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
Software Today

• Great progress in building better software
  ▪ We’ve been working on this problem for 50 years!

• But, we still have a long way to go
  ▪ *The national annual costs of an inadequate infrastructure for software testing is estimated to range from $22.2 to $59.5 billion.* —NIST 2002 planning report
Tools for Software Quality

• Goal: Give programmers new tools to help them improve software reliability and security

• Approach: Static analysis
  - Analyze program source code
    - Reason about all possible program runs
    - Complements testing and code auditing

• Check limited but useful properties
  - Do the easy stuff automatically
  - Support the programmer in doing the hard stuff
Some Recent Projects at UMD

• Path Projection — User-centered software analysis tools
• DRuby — Adding static typing to Ruby
• CMod — Enforcing modularity in C
• Uno — Ownership and uniqueness inference for Java
• Pistachio — Checking network protocol implementations
• Locksmith — Data race detection in C
• Saffire — Type check multi-lingual programs
• CQual/JQual — Type qualifiers for C and Java

Overall theme: Taking informal assumptions and ideas of the programmer and enforcing them

http://www.cs.umd.edu/projects/PL
Some Recent Projects at UMD

- Path Projection — User-centered software analysis tools
- DRuby — Adding static typing to Ruby
- CMod — Enforcing modularity in C
- Uno — Ownership and uniqueness inference for Java
- Pistachio — Checking network protocol implementations
- Locksmith — Data race detection in C
- Saffire — Type check multi-lingual programs
- CQual/JQual — Type qualifiers for C and Java

Overall theme: Taking informal assumptions and ideas of the programmer and enforcing them

http://www.cs.umd.edu/projects/PL
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: \( r \@ l \)
  - 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
void foo(pthread_mutex_t *l, int *p) {
    pthread_mutex_lock(l);
    *p++;
    pthread_mutex_unlock(l);
}

... 

pthread_mutex_t L1, L2;
it int x, y;
foo1(&L1, &x);
foo2(&L2, &y);
void foo(pthread_mutex_t *l, int *p) {
    pthread_mutex_lock(l);
    *p++;
    pthread_mutex_unlock(l);
}

void foo1(pthread_mutex_t *l, int *p) {
    pthread_mutex_lock(l);
    *p++;
    pthread_mutex_unlock(l);
}

void foo2(pthread_mutex_t *l, int *p) {
    pthread_mutex_lock(l);
    *p++;
    pthread_mutex_unlock(l);
}
void foo(pthread_mutex_t *l, int *p) {
    pthread_mutex_lock(l);
    *p++;
    pthread_mutex_unlock(l);
}

...  
pthread_mutex_t L1, L2;
int x, y;
foo1(&L1, &x);
foo2(&L2, &y);
void foo(pthread_mutex_t *l, int *p) {
    pthread_mutex_lock(l);
    *p++;
    pthread_mutex_unlock(l);
}

... 
pthread_mutex_t L1, L2;
int x, y;
foo1(&L1, &x);
foo2(&L2, &y);
Locksmith: Race Detection for C

```c
void foo(pthread_mutex_t *l, int *p) {
    pthread_mutex_lock(l);
    *p++;
    pthread_mutex_unlock(l);
}
...
pthread_mutex_t L1, L2;
int x, y;
foo1(&L1, &x);
foo2(&L2, &y);
```
void foo(pthread_mutex_t *l, int *p) {
    pthread_mutex_lock(l);
    *p++;
    pthread_mutex_unlock(l);
}

... p

pthread_mutex_t L1, L2;
int x, y;
foo1(&L1, &x);
foo2(&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: -
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: -

Three possibly-racing derefs (paths)

