How Good is Static Analysis at Finding Concurrency Bugs?

Devin Kester, **Martin Mwebesa**, Jeremy S. Bradbury

Software Quality Research Group
University of Ontario Institute of Technology
Oshawa, Ontario, Canada

devin.kester@mycampus.uoit.ca, {martin.mwebesa, jeremy.bradbury}@uoit.ca

Funding provided by:

Motivation

- Bug detection in concurrent software is very challenging due to the many thread interleavings.
- Concurrency bug detection techniques often involve dynamic and/or static analysis
  - Dynamic analysis is rather costly because it needs to cover all thread interleavings
  - Static analysis offers a less costly alternative but is susceptible to spurious results

The Goals

1. How **effective** are existing static analysis tools at detecting concurrency bugs?

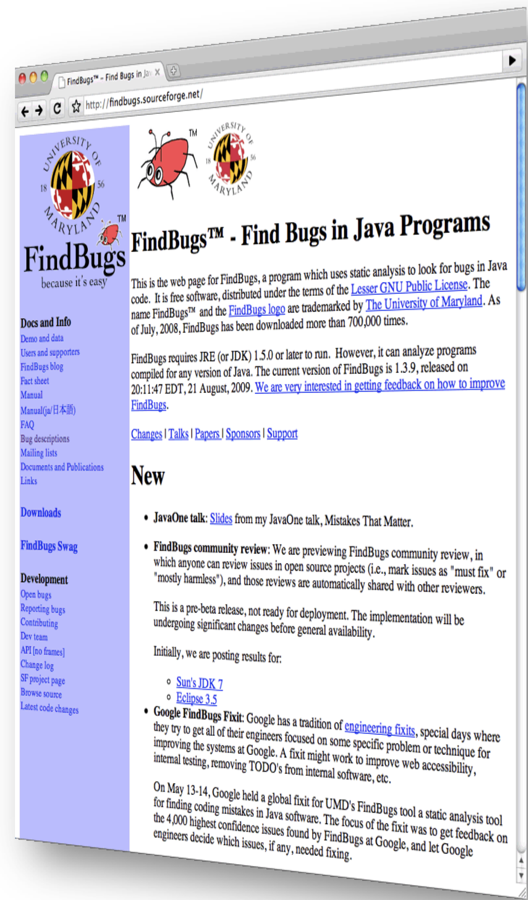
2. What is the **rate of false positives** (spurious results) in existing static analysis tools?

The Static Analysis Tools

- The tools selected for the experiment were:
  - FindBugs
  - JLint
  - Chord
- **Why these 3 tools?** The tools were selected because they vary in the kinds of static analysis they perform.

FindBugs [HP04]

- A general purpose static analysis tool that finds instances of different bug patterns in Java bytecode
  - We have focused on the multi-threaded bug patterns only
- Types of static analysis used:
  - Pattern matching
  - Data flow analysis



[HP04] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *ACM SIGPLAN Notices*, vol. 39, no. 12, pp. 92–106, 2004.

JLint [Art01]

- Similar to FindBugs, JLint is a general purpose static analysis tool that inspects Java bytecode
  - It includes concurrency bug pattern detection – specifically deadlocks, race conditions and improper use of wait-notify synchronization constructs
- Types of static analysis used:
  - Data flow analysis
  - Analysis of lock dependency graphs

[Art01] C. Artho, "Finding faults in multi-threaded programs," Master's thesis, Institute of Computer Systems, Federal Institute of Technology, Zurich/Austin, 2001.

Chord [NA07]

- A newer tool. Special purpose tool built to detect concurrency bugs – both statically and dynamically
  - For the purposes of this experiment we use only the static analysis features
- Types of static analysis used:
  - Call-graph (multi-graph) analysis
  - Alias analysis
  - Thread-escape analysis
  - Lock analysis

[NA07] M. Naik and A. Aiken, "Conditional must not aliasing for static race detection," *ACM SIGPLAN Notices*, vol. 42, no. 1, 2007.

