Cooperative Program Analysis

Bor-Yuh Evan Chang
University of Colorado Boulder

Colorado State University
September 22, 2014
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 6
1980s: Bug in Therac-25 kills 6

2000s: Conficker worm costs $9.1 billion in damages
1980s: Bug in Therac-25 kills 6

2000s: Conficker worm costs $9.1 billion in damages

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

Program Analysis for Formal Verification

Systematically examine the program to “simulate” running it on “all inputs”
The End?
Systematically examine the program to “simulate” running it on “all inputs”
The Ugly, Hidden Truth

Program Analysis for Formal Verification

Systematically examine the program to "simulate" running it on "all inputs"
The Ugly, Hidden Truth

Program Analysis for Formal Verification

Undecidability necessitates the possibility of **false alarms**. We hope not too many.
Uncooperative Program Analysis?
Uncooperative Program Analysis?

Oh Verifier, help me prove my program has no bugs
Uncooperative Program Analysis?

Oh Verifier, help me prove my program has no bugs

On line 142, there may be a bug
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!?!?
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!
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
Uncouoperative 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.

And noisily repeated over and over!
“[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.
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
  - ✘ Program
  - Alarm Report

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

- **Programming**
  - Manual Triaging
  - Thresher: Assisting Triage by Refutation Analysis [Blackshear+ PLDI’13, Blackshear+ SAS’11, under review]

- **Manual**
  - Program

- **Thresher**
Agenda: The cooperative approach addresses the whole bug mitigation process.

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

- **Programming**
  - **Specifcation**
  - **Program**

- **Verifier**
  - Thresher: Assisting Triage by Refutation Analysis
    - [Blackshear+ PLDI’13, Blackshear+ SAS’11, under review]

- **Manual Triaging**

- **Alarm Report**
Agenda: The cooperative approach addresses the whole bug mitigation process.

- Thresher: Assisting Triage by Refutation Analysis
  - [Blackshear + PLDI'13, Blackshear + SAS'11, under review]

- Fissile Types: Checking Almost Everywhere Invariants
  - [Coughlin + POPL'14, in prep]

- Enforcement Windows: Measuring Bug Avoidance
  - [Coughlin + ISSTA'12]
Agenda: The cooperative approach addresses the whole bug mitigation process.

**Enforcement**
Windows: Measuring Bug Avoidance
[Coghlin+ ISSTA'12]

**Programming**

**Fissile Types:**
Checking Almost Everywhere Invariants
[Coughlin+ POPL'14, in prep]

**Verifier**

**Thresher:**
Assisting Triage by Refutation Analysis
[Blackshear+ PLDI'13, Blackshear+ SAS'11, under review]

**Runner**

**Test Input**

**Spec-ification**

**Program**

**Test Output**

**Manual Triaging**

**Alarm Report**

proof of no bug

Enforcement Windows: Measuring Bug Avoidance
[Coghlin+ ISSTA'12]
Agenda: The cooperative approach addresses the whole bug mitigation process.

**Enforcement**
Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA'12]

**Programming**

**Fissile Types**
Checking Almost Everywhere Invariants
[Coughlin+ POPL'14, in prep]

**Manual Triaging**

**Test Input**

**Spec- ification**

**Program**

**Static Incrementalization of Data Structure Checks**
[under review]

**Thresher**
Assisting Triage by Refutation Analysis
[Blackshear+ PLDI’13, Blackshear+ SAS’11, under review]

**Verifier**

**Test Output**

**Alarm Report**

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

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

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

- **Program**
  - Manual Triaging

- **Verifier**
  - Thresher: Assisting Triage by Refutation Analysis [Blackshear+ PLDI’13, Blackshear+ SAS’11, under review]
  - Jsana: Abstract Domain Combinators for Dynamic Languages [Cox+ ECOOP’13, Cox+ SAS’14, under review]
  - Static Incrementalization of Data Structure Checks [under review]

- **Runner**
  - Test Input
  - Test Output

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

- **Output**
  - Alarm
  - Report

- **Input**
  - Manual Triaging
Agenda: The cooperative approach addresses the whole bug mitigation process.

Fissile Types: Checking Almost Everywhere Invariants [Coughlin+ POPL’14, in prep]

This Talk

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

This Talk

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

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, in prep]

Enforcement Windows: Measuring Bug Avoidance
[Coughlin+ ISSTA’12]
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
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)
**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
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?

Is there a program execution where at some time a variable of type T?

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

How can you have memory leaks with a garbage collected runtime?
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

of type Activity
Android memory leaks underly rotation-based crashes.

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

Activity objects encapsulate the UI

of type Activity

of type Activity
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 memory leaks underly rotation-based crashes.

I can't collect this dead Activity!

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

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

Android OS

Bug: Holding reference to "old" Activity

Activity objects encapsulate the UI

I can’t collect this dead Activity!

program heap

a_static_field

of type Activity

of type Activity

Android OS

I'm full of garbage!
Android memory leaks underly rotation-based crashes.

