Goal-Directed Program Analysis with Jumping

Sam Blackshear
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

Manu Sridharan
Samsung Research America

École normale supérieure
1 October 2015
Lab: Program analysis in the whole bug mitigation process
Lab: **Program analysis in the whole bug mitigation process**

![Diagram showing program analysis in the whole bug mitigation process](attachment:image.jpg)

- Program
- Verifier
- Alarm Report
- Proof of no bug
- Proof of bug
Lab: **Program analysis in the whole bug mitigation process**
Lab: **Program analysis in the whole bug mitigation process**
Lab: **Program analysis in the whole bug mitigation process**

Verifier

**Thresher: Goal-Directed Refutation Analysis**

[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
Lab: **Program analysis in the whole bug mitigation process**
Lab: Program analysis in the whole bug mitigation process

Verifier

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

proof of no bug

Alarm Report

Manual Triaging

Program

Program-
ning

Spec-
ification

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

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

Verifier

proof of no bug

Alarm Report

Manual Triaging

Program

Spec-ification

Program- ming

Lab: Program analysis in the whole bug mitigation process
Program analysis in the whole bug mitigation process

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

Intertwine type-based and separation-based analysis to verify almost-everywhere invariants

type-intertwined separation logic
Program analysis in the whole bug mitigation process

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

Verifier

proof of no bug

Alarm Report

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
Analysis in the whole bug mitigation process

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

**Program:**
- Programming

**Verifier**
- Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
- Proof of no bug

**Alarm Report**

**Manual Triaging**

**Test Input**

**Test Output**

**Runner**

**Test Input**

**Spec-ification**

**Program**

**Test Output**

**Alarm Report**
Program analysis in the whole bug mitigation process:

- **Verifier**: Synthesizing Short-Circuiting Data Structure Checks
  - [under review]

- **Divva**: Synthesizing Short-Circuiting Data Structure Checks
  - [under review]

- **Thresher**: Goal-Directed Refutation Analysis
  - [Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

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

- **Manual Triaging**

- **Test Input**

- **Test Output**

- **Alarm Report**

- **proof of no bug**
Analysis in the whole bug mitigation process:

**Lab:** Program analysis in the whole bug mitigation process

- **Verifier**
  - ✔ Proof of no bug
  - ✘ Alarm

**Program**: Program analysis in the whole bug mitigation process

- **Test Input**
- **Specification**
- **Program**

**Manual Triaging**

**Thresher**: Goal-Directed Refutation Analysis

[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

**Fissile Types**: Checking Almost Everywhere Invariants

[Coughlin+ POPL’14, in prep]

**Divva**: Synthesizing Short-Circuiting Data Structure Checks

[under review]

Use static shape analysis to synthesize short-circuiting dynamic validation of data structure invariants

uninterpreted inductive separation logic predicates

**Use static shape analysis to synthesize short-circuiting dynamic validation of data structure invariants**

**Divva**: Synthesizing Short-Circuiting Data Structure Checks

[under review]

**Use static shape analysis to synthesize short-circuiting dynamic validation of data structure invariants**

uninterpreted inductive separation logic predicates
**Lab: Program Analysis in the whole bug mitigation process**

- **Verifier**
  - **Proof of no bug**: ✔
  - **Alarm Report**: ✘

- **Thresher**: Goal-Directed Refutation Analysis
  - **Blackshear** + SAS’11, Blackshear + PLDI’13, under review

- **Divva**: Synthesizing Short-Circuiting Data Structure Checks
  - under review

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

- **Manual Triaging**
Program analysis in the whole bug mitigation process

Verifier

Test Output

proof of no bug

Alarm Report

Thresher: Goal-Directed Refutation Analysis

[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

Fissile Types: Checking Almost Everywhere Invariants

[Coughlin+ POPL’14, in prep]

Divva: Synthesizing Short-Circuiting Data Structure Checks

[under review]

Github

Program

Manual Triaging

Test Input

Spec-ification

Runner

Test Output
Program analysis in the whole bug mitigation process:

- **Verifier**
  - Proof of no bug
  - Alarm report

- **Runner**
  - Test input
  - Test output

- **Program**
  - Specification
  - Program

- **Manual Triaging**

- **Fissile Types**
  - Checking Almost Everywhere Invariants
    - [Coughlin et al. POPL'14, in prep]

- **Divva**
  - Synthesizing Short-Circuiting Data Structure Checks
    - [under review]

