Program Testing and Analysis: Path Profiling

Prof. Dr. Michael Pradel
Software Lab, TU Darmstadt
What does the following code print?

```javascript
function d(a, b) {
    var sum = 0;
    for (var i = 0; i < arguments.length; i++) {
        sum += arguments[i];
    }
    console.log(sum == (a + b + 67));
}

d(23, 45, 67);
```

true    false    Something else
What does the following code print?

```javascript
function d(a, b) {
    var sum = 0;
    for (var i = 0; i < arguments.length; i++) {
        sum += arguments[i];
    }
    console.log(sum == (a + b + 67));
}

d(23, 45, 67);
```

true false Something else

Includes all passed arguments, independent of declared parameters
Outline

1. Motivation and Challenges
2. Ball-Larus algorithm for DAGs
3. Generalization and Applications

Mostly based on this paper:
- *Efficient path profiling*, Ball and Larus, MICRO 1996

Other reading material:
- *Whole program paths*, Larus, PLDI 1999
- *HOLMES: Effective statistical debugging via efficient path profiling*, Chilimbi et al., ICSE 2009
Path Profiling

- **Goal:** Count **how often a path through a function is executed**

- **Interesting for various applications**
  - Profile-directed **compiler optimizations**
  - **Performance tuning:** Which paths are worth optimizing?
  - **Test coverage:** Which paths are not yet tested?
Challenges

■ Runtime **overhead**
  - Limit slowdown of program

■ **Accuracy**
  - Ideally: *Precise profiles* (no heuristics, no approximations)

■ **Infinitely many paths**
  - Cycles in control flow graph
<table>
<thead>
<tr>
<th>#</th>
<th>Path</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ACDF</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>ACD</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ABCDF</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ABCDE</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ABDF</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ABDF</td>
<td></td>
</tr>
</tbody>
</table>
Naive approach: Edge profiling

- Instrument each branching point
- Count how often each CFG edge is executed
- Estimate most frequent path: Always follow most frequent edge
Example: Edge profiling

frequency of execution

Quiz: What is the most frequent path?

ACDEF

Really? Two possible path profiles

<table>
<thead>
<tr>
<th>Path</th>
<th>Profile 1</th>
<th>Profile 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDF</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>ACDEF</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>ABCDF</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ABCDEF</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ABDF</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>ABDDEF</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>ABCD</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>ABCDE</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>ABDEF</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>ABCDEF</td>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>
Edge Profiling

Naive approach: Edge profiling

- Instrument each branching point
- Count how often each CFG edge is executed
- Estimate most frequent path: Always follow most frequent edge
Edge Profiling

Naive approach: **Edge profiling**

- Instrument each branching point
- Count how often each CFG edge is executed
- Estimate **most frequent path**: Always follow most frequent edge

Fails to uniquely identify most frequent path
Ball-Larus Algorithm

- Assign a number to each path
- Compute path number by incrementing a counter at branching points
- Properties of path encoding
  - Precise: A single unique encoding for each path
  - Minimal: Instruments subset of edges with minimal cost
Example: Path Encoding

```
A \quad r=0
  \quad B \quad r=2
    \quad C
  \quad D
    \quad E
        \quad r+1
            \quad r\text{ count}[r]++

instrumentation:
  - state \( r \)
  - array of counters \( \text{count} \)

<table>
<thead>
<tr>
<th>Path</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDF</td>
<td>0</td>
</tr>
<tr>
<td>ACDE</td>
<td>1</td>
</tr>
<tr>
<td>ABCDF</td>
<td>2</td>
</tr>
<tr>
<td>ABCDE</td>
<td>3</td>
</tr>
<tr>
<td>ABDF</td>
<td>4</td>
</tr>
<tr>
<td>ABDE</td>
<td>5</td>
</tr>
</tbody>
</table>
```
Algorithm for DAGs

Assumptions

- Control flow graph is a directed acyclic graph (DAG)
- $n$ paths (numbered $0$ to $n - 1$)
- Graph has unique entry and exit nodes
- Artificial back edge from exit to entry
Assumptions

unique node

A

B

C

D

E

F

graph is acyclic

artificial backedge

unique exit node
Algorithm: Overview

- **Step 1:** Assign integers to edges
  - Goal: Sum along a path yields unique number for path
  - Enough to achieve "precise" goal

- **Step 2:** Assign increment operations to edges
  - Goal: Minimize additions along edges
  - Instrument subset of all edges
  - Assumes to know/estimate how frequent edges are executed
Representing Paths with Sums