Bug: Holding reference to "old" Activity

"an Activity leak"

Activity objects encapsulate the UI
The expert recommendation ...
The expert recommendation ...

Avoiding memory leaks

Android applications are, at least on the T-Mobile G1, limited to 16 MB of heap. It's both a lot of memory for a phone and yet very little for what some developers want to achieve. Even if you do not plan on using all of this memory, you should use as little as possible to let other applications run without getting them killed. The more applications Android can keep in memory, the better it will be for the user to switch between his apps. As part of my job, I ran into memory leaks issues in Android applications and they are most of the time due to the same mistake: keeping a long-lived reference to a Context.

On Android, a Context is used for many operations but mostly to load and access resources. This is why all the widgets receive a Context as a parameter in their constructor. In a regular Android application, you usually have two kinds of Contexts: Activity and Application. It's usually the first one that the developer passes to classes and methods that need a Context:

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    TextView label = new TextView(this);
    label.setText("Leaks are bad");
```
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”

Often: A misunderstanding of a library causes the library to keep the Activity reference.

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

```
a_static_field
```

*of type* *Activity*
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
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

a_static_field

of type Activity

Can be answered with a points-to analysis

with approximation

Hidden Truth
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

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

of type \(\text{Activity}\) ?

Can be answered with a points-to analysis with approximation

Some pointer relations \textbf{may} be false

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

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

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.

**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.

Known: Precise points-to analysis challenging enough?

---

**Manual**

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

Dagstuhl 13162: Pointer Analysis (2013)
Manual triage is particularly hard for heap reachability reports.
Manual triage is particularly hard for heap reachability reports.

allocated here

MyClass1.java
Manual triage is particularly hard for heap reachability reports.

allocated here

MyClass1.java

LibraryClass1.java
Manual triage is particularly hard for heap reachability reports.

allocated here

MyClass1.java

allocated here

LibraryClass1.java

MyClass2.java
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.

allocated here

MyClass1.java
allocated here

LibraryClass1.java

MyClass2.java

Library2Class1.class

java.util.HashMap.class

MyClass3.java
Manual triage is particularly hard for heap reachability reports.

allocated here

Get abstract heap path + maybe allocation sites

Guesstimate: >1 to 2 hours per alarm to triage “well”
Examining manual triage ...
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.
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!
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).
Thresher filters out false alarms by refuting them one-by-one.
Thresher filters out false alarms by refuting them one-by-one.

Verifier

Program

Points-To Analyzer (off-the-shelf)

Points-To Facts

Manual Triaging

Filter with Thresher

Proof of no bug

Leak Alarms

✔

✘
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

proof of no bug

Manual Triaging

Filter with Thresher

Leak Alarms

¬ proof of no bug

Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triaging

¬ Manual Triag
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

*-sensitive
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

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

Leak Alarms

Proof of no bug

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

\[ o \]

\[ o' \]

\[ \vdots \]

\[ \downarrow \]

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

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

There may be an execution where at some time of type $T$.

- $o$
- $o'$
- of type $T$. 
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

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

\[ o \]

\[ \downarrow \]

\[ o' \]

\[ \vdots \]

\[ \downarrow \]

\[ \text{of type T} \]

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

B. Because statement \( s \) may execute to make \( o \) point to \( o' \)
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

There may be an execution where at some time 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

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

\[
\begin{array}{c}
o \\
\downarrow \\
o' \\
\downarrow \\
\vdots \\
\downarrow \\
\text{of type } T.
\end{array}
\]

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 \( \varphi \)?
There may be an execution where at some time

\[ o \]
\[ \downarrow \]
\[ o' \]
\[ \ldots \]

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

There may be an execution where at some time 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 \( \varphi' \)

A just asks “but why?”

B reasons about program semantics
Refutation analysis is “Proof by Contradiction” with the “But Why?” game.

There may be an execution where at some time of type T.

A just asks “but why?”

B reasons about program semantics

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

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

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

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

set of possible states

Set $\varphi$
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

$\varphi$
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

$\varphi \xrightarrow{\text{pre}_{s'}} \varphi'$
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 special ed 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

$\varphi \xrightarrow{\text{pre}_s} \varphi'$
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 \( \varphi \) ?

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

if empty, then refuted (A wins)

set of possible states

\[ \varphi \]

\[ \psi \]

\( \text{pre}_{s'} \)

\( \varphi' \)
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 $\varphi$?

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

A. Why can the state before statement *s* satisfy formula *φ*?

B. Because before statement *s*, the program state could satisfy formula *φ*

B. Because before the previous statement *s’*, the state could satisfy formula *φ’*

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 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 $\varphi$?

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 $\varphi$?

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

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

**Technical Contribution:** Specialized constraint forms makes finding refutations feasible
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><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>

Note: The table shows the effectiveness of Thresher in filtering points-to alarms across various programs. The columns include the program names, line of code (LOC), points-to alarms, Thresher refuted, true bugs, Thresher time, false alarm percentage, and filtered percentage. Thresher appears to be effective in filtering, as indicated by the high percentages of filtered bugs and low false alarm rates.
### 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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>K9Mail</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><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 ~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><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>

- **Filtered staticfield-Activity pairs**
- **Filtered**
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><strong>72K</strong></td>
<td><strong>311</strong></td>
<td><strong>172</strong></td>
<td><strong>115</strong></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>

Filtered staticfield-Activity pairs triage “well” at 10–15 minutes per Manual

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>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><strong>72K</strong></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><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>

< coffee to lunch break
<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><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>
</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>
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>

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>Alarm %</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>
<td></td>
<td></td>
<td></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>
<td></td>
<td></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>
<td></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>
<td></td>
<td></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>
<td></td>
<td></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>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></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
    }
}
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
        }
    }
}
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
    }
}
null object pattern: should not be written to

Allocate new backing array on first write
Find the Android’s HashMap bug ...

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

An “evil” implementation of the \texttt{Map} interface can corrupt \texttt{EMPTY}. Then, all \texttt{HashMaps} created in the future will be corrupted.
null object pattern: should not be written to

allocate new backing array on first write

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?

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 0
}
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.

What if you store passwords in a HashMap?

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**
  - **Program**
  - **Specification**

- **Verifier**
  - **Valid proof of no bug**

- **Alarm Report**

- **Runner**

- **Manual Triaging**

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

- **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]

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

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

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

Program- ming

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

Verifying

- Test Input

Runner

Spec- ification

Program

Veriﬁer

- Static Incrementalization of
  Data Structure Checks
  [NSF CAREER]

Alarm

Report

✘

Manual

Triaging

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

Program-
ning

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

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

Proof of no bug

Test Output

Test Input

Manual

Triaging

PhD Advisee
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)

object[string]()

type system designers

"web 2.0" developers
Method Reflection and the Great Divide

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

