Explaining Program Failures via Postmortem Static Analysis

Roman Manevich
Tel Aviv University

Manu Sridharan
UC Berkeley

Stephen Adams, Manuvir Das, Zhe Yang
Center for Software Excellence
Microsoft Corporation
Motivation

- Programs are shipped with bugs
- **Crash reports** ease bug fixing
  - Automated, sent over network
  - Give type of failure and stack trace
- But, problems remain
  - No execution trace provided
  - Reconstructing trace is **time-consuming**
**An Example Crash**

foo(rec *x, rec *z)
{
    q = z->f;  // Does dereference of z matter?
    *p = u;   // What does p point to?
    if (b)
        y = z; // Lots to keep track of!
    else
        y = x->f; // Which branch? Both?
    *y = ...;
}

NULL pointer dereference
Tool Support Needed

- **Input:** crash report
  - Program point of failure
  - Type of failure, eg. NULL dereference

- **Output:** error traces
  - Paths to point of failure that cause error
Static slicing?

foo(rec *x, rec *z)
{
    q = z->f;
    *p = u;
    if (b)
        y = z;
    else
        y = x->f;
    \*y = ...;
}

x->f NULL at entry
more informative error-specific slice
infeasible static slice
**Postmortem Symbolic Evaluation**

- **Dataflow analysis** to find traces
  - Track value backwards from error
  - Maintain flow information on each path
  - Use error type to filter traces
- **Borrow techniques from ESP [DLS02]**
  - For *scalability, precision, soundness*
Expression from which value is copied
  • Specific to path

**Single witness** per point on path
  • Demand analysis
Computing The Witness

<table>
<thead>
<tr>
<th>Formula</th>
<th>Witness</th>
<th>Witness</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u \rightarrow f = \text{NULL}$</td>
<td>done</td>
<td>$&lt;z \rightarrow f&gt;$</td>
</tr>
<tr>
<td>$*p = u$</td>
<td>$&lt;u \rightarrow f&gt;$</td>
<td>$&lt;z \rightarrow f&gt;$</td>
</tr>
<tr>
<td>$y = z$</td>
<td>$&lt;z \rightarrow f&gt;$</td>
<td>$&lt;z \rightarrow f&gt;$</td>
</tr>
<tr>
<td>$x = y \rightarrow f$</td>
<td>$&lt;y \rightarrow f&gt;$</td>
<td>$&lt;y \rightarrow f&gt;$</td>
</tr>
<tr>
<td>$*x = \ldots$</td>
<td>$&lt;x&gt;$</td>
<td>$&lt;x&gt;$</td>
</tr>
<tr>
<td></td>
<td>$p == &amp;z$</td>
<td>$p != &amp;z$</td>
</tr>
</tbody>
</table>

- Substitution like weakest preconditions
- Query aliasing oracle for indirect updates
- Still polynomial time
  - Bound number of witnesses
  - Switch to abstract location when too long
Using The Error Type

- No double deref of NULL on path
  - \( x = \text{NULL}; *x = y; *x = z \) is infeasible
  - Just check if witness is dereferenced
- In general, handle \textit{typestate errors}
  - Automaton describes behavior
  - Crash at transition to error state
- Do double derefs generalize?
Automaton Reversal

File I/O

print(f,"hi");
close(f);

infeasible
Putting It All Together

- ESP-style dataflow analysis [DLS02]
  - Interprocedural, path-sensitive
  - Engine maintains / presents traces
  - GOLF serves as aliasing oracle [DLFR01]
- Stack trace used if available
  - Restricts traversal up call stack
- Detect simple tests for NULL
  - Eg. if (p)
  - If p is witness on true branch, infeasible
Evaluation: Does It Scale?

- Test SPEC95 derefs for NULL deref
  - 2,000 - 140,000 lines of code
  - 100 random derefs per benchmark
  - If no traces for a deref, proven safe

- No stack traces

- Configurations
  - **Normal**: full analysis
  - **NoDD**: no filtering using double derefs
• Most queries fast (usually more than 90%)
• The rest are quite slow (minutes)
  • No useful analysis result, so timeout (15 seconds)
Aliasing

• Imprecise analysis for heap pointers
  • False positives + increased analysis time
• Traces with aliasing inscrutable
  • No explanation for alias
  • Thus far, useless to developers
• Configuration “Unsound”
  • No checking for indirect updates
  • No abstraction for long witnesses
### SPEC Number of Error Reports

<table>
<thead>
<tr>
<th>Bench</th>
<th>Normal</th>
<th>Unsound</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>li</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>go</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>m88ksim</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>jpeg</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>perl</td>
<td>44</td>
<td>29</td>
</tr>
<tr>
<td>vortex</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>gcc</td>
<td>18</td>
<td>2</td>
</tr>
</tbody>
</table>

- Remaining false positives
  - Global flag
  - Use of abstract locs (eg. a[i])
Evaluation: Useful traces?

- PREfix: static bug finding tool [BPS00]
- Checked five real NULL deref errors
- Five successes with “Unsound”
  - Found error-causing traces only
  - Query times under a second
  - Stack traces helpful
  - Four succeeded with “Normal”
Related Work

- Slicing [Tip95]
- Postmortem analysis [LA02]
- Typestate analysis [SY86, SY93]
- Fault localization
- Remote program sampling [LAZJ03]
- Forward analyses (Metal, ESP, model checkers)
Conclusions

- New analysis for diagnosing errors
  - Value traced back from error
  - Witnesses give useful flow information
  - False traces pruned using error type
- Results are promising
- Extensions
  - Integration with Watson
  - Evaluating other typestate errors
  - Presentation of aliases to developer
The End