Lecture 1. Overview

Wei Le

2016.1
What Is the Course About

- Program analysis aims to automatically reason about the code and program executions for predicting software behaviors. The course will teach:
  - how to determine program properties (foundational knowledge of program analysis)
  - what are the purposes to do so (applications of program analysis)
- Program properties: what facts hold for partial or all executions
http://www.cs.iastate.edu/~weile/cs513x.html
Course Objectives

- Know **terminologies and mathematical frameworks** needed to read program analysis literature,
- Know the capabilities and tradeoffs of **program analysis algorithms** and implement them,
- Gain problem solving skills and apply program analysis to solve software engineering and security problems (**problem reduction**),
- **Start program analysis research**
Course Work and Evaluation

- Reading and survey
- Implementation
- Research project (2 persons a team)
Course Topics and Survey Signup

1. Points-to analysis
2. Computability, complexity, soundness, completeness, flow-sensitivity, context-sensitivity, path-sensitivity
3. Control flow analysis and type inferences: call graphs, cfg, icfg
4. Dependency and program slicing, chopping, taint analysis
5. Abstract interpretation
6. Dataflow analysis
7. Interprocedural analysis
8. Symbolic execution
9. Infeasible paths detection
10. Typestate
11. Delta-debugging
12. Dynamic analysis
13. Program analysis and its related areas
14. Experimentation
Program Analysis and Other Areas of Computer Science

- **Programming Languages**
  - Define a language, and then design the analysis for the programs written in the language
  - Examples of language dependent features: recursion and level of pointer dereferences

- **Compilers**
  - Code optimization: first perform program analysis to determine the optimization opportunities; then transform the code
  - Semantic analysis: check if the properties of correctness holds

- **Software Engineering**
  - Program analysis tools are software that analyzes other software (like a compiler), they also require software engineering skills to build

- **Software Engineering, Security, Systems, Graphics and Robotics**
  - Problems in the domains such as test input generation, malware analysis and performance tuning need information from software.
Program Analysis and Other Areas of Computer Science

- **Theory**: An important part of the program analysis research is to develop algorithms and analyze their complexity, correctness and precision.

- **Machine Learning**: Program analysis generates data and thus we can apply machine learning to process these data and summarize the program properties.

- **Architecture**: We can develop architecture support to get information from executions to facilitate program analysis, e.g., hardware counter is useful to collect branch information for the execution paths.
Program Analysis: Problem Reduction

- From application domains to program analysis:
  - Compiler optimizations
  - Bug Findings
  - Generate exploits
  - Test input generation
  - Infer program specifications
  - Debugging
  - Repair
  - ...

- From program analysis to constraint solving:
  - Theorem prover
  - Boolean Satisfiability (SAT) solver
  - Satisfiability Modulo Theories (SMT) solver
  - ...

Program Analysis: Challenges

- Challenges: large state space
- What is a program state:
  - A program state consists of the values of all the variables at a particular program point.
  - Relations of code, paths, inputs and executions
- Two key ideas:
  - ABSTRACTION: merging states – typically done in static analysis
  - SAMPLING: collecting partial states – both in static and dynamic analysis
Static Analysis

Learning from the source or binary code:

- Analyzing code without running a program,
- Offering techniques for predicting statically at compile-time the set of configurations or behaviors dynamically at run-time
- Automating code inspection.

How to perform it:

- Convert source code to some sort of program representation (graphs or trees)
- Traverse the representation to collect information (static analysis algorithm: how to traverse it?)
- Determine if the properties hold based on the information
Theoretical Complexity of Static Analysis

Static analysis is undecidable. Thus,

- The static analysis algorithm is an approximation.
- The results of static analysis tools are imprecise in general.
Implementing Static Analysis

- Frameworks: Soot, LLVM, Phoenix
- Program representations and IR: AST, CFG, Callgraphs
- Symbol table: variable types
- Program analysis algorithms: points-to, loop analysis, value range information, definition and use of variables
- Other supporting facilities: optimizing program analysis information, storing in-progress program analysis, mapping between source and IR
- Standalone tools: codesurfer, atlas, findbugs ...
Dynamic Analysis

Learning from executions:

- Run code/Execution (what is the input)
- Profiling and monitoring (how to get useful information while minimizing the overhead)
- Data analysis (offline or online):
  - Frequency, coverage and correlation of program entities
  - Build graph representation for the data
  - ...
Combine and Contrast Static and Dynamic Analysis

The two analyses can be performed on
- Different parts of code [Le:ICSE13]
- Different parts of executions [Godefroid:PLDI05]
- Different passes [Csallner:ICSE05]

Table: An Instance Comparison of Static and Dynamic Analyses: Bug Detection

<table>
<thead>
<tr>
<th>Requirement</th>
<th>static</th>
<th>dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>code (compile, build, source, binary)</td>
<td>executable, input</td>
</tr>
<tr>
<td>Advantages</td>
<td>apply early in lifecycle</td>
<td>easy for debugging</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>false positives</td>
<td>false negatives</td>
</tr>
</tbody>
</table>
Program Analysis: Research Directions

- Algorithms to enable more precise and scalable analysis
- Algorithms to compute new type of program information (e.g., probabilistic symbolic execution)
- Algorithms to address new types of programming languages and paradigms (e.g., javascripts, web applications, Android)
- Problem reduction: discover new types of problems (e.g., program synthesis)
- ...