```java
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

Type system designers worry.

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

“web 2.0” developers

“Web 2.0” developers think it’s cool.

I can flexible and compact code, so I will take it over static safety.
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

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

safe assuming a relationship invariant between .o and .m
Programs are often (1) safe, (2) not type safe, (3) but almost so

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

invariant holds

safe assuming a relationship invariant between .o and .m
Programs are often (1) safe, (2) not type safe, (3) but almost so.

```plaintext
safe assuming a relationship invariant between .o and .m
```
Programs are often (1) safe, (2) not type safe, (3) but almost so

\begin{align*}
\text{callback.o[callback.m]}() \\
\end{align*}

invariant holds

safe assuming a relationship invariant between .o and .m

invariant broken

but only temporarily

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

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

Tolerate "temporary" violation with

```
```

invariant holds

safe assuming a relationship invariant between .o and .m

Program

invariant broken

but only temporarily
Safe but not type safe ...
class Callback:
    var sel: Str
    var obj: Obj

def call():
    this.obj[this.sel]()

def update(s: Str, o: Obj | respondsTo s):
    this.sel = s
    this.obj = o
class Callback:
    var sel: Str
    var obj: Obj | respondsTo sel

    def call():
        this.obj[this.sel]()

    def update(s: Str, o: Obj | respondsTo s):
        this.sel = s
        this.obj = o
class Callback:
    var sel: Str | respondsTo sel
    var obj: Obj | respondsTo sel

    def call():
        this.obj[this.sel]()

    def update(s: Str, o: Obj | respondsTo s):
        this.sel = s
        this.obj = o
class Callback:
    var sel: Str
    var obj: Obj

def call():
    this.obj[this.sel]()

def update(s: Str, o: Obj | respondsTo s):
    this.sel = s
    this.obj = o
class Callback:
    var sel: Str
    var obj: Obj | respondsTo sel

def call():
    this.obj[this.sel]()

def update(s: Str, o: Obj | respondsTo s):
    this.sel = s
    this.obj = o
class Callback:
    var sel: Str
    var obj: Obj | respondsTo sel

    def call():
        this.obj[this.sel]()

    def update(s: Str, o: Obj | respondsTo s):
        this.sel = s
        this.obj = o

Type specifies a global relationship invariant

Call is safe because of the invariant

relationship invariant violated
class Callback:
    var sel: Str
    var obj: Obj | respondsTo sel

def call():
    this.obj[this.sel]()

def update(s: Str, o: Obj | respondsTo s):
    this.sel = s
    this.obj = o
Safe but not type safe ...

class Callback:
    var sel: Str
    var obj: Obj | respondsTo sel

def call():
    this.obj[this.sel]()

def update(s: Str, o: Obj | respondsTo s):
    this.sel = s
    this.obj = o

Call is safe because of the invariant

Type specifies a global relationship invariant

Tolerate “temporary” violation with
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

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 (~4 to 90 kloc/s)
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)
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.

- **Verifier**
  - ✔ proof of no bug
  - Alarm Report

- **Program**
  - Manual Triaging
  - Runner
    - Test Input
    - Specification
    - Program

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

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

- **Static Incrementalization of Data Structure Checks**
  - [under review]

- **Jsana: Abstract Domain Combinators for Dynamic Languages**
  - [Cox+ ECOOP’13, Cox+ SAS’14, under review]

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