Cooperative Program Analysis

Bor-Yuh Evan Chang
University of Colorado Boulder

National Taiwan University 國立臺灣大學
August 5, 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.”
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”
How does program analysis save the day?

Program Analysis for Formal Verification

Systematically examine the program to “simulate” running it on “all inputs”

- Program
- Verifier
- Alarm Report
- proof of no bug

Program Analysis for Formal Verification
The End?
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”
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.
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!
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.

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.

Verifier

Program

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

Manual Triaging

Alarm Report

Verified: proof of no bug

Faulty: alarm report

Proof of no bug
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**
  - ✗

- **Manual Triaging**

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

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

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

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

- **Programming**
  - Spec-ification
  - Program

- **Verifier**

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

- **Manual Triage**

- **Alarm Report**
  - Proof of no bug
Agenda: The cooperative approach addresses the whole bug mitigation process.

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

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

- **Manual**
  - Triaging

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

Verifier

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

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

- **Programming**

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

- **Test Input**

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

- **Program**

- **Manual Triaging**

- **Test Output**

- **Spec-ification**

- **Runner**

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

- **Alarm**
  ✘

- **Test**

- **Input**

- **Output**

- **Specification**

- **Thresher**

- **Runner**

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

**Program**

- **Enforcement**: Windows: Measuring Bug Avoidance [Coughlin+ ISSTA’12]
- **Program**
- **Test Input**
- **Specification**
- **Program**

**Manual Triaging**

**Verifier**

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

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

**Test Output**
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]

**Programming**

**Verifier**

✔

proof of no bug

**Alarm**

Report

✗

Program

Test Input

Spec-
ification

Program-
ming

Runner

Test Output

Test

Input

Spec-
ification

Program-
ning

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]

**Static Incrementalization of Data Structure Checks**

[under review]

**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]

- **Fissile Types:** Checking Almost Everywhere Invariants
  
  [Coughlin+ POPL’14, in prep]
Agenda: The cooperative approach addresses the whole bug mitigation process.
Threscher: Precise Refutations for Heap Reachability

Assist in triage of queries about heap relations

- Idea: Assume alarms false, prove them so automatically
- Filters out ~90% of false alarms to expose true bugs
- Going from ~450 hours of manual work to ~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 \(~90\%\) of false alarms to expose true bugs
  - Going from \(~450\) hours of manual work to \(~30\) 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 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 memory leaks underly rotation-based crashes.

- `a_static_field`
- `program heap`
- `of type Activity`
- `Activity objects encapsulate the UI`
- `I can’t collect this dead Activity!`
- `of type Activity`
Android memory leaks underly rotation-based crashes.

Android OS

of type Activity

of type Activity

I'm full of garbage!

Activity objects encapsulate the UI

I can't collect this dead Activity!

program heap

a_static_field
Android memory leaks underly rotation-based crashes.

Bug: Holding reference to “old” Activity

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

**Bug**: Holding reference to “old” Activity

"an Activity leak"

Activity objects encapsulate the UI

Android OS

program heap

of type Activity

of type Activity

I can’t collect this dead Activity!

I'm full of garbage!

"an Activity leak"
The expert recommendation ...
The expert recommendation ...

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 ensure applications run without getting them killed. The more applications Android can keep in memory, the faster it'll be for the user to switch between his apps. As part of my job, I run 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 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”

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 ...

Memory Analysis for Android Applications

The Dalvik runtime may be garbage-collected, but that doesn't mean you can ignore memory management. You should be especially mindful of memory usage on mobile devices, where memory is more constrained. In this article we're going to take a look at some of the memory profiling tools in the Android SDK that can help you trim your application's memory usage.

Some memory usage problems are obvious, for example, if your app leaks memory every time the user touches the screen, it will probably trigger an OutOfMemoryError eventually and crash your app. Other problems are more subtle, and may just degrade the performance of both your app (as garbage collections are more frequent and take longer) and the entire system.

Tools of the trade

The Android SDK provides two main ways of profiling the memory usage of an app: the Allocation Tracker in DDMS, and heap dumps. The Allocation Tracker is useful when you want to get a sense of what kinds of allocation are happening over a
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
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](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
Answering “Is there an Activity leak?” with program analysis ...

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

Is there a program execution where at some time

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

Can be answered with a points-to analysis

Example

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

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

of type \text{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

```
| a_static_field |
```

of type `Activity`

Can be answered with a points-to analysis with approximation

Some pointer relations may be false
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.

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.

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

- **Program**
  - **Points-To Analyzer**
    - **proof of no bug**
    - **✘**
  - **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.

**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
Thresher addresses alarm triage in a particularly challenging domain.

**Diagram:***
- **Program** flows through a **Points-To Analyzer** to produce **Points-To Facts**.
- **Verifier** checks for **proof of no bug** and **proof of bug**.
- **Manual** triage is indicated.

