# AAA615: Formal Methods

## Lecture 0 — Course Overview

Hakjoo Oh
2017 Fall

# Basic Information

Instructor: Hakjoo Oh

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

# Motivation: Unsafe Software

Software errors cost the U.S. economy $60 billion every year.

- (1996) Explosion of the Arian-5 rocket. Cost: $8 billion

- (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.

# Current Technologies for Safe Software

Very primitive the status-quo:

- Code review
- Testing
- Debugging
- Simulation, ...

$\Rightarrow$ Manual, ad-hoc, incomplete, expensive, and postmortem.

# About This Course

Cutting-edge technologies for building software in safer and easier ways:

- Program verification
- Program analysis
- Program synthesis
- Program repair

# Program Verification

Techniques for verifying programs according to specifications:

$$
\begin{array}{l}
\texttt{@input} : \top \\
\texttt{@output} : \textbf{sorted}(rv, 0, |rv| - 1) \\
\text{bool BubbleSort (int } a[]) \ \{ \\
\quad \text{int[] } a := a_0 \\
\quad \text{for (int } i := |a| - 1;\ i > 0;\ i := i - 1) \ \{ \\
\quad\quad \text{for (int } j := 0;\ j < i;\ j := j + 1) \ \{ \\
\quad\quad\quad \text{if } (a[j] > a[j+1]) \ \{ \\
\quad\quad\quad\quad \text{int } t := a[j]; \\
\quad\quad\quad\quad \text{int } a[j] := a[j+1]; \\
\quad\quad\quad\quad \text{int } a[j+1] := t; \\
\quad\quad\quad \} \\
\quad\quad \} \\
\quad \} \\
\quad \text{return } a; \\
\}
\end{array}
$$

# Program Analysis

Automated techniques for analyzing program behaviors

- typically interested in weaker properties than verification
- static vs. dynamic approaches

Example: Sparrow
The Early Bird

Technology for "Software MRI"


- Aims to detect memory errors in C programs such as buffer-overrun, memory leak, null-dereference, etc
- fully automated, guaranteed to find all bugs

# Program Synthesis

Techniques for generating programs from specifications:

$$\text{reverse}(12) = 21,\ \text{reverse}(123) = 321$$

```
reverse(n){
  r := 0;
  while ([        ]){
    [                ]
  };
  return r;
}
```

2.5s →

```
reverse(n){
  r := 0;
  while (n > 0){
    x := n % 10;
    r := r * 10;
    r := r + x;
    n := n / 10;
  };
  return r;
}
```

# Program Repair

Automated techniques for fixing software errors:

```
p = malloc(); // o1

if(...){
    q = malloc(); // o2
    *q = *p;
} else {
    q = p;
}
// Use q
free(p);

// o2 leaks here
return;
```

buggy program

```
p = malloc(); // o1

if(...){
    q = malloc(); // o2
    *q = *p;
    free(p);
} else {
    q = p;
}
// Use q
free(q);

return;
```

fixed program

# Topics

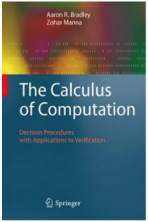
Computational logic and its application to formal approaches in software engineering, including program verification, analysis, synthesis, and repair.

- **Part 1 (Foundations)**:
  - Propositional logic
  - First-order logic
  - First-order theories
  - SAT/SMT solvers
- **Part 2 (Applications)**:
  - Program verification
  - Program analysis
  - Program synthesis
  - Program repair

**Prerequisites**: programming language theory and program analysis

# Course Materials

- Textbook: Aaron R. Bradley and Zohar Manna. The Calculus of Computation. Springer.




- Materials from related courses:
  - Computer-Aided Reasoning for Software. Univ. of Washington
    `https://courses.cs.washington.edu/courses/cse507/17wi/`
  - Automated Logical Reasoning. Univ. of Texas at Austin
    `http://www.cs.utexas.edu/~isil/cs389L/`

# Grading

- Exam – 30%
- Project – 60%
  - Goal: apply what you learn and build a prototype programming tool (analyzer, verifier, synthesizer, patch generator, etc)
  - Schedule:
    - Proposal (1–2 pages): due 10/31 (Mon) in class
    - Demo: 12/4 (Mon) in class
    - Paper (–5 pages): due 12/11 (Mon) in class
  - Make a team of 1–3 students
- Participation – 10%

# Schedule (tentative)

| Weeks | Topics |
|---|---|
| Week 1 | Propositional Logic |
| Week 2 | Propositional Logic |
| Week 3 | CDCL SAT solvers |
| Week 4 | Applications of SAT Solvers |
| Week 5 | First-order Logic |
| Week 6 | First-order theories |
| Week 7 | SMT Solvers |
| Week 8 | Program verification |
| Week 9 | Program verification |
| Week 10 | Exam |
| Week 11 | Program analysis |
| Week 12 | Program analysis |
| Week 13 | Program synthesis |
| Week 14 | Program repair |
| Week 15 | Project demo |
| Week 16 | Wrap up |