Path Projection for User-Centered Static Analysis Tools

Jeff Foster
University of Maryland, College Park

Joint work with Yit Phang Khoo, Mike Hicks, and Vibha Sazawal

Introduction

- Many recent successes in static analysis tools for defect detection / prevention
  - Lots of progress in the research community
  - Coverity, Fortify, GrammaTech, and others sell static analysis tools
  - Microsoft has had great success with static analysis

- Major research focus: Building tools that are...
  - “Precise enough”
  - And still scale to large programs

Motivation

- Static analysis tools are not perfect
  - Users must triage bug reports
    - Decide whether true or false positives, and how important
  - Users must remediate true bugs

- Conclusion: successful static analysis requires cooperation between the user and the tool

- How do we build more user-centered static analyses?

Path Projection

- A new interface to help users visualize code paths
  - E.g., call stacks, control flow paths, data flow paths
  - These are common in static analysis error reports

- Core principles
  - Remain true to original source code
  - Fit as much on one screen at a time as possible
Contributions

• Prototype implementation in WebKit

• Controlled user study
  ▪ Task: Triaging Locksmith error reports
  ▪ Compared to “standard viewer” (similar to IDEs)

• Experimental results
  ▪ Improved performance (completion time)
  ▪ Same accuracy
  ▪ Qualitatively better

Programming Against Races

• Locations \( r \)
• Locks \( l \)
• Correlation: \( r \circ l \)
  ▪ Location \( r \) accessed while lock \( l \) held

• Consistent correlation
  ▪ Any shared location only correlated with one lock
    - Then that lock guards that location
  ▪ Implies race freedom

Locksmith: Race Detection for C

```c
void foo(pthread_mutex_t *l, int *p) {
  pthread_mutex_lock(l);
  *p++;
  pthread_mutex_unlock(l);
}
```

```c
pthread_mutex_t L1, L2;
int x, y;
foo(&L1, &x);
foo(&L2, &y);
```

Sample Locksmith Error Report

```
Warning: Possible data race of g_conn_open (knot.c:<global>:61)
at:
1. <in knot.c>
   main():601   -> dereference
   locks: -
2. <in knot.c>
   main():558   -> pthread_create()
   thread_main_autospawn():458
   accept_loop():395   -> dereference
   locks: -
3. <in knot.c>
   main():577   -> pthread_create()
   thread_main():476
   accept_loop():395   -> dereference
   locks: -
```

Sample Locksmith Error Report

```
Warning: Possible data race of g_conn_open (knot.c:<global>:61)
at:
1. <in knot.c>
   main():601   -> dereference
   locks: -
2. <in knot.c>
   main():558   -> pthread_create()
   thread_main_autospawn():458
   accept_loop():395   -> dereference
   locks: -
3. <in knot.c>
   main():577   -> pthread_create()
   thread_main():476
   accept_loop():395   -> dereference
   locks: -
```

Sample Locksmith Error Report

```
Warning: Possible data race of g_conn_open (knot.c:<global>:61)
at:
1. <in knot.c>
   main():601   -> dereference
   locks: -
2. <in knot.c>
   main():558   -> pthread_create()
   thread_main_autospawn():458
   accept_loop():395   -> dereference
   locks: -
3. <in knot.c>
   main():577   -> pthread_create()
   thread_main():476
   accept_loop():395   -> dereference
   locks: -
```
Triaging a Locksmith Report

• Three things to check:
  1. Both accesses refer to same location
     - Locksmith’s alias analysis may be imprecise
  2. Locksmith has not missed a held lock
     - Could happen at join points in cfg
  3. Potentially-racing accesses can occur, simultaneously
     - Both stack traces given must be simultaneously realizable

• This work: Focus on task 3
  • Leave 1 and 2 as future work

Challenges with Standard Interface

• Requires lots of scrolling through code
  • Hyperlinks help a little
  • Still hard to keep track of the context along the path

• Need to compare multiple paths together
  • Are both simultaneously realizable?

• Need to visually switch between error report and program source code
  • Adds cognitive burden
Side-by-side paths

Checklist

Function call inlining

Code folding (non-consecutive line numbers)

Multiple simultaneous searches

Path Projection Demo
Informational Visualization Strategies

• Increase user’s memory and processing resources, and reduce the search for information
  ▪ Make important lines of code visible on screen
  ▪ Place related lines of code close together

• Use visual representation to enhance pattern matching
  ▪ Put function definition in colored boxes
  ▪ Format/color code to reveal program structure

• Encode information in a manipulable medium
  ▪ Allow users to search/highlight

User Study: Overview

• Standard Viewer (SV) vs. Path Projection (PP)
• Task: Triage a Locksmith error report
  ▪ Decide whether it is a false positive or not
• Measurements
  ▪ Completion time for the task
  ▪ Qualitative feedback from users
  ▪ Observations of user behavior

Locksmith Task Details

• All trials from Locksmith benchmarks
  ▪ E.g., web server, ftp client, etc.
  ▪ Roughly 1,500 lines each
  ▪ Unfamiliar to participants

• One warning per trial
  ▪ No need to manage warnings

Locksmith Task Details (cont’d)

