Improving Efficiency of Dynamic Analysis with Dynamic Dependence Summaries

Vijay Krishna Palepu    Guoqing Xu    James A. Jones
University of California, Irvine, USA

The Hut and The Mountain
The Hut and The Mountain
The Hut and The Mountain
The Hut and The Mountain
Method Summaries
Method Summaries
Method Summaries
Example.

```java
void main() {
    IntList k = new IntList();
    int num = 1;
    k.add(num);
    ...
    ...
    ...
}
```
Example.

```java
void main() {
    IntList k = new IntList();
    int num = 1;
    k.add(num);
    ...
    ...
    ...
}

void add(int i) {
    int t = this.size;
    int[] a = this.arr;
    a[t] = i;
    t = t + 1;
    this.size = t;
}
```
Example.

```java
void main() {
    IntList k = new IntList();
    int num = 1;
    k.add(num);
    
    k.arr[k.size] = num;
    k.size = k.size + 1;
}
```
Background.

- Rountev, Sharp and Xu. **IDE Dataflow Analysis in the Presence of Large Object-Oriented Libraries.** CC, 2008.
Our Approach

Overview

Typical Dynamic Analysis

Instrument Program & Libraries.

Monitor Execution (Program & Libraries).

Extract Dataflow (Program & Libraries).

Client Analysis. e.g., Find Bug!

Palepu, Xu, Jones (UCI) Dynamic Dependence Summaries (9/24) ASE 2013
Our Approach

Overview

Approach Overview.

Instrument Program & Libraries.

Monitor Execution (Program & Libraries).

Extract Dataflow (Program & Libraries).

After Some Waiting

Client Analysis. e.g., Find Bug!

Palepu, Xu, Jones (UCI) Dynamic Dependence Summaries (10/24) ASE 2013
Our Approach
Overview

Approach Overview.

Instrument Program & Libraries.
Monitor Execution (Program & Libraries).
After Some Waiting
Extract Dataflow (Program & Libraries).
Client Analysis. e.g., Find Bug!

Library Method’s Dataflows.
Approach Overview.

- Instrument Program & Libraries.
- Monitor Execution (Program & Libraries).
- Extract Dataflow (Program & Libraries).
- Client Analysis. *e.g.*, Find Bug!

- Library Method’s Dataflows.
- Recognize Method Inputs/Outputs.
Our Approach

Approach Overview

Instrument Program & Libraries.

Monitor Execution (Program & Libraries).

Extract Dataflow (Program & Libraries).

Client Analysis. e.g., Find Bug!

Library Method’s Dataflows.

Recognize Method Inputs/Outputs.

Summarize Dataflows.

Palepu, Xu, Jones (UCI) Dynamic Dependence Summaries (10/24)
Our Approach

Approach Overview

Instrument Program & Libraries.

Monitor Execution (Program & Libraries).

After Some Waiting

Extract Dataflow (Program & Libraries).

Client Analysis. e.g., Find Bug!

Library Method’s Dataflows.

Recognize Method Inputs/Outputs.

Summarize Dataflows.

Abstract & Store.
Our Approach

Approach Overview.

Instrument Program.

Monitor Execution (Program).

Reuse Stored Library Method Summaries.

Extract Dataflow (Program & Libraries).

Client Analysis. e.g., Find Bug!
Dynamic Dependence Summaries.

- Summarize
- Abstract
- Reuse
Summarize.

```plaintext
obj.method(param1, param2)
```

- **inputs**
  - param1
  - param2

- **outputs**
  - obj
  - return
obj.method(param1, param2)

inputs

param1

param2

outputs

obj

field1

field2

field3

field4

return
Our Approach  Analysis & Implementation

Summarize.

```
obj.method(param1, param2)
```

inputs

```
field1
field2
field3
field4
```

outputs

```
obj
field1
field2
```

return

Palepu, Xu, Jones (UCI) Dynamic Dependence Summaries (13/24)
Our Approach

Analysis & Implementation

Summarize.

```
obj.method(param1, param2)
```

inputs

Global.constant

param1

param2

outputs

obj

field1

field2

return

obj

field1

field2

field3

field4
Our Approach

Analysis & Implementation

Summarize.

```
obj.method(param1, param2)
```

inputs

- Global.constant
- param1
- param2

outputs

- Global.field
- obj
  - field1
  - field2
- return

- obj
  - field1
  - field2
  - field3
  - field4
Summarize.

```
obj.method(param1, param2)
```

Inputs:
- `Global.constant`
- `param1`
- `param2`

Outputs:
- `Global.field`
- `obj.field1`
- `obj.field2`
- `obj.field3`
- `obj.field4`
- `return`
Summarize.

```python
obj.method(param1, param2)
```
obj.method(param1, param2)

- obj -> p0
- param1 -> p1
- param2 -> p2

p0.method(p1, p2)
Our Approach

Analysis & Implementation

Reuse.

\[ p_0.\text{method}(p_1, p_2) \]
\[ p_0 \rightarrow \text{obj2} \]
\[ p_1 \rightarrow \text{param3} \]
\[ p_2 \rightarrow \text{param4} \]

\[ \text{obj2.method(param3, param4)} \]

Diagram: 
- \text{Global.constant} 
- \text{param3} 
- \text{param4} 
- \text{obj2.field2.field4} 
- \text{obj2.field2.field3} 
- \text{Global.field} 
- \text{obj2.field1} 
- \text{obj2.field2} 
- \text{return}
Key Technical Challenges Addressed.

