CSCI 699: Computer-Aided Verification

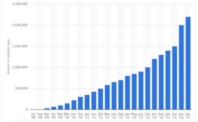
Mukund Raghothaman

Course Introduction

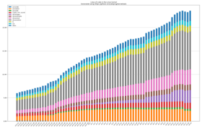
`https://r-mukund.github.io/teaching/sp2020-csci699/`

USC Viterbi
Department of Computer Science

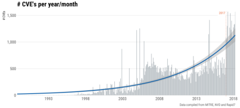
Expanding Scope and Complexity of Software



Apps on the App Store



Linux Kernel SLOC



Rate of CVEs being reported

Increasing Adoption of Formal Verification Tools by Industry

sonarqube

SonarSource

coverity®

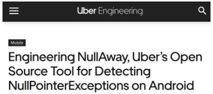
Synopsis

aws

Amazon

Infer

Facebook



Uber

Astrée

Airbus

SLAM

Microsoft

Overarching Questions of this Course

- How do we **reason** about code?
- How do we prove that they never go wrong?
- How do we prove that they eventually do good?
- How do we automate this reasoning process?
- Can we synthesize artifacts other than proofs? Code, itself?
- **How do we best help programmers write code?**

Code, as Broadly Construed

Or why you should take this course ...

- Traditional programs
- Neural networks
- Network controllers
- Controllers for cyber-physical systems
- Biological models of cells

Goals of this Course

,

- Introduce you to the art and science of program verification
- Provide exposure to using practical verification tools
- Open them up and study their inner workings
- Expose you to cutting edge research in the field

Outline of Today's Lecture

Motivation

Course Outline

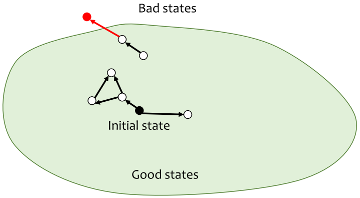
Logistics

Part 1: Techniques of Proof

- What is the **state** of a program, and how does it evolve?
- How do we show that bad states are never reached?
- How do we show that good states are eventually reached?
- Going from test cases to symbolic execution to proofs

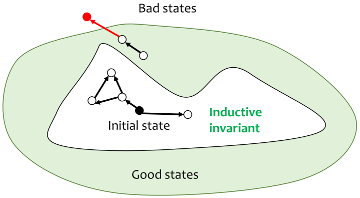
Part 1: Techniques of Proof

- What is the **state** of a program, and how does it evolve?
- How do we show that bad states are never reached?
- How do we show that good states are eventually reached?
- Going from test cases to symbolic execution to proofs



Part 1: Techniques of Proof

- What is the **state** of a program, and how does it evolve?
- How do we show that bad states are never reached?
- How do we show that good states are eventually reached?
- Going from test cases to symbolic execution to proofs



Part 2: Engines of Reason

- Propositional logic

$$a \text{ and } (\neg a \text{ or } b \text{ or } c)$$

- Satisfiability: Is an erroneous path feasible?
- Canonical NP-complete problem
- Massive progress in practical SAT solvers

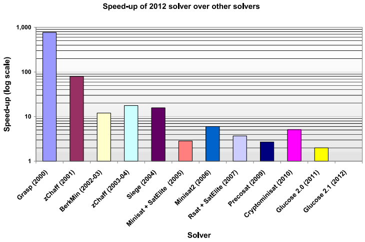


Figure: Credit: Sanjit Seshia

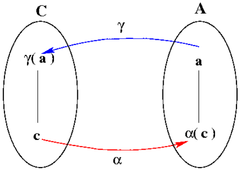
Part 2: Engines of Reason

- Formulas over theories

$$x \geq 10 \text{ and } (x \not\geq 10 \text{ or } x \leq 5 \text{ or } y > 8)$$