- **Thresher**
  - Goal-Directed Refutation Analysis
    - [Blackshear et al. SAS'11, Blackshear et al. PLDI'13, under review]

- **Github**

- **Fixr**
  - Mining Bug Fixes from Commits
    - [in progress]

- **Manual Triaging**
Analysis in the whole bug mitigation process:

**Program analysis**

- **Program**: Checking Almost Everywhere Invariants
  
  - *Fissile Types*: Checking Almost Everywhere Invariants
  
  - *Jsana*: Abstract Domain Combinators for Dynamic Languages
  
- **Verifier**: Goal-Directed Refutation Analysis

  - *Thresher*: Goal-Directed Refutation Analysis

- **Manual Triaging**

  - *Fixr*: Mining Bug Fixes from Commits

**Github**

**Manual Testing**

- **Test Input**

- **Test Output**
**Lab: Program analysis in the whole bug mitigation process**

**Divva: Synthesizing Short-Circuiting Data Structure Checks**
[Blackshear + SAS'11, Blackshear + PLDI'13, under review]

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

**Jsana: Abstract Domain Combinators for Dynamic Languages**
[Cox+ ECOOP'13, Cox+ SAS’14, Cox+ ESOP’15]

**Fixr: Mining Bug Fixes from Commits**
in progress

Heap with **set symbols partitioning “open objects”**

<table>
<thead>
<tr>
<th></th>
<th>A1</th>
<th>A2</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>V1</td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>V2</td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td></td>
<td>A2</td>
</tr>
</tbody>
</table>

Separation logic with open object predicates and desynchronized separation
Program analysis in the whole bug mitigation process:

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

- **Divva**: Synthesizing Short-Circuiting Data Structure Checks
  - [under review]

- **Jsana**: Abstract Domain Combinators for Dynamic Languages
  - [Cox+ ECOOP’13, Cox+ SAS’14, Cox+ ESOP’15]

- **Thresher**: Goal-Directed Refutation Analysis
  - [Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

- **Github**: Fixr: Mining Bug Fixes from Commits
  - [in progress]
Lab: Program analysis in the whole bug mitigation process

This Talk

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

Verifier

✔
proof of no bug

✔
proof of no bug

 verifier

Alarm

Report

✘

✘

Alarm

Report

Test

Input

Runner

Test

Output

Program

Program-

ming

Github

Fixr: Mining Bug Fixes from Commits
[in progress]

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

Fissile Types:

Divva: Synthesizing Short-Circuiting Data Structure Checks
[under review]

Jsana: Abstract Domain Combinators for Dynamic Languages
[Cox+ ECOOP’13, Cox+ SAS’14, Cox+ ESOP’15]

Jsana: Abstract Domain Combinators
for Dynamic Languages

[under review]

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
Goal-Directed Program Analysis with Jumping

Sam Blackshear
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

Manu Sridharan
Samsung Research America

École normale supérieure
1 October 2015
Roughly, 3% of all commits fix NullPointerExceptions
Callback-oriented programming
Callback-oriented programming

Android Framework

App
Callback-oriented programming

callbacks (e.g., Activity.onCreate)
Callback-oriented programming

callbacks (e.g., Activity.onCreate)
Callback-oriented programming

Android components have an ordered, event-driven lifecycle

callbacks (e.g., Activity.onCreate)
Callback-oriented programming

Android components have an ordered, event-driven lifecycle.

callbacks (e.g., Activity.onCreate)

mHostDb = null;
mService = null;
Callback-oriented programming

Android components have an ordered, event-driven lifecycle.

But, lifecycles of different components and other callbacks can interleave ...

mHostDb = null;
mService = null;
Callback-oriented programming

Challenge: Verifying safety (of dereferences) depends on **callback interleaving**

Android components have an ordered, event-driven lifecycle

callbacks (e.g., Activity.onCreate)

But, lifecycles of different components and other callbacks can interleave ...

```java
mHostDb = null;
mService = null;
```
Callback-oriented programming

```
onCreate
  ↓
onResume
  ↓
onClick
  ↓
onPause
  ↓
onDestroy
```
Callback-oriented programming

onCreate → onResume → onClick → onPause → onDestroy

onCreate → onResume → onClick → onPause → onDestroy
Callback-oriented programming

Heap

- onCreate
- onResume
- onClick
- onPause
- onDestroy
Callback-oriented programming