- Summary Abstraction and Reuse.
- Precise modeling of Array element accesses.
- Accounting for Varying Method Behavior due to polymorphism.
- Handling object-graph mismatch.
- Object sensitivity.
Implementation.

Key Parts
Implementation.

Key Parts

- Includes:
  - Java Bytecode Instrumenter
  - uses: Java; ASM (asm.org)
Implementation.

Key Parts

- Includes:
  - Java Bytecode Instrumenter
    - uses: Java; ASM (asm.org)
  - Trace Analyzer
    - (Dataflow or Dependencies)
    - uses: Java
Our Approach

Implementation.

Key Parts

- Includes:
  
  Java Bytecode Instrumenter
  uses: Java; ASM (asm.org)
  
  Trace Analyzer (Dataflow or Dependencies)
  uses: Java
  
  Dependence Summarizer
  uses: Java

Palepu, Xu, Jones (UCI) Dynamic Dependence Summaries (16/24)
Implementation.

Key Parts

- **Includes:**
  - Java Bytecode Instrumenter
    - uses: Java; ASM (asm.org)
  - Trace Analyzer (Dataflow or Dependencies)
    - uses: Java
  - Dependence Summarizer
    - uses: Java

- **Built using:** Java; ASM (asm.org)
RQ1

How does the reuse of dynamic dependence summaries affect the costs of dynamic analysis?
Experiment One: Performance.

RQ1

How does the reuse of dynamic dependence summaries affect the costs of dynamic analysis?

Metrics

Execution Trace Size.
Execution Running Time.
Experiment One: Performance.

RQ1

How does the reuse of dynamic dependence summaries affect the costs of dynamic analysis?

Metrics

Execution Trace Size.
Execution Running Time.

Treatments

Exhaustive.
Summary-based.
Experiment One: Performance.

RQ1

How does the reuse of dynamic dependence summaries affect the costs of dynamic analysis?

Metrics

Execution Trace Size.
Execution Running Time.

Treatments

Exhaustive.
Summary-based.

Client Subjects

ANTLR (35KLOCs)
BLOAT (41KLOCs)
FOP (102KLOCs)

JYTHON (245KLOCs)
PMD (60KLOCs)
Experiment One: Performance.

RQ1

How does the reuse of dynamic dependence summaries affect the costs of dynamic analysis?

Metrics

Execution Trace Size.
Execution Running Time.

Treatments

Exhaustive.
Summary-based.

Client Subjects

ANTLR (35KLOCs)
BLOAT (41KLOCs)
FOP (102KLOCs)
Jython (245KLOCs)
PMD (60KLOCs)

Library Subject

JAVA DEVELOPMENT KIT (rt.jar)
Results: Runtime (RQ1).

- 1.5× — 3.6× speedup in execution runtimes.
  - Exhaustive: 112× runtime overhead
  - Summary Times: 43× runtime overhead
Results: Trace Size (RQ1).

- 44% smaller traces with summary usage.

Palepu, Xu, Jones (UCI) Dynamic Dependence Summaries (19/24) ASE 2013
Experiment Two: Accuracy.

RQ2

How does the reuse of dynamic dependence summaries affect the accuracy of dynamic analysis?
Experiment Two: Accuracy.

RQ2

How does the reuse of dynamic dependence summaries affect the accuracy of dynamic analysis?

Metrics

Found Bugs.  
Runtime Overhead.
RQ2

How does the reuse of dynamic dependence summaries affect the accuracy of dynamic analysis?

Metrics

Found Bugs. Runtime Overhead.

Treatment

Exhaustive. Summary-based.
RQ2

How does the reuse of dynamic dependence summaries affect the accuracy of dynamic analysis?

Metrics

Found Bugs.  
Runtime Overhead.

Treatment

Exhaustive.  
Summary-based.

Client Subject

NanoXML (7KLOC)
Experiment Two: Accuracy.

RQ2
How does the reuse of dynamic dependence summaries affect the accuracy of dynamic analysis?

Metrics
- Found Bugs
- Runtime Overhead

Treatment
- Exhaustive
- Summary-based

Client Subject
- NanoXML (7KLOC)

Library Subject
- Java Development Kit (rt.jar)
Results: Found Bugs (RQ2).

NanoXML: Exhaustive vs. Summary

Palepu, Xu, Jones (UCI) Dynamic Dependence Summaries (21/24) ASE 2013
Results: Runtime Overhead (RQ2).

NanoXML: Exhaustive vs. Summary

- Runtime Overhead (ratio)

<table>
<thead>
<tr>
<th></th>
<th>Summary</th>
<th>Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>3,838</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10,000</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

Palepu, Xu, Jones (UCI) Dynamic Dependence Summaries (22/24) ASE 2013
Open Issues.
Open Issues.

- Assess suitability for summarization; adequacy criteria.
Takeaways

Open Issues.

- Assess suitability for summarization; adequacy criteria.
- Comparison with static summaries.
Open Issues.

- Assess suitability for summarization; adequacy criteria.
- Comparison with static summaries.
- Accuracy analysis with multiple test subjects and client analyses.
Takeaways.
• Theory, Models, Implementation for construction and reuse of Dynamic Summaries.
Takeaways.

- **Theory, Models, Implementation** for construction and reuse of Dynamic Summaries.
- **2× performance gains** (best case: 3.6×) while analyzing large software benchmarks.
Takeaways.

- **Theory, Models, Implementation** for construction and reuse of **Dynamic Summaries**.

- **2× performance gains** (best case: 3.6×) while analyzing large software benchmarks.

- Empirical study indicates **cost savings with modest accuracy losses**.