Shared variable

No locks held at deref
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: -

Shared variable
No locks held at deref
Thread creation site

Three possibly-racing derefs (paths)
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
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
```c
start = current_usecs();
sleep(g_timer_interval);

while (1) {
    long long bytes_sent;
    int conn_open, conn_succeed, conn_fail, conn_active, cache_hits,
        cache_misses = g_cache_misses;

    now = current_usecs();

    bytes_sent = g_bytes_sent;  g_bytes_sent = 0;
    conn_open = g_conn_open;    g_conn_open = 0;
    conn_succeed = g_conn_succeed; g_conn_succeed = 0;
    conn_fail = g_conn_fail;    g_conn_fail = 0;
    conn_active = g_conn_active; g_conn_active = 0;
    cache_hits = g_cache_hits;  g_cache_hits = 0;
    cache_misses = g_cache_misses; g_cache_misses = 0;

    //printf("rate: %.3g Mbits/sec %.0f open/sec %.0f suc/
    printf("rate: %.3g Mbits/sec %.0f open/sec %.0f suc/
        %d);  (double)bytes_sent * 8 * 1000000) / (double)(now-start
    ((double)conn_open * 1000000) / (double)(now-start
    ((double)conn_fail * 1000000) / (double)(now-start
    ((double)conn_succeed * 1000000) / (double)(now-start
    (double)conn_active, cache_misses,
    100*(double)cache_hits)/(double)(cache_hits+cache
    );
    printf("\n");
    start = now;

    sleep(g_timer_interval);
}

pthread_exit(0);
return 0;
//close(s);
//exit(0);

// Set the emacs indentation offset
// Local Variables: ***
```
Splittable file viewer

```c
start = current_usecs();
sleep(g_timer_interval);

while (1) {
    long long bytes_sent;
    int conn_open, conn_succeed, conn_fail, conn_active, cache_hits, cache_misses;
    now = current_usecs();
    
    bytes_sent = g_bytes_sent;  g_bytes_sent = 0;
    conn_open = g_conn_open;    g_conn_open = 0;
    conn_succeed = g_conn_succeed; g_conn_succeed = 0;
    conn_fail = g_conn_fail;     g_conn_fail = 0;
    conn_active = g_conn_active; g_conn_active = 0;
    cache_hits = g_cache_hits;  g_cache_hits = 0;
    cache_misses = g_cache_misses; g_cache_misses = 0;

    //printf("rate: %.3g Mbits/sec  %.0f open/sec  %.0f succ/s
    printf("rate: %.3g Mbits/sec  %.0f open/sec  %.0f succ/s
        ((double)bytes_sent * 8 * 10000000) / ((double)(now-start)
        ((double)conn_open * 10000000) / (double)(now-start)
        ((double)conn_succeed * 10000000) / (double)(now-start)
        ((double)conn_active * 10000000) / (double)(now-start)
        100*(double)cache_hits)/(double)(cache_hits+cache
    
    printf("\n");
    start = now;
    sleep(g_timer_interval);
}

pthread_exit(0);
return 0;
//close(s);
//exit(0);

// Set the emacs indentation offset
// Local Variables: ***
```

Path 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_autospawn():476
   accept_loop():395    -> dereference
   locks: -

Locksmith Checklist

Please verify that the locks reported at the end of each path in the path report, if any, are correctly acquired in the source code (list the function:line, or none):

Path 1: 
Path 2: 
Path 3: 
Hyperlinked error report

```
589     start = current_usecs();
590     sleep(g_timer_interval);
591
592     while (1) {
593         long long bytes_sent;
594         int conn_open, conn_succeed, conn_fail, conn_active, cache
595         now = current_usecs();
596
597         bytes_sent = g_bytes_sent;  g_bytes_sent = 0;
598         conn_open = g_conn_open;    g_conn_open = 0;
599         conn_succeed = g_conn_succeed; g_conn_succeed = 0;
600         conn_fail = g_conn_fail;   g_conn_fail = 0;
601         conn_active = g_conn_active; g_conn_active = 0;
602         cache_hits = g_cache_hits; g_cache_hits = 0;
603         cache_misses = g_cache_misses;  g_cache_misses = 0;
604
605         // printf("rate: %.3g Mbits/sec  %.0f open/sec  %.0f succeed/succ/\n606         // (double)bytes_sent * 8 * 10000000) / (double)(now-start)\n607         // (double)conn_open * 10000000) / (double)(now-start)\n608         // (double)conn_succeed * 10000000) / (double)(now-start)\n609         // (double)conn_fail * 10000000) / (double)(now-start)\n610         // (double)conn_active * 10000000) / (double)(now-start)\n611         // 100*(double)cache_hits)/(double)(cache_hits+cache
612         // misses);\n613         printf("\n");
614
615         start = now;
616
617         pthread_exit(0);
618         return 0;
619         } // close(s);
620     } // exit(0);
621
622     // Set the emacs indentation offset
623     // Local Variables: ***
624
```

