Overview

CS560: Reasoning About Programs

Roopsha Samanta
Roadmap

Today

- Motivation
- Overview
- Logistics
Roadmap

Today

- Motivation
- Overview
- Logistics
# What is this course about?

Logical foundations & algorithmic techniques to ensure program correctness

<table>
<thead>
<tr>
<th>Specification</th>
<th>Logics to express expected program behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verification</td>
<td>Methods to automatically check if a program satisfies a specification</td>
</tr>
<tr>
<td>Repair</td>
<td>Methods to automatically fix an incorrect program</td>
</tr>
<tr>
<td>Synthesis</td>
<td>Methods to automatically generate a correct program</td>
</tr>
</tbody>
</table>
Why should you care?

Programmers make mistakes

99 little bugs in the code.
99 little bugs in the code.
Take one down, patch it around.
127 little bugs in the code...
Why should you care?

Software bugs can be expensive, or fatal, or both!
Therac-25 radiotherapy machine overdose, mid-1980s

6 deaths. Overflow error, race conditions
Ariane 5 explosion

$7 billion loss. Overflow error
North American power blackout

11 deaths, $6 billion loss. Race condition
Turing Awards

Dijkstra

Floyd

Hoare

Milner

Pnueli

Clarke

Emerson

Sifakis

Lamport
Success stories

- Intel CPU arithmetic and logical operations
- Microsoft device drivers
- Rockwell Collins AAMP7G microprocessor’s partition management
- Rolls Royce Trent Series Health Monitoring Units
- Lockheed Martin C130J Mission Computers
- Boeing “Little Bird” helicopter (seL4 OS-based mission computer)
- Royal Navy Ship/Helicopter Operating Limits Unit
- Airbus 380 primary flight control software
- Paris Metro (RATP)
- NASA Mars Rover data management subsystem
- Bombardier ILLBV950L2 railway interlocking system
- Apple, ARM/SoftBank, Nvidia, IBM, Oracle RTL
- AMD K5 floating point square root microcode
- Micrium OS μC/OS-II real-time kernel
Roadmap

Today

- Motivation
- Overview
- Logistics
Testing shows the presence, not the absence of bugs.

Ergo, testing can fail to show the presence of some bugs!
Write program

Test

Fix

Debug

Test-suite

Pass

Correct?

Fail
Write program

```c
int max (int x, int y)
    m = 0;
    if (x > y) m = x;
    return m;
```

Test

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>m</th>
<th>Pass?</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>Pass</td>
</tr>
<tr>
<td>100</td>
<td>99</td>
<td>100</td>
<td>Pass</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>Pass</td>
</tr>
</tbody>
</table>

Debug

Correct?
```c
int max (int x, int y)
{
    int m = 0;
    if (x > y) m = x;
    return m;
}
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>m</th>
<th>Pass?</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>✓</td>
</tr>
<tr>
<td>100</td>
<td>99</td>
<td>100</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>✗</td>
</tr>
</tbody>
</table>
```c
int max (int x, int y)
    m = 0;
    if (x >= y) m = x;
    return m;
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>m</th>
<th>Pass?</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>✓</td>
</tr>
<tr>
<td>100</td>
<td>99</td>
<td>100</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>✓</td>
</tr>
</tbody>
</table>

Correct??

- Write program
- Test
- Fix
- Debug

Pass

Fail
Dijkstra

Testing shows the presence, not the absence of bugs.

Formal specifications can precisely capture correctness requirements.

Formal verification can **prove** the absence of bugs!

Formal repair can ensure the absence of bugs for programs with bugs!
Write program

Formally verify

Formally Fix

Formally Debug

Formal specification

Pass

Correct.

Fail
Write program

```
int max (int x, int y) {
    m = 0;
    if (x > y) m = x;
    return m;
}
```

Formally verify

\[ \forall x, y. (m \geq x) \land (m \geq y) \land [(m = x) \lor (m = y)] \]

Pass

Correct.

Formally Fix

Formally Debug
\[
\forall x, y. (m \geq x) \land (m \geq y) \land \\
[(m = x) \lor (m = y)]
\]

```c
int max (int x, int y)
    m = 0;
    if (x > y) m = x;
    return m;
```

Correct.
Testing shows the presence, not the absence of bugs.

