m) {
        if (m.size() < 1) { \texttt{this.tbl = EMPTY}; }
        else { \texttt{this.tbl = new Object[at least m.size()];}
            \textit{copy from m}
        }
    }
}
Find the Android’s HashMap bug ...

class HashMap {
    static Object[] EMPTY = new Object[2]; ... 
    HashMap() { this.tbl = EMPTY; capacity initially empty }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) { this.tbl = EMPTY; }
        else { this.tbl = new Object[at least m.size()]; }
        copy from m
    }
}
Find the Android’s HashMap bug ...

class HashMap {
    static Object[] EMPTY = new Object[2]; ...
    HashMap() { this.tbl = EMPTY; }  
    void put(Object key, Object val) {  
        if (need capacity) {  
            this.tbl = new Object[more capacity];  
            copy from old table  
        }
        this.tbl[bucket using hash of key] = val;  
    }
    HashMap(Map m) {  
        if (m.size() < 1) { this.tbl = EMPTY; }  
        else { this.tbl = new Object[at least m.size()]; }  
        copy from m  
    }
}
class HashMap {

    static Object[] EMPTY = new Object[2]; ... 

    HashMap() { this.tbl = EMPTY; capacity initially empty }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) { this.tbl = EMPTY; }
        else { this.tbl = new Object[at least m.size()]; }
        copy from m
    }
}

An “evil” implementation of the Map interface can corrupt EMPTY. Then, all HashMaps created in the future will be corrupted.
class HashMap {

    static Object[] EMPTY = new Object[2]; ...

    HashMap() {
        this.tbl = EMPTY; capacity initially empty
    }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }

        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) {
            this.tbl = EMPTY;
        } else {
            this.tbl = new Object[at least m.size()];
            copy from m
        }
    }

    return "evil" content

    An "evil" implementation of the Map interface can corrupt EMPTY. Then, all HashMaps created in the future will be corrupted.
class HashMap {
    static Object[] EMPTY = new Object[2];

    HashMap() {
        this.tbl = EMPTY;
    }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
            copy from old table
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) {
            this.tbl = EMPTY;
        } else {
            this.tbl = new Object[at least m.size()];
        }
        copy from m
    }
}

HashMap(Map m) {
    if (m.size() < 1) {
        this.tbl = EMPTY;
    } else {
        this.tbl = new Object[at least m.size()];
    }
    copy from m
}

return 0

An “evil” implementation of the Map interface can corrupt EMPTY. Then, all HashMaps created in the future will be corrupted.

What if you store passwords in a HashMap?
class HashMap {
    static Object[] EMPTY = new Object[2];

    HashMap() {
        this.tbl = EMPTY;
    }

    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object[more capacity];
        }
        this.tbl[bucket using hash of key] = val;
    }

    HashMap(Map m) {
        if (m.size() < 1) {
            this.tbl = EMPTY;
        } else {
            this.tbl = new Object[at least m.size()];
        }
        copy from m
    }

    return "evil" content

    HashMap(Map m) {
        if (m.size() < 1) {
            this.tbl = EMPTY;
        } else {
            this.tbl = new Object[at least m.size()];
        }
        copy from m
    }

    return "evil" content

    An “evil” implementation of the Map interface can corrupt EMPTY. Then, all HashMaps created in the future will be corrupted.

    We reported this, Google fixed it: https://android-review.googlesource.com/#/c/52183/
Contribution: Addressed the false alarm problem with a “smart and precise filter” and a refutation analysis.
Agenda: The cooperative approach addresses the whole bug mitigation process.

**Enforcement**
Windows: Measuring Bug Avoidance  
[Coughlin+ ISSTA’12, NSF EAGER]

**Program-ming**

- **Test Input**
- **Spec-ification**
- **Program**

**Verifier**

- **Runnner**
- **Jsana: Abstract Domain Combinators for Dynamic Languages**
  [Cox+ ECOOP’13, NSF SHF]

**Manual Triaging**

- **Thresher: Assisting Triage by Refutation Analysis**
  [Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]

**Static Incrementalization of Data Structure Checks**
[NSF CAREER]

**Fissile Types: Checking Almost Everywhere Invariants**
[Coughlin+ POPL’14, NSF SHF]
Agenda: The cooperative approach addresses the whole bug mitigation process.

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Program-
m sings

Verifier

Runn e

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]

Enforcement
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12, NSF EAGER]

Test Input

Test Output

Alarm
Report

proof of no bug

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, NSF SHF]
Fissile Types: Checking Reflection with Almost Everywhere Invariants
Method Reflection and the Great Divide
object[string]()
Method Reflection and the Great Divide

reflective method call: dispatch based on run-time value (in string)

object[string]()
Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in **string**)
Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in **string**)

object[string]()

type system designers “web 2.0” developers

Type system designers **worry**.

What gets called? What if object has no method named by string?
Method Reflection and the Great Divide

reflective method call: dispatch based on run-time value (in string)

object[string]()

type system designers

"web 2.0" developers

Type system designers **worry**.

What gets called? What if object has no method named by string?

"Web 2.0" developers think it's **cool**.

I can flexible and compact code, so I will take it over static safety.
Method Reflection and the Great Divide

reflective method call: dispatch based on run-time value (in string)

object[string]()

type system designers

“web 2.0” developers

Type system designers worry.