Heap

... and operates over a shared, global heap

onCreate

onResume

onClick

onPause

onDestroy

onCreate

onResume

onClick

onPause

onDestroy
Callback-oriented programming

Challenge: Verifying safety (of dereferences) depends on the ordering of heap writes.
But it shouldn’t be so hard ...
But it shouldn’t be so hard ...

```java
mHostDb.s();
```

```
void onCreate()
void onResume()
void onClick()
void onPause()
void onDestroy()
```
But it shouldn’t be so hard …

```java
mHostDb.s();
```

```
safe?
```
But it shouldn’t be so hard ...

safe?
mHostDb.s();

Idea: Safety of a particular dereference should not require reasoning about all callback interleavings
But it shouldn’t be so hard …

Idea: Safety of a particular dereference should not require reasoning about all callback interleavings. A “smart” goal-directed analysis could consider relevant callback orderings without considering all of them.
Goal-directed program analysis

mHostDb.s();
safe?
Goal-directed program analysis

Given a program configuration **goal**, derive a **contradiction** w.r.t. its reachability

```
mHostDb.s();
safe?
```
Goal-directed program analysis

Given a program configuration \textbf{goal}, derive a \textbf{contradiction} w.r.t. its reachability

\begin{align*}
\text{safe?} \quad & \text{mHostDb.s();} \\
& \text{mHostDb == null}
\end{align*}
Goal-directed program analysis

Given a program configuration **goal**, derive a **contradiction** w.r.t. its reachability

```java
mHostDb == null
mHostDb.s();
```

`safe?`
Goal-directed program analysis

Given a program configuration goal, derive a contradiction w.r.t. its reachability

\[(\text{this} \leftrightarrow \hat{t} \ast \hat{t} \cdot \text{mHostDb} \leftrightarrow \hat{a} \ast \text{true}) \land \hat{a} = \text{null}\]
Given a program configuration goal, derive a contradiction w.r.t. its reachability.

\[(\text{this} \leftrightarrow \hat{t} \ast \hat{t} \cdot \text{mHostDb} \leftrightarrow \hat{a} \ast \text{true}) \land \hat{a} = \text{null}\]

A precise backwards abstract interpretation with separation logic constraints to refute error conditions [PLDI’13]
Two dereferences: one safe and one buggy
void onClick(...) {
    mHostDb.s(mService.g());
}

Two dereferences: one safe and one buggy
Two dereferences: one safe and one buggy

```java
void onClick(...) {
    mHostDb.s(mService.g());
}
```

1 safe? 2
Two dereferences: one safe and one buggy

```java
void onClick(…) {
    mHostDb.s(mService.g());
}
```

```java
void onDestroy(…) {
    mHostDb = null;
    mService = null;
}
```
Two dereferences: one safe and one buggy

```
void onClick(...) {
    mHostDb.s(mService.g());
}
```

```
void onDestroy(...) {
    mHostDb = null;
    mService = null;
}
```
Two dereferences: one safe and one buggy

```java
void onCreateView() {
    bindService(..., new ServiceConn {
        void onConnected(@Nonnull Service s) {
            mService = s;
        }
    });
    mHostDb = new Db();
}

void onClick(…) {
    mHostDb.s(mService.g());
}

void onDestroy(…) {
    mHostDb = null;
    mService = null;
}
```
void **onCreate**(...) {
    bindService(..., new ServiceConn {
        void onConnected(@Nonnull Service s) {
            mService = s;
        }
    });
    mHostDb = new Db();
}

void **onClick**(...) {
    mHostDb.s(mService.g());
}

void **onDestroy**(...) {
    mHostDb = null;
    mService = null;
}
void **onCreate**() {
    bindService(…, new ServiceConn {
        void onConnected(@Nonnull Service s) {
            mService = s;
        }
    });
    mHostDb = new Db();
}

void **onClick**(...) {
    mHostDb.s(mService.g());
}

void **onDestroy**(...) {
    mHostDb = null;
    mService = null;
}
Two dereferences: one safe and one buggy

```java
void onCreate() {
    bindService(…, new ServiceConn {
        void onConnected(@Nonnull Service s) {
            mService = s;
        }
    });
    mHostDb = new Db();
}

void onClick(...) {
    mHostDb.s(mService.g());
}

void onDestroy(...) {
    mHostDb = null;
    mService = null;
}
```

Need to consider **some but not all** callback ordering constraints.
Idea: Jumping to relevant callbacks

- onCreate
- onResume
- onClick
- onPause
- onDestroy
Idea: Jumping to relevant callbacks

- onCreate
- onResume
- onClick
- onPause
- onDestroy

- onCreate
- onResume
- onClick
- onPause
- onDestroy

- onCreate
- onResume
- onClick
- onPause
- onDestroy
Idea: *Jumping* to relevant callbacks

- onCreate
- onResume
- onClick
- onDestroy
- onPause

safe?