Experimental Setup I

- We used 12 example programs in our experiment
  - 6 programs provided by developers of Java Pathfinder – NASA
  - 6 programs provided by the developers of ConTest – researchers at IBM's Haifa Lab
- The programs contained examples of deadlock bugs, data race bugs and weak reality synchronization bugs (caused by improper synchronization)

Experimental Setup II

- **Why these 12 programs?**
  - Publicly available sources – allow for reproducing results
  - Developed by third party (not used by the developers of the 3 static analysis tools under experiment)
  - Each program has a single documented concurrency fault
  - Each program is small enough to do a manual assessment of the experimental results

Experimental Procedure

1. Dynamic analysis preprocessing
   - Confirmed that the concurrency bugs in the 12 example programs could be reproduced in JPF and ConTest
2. Analysis with FindBugs, JLint and Chord
   - Analyzed each of the 12 example programs using each of the 3 static analysis tools – default settings were used
3. Assessment of the static analysis output
   - Each warning produced in Step 2 is examined and the cause of the warning is attributed to a known bug or the warning is identified as a false positive - done manually

Results – Effectiveness

Static Analysis Tool	FindBugs	JLint	Chord
Programs Analyzed	12	11	12
Concurrency Bugs Present	13	12	13
Warnings Generated	39	31	8
Multi-threaded Warnings Generated	12	9	8
Warnings Exhibiting Real Bugs	6 (50.00%)	7 (77.78%)	8 (100.00%)
Known Bugs Successfully Found	4 (30.77%)	4 (33.33%)	4 (30.77%)
Known Bugs Not Found	9	8	9

