List of Topics to be covered (8/29/2015)

(Note ** papers are classics; papers to be presented are indicated by a ## prefix and are listed first in each category; all IEEE and ACM conferences and journals available through VT library electronic subscription)

--- *Foundations of Dataflow Analysis* (lectures 1+2)

- CFGs, call graphs, classical dataflow equations, Lattice theory and its relation to dataflow analysis, worklist algorithm as a solution procedure; fixed point iteration and dataflow transfer functions to define a monotone framework,

References:

Matthew S. Hecht, *Flow Analysis of Computer Programs*, Ch 9, Monotone Dataflow Frameworks**
(book is out of print, but will be on reserve in the library on campus)


Or any standard compiler textbook, chapter on optimization technology, look at definitions of the classical dataflow problems: reaching definitions, live uses of variables, available expressions

--- *Advanced Foundations of Dataflow Analysis* (lecture 3)

dataflow solution safety and what it means; meet-over-all-paths solution and what it means for static analysis;

--- *Data dependence and control dependence analysis* (lecture 3)

data dependence and control dependence

Jeanne Ferrante, Karl Ottenstein, & Joe Warren, “The Program Dependence Graph and its Use in Optimization”, ACM TOPLAS vol 9, no 3, July 1987 ** (first sections of the paper have good definitions of data and control dependence. PDG was used to parallelize programs)


--- *Reference Analysis and Call Graph Construction for Statically Typed OOPLs* (e.g., Java, C++)

*Type-based techniques – Class Hierarchy Analysis (CHA), Rapid Type Analysis (RTA)*


--- *Flow-based techniques – flow-insensitive, context-insensitive approach to points-to analysis (i.e., reference analysis)*


*Other sources:*

Bjarne Steensgaard, “Points-to Analysis in Almost Linear Time”, POPL 1996, pp32-41. (C pointers)
---Adding calling context sensitivity (Java)


##Dave Grove & Craig Chambers, "Call graph Construction in OO Languages", OOPSLA 1997.;


## O. Lhotak & L. Hendren, "Context-sensitive Points-to Analysis: is it worth it?", CC 2006

Other sources:
Y. Smaragdakis, M. Bravenboer, & O. Lhotak, “Pick your contexts well: understanding object sensitivity”, POPL 2011
G. Kastrinis, Y. Smaragdakis, "Hybrid Context Sensitivity for Points-to Analysis", PLDI 2013

---Reference Analysis and Call Graph Construction for JavaScript (an example of a dynamically typed language, related to webpage information flow security problems) Dealing with dynamic code construction, dynamic object types, objects whose properties can vary at runtime (adds and deletes of properties are possible)

1st set of papers to present:
##V. Kashyap et al, “JSAI: A Static Analysis Platform for JavaScript”, FSE 2014

2nd set of papers to present:
Other sources:


V. Kashyap et al, “Type Refinement for Static Analysis of JavaScript”, DLS 2013

Hybrid analysis approaches – applying mixes of different sorts of context sensitivity during analysis


---Dynamic analysis – profiling and sampling

(selective instrumentation, profiling procedural acyclic paths)

##James R. Larus, “Whole Program Path”, PLDI 1999**

##Richards, G., Lebresne, S., Burg, B., & Vitek, J. An analysis of the dynamic behavior of JavaScript programs. PLDI, pp. 1-12, 2010 (an empirical study of the dynamic behavior of JavaScript programs using JS tracing, focusing on the dynamism of this language; tool used: instrumented version of WebKit, tracingSafari, traceAnalyzer)

Other sources:


Wei, S., Xhakaj, F., & Ryder, B. G. Empirical study of the dynamic behavior of JavaScript objects. Software: Practice and Experience, 2015. (more focused empirical study on the dynamism of JS with respect to the objects used in JS programs, using traces of JS executions; tools used: tracingSafari, traceAnalyzer)

Zhang, Xiangyu, and Rajiv Gupta. Whole execution traces and their applications. ACM Transactions on Architecture and Code Optimization (TACO). pp. 301-334, 2005. (whole-execution tracing that records fine-grained runtime artifacts including control flow, values, addresses, and data/control
dependence histories; the trace is represented by a labeled graph)

---Static and Dynamic Slicing techniques – related to dependences

**Static:**

##Mark Weiser, Program Slicing. TSE, 1984; Mark Weiser, “Programmers use Slices when Debugging”, CACM vol 25, no 7, pp 446-452, July 1982.** (the original static slicing, intraprocedurally precise but interprocedurally imprecise because it does not model calling contexts)


**Other sources:**


Gupta, R., Soffa, M. L., & Howard, J. Hybrid slicing: integrating dynamic information with static analysis. TOSEM,1997. (hybrid slicing, breakpoints + dynamic call graph + static slice)