- onCreate
- onResume
- onClick
- OnDestroy

- onCreate
- onPause
- OnDestroy

- onClick
- OnDestroy
Idea: Jumping to relevant callbacks

Find data relevant callbacks
Idea: Jumping to relevant callbacks

Find data relevant callbacks
Idea: **Jumping to relevant callbacks**

- `onCreate`
- `onResume`
- `onClick`
- `onDestroy`
- `onPause`
Idea: Jumping to relevant callbacks

Find data relevant callbacks
Idea: Jumping to relevant callbacks

Find data relevant callbacks
Filter using control-feasibility
Idea: Jumping to relevant callbacks

Find data relevant callbacks

Filter using control-feasibility
Idea: Jumping to relevant callbacks

Find data relevant callbacks

Filter using control-feasibility
Idea: Jumping to relevant callbacks

Find data relevant callbacks

Filter using control-feasibility

backwards analysis “jumps”
onCreate \rightarrow onClick \rightarrow onCreate

backwards analysis
“jumps”
Contributions

backwards analysis
"jumps"

\[ l_1 \xrightarrow{c} l_2 \]

Framework for jumping analyses
Contributions

1. Framework for jumping analyses
   \[ \ell_1 \xrightarrow{[c]} \ell_2 \]

2. Applied to Android events
   "jumps"
A program model for a jumping analysis

\[
l_1 \xrightarrow{[c]} l_2
\]
A program model for a jumping analysis

\[ l_1 \rightarrow [c] \rightarrow l_2 \]
A program model for a jumping analysis

Unstructured control-flow: atomic commands connected by pre/post labels
A program model for a jumping analysis

\[ \ell_1 \rightarrow [c] \rightarrow \ell_2 \]

Program

Unstructured control-flow: atomic commands connected by pre/post labels

Represent conditionals with \texttt{assume}, loops with \texttt{assume} and a back edge.
A program model for a jumping analysis

Unstructured control-flow: atomic commands connected by pre/post labels

Represent conditionals with assume, loops with assume and a back edge.

Analysis parameters
A program model for a jumping analysis

Unstructured control-flow: atomic commands connected by pre/post labels

Represent conditionals with \texttt{assume}, loops with \texttt{assume} and a back edge.

Analysis parameters

\[ \{Q'\} \ c \ \{Q\} \]

Backward transfer functions
A program model for a jumping analysis

Unstructured control-flow: atomic commands connected by pre/post labels

Represent conditionals with `assume`, loops with `assume` and a back edge.

Analysis parameters

\[ \{Q'\} \ c \ \{Q\} \]

- Backward transfer functions
- Current location

Current location: \( \ell_{\text{cur}} \)
A program model for a jumping analysis

Unstructured control-flow: atomic commands connected by pre/post labels

Represent conditionals with `assume`, loops with `assume` and a back edge.

Analysis parameters

\[ \{ Q' \} \quad c \quad \{ Q \} \]

- Backward transfer functions
- Current location \( \ell_{\text{cur}} \)
- Current query \( Q_{\text{cur}} \)
Data-relevance identifies writes

\( (Q, \ell_{\text{cur}}) \)
Data-relevance identifies writes

\[(Q, \ell_{\text{cur}})\]

Identify locations that can write to the query \(Q\).
Data-relevance identifies writes

\[(Q, l_{\text{cur}})\]

Data-relevance

\[l_i \quad l_j \quad l_k \quad \ldots\]

Identify locations that can write to the query \(Q\).

Computed using pre-pass points-to analysis, types, field-based, ...
Data-relevance identifies writes

Identify locations that can write to the query $Q$.

Computed using pre-pass points-to analysis, types, field-based, ...

