Seminar on Program Analysis

ECS 289C (Programming Languages and Compilers)

WINTER 2015

Preliminary Reading List

1. Abstract Interpretation and Dataflow Analysis
   (a) Global Data Flow Analysis and Iterative Algorithms [J.ACM’76]
   (b) Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints [POPL’77]
   (c) Precise Interprocedural Dataflow Analysis via Graph Reachability [POPL’95]

2. Pointer Analysis
   (a) Points-to Analysis in Almost Linear Time [POPL’96]
   (b) Pointer Analysis: Haven’t We Solved This Problem Yet? [PASTE’01]
   (c) The Ant and the Grasshoper: Fast and Accurate Pointer Analysis for Millions of Lines of Code [PLDI’07]

3. Program Slicing
   (a) Interprocedural Slicing Using Dependence Graphs [PLDI’88]
   (b) Dynamic Program Slicing [PLDI’90]

4. Dynamic Symbolic Execution and Test Generation
   (a) DART: Directed Automated Random Testing [PLDI’05]
   (b) CUTE: A Concolic Unit Testing Engine for C [FSE’05]
   (c) KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex System Programs [OSDI’08]

5. Model Checking
   (a) Counterexample-Guided Abstraction Refinement [CAV’00]
   (b) Automatic predicate abstraction of C programs [PLDI’01]
   (c) Lazy Abstraction [POPL’02]

6. Concurrency
   (a) Learning from Mistakes: A Comprehensive Study on Real World Concurrency Bug Characteristics [ASPLOS’08]
   (b) Finding and Reproducing Heisenbugs in Concurrent Programs [OSDI’08]
(c) Life, Death, and the Critical Transition: Finding Liveness Bugs in Systems Code [NSDI’07]
(d) RacerX: Effective, Static Detection of Race Conditions and Deadlocks [SOSP’03]
(e) FastTrack: Efficient and Precise Dynamic Race Detection [PLDI’09]
(f) A Randomized Dynamic Program Analysis Technique for Detecting Real Deadlocks [PLDI09]

7. Automated Debugging / Fault Localization
   (a) Simplifying and Isolating Failure-Inducing Input [TSE’02]
   (b) Scalable Statistical Bug Isolation [PLDI’05]
   (c) Angelic Debugging [ICSE’11]

8. Performance Analysis
   (a) Understanding and Detecting Real-World Performance Bugs [PLDI’12]
   (b) Go with the Flow: Profiling Copies to Find Runtime Bloat [PLDI’09]

9. JavaScript and Web Apps
   (a) Jalangi: A Selective Record-Replay and Dynamic Analysis Framework for JavaScript [FSE’13]
   (b) Dynodroid: An Input Generation System for Android Apps [FSE’13]
   (c) A Framework for Automated Testing of JavaScript Web Applications [ICSE’11]

10. Floating-Point Programs
    (a) Automatic Detection of Floating-Point Exceptions [POPL’13]
    (b) Precimonious: Tuning Assistant for Floating-Point Precision [SC’13]
    (c) Efficient Search for Inputs Causing High Floating-Point Errors [PPoPP’14]
    (d) On-the-fly Detection of Instability Problems in Floating-Point Program Execution [OOPSLA’13]

11. Approximate Computing
    (a) Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware [OOPSLA’13]
    (b) Chisel: Reliability- and Accuracy-Aware Optimization of Approximate Computational Kernels [OOPSLA’14]

12. Error Handling
    (a) Error Propagation Analysis for File Systems [PLDI’09]
    (b) Amplifying Tests to Validate Exception Handling Code [ICSE’12]
(c) Finding and Preventing Run-Time Error Handling Mistakes [OOPSLA’04]
(d) Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems [OSDI’14]

13. Security
(a) Finding User/Kernel Pointer Bugs With Type Inference [Usenix’04]
(b) Automatic Creation of SQL Injection and Cross-Site Scripting Attacks [ICSE’09]
(c) TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones [OSDI’10]
(d) Path Sensitive Static Analysis of Web Applications for Remote Code Execution Vulnerability Detection [ICSE’13]

14. User Annotations and Specifications
(a) Extended Static Checking for Java [PLDI’02]
(b) Graph-Based Mining of Multiple Object Usage Patterns [FSE’09]

15. Automatic Program Repair
(a) Automatically Finding Patches Using Genetic Programming Automatically Finding Patches Using Genetic Programming [ICSE’09]
(b) Automated Atomicity-Violation Fixing [PLDI’11]

16. Compilers
(a) Formal Certification of a Compiler Back-End, or: Programming a Compiler with a Proof Assistant [POPL’06]
(b) Trace-based Just-in-Time Type Specialization for Dynamic Languages [PLDI’09]
(c) Finding and Understanding Bugs in C Compilers [PLDI’11]
(d) Compiler Validation via Equivalence Modulo Inputs [PLDI’14]

17. Miscellaneous
(a) What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems [SOCC’14]
(b) A System and Language for Building System-Specific, Static Analyses [PLDI’02]
(c) LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation [CGO’04]