Lecture 1

Course Overview & Introduction

Zvonimir Rakamarić
University of Utah

About Me

- Name: Zvonimir Rakamarić
- Born and raised in Croatia
  - BS from the University of Zagreb
- Moved to Canada in 2004
  - MS and PhD from the University of British Columbia
- Worked for a year with NASA Ames
- Started at the University of Utah in 2012
  - Leading SOARlab
    - Software Analysis Research Laboratory
    - http://soarlab.org/
  - Always looking for great students to join the lab

Course Overview

- Course page is on Canvas
- Main goals
  - Gain solid understanding of basic theory and practice behind proving correctness of programs
  - Cover advanced topics (interpolants, dealing with concurrency) in second part of the course
- Textbook: The Calculus of Computation by Aaron R. Bradley and Zohar Manna
  - Electronic version is free through SpringerLink

Topics

- Propositional logic and SAT
- First-order logic and SMT
- Verification conditions
  - Weakest precondition
- Proving program correctness
  - Pre- and post-conditions
  - Loop invariants
- Symbolic and concolic execution
- Advanced topics
  - Analyzing concurrent programs

Course Organization

- Lectures
  - Discuss basic and advanced verification topics
  - Emphasize on lasting foundations and theory
  - Reading research papers
- Homework assignments
  - Hands-on exercises accompanying presented material
  - Coding in your programming language of choice
- Projects
  - Focused, practical exploration of a topic related to software verification (and ideally your interests!)

Course Communication

- Leverage Canvas
  - Post questions
  - Discuss concerns
  - Ask for help and clarifications
- No fixed time for office hours
  - Catch me after class
  - Find me in my office
  - Message me
- Email: zvonimir@cs.utah.edu
  - Private questions (e.g., questions related to your grade)

Grading

- 50% homework assignments
  - 5-6 practical homework assignments
  - Each assignment is worth the same
- 50% course project
  - Project proposal (10 points)
  - Final presentation (30 points)
  - Final report (50 points)
  - Peer review (10 points)

- 5110 students are graded slightly differently (see course syllabus)

Course Projects

- Mini research projects
  - Publishing a (workshop) paper is the ultimate goal
- Deadlines still not defined
  - I will update the webpage by the end of this week
- I will also come up with a list of potential topics

- Team work
  - Allowed (up to 2 students)
  - You have to do twice as much work
  - If it is not clearly specified who did how much work, both students will get the same grade

Collaboration vs Cheating

- Discussing homework and project solutions at high-level is fine and encouraged
- **Basing your code/write-up on any other code/write-up is cheating**
  - **do not copy solutions from another student**
  - **do not copy solutions from the internet**
  - **do not even look at solutions from another student**
  - **do not ask for solutions on online forums**
  - **………**
- **Acknowledge appropriately any outside materials you used or rely on**

Collaboration vs Cheating cont.

- **I will officially report instances of cheating**
  - **<u>I will request that you fail this class</u>**
  - **If confirmed, cheating will be on your record with this department**

- **Ignorance is not a valid excuse**
  - **Read our policies on cheating**
  - **Talk to professors if you are still not sure**

Typical Cheating Scenario I

- Part of a student report copied from Wikipedia

In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.

Typical Cheating Scenario II

In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics [1].

[1] https://en.wikipedia.org/wiki/Formal_verification

Typical Cheating Scenario III

Wikipedia defines formal verification as follows [1]:

“In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.”

[1] https://en.wikipedia.org/wiki/Formal_verification

Typical Cheating Scenario IV

Formal verification encompasses tools and techniques for proving correctness of complex systems [1].

[1] https://en.wikipedia.org/wiki/Formal_verification

Late Policy

- Late homework assignments and project deliverables will not be accepted unless you contact me well ahead of the deadline and have a good excuse

Introduction to Software Verification

Discussion

- Where can software be found nowadays?
- Any bad software bugs you heard about?

Introduction to Software Verification

- Software is everywhere
  - Personal computers, mobile phones, in cars, ATMs, banks, planes, pacemakers, hospitals…
- Software has errors
  - Software systems are generally large, complex, and prone to errors…
  - And getting larger and more complex…
    - Heterogeneous hardware (multicore, GPUs)
  - …and more error prone!

Infamous Software Bugs

- 1962: Mariner I space probe
- 1982: Soviet gas pipeline
- 1985-87: Therac-25 medical accelerator
- 1988: Berkeley Unix finger daemon
- 1988-96: Kerberos Random Number Generator
- 1990: AT&T Network Outage
- 1993: Intel Pentium floating point divide
- 1995-96: The Ping of Death
- 1996: Ariane 5 Rocket
- 2000: Cancer institute's therapy planning software

Therac-25 Medical Accelerator

- Radiation therapy machine produced by Atomic Energy of Canada Limited (AECL)
- Bug: Race condition (concurrency error) between concurrent tasks in the Therac-25 software
  - Massive overdoses of radiation
- Between 1985-87 at least five patients die; others are seriously injured