• No potential aliasing issues
  ▪ All shared variables and locks are global
  ▪ (Just a property of these particular warnings)

• Semantics-preserving simplifications:
  ▪ Made local static variables global
  ▪ Changed `wait()`/`signal()` to `join()`
  ▪ Deleted `#if 0` or other conditional macros
  ▪ Converted some `goto/switch` statements to `if`
Within-subjects Design

• Two possible schedules for a participant
  ▪ Same problems in same order
    | Session 1 | Session 2 |
    | Group 1   | PP (1.1, 1.2, 1.3) | SV (2.1, 2.2, 2.3) |
    | Group 2   | SV (1.1, 1.2, 1.3) | PP (2.1, 2.2, 2.3) |
• Pros:
  ▪ Participants can directly compare both interfaces
  ▪ Fewer problems due to individual variances

• Cons:
  ▪ Order effect: may prefer first interface
  ▪ Learning effect: may become better at task over time

Experimental Procedure

• Pre-questionnaire (background/demographic)
• Each session
  ▪ Tutorials on data races, Locksmith, the interface
  ▪ One practice trial
  ▪ Three measured trials
    - First, complete task; measure time
    - Then, repeat same task and explain aloud
      - Helps users learn faster, and gives us some insights
      - We do not tell users whether their reasoning is correct

Triaging Procedure

• We observed a major issue in our pilot study
  ▪ Many participants could not develop an effective procedure for triaging reports
    - Even with pre-experiment tutorials
• Problem: they don’t know how Locksmith works
  ▪ Would double-check facts guaranteed true
    - E.g., if Locksmith says a lock is acquired, it definitely is
  ▪ Would miss checking facts that may be imprecise
• Wasted a lot of time, led to a high variance among users, unclear were using tool right

Checklist

• Developed a Locksmith-specific checklist of tasks that must be completed to perform triaging
  ▪ Encodes a key human component of using the tool
• Two concerns of such a checklist:
  ▪ What form do error reports take?
    - Locksmith reports include program paths that may race
    - Checklist: Enumerates pairs of paths to check for races
  ▪ What are the sources of imprecision?
    - Unrealizable paths (for this study)
    - Checklist: Are both paths simultaneously realizable?
Warm-up task
Pairs of racing paths (and self-racing paths)

Checklist Demo

Checklist Improvement

- Participants improved completion times by 41% using the checklist
  - Comparing pilot study to final study
  - But only anecdotal, because there were other changes in the interface etc

- Interesting direction for future study

Participants and Equipment

- 8 student participants
  - 3 undergraduates, 5 graduates
  - Prior experience in C, multithreading (not necc. C)
  - Self-rated experience: 3 to 4
    - Scale of 1: no experience to 5: very experienced
  - 2 participants had experience in Locksmith and Eraser

- Apparatus
  - 24" 1920-by-1200 LCD
  - Mac OS X 10.5.2
    - All shortcuts disabled except for cut/copy/paste/find/find-next
Mean Time for All Participants

- Both improvements statistically significant
- PP improvement large
- SV improvement small-med
- PP-SV: 55s
- SV-PP: 188s
- Still a learning effect

Accuracy and Detailed Times

- Completion times and accuracy for each trial

<table>
<thead>
<tr>
<th>Trial</th>
<th>Session 1</th>
<th>Session 2</th>
<th>User</th>
<th>Session 1</th>
<th>Session 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1</td>
<td>1.2</td>
<td>1.3</td>
<td>mean</td>
<td>2.1</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8:36</td>
<td>14:14</td>
<td>9:44</td>
<td>10:51</td>
<td>7:07</td>
</tr>
<tr>
<td>2</td>
<td>5:07</td>
<td>3:10</td>
<td>5:50</td>
<td>4:42</td>
<td>4:16</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- # tabs: 3, 2, 6, 6, 3, 3
- One incorrectly answered tab in the checklist

- Similar # of mistakes: 10 PP (10.9%), 9 SV (9.8%)

Mouse Hover Time in Error Report

- 20s on average for PP, compared to 1:34 for SV
  - Little need to use hyperlinks under PP

Overall Impression (Qualitative)

- Boxplot
  - Centerline = median
  - Box extent = quartiles
  - Whiskers = min/max
  - Dots = outliers
Overall Impression (Qualitative)

- No statistically significant differences in answers
  - Small sample? Limited exposure?
- All but one preferred PP

PP Feature Ratings (Qualitative)

- Surprisingly, liked code folding/function inlining
  - Code folding was “the best feature” or “my favorite feature”
- Checklist: “saved me from having to memorize rules”
- Two participants did not favor multi-query
  - But forgot multi-query had 4 default items
Threats to Validity

- Results may not generalize
  - Small population, students, not data race experts
  - Small set of programs
  - Learning effect still present
- Changes to programs to make task easier
  - Task in experiments is very focused
  - Understanding error reports generally requires wider range of activities
- SV interface is not production quality
  - Deliberate choice, to avoid giving any advantages

Summary

- Introduced Path Projection, a new interface
  - Side-by-side display of paths
  - Function call inlining
  - Code folding
  - In general, tries to follow InfoViz principles
- Experimental results suggest PP
  - Improves completion times
  - Is liked by users
- Lots more to do on this topic!