Classic idea: Following data dependencies yields a \textbf{sparse} analysis (but, here, flow-insensitive)
Control-feasibility \textit{selectively} recovers flow-sensitivity

\[(Q, \ell_{\text{cur}})\]

\textbf{Data-relevance}

\[l_i \quad l_j \quad l_k \quad \ldots\]
Control-feasibility *selectively* recovers flow-sensitivity

\[(Q, \ell_{\text{cur}})\]

Data-relevance

\[\ell_i \quad \ell_j \quad \ell_k \quad \ldots\]

Control-feasibility

\[\ell_i \quad \ell_j \quad \ldots\]
Control-feasibility selectively recovers flow-sensitivity

Filter the set of data-relevant locations using control flow and the current program point.
Filter the set of data-relevant locations using control flow and the current program point.
Control-feasibility selectively recovers flow-sensitivity

\[(Q, \ell_{\text{cur}})\]

Data-relevance

\[\ell_i, \ell_j, \ldots, \ell_k\]

Control-feasibility

\[\ell_{\text{cur}}, \ell_i, \ell_j, \ldots\]

Filter the set of data-relevant locations using control flow and the current program point

Not backward-reachable from current location
Control-feasibility selectively recovers flow-sensitivity

\[(Q, \ell_{\text{cur}})\]

Data-relevance

\[l_i \quad l_j \quad l_k \quad \ldots\]

Control-feasibility

\[l_i \quad l_j\]

\[\text{Must visit another relevant location first.}\]

Filter the set of data-relevant locations using control flow and the current program point

\[\text{Not backward-reachable from current location}\]
Jumping enables sparse, selective, on-the-fly control-flow abstraction
Jumping enables sparse, selective, on-the-fly control-flow abstraction.
Jumping enables sparse, selective, on-the-fly control-flow abstraction.
Jumping enables sparse, selective, on-the-fly control-flow abstraction.
Jumping enables sparse, selective, on-the-fly control-flow abstraction.
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Next locations

Control-feasibility

Data-relevance

\( \ell_{\text{cur}} \)

Current location

\( Q_{\text{cur}} \)

Current query

\( \ell_1 \xrightarrow{[c]} \ell_2 \)
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ l_1 \xrightarrow{[c]} l_2 \]

Control-feasibility

Data-relevance

\[ l_{\text{cur}} \]

\[ Q_{\text{cur}} \]

Current location

Current query

no filter
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Current location $\ell_{\text{cur}}$, Current query $Q_{\text{cur}}$,

Data-relevance $\ell$,

Control-feasibility $[c] \rightarrow \ell_{2}$

No filter, all locations.
Jumping enables sparse, selective, on-the-fly control-flow abstraction

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

no filter

all locations

\[
\ell_1 \xrightarrow{c} \ell_2
\]
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Current location \(l\) ➔ \(l'\) via query \(Q\) cur

Data-relevance

Control-feasibility

Next locations

no filter

all locations
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Next locations:

- only preds($\ell_{\text{cur}}$)
- all locations

Data-relevance

Control-feasibility

Current location

Current query

$l_1 \xrightarrow{c} l_2$
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

only preds($\ell_{\text{cur}}$)

all locations
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

only \( \text{preds}(\ell_{\text{cur}}) \)

all locations
Jumping enables sparse, selective, on-the-fly control-flow abstraction

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
<th>✔</th>
<th>✔</th>
<th>✔</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>path</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

only feasible-preds\((l_{\text{cur}}, Q_{\text{cur}})\)

all locations
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>path</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>context</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

only feasible-preds($l_{cur}, Q_{cur}$)

all locations

Current location

Current query

Control-feasibility

Data-relevance

Next locations

$1 \xrightarrow{[c]} l_2$
Jumping enables sparse, selective, on-the-fly control-flow abstraction

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>path</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>context</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

only feasible-preds($\ell_{\text{cur}}, Q_{\text{cur}}$)

all locations
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>path</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>context</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

only feasible-preds($\ell_{\text{cur}}, Q_{\text{cur}}$)

all locations, except assumes
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>path</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>context</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

only feasible-preds($\ell_{cur}$, $Q_{cur}$)

all locations, except assumes
Jumping enables sparse, selective, on-the-fly control-flow abstraction.

