AAA616: Program Analysis

Course Overview

Hakjoo Oh
2019 Fall

Basic Information

Instructor: Hakjoo Oh

- **Position:** Associate professor in Computer Science and Engineering, Korea University
- **Expertise:** Programming Languages
- **Office:** 616c, Science Library
- **Email:** `hakjoo_oh@korea.ac.kr`
- **Office Hours:** 1:00pm–3:00pm Mondays (by appointment)

Course Website:

- `http://prl.korea.ac.kr/~pronto/home/courses/aaa616/2019/`
- Course materials will be available here.

Unsafe Software

- (1996) The Arian-5 rocket, whose development required 10 years and $8 billion, exploded just 37s after launch due to software.

- (1998) NASA's Mars climate orbiter lost in space. Cost: $125 million
- (2000) Accidents in radiation therapy system. Cost: 8 patients died
- (2007) Air control system shutdown in LA airport. Cost: 6,000 passengers stranded
- (2012) Glitch in trading software of Knight Captal. Cost: $440 million
- (2014) Airbag malfunction of Nissan vehicles. Cost: $1 million vehicles recalled
- ... Countless software projects failed in history due to bugs.

Towards Safe Software Technology

- The technology for efficient software is mature.

$$\text{Program} \rightarrow \boxed{\text{Interpreter}} \rightarrow \text{Result}$$

- However, technology for safe software is not. Current language systems put almost all the burden of writing safe programs on the programmers. This manual approach to safe software has proven extremely unsuccessful.
- Automated technology for analyzing the safety of programs:

$$\text{Program} \rightarrow \boxed{\text{Analyzer}} \rightarrow \boxed{\text{Interpreter}} \rightarrow \text{Result}$$

Static Program Analysis

- Technology for predicting SW behavior statically and automatically
  - static: before execution, before sell / embed
  - automatic: sw is analyzed by sw ("static analyzer")
- Applications
  - bug-finding: e.g., find runtime failures of programs
  - security: e.g., is this app malicious or benign?
  - verification: e.g., does the program meet its specification?
  - compiler optimization: e.g., automatic parallelization
  - program synthesis, automatic patch generation, etc

Topics

In this course, we will focus on principles of program analysis:

- Programming language theory: semantic formalism, operational semantics, denotational semantics
- Abstract interpretation: a unified framework for designing static analyses

Prerequisites:

- Undergraduate-level programming languages, compilers, theory of computation, and discrete math

Course Materials

- Xavier Rival and Kwangkeun Yi. Introduction to Static Analysis: An Abstract Interpretation Perspective. MIT Press

Grading

- Quiz –50%
  - Every class will begin with a short quiz (closed book)
- Final exam – 30%
- Paper reading – 20%
  - 3–5 papers will be given as reading assignments

Schedule

Weeks	Topics
Week 1	Introduction & Notation
Week 2	Preliminaries: Operational Semantics
Week 3	Preliminaries: Denotational Semantics
Week 4	Abstract Interpretation
Week 5	Abstract Interpretation
Week 6	Abstract Interpretation
Week 7	Advanced Static Analysis Techniques
Week 8	Advanced Static Analysis Techniques
Week 9	Mid-term exam
Week 10	Static Analysis Tool Implementation
Week 11	Static Analysis Tool Implementation
Week 12	Static Analysis for Advanced Programming Features
Week 13	Static Analysis for Advanced Programming Features
Week 14	Classes of Semantic Properties and Verification by Static Analysis
Week 15	Type and effect system
Week 16	Specialized Static Analysis Frameworks