Path 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_autospawn():476
   accept_loop():395  --> dereference
   locks: -

Locksmith Checklist

Please verify that the locks reported at the end of each path in the path report, if any, are correctly acquired in the source code (list the function:line, or none):

Path 1: ????
Path 2: ????
Path 3: ????
Basic searching
Standard Interface Demo
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
Error report

Checklist
Side-by-side paths
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

<table>
<thead>
<tr>
<th></th>
<th>Session 1</th>
<th>Session 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>PP (1.1, 1.2, 1.3)</td>
<td>SV (2.1, 2.2, 2.3)</td>
</tr>
<tr>
<td>Group 2</td>
<td>SV (1.1, 1.2, 1.3)</td>
<td>PP (2.1, 2.2, 2.3)</td>
</tr>
</tbody>
</table>

• 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?
3. `<in knot.c>`
   - `main()`: 577
   - `thread_main()`: 476
   - `accept_loop()`: 395
     - `-> pthread_create()`
     - `-> dereference`

### Locksmith Checklist

<table>
<thead>
<tr>
<th>Locks</th>
<th>Path 1</th>
<th>Path 2</th>
<th>Path 3</th>
<th>1-2</th>
<th>1-3</th>
<th>Submit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please verify that the locks reported at the end of each path in the path report, if any, are correctly acquired in the source code (list the `function:line`, or none):

- **Path 1**: ?????
- **Path 2**: ?????
- **Path 3**: ?????
3. `<in knot.c>`
   main(): 577
   thread_main(): 476
   accept_loop(): 395
   `-> pthread_create()`
   `-> dereference`

Pairs of racing paths (and self-racing paths)

---

### Locksmith Checklist

<table>
<thead>
<tr>
<th>Locks</th>
<th>Path 1</th>
<th>Path 2</th>
<th>Path 3</th>
<th>1-2</th>
<th>1-3</th>
<th>Submit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please verify that the locks reported at the end of each path in the path report, if any, are correctly acquired in the source code (list the `function:line`, or none):

- Path 1: ?????
- Path 2: ?????
- Path 3: ?????
Warm-up task

Pairs of racing paths (and self-racing paths)
**Warm-up task**

Pairs of racing paths (and self-racing paths)

---

**Locksmith Checklist**

<table>
<thead>
<tr>
<th>Locks</th>
<th>Path 1</th>
<th>Path 2</th>
<th>Path 3</th>
<th>1-2</th>
<th>1-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please verify that the locks reported at the end of each path in the report, if any, are correctly acquired in the source code (list the function:line or none):

- Path 1: ????
- Path 2: ????
- Path 3: ????

Submit completes checklist (and ends trial)
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

Completion time (sec)

- Standard Viewer
- Path Projection

Session 1

Session 2
Mean Time for All Participants

Completion time (sec)

- Standard Viewer
- Path Projection

Session 1
- PP-SV: 55s
- SV-PP: 188s

Session 2

Both improvements statistically significant

PP improvement large
SV improvement small-med

Still a learning effect
Accuracy and Detailed Times

<table>
<thead>
<tr>
<th>Completion times and accuracy for each trial</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trial</strong></td>
</tr>
<tr>
<td><strong>User</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</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%)
PP improvement was large compared to the standard mean difference, which was statistically significant. We applied Cohen’s d, and we found that both of these improvements were statistically significant when analyzed separately. For each presentation order, the mean time improved by 1 second. We ran two one-way ANOVAs, and the difference in time was statistically significant when we compared PP to SV. This showed that the difference in time was statistically significant.

We also analyzed the number of mistakes participants made in each session. For each trial, we compared the number of mistakes participants made under PP to SV. We found that participants made significantly fewer mistakes under PP compared to SV. For each participant, we summed the number of mistakes they made in each session. For each trial, we compared the number of mistakes participants made under PP to SV. We found that participants made significantly fewer mistakes under PP compared to SV.