**References:**
- Hind. “Pointer Analysis: Haven’t We Solved This Problem Yet?” (2001)
- Dagstuhl 13162: Pointer Analysis (2013)

**Known:** Precise points-to analysis challenging enough?
Manual triage is particularly hard for heap reachability reports.
Manual triage is particularly hard for heap reachability reports.

allocated here

MyClass1.java

allocated here
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.

allocated here

MyClass1.java
allocated here

LibraryClass1.java

MyClass2.java

Library2Class1.class

java.util.HashMap.class
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!
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).

Examining manual triage ...
Thresher filters out false alarms by refuting them one-by-one.
Thresher filters out false alarms by refuting them one-by-one.
Thresher filters out false alarms by refuting them one-by-one.
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

* -sensitive
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

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

\[ o \]

\[ o' \]

\[ \vdots \]

\[ \text{of type } T \]
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

$$o$$

$$o'$$

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

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

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

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

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 \(o'\)?
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 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
↓
o'
↓
...
```

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 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 $\varphi'$
Refutation analysis is “Proof by Contradiction” with the “But Why?” game

There **may** be an execution where at some time $o'$. 

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

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
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
**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

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

$\varphi$

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

$\varphi'$
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 $\varphi$

If empty, then refuted (A wins)

Set of possible states

$\varphi$ -> $\psi$ -> $\varphi'$

$\text{pre}_s'$
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

$\varphi$ $\psi$

pre$_s'$ $\varphi'$
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 $\varphi'$.

pre$_s'$
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 $\varphi'$
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

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

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 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>

**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>Total</td>
<td></td>
<td>311</td>
<td>172</td>
<td>115</td>
<td>1602</td>
<td>17</td>
<td>88</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>

The table above shows the effectiveness of Thresher in filtering static field-Activity pairs. The columns represent the program, LOC, points-to alarms, and thresher refuted. The total LOC is 72K with 311 points-to alarms filtered by Thresher, resulting in 172 refuted bugs.
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>
</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**
- **Manual**

Triage “well” at 10–15 minutes per 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>
</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><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>Total</td>
<td>72K</td>
<td>311</td>
<td>172</td>
<td>115</td>
<td>1602</td>
</tr>
</tbody>
</table>

< ∼coffee to lunch break>
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>Total</td>
<td>72K</td>
<td>311</td>
<td>172</td>
<td>115</td>
<td>1602</td>
<td>17</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>
Is Thresher effective at filtering?

False alarms down to 17% from 63% (points-to analysis only)

Thresher filters 88% of false alarms from points-to analysis
Guesstimate

Triage “well” without versus with: ∼450 hours versus ∼30 hours
Triage “ok” without: ∼30 hours

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Alarms</th>
<th>Related</th>
<th>Bugs</th>
<th>Time (s)</th>
<th>Alarm % 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>
</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>

False alarms down to 17% from 63% (points-to analysis only)
Thresher filters 88% of false alarms from points-to analysis
... in the process of finding leaks in apps
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
    }
}
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 ...

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 ...

```java
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

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

    return 0

    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;
        capacity initially empty
    }
    
    void put(Object key, Object val) {
        if (need capacity) {
            this.tbl = new Object更多容量;
            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
}

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?
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
    }
    return 0
  }

  return “evil” content

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

  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
  }

  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” a refutation analysis
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]

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

**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.

- **Fissile Types**: Checking Almost Everywhere Invariants [Coughlin+ POPL’14, NSF SHF]
- **Enforcement Windows**: Measuring Bug Avoidance [Coughlin+ ISSTA’12, NSF EAGER]
- **Thresher**: Assisting Triage by Refutation Analysis [Blackshear+ PLDI’13, Blackshear+ SAS’11, NSF CAREER]
- **Jsana**: Abstract Domain Combinators for Dynamic Languages [Cox+ ECOOP’13, NSF SHF]
- **Static Incrementalization of Data Structure Checks** [NSF CAREER]
Fissile Types: Checking Reflection with Almost Everywhere Invariants
Method Reflection and the Great Divide
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 \textit{run-time value (in string)}

\texttt{object[string]()}\

\begin{itemize}
  \item \begin{itemize}
    \item type system designers
  \end{itemize}
  \item \begin{itemize}
    \item "web 2.0" developers
  \end{itemize}
\end{itemize}
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 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.

“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

```c
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[callbacl.m]()
Programs are often (1) safe, (2) not type safe, (3) but almost so

![Diagram](image)

- invariant holds

- 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

**Invariant broken**
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

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

Invariant holds

safe assuming a relationship invariant between .o and .m

Invariant broken

but only temporarily

callback.o[callback.m]()
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)

benchmarks (total)
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.