<table>
<thead>
<tr>
<th>Precision</th>
<th>-insensitive</th>
<th>-sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>path</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>context</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

only feasible-preds($\ell_{\text{cur}}, Q_{\text{cur}}$)

all locations, except assumes
Jumping enables sparse, selective, on-the-fly control-flow abstraction

only feasible-preds($\ell_{\text{cur}}, Q_{\text{cur}}$)

all locations
Jumping enables sparse, selective, on-the-fly control-flow abstraction

only feasible-preds($\ell_{cur}, Q_{cur}$)

all mods($Q_{cur}$)
Jumping enables sparse, selective, on-the-fly control-flow abstraction

Be **sparse** by using data relevance with desired control-flow abstraction

only feasible-preds($\ell_{\text{cur}}, Q_{\text{cur}}$)

all mods($Q_{\text{cur}}$)

Data-relevance

Control-feasibility

$\ell_{\text{cur}}$

$Q_{\text{cur}}$

Current location

Current query
Jumping enables sparse, selective, on-the-fly control-flow abstraction

Be **sparse** by using data relevance with desired control-flow abstraction

- only feasible-preds($\ell_{\text{cur}}, Q_{\text{cur}}$)
- all mods($Q_{\text{cur}}$)

Be **selective** by varying the relevance relation at each analysis step
Contributions

1. Framework for jumping analyses
   \[ l_1 \xrightarrow{c} l_2 \]

2. Applied to Android events
Contributions

1. Framework for jumping analyses

2. Applied to Android events

backwards analysis “jumps”
Formalizing lifecycle graphs

onCreate

onResume

onClick

onPause

onDestroy
Formalizing lifecycle graphs

“Lifecycle automata” accepts the concrete trace of callbacks projected onto its transitions
Formalizing lifecycle graphs

“Lifecycle automata” accepts the concrete trace of callbacks projected onto its transitions

Further scalability challenge: Lifecycle spec per class but analysis applies per object
A jumping policy for Android analysis
Within an event-callback (intra-event), follow predecessor transitions

no jumping, path/context-sensitive
Within an event-callback (intra-event), follow predecessor transitions

no jumping, path/context-sensitive

Between event-callbacks (inter-event), jump using lifecycle graphs for control-feasibility filtering
Hypothesis: Jumping is an effective approach to path-sensitive, inter-event analysis
Setup: Proving dereferences safe
Setup: Proving dereferences safe

10 open source Android apps
3,000 to 55,000 lines of code
10 to 100 components
120 to 1,320 callbacks
Setup: Proving dereferences safe

10 open source Android apps
3,000 to 55,000 lines of code
10 to 100 components
120 to 1,320 callbacks

Event product graph would have $10^{10}$ to $10^{111}$ nodes (with one instance per class)
Setup: Proving dereferences safe

10 open source Android apps
3,000 to 55,000 lines of code
10 to 100 components
120 to 1,320 callbacks
Setup: Proving dereferences safe

10 open source Android apps
3,000 to 55,000 lines of code
10 to 100 components
120 to 1,320 callbacks

Compared 3 analyses

**Nit**: flow-insensitive

**Thresher**: no jumping

**Hopper**: jumping
Is jumping effective?

<table>
<thead>
<tr>
<th></th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr</th>
<th>Hop (Impr %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179</td>
<td>72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181</td>
<td>51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518</td>
<td>143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954</td>
<td>477 (50)</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601</td>
<td>290 (52)</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986</td>
<td>625 (37)</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316</td>
<td>74 (77)</td>
</tr>
<tr>
<td>textsecure</td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698</td>
<td>330 (53)</td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104</td>
<td>1988 (36)</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431</td>
<td>1362 (44)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>266</strong></td>
<td><strong>62562</strong></td>
<td><strong>41071</strong></td>
<td><strong>9968</strong></td>
<td><strong>5412 (54)</strong></td>
</tr>
</tbody>
</table>
Is jumping effective?

<table>
<thead>
<tr>
<th></th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr</th>
<th>Hop (Impr %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179</td>
<td>72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181</td>
<td>51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518</td>
<td>143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954</td>
<td>477 (50)</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601</td>
<td>290 (52)</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986</td>
<td>625 (37)</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316</td>
<td>74 (77)</td>
</tr>
<tr>
<td></td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698</td>
<td>330 (53)</td>
</tr>
<tr>
<td>Total</td>
<td>266</td>
<td>62562</td>
<td>41071</td>
<td>9968</td>
<td>5412 (54)</td>
</tr>
</tbody>
</table>

Many derefs
Is jumping effective?