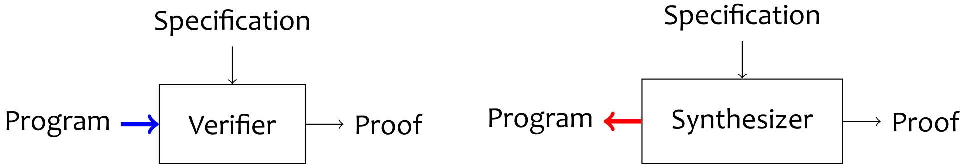
- Programs operate with data
- Verification conditions combine Boolean structure ($\wedge$ and $\vee$) with rich theories (integers, floating point numbers, arrays, heaps)
- Satisfiability, **modulo theory**

Part 3: Abstract Interpretation



- How do we soundly **abstract** the behaviors of programs?
- Focus on specific parts of the program behavior
  - Flow of values through the program
  - Range of values taken by a variable
  - Locks held at a program point
- Widely applicable framework. Examples:
  - Neural network certification (Gehr et al., SP 2018)
  - Verifying computer networks (Alpernas et al., SAS 2018)

Part 4: Program Synthesis



- Repurpose the constraint solving machinery of Part 2 to *synthesize* programs
- The holy grail: Systems that are **correct-by-construction**
- **Question 1:** How do users specify their **intent**?
- **Question 2:** How do we do synthesis?
- Exciting research area over the last 10 years
- Deep connections to AI and machine learning

Outline of Today’s Lecture

Motivation

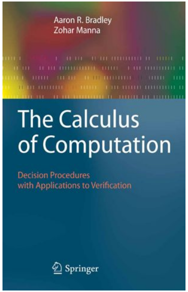
Course Outline

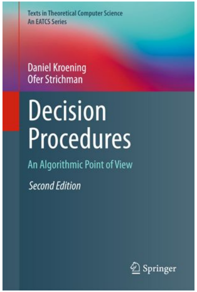
Logistics

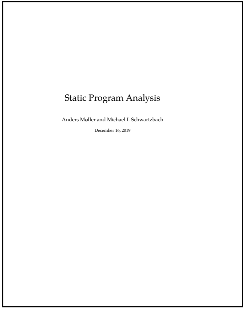
Grading

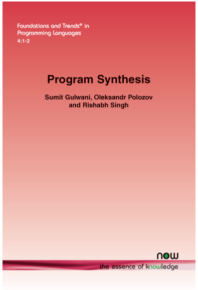
- Homework assignment associated with each unit (four in all)
- Class project
- All graded equally

Readings









- Will be assigned for each part
- Freely accessible from within the USC network

Class Project

- Either alone or in pairs
- Submit proposal by February 19
  - Describe problem, state deliverables, propose grading rubric
- Class presentations on April 22 and 27
- Final report by May 6

Class Project

- **Set up meeting with instructor to discuss project topics**
- Research project
  - Pick a research problem, devise possible solutions, write research paper
  - Ideally related to your own PhD research
- Survey project
  - Write a comprehensive survey on a research area
- Reimplementation project
  - Pick research paper (*not your own*), and reimplement the proposed techniques
  - Summarize experience

Logistics

- Class timings: Mondays and Wednesdays, 5–6:50pm, GFS 220
- Expected breakdown:
  - First 5 minutes: Recap of previous lecture, outline of present lecture, announcements
  - Last 10 minutes: Optional questions and discussion
  - 5 minute break at 6pm
- Office hours: Fridays, 3–5pm, SAL 308, **or by appointment**

- Course website: `https://r-mukund.github.io/teaching/sp2020-csci699/`
- Watch website regularly for announcements and updates

Introductions …

- **Mukund Raghothaman**
- New Assistant Professor (joined in August 2019)
- PhD and postdoc from the University of Pennsylvania
- Research in program verification and synthesis
- Applications of machine learning and probabilistic methods

Introductions …

- What is your background?
- Why do you feel like taking this course?
- What do you expect to get out of it?
- Previous experience in software engineering / programming languages?