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

MPI-SWS
3 October 2015

To appear at OOPSLA 2015
Roughly, 3% of all commits fix NullPointerExceptions
Callback-oriented programming

App
Callback-oriented programming
Callback-oriented programming

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

- App
- Android Framework
- Activity
  - onCreate
  - onResume
  - onClick
  - onPause
  - onDestroy

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 Framework

Activity

onCreate

onResume

onClick

onPause

onDestroy
Callback-oriented programming

Android components have an ordered, event-driven lifecycle

callbacks (e.g., Activity.onCreate)

Activity

onCreate

onResume

onClick

onPause

onDestroy

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

Android components have an ordered, event-driven lifecycle.

Activity

- onCreate
- onResume
- onClick
- onPause
- onDestroy

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

mHostDb = null;
mService = null;
Challenge: Verifying safety (of dereferences) depends on callback interleaving

Android components have an ordered, event-driven lifecycle

callbacks (e.g., Activity.onCreate)

onCreate

onResume

onPause

onDestroy

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

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

onCreate

onResume

onPause

onClick

onDestroy
Callback-oriented programming
Callback-oriented programming

Heap

onCreate

onResume

onClick

onPause

onDestroy

onCreate

onResume

onClick

onPause

onDestroy
Callback-oriented programming

Heap

.onCreate

.onResume

.onClick

.onPause

.onDestroy

.onCreate

.onResume

.onClick

.onPause

.onDestroy

... and operates over a shared, global heap
Callback-oriented programming

Heap

onCreate \rightarrow onResume

onCreate \rightarrow onResume

onClick \rightarrow onClick

... and operates over a shared, global heap

Challenge: Verifying safety (of dereferences) depends ordering of heap writes
Verification challenge: Interleavings

Diagram showing a sequence of events labeled $e_1$, $e_2$, $e_3$, $e_4$, $e_5$, $e_6$, $e_7$, $e_8$, and $e_9$. The events are connected by arrows indicating a possible order or flow.
Verification challenge: Interleavings
Verification challenge: Interleavings

Previous "lifecycle-sensitive" analyses do not consider inter-component interleavings
But it shouldn’t be so hard ...

onCreate

onResume

onClick

onPause

onDestroy
mHostDb.s();

But it shouldn’t be so hard ...
But it shouldn’t be so hard ...

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

```java
onCreate
```

```java
onResume
```

```java
onClick
```

```java
onPause
```

```java
onDestroy
```

safe?
But it shouldn’t be so hard …

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

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

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

safe?

mHostDb.s();
Goal-directed program analysis

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

safe?

mHostDb.s();
Goal-directed program analysis

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

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

```java
mHostDb == null
```
Given a program configuration \textbf{goal}, derive a \textbf{contradiction} \textit{w.r.t.} its reachability.

\texttt{safe?}

\texttt{mHostDb.s()};

\texttt{mHostDb == null}
Goal-directed program analysis

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

\[(\text{this} \mapsto \hat{t} \ast \hat{t} \cdot \text{mHostDb} \mapsto \hat{a} \ast \text{true}) \land \hat{a} = \text{null}\]
Goal-directed program analysis

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

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

Thresher: A precise backwards abstract interpretation with separation logic constraints to refute error conditions [PLDI’13]
Two dereferences: one safe and one buggy
Two dereferences: one safe and one buggy

```
void onClick(...) {
    mHostDb.s(mService.g());
}
```
Two dereferences: one safe and one buggy

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

```java
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;
}
```
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;
}
```
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

onResume -> onClick

onClick -> onPause

onPause -> onDestroy

onCreate -> onResume

onResume -> onClick

onClick -> onPause

onPause -> onDestroy

onCreate -> onResume

onResume -> onClick

onClick -> onDestroy

onPause -> onDestroy
Idea: Jumping to relevant callbacks

- onCreate
- onResume
- onClick
- onDestroy
- onPause
- onClick
- onDestroy
- onCreateView
- onResume
- onClick
- onDestroy
- onPause
- 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

Find data relevant callbacks
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**

- `onCreate`
- `onResume`
- `onResume`
- `onClick`
- `onDestroy`
- `onCreate`
- `onclick`
- `onDestroy`
- `onCreate`
- `onPause`
- `onPause`
- `onDestroy`

**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
Contributions

backwards analysis “jumps”
Contributions

Framework for jumping analyses

