Lecture 1. A Glossary of Terms

Wei Le

2014.8
Syllabus

- What this course is about
- Office hours and TA
- References
- Evaluation
- Class policies
- Tentative schedule and topics
- Course web: http://www.cs.iastate.edu/~weile/cs641.html
Program Analysis: Goals

- Get information from the code and executions
- Reason about the code and executions to predict program behaviors
- Determine program properties: given code or a set of executions, what facts (or constraints) hold for partial or all executions (all inputs, all execution paths)
Program Analysis: Applications and Problem Reduction

From application domains to program analysis:
- Compiler Optimization
- Bug Findings
- Test Input Generation
- Understanding Programs
- Debugging
- ...

(example on the board)

From program analysis to constraint solving [4]:
- Theorem prover
- Boolean Satisfiability (SAT) solver
- Satisfiability Modulo Theories (SMT) solver
Program Analysis and Other Areas of Computer Science

- **Programming Languages**: The analysis is based on programs written in a particular language (e.g., recursion and level of pointer dereferences).

- **Compilers**: Code optimization needs to first perform program analysis and then transform the code.

- **Software Engineering**: Program analysis tools are software that analyzes other software (like a compiler).

- **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.

- **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: Types

- **Static Analysis:**
  - Analyzing code without running a program,
  - Offering techniques for predicting statically at compile-time safe and efficient approximations to the set of configurations or behaviors dynamically at run-time [7],
  - Automating code inspection.

- **Dynamic Analysis:** Learning from executions

- **Hybrid Analysis:** The two analyses are performed on
  - Different parts of code [6]
  - Different parts of executions [5]
  - Different passes [3]

### 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></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: Challenges

- Challenges: 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 inputs, paths, executions and code
  - infeasible paths

- Two key ideas:
  - ABSTRACTION: merging states – typically done in static analysis
  - SAMPLING: collecting partial states – both in static and dynamic analysis
Theoretical Complexity of Program Analysis

Generally speaking, program analysis is *undecidable*, which implies it is either [8]:

- *Unsound*
- *Incomplete* or
- *Non-terminating*: always gives correct answer when it terminates, but may run forever

*Soundness and completeness*
Problem Definition: We are given a set of pointers, a program (or say, its control flow graph) and two pointers $p$ and $q$. Three types of assignment statements are allowed in the program: (1) $* * * . . . * x = & y$, (2) $* * * . . . * x = * * * . . . * y$ and (3) $* * * . . . * x = \text{NEW}$. The third statement creates a new unnamed variable and makes $* * * . . . * x$ point to it. Two types of control flow statements are allowed: $if(\ldots)then\ldots else\ldots$ and $while(\ldots)\ldots$. Now the goal is to check if there is some path from the start node to the exit node in the control flow graph, such that, at the end of executing the statements along the path, $p$ points to $q$. 
Theoretical Complexity of Program Analysis [2]

Points-to Analysis as an Example

- Restrictions (boundary of the state space) on the program:
  - variables (scalar or structure)
  - dynamic memory (allocate memory at run-time) – the configuration space of pointers is infinite
  - procedures (single or multiple)
  - level of pointer dereference
  - typing: int* only can points to int

- Types of program analysis, e.g., whether it is flow-sensitive or flow-insensitive points-to analysis
Theoretical Complexity of Program Analysis [2]

Points-to Analysis as an Example

- Points-to Analysis
  - With dynamic memory (Undecidable)
  - Without dynamic memory
    - With well defined types
      - Levels $\geq 2$ (PSPACE-Complete)
    - Without well defined types (PSPACE-Complete)
Program Analysis: Research Directions

- Algorithms to enable more precise and scalable analysis
- Algorithms to address new types of programs
- Reduction to program analysis: new types of properties and problems, program synthesis
- ...

