

## Lecture 1 Course Overview & Introduction

Zvonimir Rakamarić  
University of Utah

### Course Overview

- ▶ Instructor: Zvonimir Rakamarić
- ▶ Webpage:  
<http://www.zvonimir.info/teaching1/cs-6110-formal-methods-in-system-design-spring-2015/>
- ▶ 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

## Course Organization

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

## Course Communication

- ▶ **Email** is the king and **mailing list** the queen of communication related to this course
- ▶ Mailing list: [sv@list.eng.utah.edu](mailto:sv@list.eng.utah.edu)
  - ▶ Subscribe at:  
<https://sympa.eng.utah.edu/sympa/info/sv>
  - ▶ Everyone enrolled must subscribe; subscribe today
  - ▶ Questions related to homework, projects, general questions...
- ▶ Email: [zvonimir@cs.utah.edu](mailto:zvonimir@cs.utah.edu)
  - ▶ Questions related to your grade...

## Grading

- ▶ 40% homework assignments
  - ▶ 4-5 practical homework assignments
  - ▶ Each assignment is worth the same
- ▶ 60% course project
  - ▶ Project proposal (5 points)
  - ▶ Related work presentation (20 points)
  - ▶ Project update report (5 points)
  - ▶ Final presentation (20 points)
  - ▶ Final report (50 points)

## 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
  - ▶ I am not sure if this time around I will allow team work

## 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

## Late Policy

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

## Introduction to Software Verification

- ▶ Software is everywhere
  - ▶ Personal computers, mobile phones, in cars, ATMs, banks, planes, space shuttles, hospitals...
- ▶ Software has errors
  - ▶ Software systems are generally large, complex, and prone to errors...
  - ▶ And getting larger and more complex...
    - ▶ Multi-cores and many-cores
  - ▶ ...and more error prone!

## Worst Software Bugs (Wired, 2005)

[<http://www.wired.com/software/coolapps/news/2005/11/69355>]

- ▶ 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>]

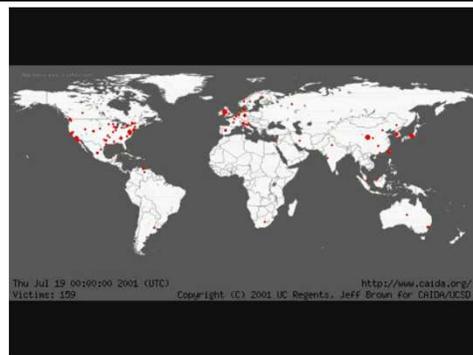
- ▶ 2001: 52,000 Jeeps recalled due to a software error that can shut down the instrument cluster.
- ▶ 2002: BMW recalls the 745i since the fuel pump would shut off if the tank was less than 1/3 full.
- ▶ 2003: A BMW trapped a Thai politician when the computer crashed. The door locks, windows, A/C and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



## Motivation

- ▶ Software errors are costly
  - ▶ US National Institute of Standards & Technology:  
Software errors cost the US economy alone an estimated \$60 billion each year
- ▶ Improving software quality and reliability is a major software engineering concern

## 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

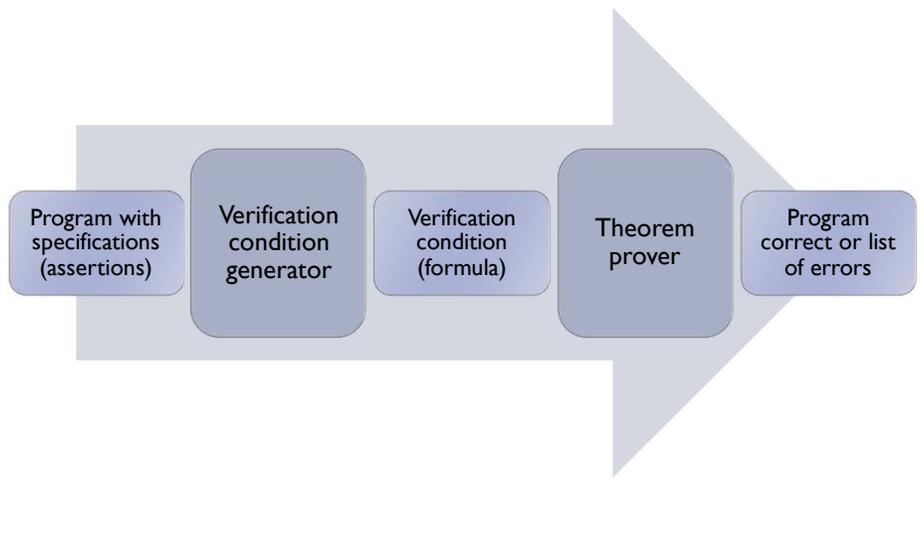
## Brief History

- ▶ 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
- ▶ Startups
  - ▶ Coverity, Polyspace, Fortify...
- ▶ Astree project in France
  - ▶ Used by Airbus
- ▶ Verified software efforts
  - ▶ NICTA's secure microkernel
  - ▶ Microsoft Hypervisor

## 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

## Summary

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

## Next Lecture

- ▶ Propositional logic
- ▶ SAT solvers