<table>
<thead>
<tr>
<th></th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr Hop (Impr %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179, 72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181, 51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518, 143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954, 477 (50)</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601, 290 (52)</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986, 625 (37)</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316, 74 (77)</td>
</tr>
<tr>
<td>textsecure</td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698, 330 (53)</td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104, 1988 (36)</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431, 1362 (44)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>266</strong></td>
<td><strong>62562</strong></td>
<td><strong>41071</strong></td>
<td><strong>9968</strong>, <strong>5412 (54)</strong></td>
</tr>
</tbody>
</table>
## Is jumping effective?

<table>
<thead>
<tr>
<th></th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr</th>
<th>Hop (Impr %)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flow-insensitive</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179</td>
<td>72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181</td>
<td>51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518</td>
<td>143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954</td>
<td>477 (50)</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601</td>
<td>290 (52)</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986</td>
<td>625 (37)</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316</td>
<td>74 (77)</td>
</tr>
<tr>
<td>textsecure</td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698</td>
<td>330 (53)</td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104</td>
<td>1988 (36)</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431</td>
<td>1362 (44)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>266</strong></td>
<td><strong>62562</strong></td>
<td><strong>41071</strong></td>
<td><strong>9968</strong></td>
<td><strong>5412 (54)</strong></td>
</tr>
</tbody>
</table>

**unproven derefs**

- Flow-insensitive
- No jumping
- Jumping
Is jumping effective?

### unproven derefs

<table>
<thead>
<tr>
<th></th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr</th>
<th>Hop (Impr %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179</td>
<td>72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181</td>
<td>51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518</td>
<td>143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954</td>
<td>477 (50)</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601</td>
<td>290 (52)</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986</td>
<td>625 (37)</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316</td>
<td>74 (77)</td>
</tr>
<tr>
<td>textsecure</td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698</td>
<td>330 (53)</td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104</td>
<td>1988 (36)</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431</td>
<td>1362 (44)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>266</strong></td>
<td><strong>62562</strong></td>
<td><strong>41071</strong></td>
<td><strong>9968</strong></td>
<td><strong>5412 (54)</strong></td>
</tr>
</tbody>
</table>
Is jumping effective?

<table>
<thead>
<tr>
<th>Project</th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr</th>
<th>Hop (Impr %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179</td>
<td>72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181</td>
<td>51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518</td>
<td>143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954</td>
<td>477 (50)</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601</td>
<td>290 (52)</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986</td>
<td>625 (37)</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316</td>
<td>74 (77)</td>
</tr>
<tr>
<td>textsecure</td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698</td>
<td>330 (53)</td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104</td>
<td>1988 (36)</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431</td>
<td>1362 (44)</td>
</tr>
<tr>
<td>Total</td>
<td>266</td>
<td>62562</td>
<td>41071</td>
<td>9968</td>
<td>5412 (54)</td>
</tr>
</tbody>
</table>

Thr and Hop: 10 second budget per deref
Is jumping effective?

<table>
<thead>
<tr>
<th></th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr</th>
<th>Hop (Impr %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179</td>
<td>72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181</td>
<td>51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518</td>
<td>143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954</td>
<td>477 (50)</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601</td>
<td>290 (52)</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986</td>
<td>625 (37)</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316</td>
<td>74 (77)</td>
</tr>
<tr>
<td>textsecure</td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698</td>
<td>330 (53)</td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104</td>
<td>1988 (36)</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431</td>
<td>1362 (44)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>266</strong></td>
<td><strong>62562</strong></td>
<td><strong>41071</strong></td>
<td><strong>9968</strong></td>
<td><strong>5412 (54)</strong></td>
</tr>
</tbody>
</table>
Is jumping effective?

<table>
<thead>
<tr>
<th></th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr Hop (Impr %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179 72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181 51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518 143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954 477 (50)</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601 290 (52)</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986 625 (37)</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316 74 (77)</td>
</tr>
<tr>
<td>textsecure</td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698 330 (53)</td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104 1988 (36)</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431 1362 (44)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>266</td>
<td>62562</td>
<td>41071</td>
<td>9968 5412 (54)</td>
</tr>
</tbody>
</table>

Thresher has 74% fewer than Nit
Is jumping effective?

<table>
<thead>
<tr>
<th></th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr</th>
<th>Hop (Impr %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179</td>
<td>72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181</td>
<td>51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518</td>
<td>143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954</td>
<td>477 (50)</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601</td>
<td>290 (52)</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986</td>
<td>625 (37)</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316</td>
<td>74 (77)</td>
</tr>
<tr>
<td>textsecure</td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698</td>
<td>330 (53)</td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104</td>
<td>1988 (36)</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431</td>
<td>1362 (44)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>266</strong></td>
<td><strong>62562</strong></td>
<td><strong>41071</strong></td>
<td><strong>9968</strong></td>
<td><strong>5412 (54)</strong></td>
</tr>
</tbody>
</table>

Hopper has 54% fewer than Thresher
Is jumping effective?