Program synthesis can generate programs that are **correct-by-construction**!
Write partial program \rightarrow Synthesize completion \rightarrow Formal specification \rightarrow Correct.
Write partial program

```
int max (int x, int y
m = ??;
if (??) m = ??;
return m;
```

Synthesize completion

```
∀x,y. (m >= x) ∧ (m >= y) ∧
[(m = x) ∨ (m = y)]
```

Correct.

```
int max (int x, int y
m = y;
if (x >= y) m = x;
return m;
```
Verifier

Program/Model
Specification

Yes/Proof
No/Bug

Type Systems
Deductive Verification
Model Checking
Abstract Interpretation
Deductive Verification

Model Checking

Abstract Interpretation

Type Systems

Interactive theorem provers

Automatic theorem provers

SAT/SMT solvers

Static analysis

Expressiveness

Automation

Scalability

Precision

Applicability
Unit 1: Introduction, Logics, and Proof Engines
Unit 2: Program Verification & Analysis
Unit 3: Program Synthesis & Repair
Unit 4: Advanced Topics, New Frontiers
Roadmap

Today
- Motivation
- Overview
- Logistics
Hi! I am Roopsha.
Developing algorithms/tools to assist programmers in writing reliable programs
Formal Verification and Synthesis for Distributed Systems

Semantics-guided Inductive Program Synthesis
Your turn!
Name?

CS/Math/ECE?

Undergrad/Grad?

Research Interests?

Why this course?
COVID-19 Impact

Lectures  Zoom SYNC-ONLINE
Syllabus  Course Website
Resources Brightspace
Discussions Piazza

If you feel sick, contact Protect Purdue Health Center at 765-496-4636!
Research papers, survey papers, handbook chapters
### Grading

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Project</td>
<td>40%</td>
</tr>
<tr>
<td>Midterm</td>
<td>20%</td>
</tr>
<tr>
<td>Homeworks</td>
<td>35%</td>
</tr>
<tr>
<td>Participation</td>
<td>5%</td>
</tr>
</tbody>
</table>
Class project

- You will write a paper and present a talk at our end-of-semester Workshop on Reasoning About Programs (WRAP) 2021!

- You will work in teams of 2-3 students for your project. Use Piazza to find teammates.

- Each of you will also *review* your peers’ papers!

- **Double-blind reviewing**: Reviewer and team identities are concealed.
Class project

- We will do some **Semantics-guided Inductive Program Synthesis (MANTIS)**!

- You will identify a domain and adapt MANTIS to it.

- *What is MANTIS?* Next class.
## Project deliverables

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposal</td>
<td>Identify team, domain</td>
<td>Feb 11</td>
</tr>
<tr>
<td>Partial Paper</td>
<td>Some sections of final paper</td>
<td>Mar 25</td>
</tr>
<tr>
<td>WRAP paper</td>
<td>Final paper</td>
<td>Apr 20</td>
</tr>
<tr>
<td>WRAP talk</td>
<td>Final presentation</td>
<td>Apr 29</td>
</tr>
</tbody>
</table>
Project grading

- WRAP Paper and Talk: 100% of Project Grade Proposal and Partial Paper will not be graded.

- WRAP Paper will be graded by peer reviewers and me.

- WRAP Talk will be graded by me.
Peer Review

- Each of you will serve on the Program Committee (PC) of WRAP 2021!

- Reviewing load: 2-3 papers.

- Each WRAP paper will be reviewed by a subset of your peers and discussed in a PC meeting on Apr 27.

- Goal of PC meeting: Rank papers.

- Reviewing criteria: Contribution, Originality, Presentation
Homeworks

- 5 homeworks
- Theoretical problem sets, programming assignments, paper reviews
- All homeworks will be weighted equally
- Upload to Piazza by 6:00pm on due dates
  
  Reviews (HW 5) will be due during PC meeting
Policies

- Be honest, reasonable and respectful.

- Presentations, write-ups and homeworks must be your own work.

- Teams are *not* allowed to discuss their project specifics with other teams.

- Do not copy text and figures from papers, websites, etc. Use your own words. Draw your own figures.

- See course website for all policies.
Summary

Today
- Motivation
- Overview
- Logistics

Next
- Introduction to program synthesis
- Project description