We also analyzed the duration of the mouse cursor idle time over the checklist. For each trial, we compared the duration of the mouse cursor idle time under PP to SV. We found that participants spent much less time with the mouse hovering over the checklist compared to SV. The chart in the bottom right of Figure 1 also shows the mean times for each session and presentation order. We found that on average, participants spend 61 seconds in the error report under PP compared to 63 seconds in the error report under SV. The chart in the bottom right of Figure 1 shows the mean times for each session and presentation order. We found that on average, participants spend 61 seconds in the error report under PP compared to 63 seconds in the error report under SV.

In addition to quantitative data, we also examined participants’ answers to the questionnaires we administered. The chart at the top of Figure 1 also indicates the distribution of errors. We see that participants felt PP took longer to use hyperlinks in the error report are heavily used to bring up relevant parts of the code. This indicated a statistically significant result. We also found that participants reported that they were unable to fairly assess PP due to the novelty of the interface and the limited amount of exposure to it in the experiment. One participant rated PP worse than SV in terms of user performance, and they preferred PP due to its potential.

Overall impressions.

• 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

**Usefulness of features.**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Standard Viewer</th>
<th>Path Projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick to learn</td>
<td>Strongly Agree</td>
<td>Strongly Agree</td>
</tr>
<tr>
<td>Confident of answer</td>
<td>Strongly Agree</td>
<td>Agree</td>
</tr>
<tr>
<td>Easy to verify race</td>
<td>Agree</td>
<td>Strongly Agree</td>
</tr>
<tr>
<td>Prefer Path Projection</td>
<td>Strongly Agree</td>
<td>Strongly Agree</td>
</tr>
</tbody>
</table>

Participants generally rated all the features of PP as somewhat or very useful. The boxplot in Figure 6 shows a box plot of participants' responses across our trials. The core threat is whether our results generalize to other forms of IDEs used in practice. We had a small number of participants, all of whom were students rather than expert programmers who had significant experience debugging data races. The set of profs we produced statistically significant and anecdotally useful information for our population and sample trials. We leave for future work a wider range of experiments and participants. The small size of our experiment did not obviate the need for this. Also, the SV interface represented a typical IDE, but is not an actual production-quality IDE. We chose this approach on purpose, so that familiarity with a particular IDE would not be a factor but we may have omitted features that would be useful for our task. For example, carrying out a wider range of experiments would obviate the need for this. Several tools provide features not present in Path Projection, though the implementation or an erroneous execution trace. We surveyed these tools using program paths to illustrate a suspect data flow. A number of potential threats to the validity of our result. The core threat is whether our results generalize to other forms of IDEs used in practice.
Overall Impression (Qualitative)

- No statistically significant differences in answers
  - Small sample? Limited exposure?
- All but one preferred PP
PP Feature Ratings (Qualitative)

- All statistically significant vs. neutral response
- Generally favorable towards PP features
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!

Fin
### Checklist for Single Path

**Path \( i \)**

<table>
<thead>
<tr>
<th>Is the thread created in a loop (loop count &gt; 1)?</th>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>If yes, there is likely a race. Are there reasons to show otherwise?</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

**Explain:**

---

- **Deference with no locks held**
  - May race with itself if called from multiple threads
    - `while(1) { fork { *p++ } }`

---

---
Checklist for Pairs of Paths

For threads leading to dereferences in Paths $i$ and $j$:

Are they parent-child (or child-parent), or child-child?
- ○ Parent-child / ○ Child-child

<table>
<thead>
<tr>
<th>Parent-child (or child-parent) threads.</th>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does the parent’s dereference occur after the child is spawned?</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Before its dereference, does the parent wait (via <code>pthread_join()</code>) for the child?</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>If no, there is likely a race. Are there reasons to show otherwise?</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Explain:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Child-child threads.</th>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Are the children mutually exclusive (i.e., only one can be spawned by their common parent/ancestor)?</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>If no, there is likely a race. Are there reasons to show otherwise?</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Explain:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>