```
1

l_1 \xrightarrow{c} l_2
```

backwards analysis "jumps"

Framework for jumping analyses
Contributions

1. \( l_1 \xrightarrow{c} l_2 \) Framework for jumping analyses

2. Applied to Android events

backwards analysis "jumps"
A program model for a jumping analysis

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

Program

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

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

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

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

Analysis parameters

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

Backward transfer functions
A program model for a jumping analysis

Program

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

\[ \ell_{\text{cur}} \]

\[ \ell_1 \rightarrow [c] \rightarrow \ell_2 \]
A program model for a jumping analysis

Program

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

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

\[ Q_{\text{cur}} \]
Current query
Data-relevance identifies writes

$\mathcal{Q}, \ell_{\text{cur}}$
Data-relevance identifies writes

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

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

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

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 **sparse** analysis (but, here, flow-insensitive)
Control-feasibility \textit{selectively} recovers flow-sensitivity

$$(Q, \ell_{\text{cur}})$$

Data-relevance

$$\ell_i \quad \ell_j \quad \ell_k \quad \ldots$$

$$_1 \ell_1 \quad [c] \Rightarrow \ell_2$$
Control-feasibility \textit{selectively} recovers flow-sensitivity
Control-feasibility selectively recovers flow-sensitivity

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

**Data-relevance**

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

**Control-feasibility**

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

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

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

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

Must visit another relevant location first.

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

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>

Current location: $\ell_{\text{cur}}$

Current query: $Q_{\text{cur}}$

Control-feasibility

Data-relevance

Next 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>

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

![Diagram](image)

- **Current location** \( l_{\text{cur}} \)
- **Current query** \( Q_{\text{cur}} \)
- **Data-relevance**
- **Control-feasibility**
- **Next locations**

**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

Current query

Data-relevance

Control-feasibility

Next locations

No filter

All locations

$l_1 \xrightarrow{\text{[c]}} l_2$
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>

- **Current location** $\ell_{\text{cur}}$
- **Current query** $Q_{\text{cur}}$

**Data-relevance**

**Control-feasibility**

**Next locations**

- **No filter**
- **All locations**

1. $\ell_1 \rightarrow [c] \rightarrow \ell_2$
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 query `Q_{cur}`

Current location `\ell_{cur}`

Control-feasibility

Data-relevance

Next locations

No filter

All locations
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(\(Q_{\text{cur}}\))
Jumping enables sparse, selective, on-the-fly control-flow abstraction

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

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

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

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

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

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
Evaluation 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>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?

### 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></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>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</td>
</tr>
<tr>
<td>npr</td>
<td>5</td>
<td>829</td>
<td>617</td>
<td>181</td>
</tr>
<tr>
<td>duckduckgo</td>
<td>11</td>
<td>1969</td>
<td>1341</td>
<td>518</td>
</tr>
<tr>
<td>lastfm</td>
<td>13</td>
<td>4840</td>
<td>3528</td>
<td>954</td>
</tr>
<tr>
<td>github</td>
<td>19</td>
<td>3603</td>
<td>2520</td>
<td>601</td>
</tr>
<tr>
<td>seriesguide</td>
<td>32</td>
<td>8184</td>
<td>5438</td>
<td>986</td>
</tr>
<tr>
<td>connectbot</td>
<td>33</td>
<td>2190</td>
<td>1562</td>
<td>316</td>
</tr>
<tr>
<td>textsecure</td>
<td>38</td>
<td>5921</td>
<td>3643</td>
<td>698</td>
</tr>
<tr>
<td>k-9</td>
<td>55</td>
<td>19032</td>
<td>11968</td>
<td>3104</td>
</tr>
<tr>
<td>wordpress</td>
<td>57</td>
<td>15066</td>
<td>9775</td>
<td>2431</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>
</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>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><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>

Thresher has 74% fewer unproven derefs 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></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
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

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

5 bugs due to **bad ordering assumptions**

10/11 patches accepted

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
Selective control-flow abstraction via a sound relevance relation
Summary

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

**On-the-fly**
- 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.
- **Coarsening**
  - Start with precise abstraction, become less precise.

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