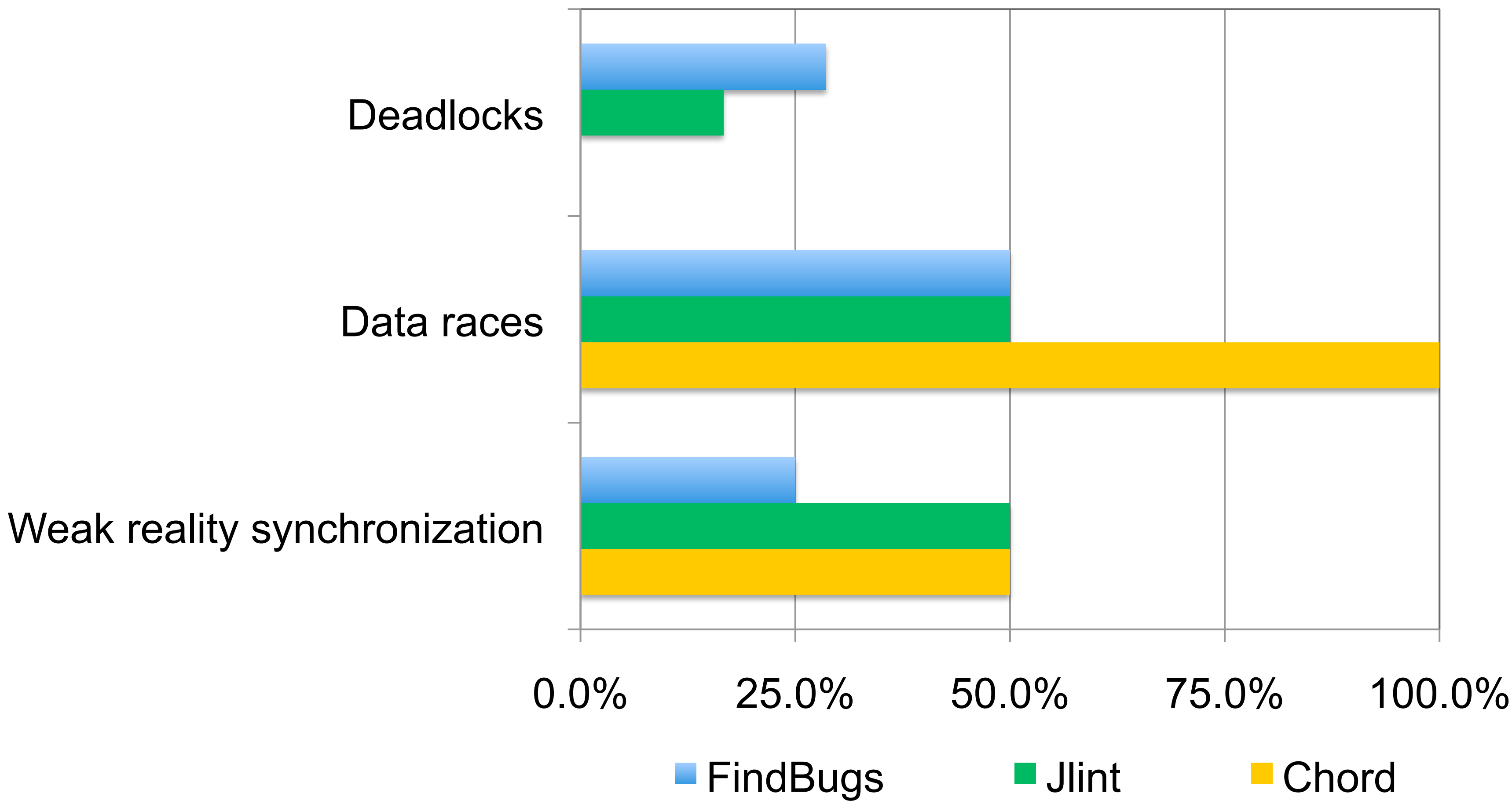
Results – Effectiveness

Static Analysis Tool	FindBugs	JLint	Chord
Programs Analyzed	12	11	12
Concurrency Bugs Present	13	12	13
Warnings Generated	39	31	8
Multi-threaded Warnings Generated	12	9	8
Warnings Exhibiting Real Bugs	6 (50.00%)	7 (77.78%)	8 (100.00%)
Known Bugs Successfully Found	4 (30.77%)	4 (33.33%)	4 (30.77%)
Known Bugs Not Found	9	8	9