<table>
<thead>
<tr>
<th>Repository</th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr Hop (Impr %)</th>
<th>% Proven</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179</td>
<td>72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181</td>
<td>51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518</td>
<td>143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954</td>
<td>477 (50)</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601</td>
<td>290 (52)</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986</td>
<td>625 (37)</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316</td>
<td>74 (77)</td>
</tr>
<tr>
<td>textsecure</td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698</td>
<td>330 (53)</td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104</td>
<td>1988 (36)</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431</td>
<td>1362 (44)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>266</strong></td>
<td><strong>62562</strong></td>
<td><strong>41071</strong></td>
<td><strong>9968</strong></td>
<td><strong>5412 (54)</strong></td>
</tr>
</tbody>
</table>
Is jumping effective?

<table>
<thead>
<tr>
<th>unproven derefs</th>
<th>KLOC</th>
<th>Deref</th>
<th>Nit</th>
<th>Thr Hop (Impr %)</th>
<th>% Proven</th>
</tr>
</thead>
<tbody>
<tr>
<td>drupaleditor</td>
<td>3</td>
<td>928</td>
<td>679</td>
<td>179</td>
<td>72 (60)</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181</td>
<td>51 (72)</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518</td>
<td>143 (72)</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954</td>
<td>477 (50)</td>
</tr>
<tr>
<td>github</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>seriesguide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>connectbot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>textsecure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104</td>
<td>1988 (36)</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431</td>
<td>1362 (44)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>266</strong></td>
<td><strong>62562</strong></td>
<td><strong>41071</strong></td>
<td><strong>9968</strong></td>
<td><strong>5412 (54)</strong></td>
</tr>
</tbody>
</table>

Compare with state-of-the-art NPE checking work that reports 84-91% proven on normal Java programs!
Triaging alarms to find bugs
Triaging alarms to find bugs

Triaged 200 alarms (from Hopper), 189 false
Triaging alarms to find bugs

Triaged 200 alarms (from Hopper), 189 false

Reasons: insufficient Android modeling, imprecise container and string domains
Triaging alarms to find bugs

Triaged 200 alarms (from Hopper), 189 false

Reasons: insufficient Android modeling, imprecise container and string domains

Only 17 false alarms due to timeouts
Triaging alarms to find bugs

Triaged 200 alarms (from Hopper), 189 false

Reasons: insufficient Android modeling, imprecise container and string domains

Only 17 false alarms due to timeouts

Found 11 bugs in 4 apps
(lastfm, seriesguide, connectbot, wordpress)

5 bugs due to **bad ordering assumptions**

10/11 patches accepted
Triaging alarms to find bugs

Found 11 bugs in 4 apps (lastfm, seriesguide, connectbot, wordpress)

5 bugs due to **bad ordering assumptions**

10/11 patches accepted
Summary
Selective control-flow abstraction via a sound relevance relation
Selective control-flow abstraction via a sound relevance relation

Effective inter-event ordering-sensitive reasoning via data-relevance and control-feasibility
Selective control-flow abstraction via a sound relevance relation

Effective inter-event ordering-sensitive reasoning via data-relevance and control-feasibility

www.cs.colorado.edu/~bec
pl.cs.colorado.edu

Cerny
Chang
Hammer
Sankaranarayanan
Somenzi

PLV
Goal-directed: abstraction-refinement versus abstraction-coarsening

staged abstraction refinement (e.g., CEGAR) on-the-fly abstraction coarsening
Goal-directed: abstraction-refinement versus abstraction-coarsening

Staged abstraction refinement (e.g., CEGAR)

Run analysis multiple times, abstraction changes only between runs.

On-the-fly abstraction coarsening

Run analysis once, abstraction changes during analysis.
Goal-directed: abstraction-refinement versus abstraction-coarsening

**Staged**
- Run analysis multiple times, abstraction changes only between runs.

**Refinement**
- Start with imprecise abstraction, become more precise.

**On-the-fly**
- Run analysis once, abstraction changes during analysis.

**Coarsening**
- Start with precise abstraction, become less precise.