“Web 2.0” developers think it’s cool.

“MethodNotFound” checked at run time

static safety.
Programs are often (1) safe, (2) not type safe, (3) but almost so
Programs are often (1) safe, (2) not type safe, (3) but almost so.
Programs are often (1) safe, (2) not type safe, (3) but almost so

callback.o[callback.m]()
Programs are often
(1) safe, (2) not type safe, (3) but almost so

safe assuming a relationship
invariant between .o and .m

callback.o[callback.m]()
Programs are often (1) safe, (2) not type safe, (3) but almost so

\[
\text{callback.o[\text{callback.m}]}()
\]
Programs are often
(1) safe, (2) not type safe, (3) but almost so

invariant holds

safe assuming a relationship invariant between .o and .m

callback.o[callback.m]()

invariant broken
Programs are often (1) safe, (2) not type safe, (3) but almost so

callback.o[callback.m]()

safe assuming a relationship invariant between .o and .m

invariant holds

but only temporarily
Programs are often (1) safe, (2) not type safe, (3) but almost so

Tolerate “temporary” violation with

```
callback.o[callback.m]()
```

Invariant holds

Invariant broken but only temporarily

Program
Programs are often 
(1) safe, (2) not type safe, (3) but almost so

Tolerate “temporary” violation with

invariant holds

invariant broken
Fissile analyzes **Objective-C** source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

Type annotations

seeded with 76 `respondsTo` in system libraries

needed only 136 annotations in benchmarks (total)
Is Fissile effective at proving reflective call safety?

Fissile analyzes **Objective-C** source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

Type annotations

Proved **86%** of check sites (up from **76%**) at interactive speeds ($\sim$4 to 90 kloc/s)

benchmarks (total)
Is Fissile effective at proving reflective call safety?

Fissile analyzes **Objective-C** source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

*places requiring a check of the invariant*

Proved **86%** of check sites (up from 76%) at **interactive speeds** (~4 to 90 kloc/s)

benchmarks (total)
Is Fissile effective at proving reflective call safety?

Fissile analyzes **Objective-C** source

9 benchmarks (6 libraries + 3 apps)

1,000 to 176,000 lines of code

461,000 lines in total

**Proved 86% of check sites (up from 76%) at interactive speeds (∼4 to 90 kloc/s)**

Big Deal: makes IDE integration possible
Summary: The cooperative approach addresses the whole bug mitigation process.

- **Enforcement**
  - Windows: Measuring Bug Avoidance
    - [Coughlin+ ISSTA’12, NSF EAGER]

- **Programming**
  - Fissile Types: Checking Almost Everywhere Invariants
    - [Coughlin+ under review, Khoo+ PLDI’10, NSF SHF]

- **Verifier**
  - Thresher: Automated Triage by Refutation Analysis
    - [Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]

- **Runner**
  - Jsana: Abstract Domain Combinators for Dynamic Languages
    - [Cox+ ECOOP’13, NSF SHF]

- **Manual Triaging**
  - Static Incrementalization of Data Structure Checks
    - [NSF CAREER]
Future Directions
Future Directions

The Cooperative Approach

Unit Test Synthesis: test input + test code to unequivocally show bug
Evidence for Alarms: alarm explanations, probability of bug
Hardening: synthesize efficient dynamic checks
Patch Synthesis: synthesize bug fixes
Performance and Scalability Bugs: beyond correctness bugs
Input Debugging: bugs in input
Analysis Engines: in new software domains
Future Directions

The Cooperative Approach

Unit Test Synthesis: test input + test code to unequivocally show bug
Evidence for Alarms: alarm explanations, probability of bug
Hardening: synthesize efficient dynamic checks
Patch Synthesis: synthesize bug fixes
Performance and Scalability Bugs: beyond correctness bugs
Input Debugging: bugs in input
Analysis Engines: in new software domains

Software Domains

Dynamic: web, science+engineering
Concurrent: event-driven systems, user-interactive systems, servers
Distributed: “big data” software
Acknowledgments

PhD Advisees
Sam Blackshear
Devin Coughlin
Arlen Cox
Yi-Fan Tsai

Student Collaborators
Aleks Chakarov (PhD)
Robert Frohardt (PhD)
Christoph Reichenbach (PhD 2009)
Khoo Yit Phang (PhD 2013, Maryland)
Antoine Toubhans (PhD, ENS Paris)
Sid Gracias (MS)
Daniel Stutzman (MS)
Vincent Laviron (MS, ENS Paris)
Nick Vanderweit (BS)
Alex Beal (BS)

Faculty/Researcher Collaborators
Sriram Sankaranarayanan
Pavol Cerny
Tom Yeh
Rick Han
Eric Keller
Dirk Grunwald
John Black
Amer Diwan (Google)
Jeremy Siek (Indiana)
Manu Sridharan (IBM Research)
Xavier Rival (INRIA/ENS Paris)
Jeff Foster (Maryland)
Atif Memon (Maryland)
Cesar Sanchez (IMDEA)
“Nothing so needs reforming as other people’s habits.” – Mark Twain
“Nothing so needs reforming as other people’s habits.” – Mark Twain

Criticize with care and empathy not condemnation
“Nothing so needs reforming as other people’s habits.” – Mark Twain

Criticize with care and empathy not condemnation

“cooperative program analysis principle and challenge”