Therapy Planning Software

- November 2000, National Cancer Institute, Panama City
  - Therapy planning software miscalculates the proper dosage of radiation for patients undergoing radiation therapy
- At least 8 patients die, another 20 receive overdoses likely to cause significant health problems

Ariane 5 Rocket

- June 4, 1996: Ariane 5 Flight 501 crash
- Working code for the Ariane 4 rocket is reused in the Ariane 5
- Ariane 5's faster engines trigger an overflow condition in an arithmetic routine inside the rocket's flight computer
- Flight computer crashes
  - The rocket explodes 40 seconds after launch

Automotive Industry

[http://www.embedded.com/columns/embeddedpulse/179100752]

- 2001: 52,000 Jeeps recalled due to a software error that can shut down the instrument cluster.
- 2002: BMW recalls the 745i since the fuel pump would shut off if the tank was less than 1/3 full.
- 2003: A BMW trapped a Thai politician when the computer crashed. The door locks, windows, A/C and more were inoperable. Responders smashed the windshield to get him out.

Automotive Industry cont.

- 2004: Pontiac recalls the Grand Prix since the software didn’t understand leap years. 2004 was a leap year.
- 2005: Toyota recalls 75,000 Prius hybrids due to a software defect
  - Cars stall or shut down while driving at highway speeds
  - Owners advised to bring their cars into dealers for an hour-long software upgrade
- 2010: Toyota recalls 300,000 Prius cars
  - Software bug?

Code Red Worm

- 2001: Code Red worm attacks the Index Server ISAPI Extension in Microsoft Internet Information Services
- Exploit used: Buffer overflow bug
- Worm released on July 13
- The number of infected hosts reached 359,000 on July 19
- Estimated damages are $2.6 billion

Heartbleed Bug

- Vulnerability in the OpenSSL cryptographic software library
- Simple problem, but discovered only in 2014
- Affected millions of machines

Motivation

- Software errors are costly
  - Software Fail Watch report for 2016: [https://www.tricentis.com/resource-assets/software-fail-watch-2016/] "The report identified 548 recorded software fails impacting 4.4 billion people and $1.1 trillion in assets."
- Improving software quality and reliability is a major software engineering concern
- 2016 NIST Report to the White House Office of Science and Technology Policy titled "Dramatically Reducing Software Vulnerabilities"
  - Software verification is prominently featured

Testing

- Quality assurance relies heavily on testing
- Pros
  - Scalable, precise (no false bugs)
  - Easy to adopt and understand
  - Testing (even random) does find lots of bugs
- Cons
  - Time consuming and costly
    - Writing (good) test cases
    - Tester:Developer ratio at Microsoft around 1:1
  - Coverage
    - Important bugs still escape

Simple Testing Example

```
void foo(int x) {
    ...
    ...
    ...
}
```

```
foo(???);

foo(INT_MAX);
foo(INT_MIN);
foo(0);
foo(random());
foo(random());
foo(random());
.........
```

Example Where Testing Works

```
void foo(int x) {
  if (x == 0) {
    BUG!
  }
}
```

Example Where Testing Fails

```
void foo(int x) {
   if (x == 914) {
      BUG!
   }
}
```

Formal Software Verification

- Definition from Wikipedia:
  - "Statically proving or disproving the correctness of a program with respect to a certain formal specification or property using formal methods of mathematics."
- Could be a very effective way to deal with the software reliability problem

