Program Analysis – Lecture 11
Path Profiling

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Winter 2019/2020
What does the following code print?

```javascript
var arr = [0, 1, 2, 3, 4];
arr[6] = 6;
console.log(arr[5] + arr[6]);
```

11  undefined  undefined 6  Something else
Warm-up Quiz

What does the following code print?

```javascript
var arr = [0, 1, 2, 3, 4];
arr[6] = 6;
console.log(arr[5] + arr[6]);
```

11  undefined  undefined  6  NaN

11 undefined undefined6 Something else
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
Running example

<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>ACDEF</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ABCDF</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ABCDF</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ABD</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ABDE</td>
<td></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
Example: Edge profiling

```
A
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>150</td>
</tr>
</tbody>
</table>
B   D  C
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>250</td>
</tr>
</tbody>
</table>
E   F
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>160</td>
<td>110</td>
</tr>
</tbody>
</table>
```

Frequency of execution

Query: 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>ACDEF</td>
<td>90</td>
<td>110</td>
</tr>
<tr>
<td>ACDEEF</td>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td>ABCDF</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ABCDEF</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>ABDF</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>ABDEEF</td>
<td>0</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**
  - **B** (r=0)
  - **C** (r=2)
  - **D** (r=4)
  - **E** (r=5)

- **F** (count[r]++)

**Instrumentation**
- state / counter: r
- array of counts: counts

<table>
<thead>
<tr>
<th>Path</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDF</td>
<td>0</td>
</tr>
<tr>
<td>ACDEF</td>
<td>1</td>
</tr>
<tr>
<td>ABCDF</td>
<td>2</td>
</tr>
<tr>
<td>ABCDEF</td>
<td>3</td>
</tr>
<tr>
<td>ABDF</td>
<td>4</td>
</tr>
<tr>
<td>ABDDEF</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 entry node
- Graph is acyclic
- Artificial back edge
- 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:
    \[ \text{NumPaths}(n) = \text{sum of } \text{NumPaths} \text{ of destination of outgoing edges} \]
Representing Paths with Sums (2)

For each node in reverse topological order:

■ **If** \( n \) **is leaf node:**

\[
NumPaths(n) = 1
\]

■ **Else:**

□ \( NumPaths(n) = 0 \)

□ For each edge \( n \rightarrow m \):

- \( Val(n \rightarrow m) = NumPaths(n) \)
- \( NumPaths(n) += NumPaths(m) \)
Example: Num Paths

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

<table>
<thead>
<tr>
<th>Node $n$</th>
<th>Num Path($n$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>1</td>
</tr>
<tr>
<td>$E$</td>
<td>1</td>
</tr>
<tr>
<td>$D$</td>
<td>2</td>
</tr>
<tr>
<td>$C$</td>
<td>2</td>
</tr>
<tr>
<td>$A$</td>
<td>4</td>
</tr>
<tr>
<td>$B$</td>
<td>6</td>
</tr>
<tr>
<td>$F_1$</td>
<td></td>
</tr>
<tr>
<td>$F_2$</td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
</tr>
<tr>
<td>$C_4$</td>
<td></td>
</tr>
<tr>
<td>$E$</td>
<td></td>
</tr>
<tr>
<td>$B$</td>
<td></td>
</tr>
<tr>
<td>$A_6$</td>
<td></td>
</tr>
</tbody>
</table>
Quiz: Values for Edges

```
  A  2  
   0  
  B   C
   
  0  0
 D

  0  1
E  
   0

F```

<table>
<thead>
<tr>
<th>n</th>
<th>NumPaths (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 of 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: $r+=..$
  - At exit, increment counter for path:
    \[ \text{count}[r]++ \]

- **Optimization**
  - Initialize with incremented value, if first chord on path: $r=..$
  - Increment sum and counter for path, if last chord on path: \[ \text{count}[r+..]++ \]
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

\[ r = 4 : \quad ABDF \]
\[ r = 1 : \quad ACDEF \]
Generalizing to Cyclic CFGs

- 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
  - $count[r]++; \ r=0$
Generalizing to Cyclic CFGs (2)

- Leads to four kinds of paths
  - From entry to exit
  - From entry to backedge
  - From end of backedge to beginning of (possibly another) backedge
  - From end of backedge to exit

- Full path information can be constructed from these four kinds
Example: Generalizing

\[
\begin{array}{c}
A \\
\downarrow \\
B \\
\downarrow \\
C \\
\downarrow \\
D \\
\downarrow \\
E \\
\downarrow \\
F
\end{array}
\]

Path Encoding
\begin{align*}
A & \rightarrow F & 0 \\
ABC & \rightarrow E & 1 \\
ABC & \rightarrow E & 2 \\
A & \rightarrow B & 3 \\
A & \rightarrow B & 4 \\
ABC & \rightarrow E & 5 \\
B & \rightarrow C & 6 \\
B & \rightarrow C & 7 \\
B & \rightarrow C & 8 \\
\end{align*}

dummy edges
backedge
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