**Dynamic:**

##Zhang, Xiangyu, Rajiv Gupta, and Youtao Zhang. Precise dynamic slicing algorithms. ICSE, 2003. ( best paper award, trace-based dynamic slicing with enhanced precision; three algorithms of same precision but different efficiency)

**Other sources:**

H. Agrawal & J. Horgan, “Dynamic Program Slicing”, PLDI 1990. (the original dynamic slicing, three algorithms of different degrees of precision and costs)

Gyimóthy, Tibor, Árpád Beszédes, and István Forgács. An efficient relevant slicing method for debugging. FSE, 1999. (relevant slicing, dynamic slice + statements that actually did not affect the variable but could have affected it had they been evaluated differently)

Zhang, Xiangyu, and Rajiv Gupta. Cost effective dynamic program slicing. PLDI, 2004. (improving efficiency of dynamic slicing via optimizing dynamic dependence graph on which dynamic slices are computed; this seems to be the fastest backward computation algorithm for backward dynamic slicing )


*Specialized applications of slicing:*
DeMillo, R. A., Pan, H., & Spafford, E. H. *Critical slicing for software fault localization*. ISSTA, 1996. (critical slicing, deletion-based slicing, including critical statements in slice only --- a statement is critical if deleting it results in changed observed behaviour for the slicing criterion)


Beszédes, Á., Faragó, C., Szabó, Z. M., Csirik, J., & Gyimóthy, T. Union slices for program maintenance. ICSM, 2002. (union slicing, unionizing traditional dynamic slices of multiple inputs, combined with static slices for software maintenance tasks)

Hall, Robert J. *Automatic extraction of executable program subsets by simultaneous dynamic program slicing*. Automated Software Engineering, 1995. (simultaneous dynamic slicing, computing dynamic slices of multiple inputs simultaneously to address the unsoundness/incorrectness of union slicing)

--- Information flow -- integrity & confidentiality of information (static and dynamic approaches)


***O. Tripp et al. “TAJ: Effective Taint Analysis of Web Applications”, PLDI 2009***

Other Sources:


--- Android security – dealing with a Java-family implementation language


***Feng, Y., Anand, S., Dillig, I., & Aiken, A. Apposcopy: Semantics-based detection of android malware***
through static analysis. FSE, pp. 576-587, 2014. (identifying Android malware that steals private user information, using static taint analysis and Inter-component call graph)


##Dynamic Detection of Inter-application Communication Vulnerabilities in Android. Roee Hay, Omer Tripp and Marco Pistoia (IBM, Israel; IBM Research, USA), ISSTA 2015.


Other sources:

Li, Li, Alexandre Bartel, Tegawendé François D. Assise Bissyande, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. iccTA: detecting inter-component privacy leaks in android apps. ICSE. 2015. (static taint analyzer detecting privacy leaks during ICC in android apps)

Grace, M. C., Zhou, Y., Wang, Z., & Jiang, X. Systematic Detection of Capability Leaks in Stock Android Smartphones. NDSS, 2012. m(known as Woodpecker, accidental capability leak detection for android apps, not implemented in a program-analysis framework but a tool-based approach mixing Java, Shell, and Python code atop an off-the-shelf dissembler; a forward context-insensitive analysis)


Scalable and Precise Taint Analysis for Android. Wei Huang, Yao Dong, Ana Milanova and Julian Dolby (Google, USA; Rensselaer Polytechnic Institute, USA; IBM Research, USA) , ISSTA 2015.

---Unsafe analysis techniques (e.g., Symbolic execution and Concolic Testing)

Concolic Testing


Other sources:


R. Majumdar, K. Sen, "Hybrid Concolic Testing" ICSE 2007


Combinations of dynamic and static analyses and soundness.


Other sources:


--- Change impact analysis & program understanding –a SE client of dynamic and static analyses

Ren, X., Shah, F., Tip, F., Ryder, B. G., & Chesley, O. Chianti: a tool for change impact analysis of java programs. OOPSLA, 2004. (descriptive test-case-level static impact analysis, finding impacted and impacting test cases based on change-type information)


Orso, A., Apiwattanapong, T., & Harrold, M. J. Leveraging field data for impact analysis and regression testing. FSE, 2003. (predictive coverage-based dynamic method-level impact analysis; intersect approximate method-level static forward slice and method-level coverage)

Other sources:

Badri, L., Badri, M., & St-Yves, D. Supporting predictive change impact analysis: a control call graph based technique. APSEC, 2005. (predictive static impact analysis based on call graph)

Ramanathan MK, Grama A, Jagannathan S. Sieve: a tool for automatically detecting variations across program versions. ASE, 2006. (descriptive statement-level dynamic impact analysis using longest-common-sequence based execution history differencing)

Gethers, M., Dit, B., Kagdi, H., & Poshyvanyk, D. Integrated impact analysis for managing software changes. ICSE, 2012. (change-request-level impact analysis combining information retrieval, dynamic analysis, and data mining techniques; can work in both predictive and descriptive settings, depending on the classes of information provided)