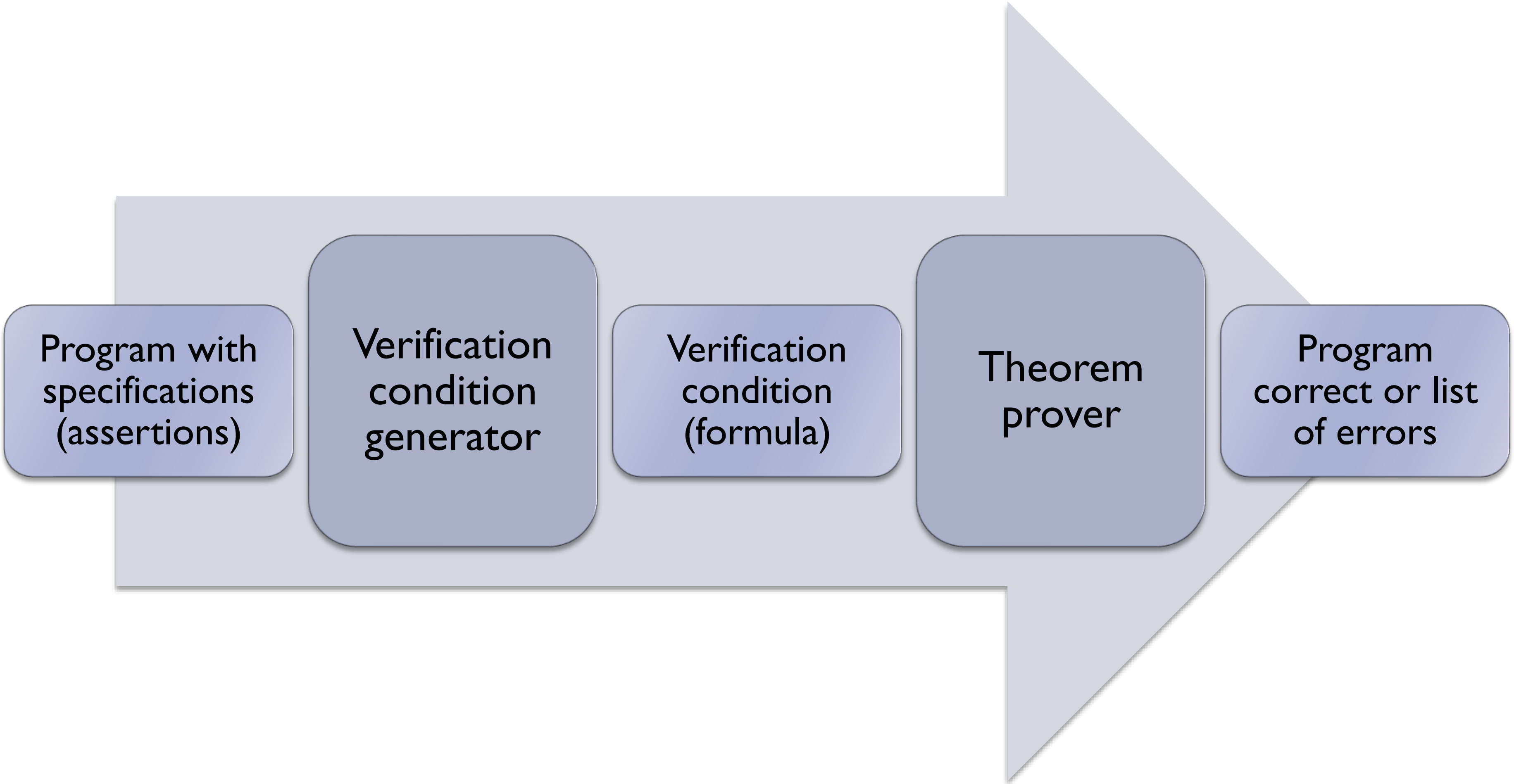
Brief History

- Turing, “Checking a Large Routine”, 1949.
  - We need proofs of programs
  - Mentions modularity
  - Early attempt at a general proof method
- Floyd, “Assigning Meaning to Programs”, 1967.
  - Workable proof method
- Hoare, “An Axiomatic Basis for Computer Programming”, 1969.
  - Further formalized
- Dijkstra, “A Discipline of Programming”, 1976.
  - Further formalized

Why Formal Verification?

- Static (or source code) analysis
  - Doesn’t execute code, no test cases
  - High coverage
    - Explores all possible paths through code
  - Finds more hard bugs
- Lower costs and turn-around time
- No silver bullet
  - Undecidable in general
    - Either misses bugs or returns false errors
  - Scalability and precision

Basic Verifier Architecture



Some Industry Success Stories

- Microsoft
  - SLAM – device drivers
  - Pex – automatic unit testing of .NET
  - Code Contracts – contracts for .NET
  - SAGE – whitebox fuzzing for security
- Facebook
  - Infer verifier
- Startups
  - Coverity, Polyspace, Fortify…
- Astree project in France
  - Used by Airbus
- Verified software efforts
  - NICTA's secure microkernel
  - Microsoft project Everest (verified https stack)

SAGE

- Finding security bugs using whitebox fuzzing
- Security bugs are expensive (MSR report)
  - Cost of each serious security bug: $Millions
  - Cost due to worms: $Billions
- Running on 100s machines 24/7
- Fuzzing 100s of applications
  - Media players, image processors, file decoders, document parsers…
- Finding 100s of security bugs
  - Saves tons of money/time/energy

SAGE cont.

“Every second Tuesday of every month, also known as "Patch Tuesday," Microsoft releases a list of security bulletins and associated security patches to be deployed on hundreds of millions of machines worldwide. Each security bulletin costs Microsoft and its users millions of dollars. If a monthly security update costs you $0.001 (one tenth of one cent) in just electricity or loss of productivity, then this number multiplied by a billion people is $1 million. Of course, if malware were spreading on your machine, possibly leaking some of your private data, then that might cost you much more than $0.001. This is why we strongly encourage you to apply those pesky security updates.”

Verification and Microbrewing ☺

- Deschutes Brewery uses SAGE-based software testing service to find bugs in their automation software:
  https://www.microsoft.com/en-us/research/video/osisoft-deschutes-brewery-used-project-springfield-full/

Summary

- Software has bugs
- Bugs can be very expensive
- Catch easy bugs with testing, etc.
- Use software verification techniques to catch hard bugs

- Understanding basics of software verification will be a requirement for future software engineers

Next Lecture

- Propositional logic
- SAT solvers