Program Analysis And Its Support in Software Development

Qing Yi

class web site:
www.cs.utsa.edu/~qingyi/cs6463

A little about myself

Qing Yi

- *B.S.* Shandong University, China.
- *Ph.D.* Rice University, USA.
- Assistant Professor, Department of Computer Science
- **Office: SB 4.01.30**
- **Phone : 458-5671**

Research Interests

- Compiler construction
  program analysis and optimizations for high-performance
- Programming languages
  type systems, object-oriented design.
- Software engineering
  automatic code generation; structure discovery of software systems; systematic error-discovery and verification of software.

General Information

- Class website
  - www.cs.utsa.edu/~qingyi/cs6463
    - Check for class handouts and announcements
  - Office hours: by appointment
- Textbook
  - Principles of Program Analysis
    - Flemming Nielson, Hanne R. Nielson, Chris Hankin.
    - ISBN 3-540-65410-0
- Requirements
  - Familiarity with C/C++
  - Basic understanding of algorithms, programming languages and compilers
- Grading
  - Fundamental theories
    - In class exercises: 30%
  - Research experience
    - Research paper presentation and discussion: 30%
    - Research project: 40%

What is Program Analysis?

- The process of automatically analyzing the behavior of computer programs
  - From Wikipedia, the free encyclopedia
- The key idea about *static* program analysis
  - Discover information without executing the program
- Compare to alternative dynamic approaches
  - Trace program behavior on a collection of sample inputs
    - Profiling --- for the purpose of performance optimization (sometimes also called performance analysis)
    - Testing --- for the purpose of validating program correctness
    - Debugging --- for the purpose of discovering erroneous behavior
  - Why do we need both?

Program Analysis in Software Development and Adoption

- Software development includes four steps
  - Requirements analysis --- what are the execution conditions? What is the desired behavior in every situation?
  - Software design --- how to guarantee the desired behavior?
  - Programming/revision --- write code that implements the plan
  - Compilation and Testing --- does your code compile and run? Does it work as planned? If not, back to programming/revision.
  - Maintenance --- what if demands have changed, or new bugs are found?
- Software adoption and integration
  - Most companies don't write all their code. They buy projects from other companies and sometimes use open-source projects
  - How do you make sure whether other people's code are safe? How do you understand other people's code?

Program analysis --- can apply anytime during code development

Program Analysis in Software Development and Adoption

- Program analysis --- support software development and adoption
  - Compilation and translation --- syntax and semantics analysis, identify syntax and type errors without running the program
    - Smart development environment (check simple errors as you type the code)
  - Program Optimization --- program transformations that do not change the meaning of the program
    - Improve performance, reduce resource consumption, reduce code size, …
    - Code revision/re-factoring ==> reusability, maintainability,
  - Program correctness --- Is the program safe? Is it secure? Is it dependable?
    - Testing --- path coverage: are the given set of sample input sufficient to test all execution paths (all the required behaviors) of the program?
    - Program validation --- is the program guaranteed to satisfy certain properties? Is the implementation safe, secure, and dependable?
    - Program integration --- are there any communication errors among different components of a collaborated project?
  - Program understanding --- extract high-level semantics from low-level implementations (reverse engineering)

Static vs. Dynamic Program Analysis

- When not explicit, program analysis normally implies *static program analysis*
- Static Program Analysis
  - + Does not need program to be executable
  - + Can prove properties of a program => if a program is proven to be correct, it is correct
  - – Does not have program input; cannot figure out many things
  - – Must approximate information (it is NP complete to enumerate all program execution paths)
- In contrast, dynamic analysis
  - Need program to be executable, cannot prove properties of programs (testing alone cannot ensure a program is correct)…

Static and dynamic analysis complement each other and are often used together

Focus of this class

- Fundamental approaches in program analysis
  - There are many different program analysis algorithms, but they are fundamentally very similar
  - There are only about 4-5 commonly used approaches. These approaches are formulated differently to solve different problems
- Analyzing different properties of a program
  - Control flow analysis, data flow analysis, dependence analysis
  - Aliasing analysis, pointer analysis, shape analysis,
  - type analysis, path analysis, security property analysis
  - And others
- Research experience
  - Study literature, identify an important problem, solve the problem with a program analysis approach, evaluate the solution.

Literature Study

- We will study a wide variety of program analysis applications
  - Analyzing different input applications
    - Computation intensive code, data intensive code, current threads, distributed applications, web applications, database applications, and others
  - In different programming languages
    - Fortran, C, C++, Java, C#, assembly, and others
  - To support performance optimizations
    - Redundancy elimination, loop optimizations, inter-procedural optimization, and others
  - To support program debugging
    - Uninitialized variables, null pointer dereference, array out-of-bound access, and others
  - To support program understanding, validation, testing, security,…

Building Program Analysis Tools

- In order to analyze a program, a tool must understand the programming language, although the analysis algorithms are often independent of specific programming languages

- Ways to resolve the problem
  - Build a small parser for a small prototyping language --- eg., the While language used in the textbook
    - Easy and flexible if publishing paper is all you want
  - Use existing infrastructures from open-source compilers/interpreters and other language processors
    - These are many infrastructures available
  - We will study both approaches.
    - We will study the use of some existing infrastructures

Roadmap

- Week1-4 --- Fundamental theories and program analysis approaches
  - Materials from the textbook
  - Instructors giving lectures and in-class exercises
  - Students select from a pool of papers and project ideas
  - Initial project plan due
    - Each project must resolves at least one non-trivial problem and evaluates the solution
- Week5-13--- Program analysis applied to solve real problems
  - Materials both from the textbook and from the research literature
  - Student paper presentations interspersed with instructor lectures
  - Class discussion and in-class exercises
  - Project intermediate and final report