- Associate with each node a value:
  \[ \text{NumPaths}(n) = \text{number of paths from } n \text{ to exit} \]

- **Computing** \( \text{NumPaths} \)
  - Visit nodes in reverse topological order
  - If \( n \) is leaf node:
    \[ \text{NumPaths}(n) = 1 \]
  - Else:
    \[ \text{NumPaths}(n) = \text{sum of } \text{NumPaths} \text{ of destination of outgoing edges} \]
Example: Num Paths

Reverse topological order:
Successor of $n$ visited before $n$

<table>
<thead>
<tr>
<th>Node $n$</th>
<th>Num Paths($n$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>A</td>
<td>6</td>
</tr>
</tbody>
</table>

Values for edges
Representing Paths with Sums (2)

For each node in reverse topological order:

- **If** $n$ **is leaf node:**
  \[
  \text{NumPaths}(n) = 1
  \]

- **Else:**
  - $\text{NumPaths}(n) = 0$
  - For each edge $n \rightarrow m$:
    - $\text{Val}(n \rightarrow m) = \text{NumPaths}(n)$
    - $\text{NumPaths}(n) += \text{NumPaths}(m)$
Quiz: Values for Edges

```
<table>
<thead>
<tr>
<th>n</th>
<th>NumPath (n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

* Sum of all NumPaths? 12
* Encoding for ACDF? 3
```
Algorithm: Overview

- **Step 1:** Assign integers to edges
  - Goal: Sum along a path yields unique number for path
  - Enough to achieve "precise" goal

- **Step 2:** Assign increment operations to edges
  - Goal: Minimize additions along edges
  - Instrument subset of all edges
  - Assumes to know/estimate how frequent edges are executed
Spanning Tree

- Given: Graph $G$

- Spanning tree $T$:
  Undirected subgraph of $G$ that is a tree and that contains all nodes of $G$

- Chord edges: Edges in $G$ but not in $T$
Example: Spanning Tree

G:

A

B → C

D

E → F

Which of these is a spanning tree of G?

1. 

2. 

3. 

4. 

5. 

Solution:

2, 4, 5
Increments for Edges

Goal: Increment sum at subset of edges

- **Choose spanning tree with maximum edge cost**
  - Cost of individual edges is assumed to be known

- **Compute increments at the chords of the spanning tree**
  - Based on existing event counting algorithm
Example: Increments for Edges

- Edge cost
- Most expensive spanning tree

Chord edges
- Non-minimal increments

Minimal increments
- Path encoding
Instrumentation

### Basic idea
- Initialize sum at entry: \( r = 0 \)
- Increment at edges: \( r += \ldots \)
- At exit, increment counter for path:
  \[
  \text{count}[r]++
  \]

### Optimization
- Initialize with incremented value, if first chord on path: \( r = \ldots \)
- Increment sum and counter for path, if last chord on path: \( \text{count}[r+\ldots]++ \)
Regenerating the Path

Knowing the sum $r$, how to determine the path?

- Use edge values from step 1 ("non-minimal increments")
- Start at entry with $R = r$
- At branches, use edge with largest value $v$ that is smaller than $R$ and set $R \leftarrow v$
Example: Regenerating the Path

\[ r = 4 : ABD \rightarrow F \]

\[ r = 1 : ACDE \rightarrow F \]
Generalizing to Cyclic CGFs

- For each backedge \( n \rightarrow m \), add dummy edges
  - \( Entry \rightarrow m \)
  - \( n \rightarrow Exit \)

- Remove backedges and add DAG-based increments

- In addition, add instrumentation to each backedge
  - \( count[r]++; \ r=0 \)
Example: Generalizing

Diagram:
- A → B → C → D → E → F
- Backedge: E → D
- Dummy edges: increments without backedge

Table:

<table>
<thead>
<tr>
<th>Path</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF</td>
<td>0</td>
</tr>
<tr>
<td>ABCEF</td>
<td>1</td>
</tr>
<tr>
<td>ABECE</td>
<td>2</td>
</tr>
<tr>
<td>BCEF</td>
<td>5</td>
</tr>
<tr>
<td>BCE</td>
<td>6</td>
</tr>
<tr>
<td>BDE</td>
<td>8</td>
</tr>
</tbody>
</table>
Conclusions

- **Ball-Larus profiling:**
  Beautiful algorithm with various applications

- Path profiles also determine accurate edge and basic block profiles

- Can generalize to other graphs, e.g., to compute call graph profiles