AAA616: Program Analysis

Lecture 0 — Course Overview

Hakjoo Oh
2018 Spring

Basic Information

Instructor: Hakjoo Oh

- **Position:** Assistant 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/aaa616/2018/`
- Course materials will be available here.

Software Failures in History

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

- (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.

Recent Security Vulnerabilities

SHELLSHOCK
BASH BUG

Towards Safe Software Technology

- The technology for efficient software is mature.

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

- 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:

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

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

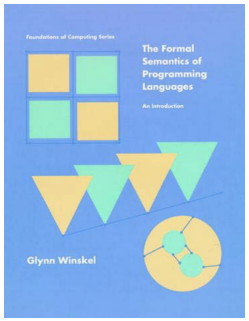
In this course, we will focus on principles of program analysis (rather than applications):

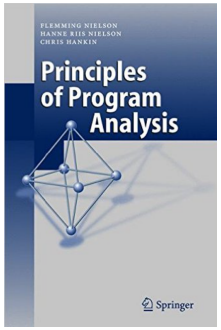
- Programming language theory: semantic formalism, operational semantics, denotational semantics
- Static analysis approaches: Data-flow analysis, Constraint-based analysis, Abstract Interpretation, Type and effect system

Prerequisites:

- Programming languages, compiler, theory of computation

Course Materials





- The Formal Semantics of Programming Languages by Glynn Winskel. MIT Press.
- Principles of Program Analysis by Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. Springer.

Grading (tentative)

- Homework – 20%
- Mid/Final exams – 30/40%
- Attendance – 10%