Wei Le
Lecture 1. A Glossary of Terms
2014.8 12 / 20
Reading

- Reading 1: A Survey of Automated Techniques for Formal Software Verification: Sections 1, 2 and 5
- Reading 2: New results on the computability and complexity of points-to analysis

Reading notes (within a page) due Fri 2:15pm: a list of 1) questions and 2) insights and ideas
Control flow analysis: Determine the execution order of the statements in a program (and sometimes specify them in a graph representation).

```java
class MaliciousActivity extends Activity{
    void onCreate(Bundle b) {
        setContentView(...);
    }
    void onStart() {
        ...
        this.id = tm.getDeviceId();
        Intent s = new Intent(this, VulnerableRecord.class);
        s.putExtra("data", this.id);
        s.putExtra("url", URL);
        sendBroadcast(s);
    }
    void onStop() {
        ...
    }
}
```

(example on the board)
**Dataflow analysis**: Determine the program variable usage patterns.

A set of dataflow problems:
- whether the variable always gets constant at a program point along a particular program path?
- whether a definition can reach a use?
- whether a variable is never used after a program point?
- whether an expression has already been computed?

(example on the board)
Static Analysis: Types – From Goal Point of View

- **Value flow analysis [1]**: When analyzing programs for value recomputation, one faces the problem of naming the value that flows between equivalent computations with different lexical names.

- **Points-to analysis (alias analysis, pointer analysis)**: Given a program and two variables p and q, the goal of points-to analysis is to check if p can point to q in some execution of the program. If so, the two pointers are *aliasing*.

- **Shape analysis (a type of points-to analysis)**: Analyzing linked, dynamically allocated data structures in programs.

- **Symbolic analysis (symbolic execution)**: Determine symbolic values for program variables – how to represent arbitrary program variables at a program point using input variable.

  (example on the board)
Static Analysis: Types – From Methodology Point of View

- **Whole-program analysis**: Requiring the entire program to perform analysis
- **Modular program analysis [8]**:
  - Can analyze parts of a program in isolation and then compose the information if needed.
  - A standard approach to modular analysis is to use formulas to represent program states, with free variables in the formulas capturing the unknown state of the partial program’s environment.
- **Intra-procedural vs Inter-Procedural**: Get the information within a procedure (or across procedures)
- **Path-sensitive vs Path-insensitive**: Whether the analysis distinguishes (or merges) the information collected from different program paths
- **Context-sensitive vs Context-insensitive**: Whether the analysis considers calling context when performing inter-procedural analysis
- **Flow-sensitive and Flow-insensitive**: Whether the analysis considers the statement orders when performing analysis; (Flow-insensitive analysis assumes statements can execute in any order, rather than producing a solution for each program point, producing a valid single solution for the whole program)
Reading

- Reading 1: The Concept of Dynamic Analysis
- Reading 2: The Heartbleed bug (http://www.pl-enthusiast.net/2014/07/01/how-did-heartbleed-remain-undiscovered-and-what-should-we-do-about-it/)
Dynamic Analysis

- Two types: *Online* and *Offline* dynamic analysis
- Three general steps: execution, monitoring and data analysis
- A *program profile* counts the number of times program entities occur in a program execution:
  - Program instrumentation (profiling)
- Frequency, coverage and correlation
Assignment

Select one of the important, well-known, interesting or yourself-discovered bugs in an open source program, and write an report on:

- meta information of the bug: when it is discovered, how it is found, what is the consequence of the bug, where is the bug report (if applied)
- analysis of the bug: what is the source code that contains the bug, what are the tests (if any) that can trigger the bug
Rastislav Bodík and Sadun Anik.
Path-sensitive value-flow analysis.

Venkatesan T. Chakaravarthy.
New results on the computability and complexity of points–to analysis.

Christoph Csallner and Yannis Smaragdakis.
Check ’n’ crash: Combining static checking and testing.

Leonardo de Moura.
SMT Solvers, Theory and Practice.

Dart: Directed automated random testing.
Wei Le.
Segmented symbolic analysis.

Flemming Nielson, Hanne R. Nielson, and Chris Hankin.
*Principles of Program Analysis.*

A Gentle Introduction to Program Analysis.
SMT Solvers, Theory and Practice.