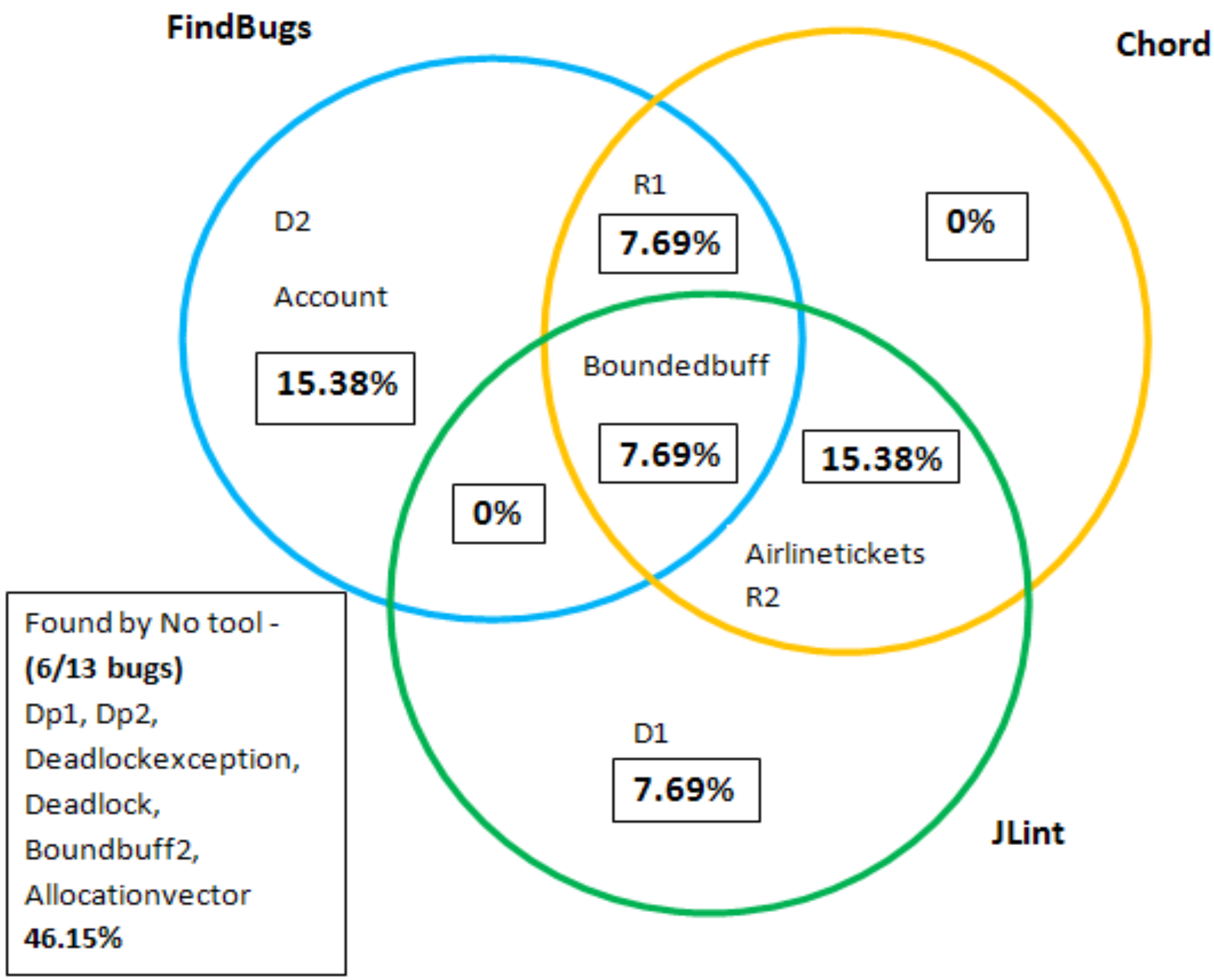
Results – Effectiveness

Static Analysis Tool	FindBugs	JLint	Chord
Programs Analyzed	12	11	12
Concurrency Bugs Present	13	12	13
Warnings Generated	39	31	8
Multi-threaded Warnings Generated	12	9	8
Warnings Exhibiting Real Bugs	6 (50.00%)	7 (77.78%)	8 (100.00%)
Known Bugs Successfully Found	4 (30.77%)	4 (33.33%)	4 (30.77%)
Known Bugs Not Found	9	8	9

Results – Percentage of Bugs Detected By Type



Results – Tools in Combination



Some Observations

- Spurious results
  - FindBugs and JLint – numerous
  - Chord - none
- All tools had issues with deadlock detection
- All tools performed better in detecting data races
  - FindBugs and JLint – 50 % effective
  - Chord – 100 % effective
- Efficiency
  - Chord took about 2 minutes, FindBugs between 7 and 14 seconds, JLint under a second

Threats to Validity

- We designed and ran our experiment with the goal of minimizing the impact of threats to validity
- Potential threats to the validity of our results:
  - Does not generalize to tools not included in our study
  - The 12 sample programs used may not be representative of concurrency programs in general (especially since all are small in size)

Conclusion

- Effectiveness of finding concurrency bugs was about the same for all tools
- All of the tools had trouble detecting deadlocks statically
- Chord had the least (zero) spurious results most likely due to the effective use of multiple forms of static analysis
- For consideration - Active testing:
  - Use of static analysis techniques to find potential bugs then dynamic analysis on the potential bugs, to isolate the real bugs (CalFuzzer [JNPS09])

[JNPS09] P. Joshi, M. Naik, C.-S. Park, and K. Sen, "CalFuzzer: an extensible active testing framework for concurrent programs," in Proc. of the 21st International Conference on Computer Aided Verification (CAV'09), 2009, pp. 675–681.

Future Work

Need more experiments!

- Need to include more static analysis tools (e.g., RacerX [EA03] and RELAY [VJL07])
  - RacerX detects both deadlocks and data race conditions
  - Relay detects data races and was developed with scalability as one of if it's main goals
- Need to increase the number of sample programs

[EA03] D. Engler and K. Ashcraft, "RacerX: effective, static detection of race conditions and deadlocks," in Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), 2003, pp. 237–252.

[VJL07] J. W. Voung, R. Jhala, and S. Lerner, "RELAY: static race detection on millions of lines of code," in Proc. of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC-FSE '07), 2007, pp. 205–214.

“Controversial” Question

- Can static analysis techniques be made as effective (or close to as effective) as dynamic analysis techniques in finding concurrency related bugs?
  - By effective I mean:
    - Finding the same number of concurrency related bugs
    - Reducing spurious results to a negligible level

How Good is Static Analysis at Finding Concurrency Bugs?

Devin Kester, **Martin Mwebesa**, Jeremy S. Bradbury

Software Quality Research Group
University of Ontario Institute of Technology
Oshawa, Ontario, Canada

devin.kester@mycampus.uoit.ca, {martin.mwebesa, jeremy.bradbury}@uoit.ca

Funding provided by: