Lecture 1
Course Overview
Instructor: Hossein Hojjat

August 28, 2017
What is this course about?
<table>
<thead>
<tr>
<th>Year</th>
<th>Project</th>
<th>Lines of code</th>
</tr>
</thead>
<tbody>
<tr>
<td>∼1960s</td>
<td>Apollo 11 mission</td>
<td>145K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Goodman 2015]</td>
</tr>
<tr>
<td>∼1970s</td>
<td>Safeguard Program (US Army anti-ballistic</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td>missile system)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Lamb 1985]</td>
</tr>
<tr>
<td>∼1980s</td>
<td>IBM air traffic control systems</td>
<td>2M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Computerworld 1988]</td>
</tr>
<tr>
<td>∼1990s</td>
<td>Seawolf Submarine</td>
<td>3.6M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Kelly 1995]</td>
</tr>
<tr>
<td>∼1990s</td>
<td>Boeing 777</td>
<td>4M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Pehrson 1996]</td>
</tr>
<tr>
<td>System</td>
<td>LOC</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>Android</td>
<td>~12M</td>
<td></td>
</tr>
<tr>
<td>Philips Healthcare MRI scanner</td>
<td>~10M</td>
<td></td>
</tr>
<tr>
<td>Ford GT</td>
<td>~10M</td>
<td></td>
</tr>
<tr>
<td>Pacemaker Device</td>
<td>~100K</td>
<td></td>
</tr>
<tr>
<td>Appolo 11</td>
<td>145K</td>
<td></td>
</tr>
<tr>
<td>Safeguard</td>
<td>2M</td>
<td></td>
</tr>
<tr>
<td>Traffic Control</td>
<td>2M</td>
<td></td>
</tr>
<tr>
<td>Seawolf Submarine</td>
<td>3.6M</td>
<td></td>
</tr>
<tr>
<td>Boeing 777</td>
<td>4M</td>
<td></td>
</tr>
</tbody>
</table>
Android
~ 12M LOC
[Geoff Varrall 2012]

Philips Healthcare MRI scanner
~ 10M LOC
[Pierre Van de Laar, Teade Punter 2011]

Ford GT
~ 10M LOC
[Jamal Hameedi 2015]

Pacemaker Device
~ 100K LOC
[Dev Raheja 2015]
German Air Force can't fly at night after software bug blinds pilots

By James Billington
January 19, 2016 13:57 GMT

Nest Thermostat Glitch Leaves Users in the Cold

By NICK BILTON JAN. 13, 2016

The Nest Learning Thermostat is dead to me, literally. Last week, my once-beloved “smart” thermostat suffered from a mysterious software bug that drained its battery and sent our home into a chill in the middle of the night.

Setting the date to 1 January 1970 will brick your iPhone iPod touch

Date bug will prevent 64-bit iOS devices from booting up, rendering them inoperable even through full-safe restore methods

By SAMUEL GIBBS
Friday 12 February 2016 08:23 EST

Bug displays Chrome user's porn hours later on Apple computer

Student’s incognito mode browsing reappeared after closing private window when he loaded video game Diablo III

By STUART DREDGE
Thursday 14 January 2016 06:16 EST

Volvo recalls 59,000 cars over software fault

Sweden carmaker Volvo is recalling 59,000 cars across 40 markets over a flaw that can temporarily shut down the engine.

20 February 2016 | Europe

'Bug' Exposes Uber Driver's Tax Info, Including Name and Social Security Number

Thursday 13 October 2016 06:12 EDT

Forbes
http://onforb.com/iJFAnKy

TAXES | 02/09/16 @ 2:48PM | 2,403 views
Course Goals

- How can we define the precise meanings of programs?
- How can we prove theorems about the behavior of programs?
- How can we design programming tools to automate reasoning about programs?

Applications:
- Finding bugs
- Designing languages to prevent bugs
- Synthesizing correct-by-construction programs
- Manipulating programs automatically and correctly
  - e.g. refactoring, optimization
Course Goals

• How can we define the precise meanings of programs?
• How can we prove theorems about the behavior of programs?
• How can we design programming tools to automate reasoning about programs?

Applications:

• Finding bugs
• Designing languages to prevent bugs
• Synthesizing correct-by-construction programs
• Manipulating programs automatically and correctly
  • e.g. refactoring, optimization
Course Outline

**Functional Programming**
- Learn about lambda calculus and Scala
- Learn to make formal arguments about program behavior

**Type Theory**
- Learn how to design and reason about type systems
- Learn to use type-based analysis to avoid information leaks

**Axiomatic Semantics/Program Logics**
- A different view of program semantics
- Learn how to make logical arguments about program correctness
Abstract Interpretation

- Use abstraction to reason about the behavior of the program under all possible inputs

Model checking

- Learn how to reason exhaustively about program states
- Learn how abstraction and symbolic reasoning can help you find bugs in device drivers and protocol designs
Big Ideas

- Operational Semantics
  - Give programs meanings via stylized interpreters
- Program Proofs as Inductive Invariants
  - All induction, all the time!
- Abstraction
  - Model programs with specifications
- Modularity
  - Break programs into pieces to analyze separately
Tools

The Scala Programming Language
https://www.scala-lang.org/

The Coq Proof Assistant
https://coq.inria.fr/

SPIN Model Checker
http://spinroot.com/
Course Work

- 10% attendance & participation
- 60% 6 homework assignments (each ~ 10%)
- 30% final examination

- Assignments must be completed individually
  - Unless the assignment explicitly says that collaboration is possible
- Workload depends on planning well: Start early!
There is no required textbook

Materials for reading will be posted with lecture notes

**Suggested Book**

- “Types and Programming Languages”
  - Benjamin C. Pierce
- Covers some of the course material
  - (type systems and lambda calculus)
Course Staff

- **Instructor:** Hossein Hojjat ([https://www.cs.rit.edu/~hh/](https://www.cs.rit.edu/~hh/))
  - University of Tehran
    (Bs. Software Engineering 2001 - 2005)
  - University of Tehran & TU Eindhoven
    (Msc. Software Engineering 2005 - 2007)
  - EPFL Lausanne, Switzerland
    (PhD Computer Science 2008 - 2013)
  - Cornell University
    (Postdoctoral Researcher 2014 - 2016)

- **Email:** hh@cs.rit.edu
- **Office:** GOL(70)-3545
- **Class Hours:** MWF 1:25PM-2:20PM
- **Office Hours:** Tu 11am - 12am, Th 11am - 12am
  - Send email for alternative time

- **Webpage:**
  - [https://mycourses.rit.edu/](https://mycourses.rit.edu/)
  - [https://cs.rit.edu/~hh/teaching/plt17/](https://cs.rit.edu/~hh/teaching/plt17/)
Tell us about your background, how do you (usually) ensure that your programs are correct, story of a nasty bug that took you a while to debug! (if any)
Why functional programming?
Parallelism

- Moore’s law: Transistors of CPU doubles approximately every two years
- No longer true: Number of cores has been increasing recently

GPU programs can spawn millions of threads during execution

- Software has to take advantage of all the additional processors
- Programmers use sequential algorithms
Models

- Shared Memory with locking (mutex, semaphore,...)
- Message Passing (Actor model)
- Software transactional memory

None of the concurrent models is the ultimate solution

Fundamental problem: Non-determinism

Heisenbug: Bug that seems to disappear when attempting to study it
Concurrent Programming

Models
- Shared Memory with locking (mutex, semaphore, ...)
- Message Passing (Actor model)
- Software transactional memory

class Person(val name: String, 
              val age: Int)
class actor extends Actor {
def receive = {
case people: Set[Person] => 
  val (minors, adults) = 
  people partition (_.age < 18)
  Facebook ! minors
  LinkedIn ! adults
}
}
Concurrent Programming

Models
• Shared Memory with locking (mutex, semaphore,...)
• Message Passing (Actor model)
• Software transactional memory

class Person(val name: String,  
             val age: Int)

class actor extends Actor {
  def receive = {
    case people: Set[Person] =>
      val (minors, adults) = people partition (_.age < 18)
      Facebook ! minors
      LinkedIn ! adults
  }
}

• None of the concurrent models is the ultimate solution
• Fundamental problem: Non-determinism
• Heisenbug: Bug that seems to disappear when attempting to study it
Non-determinism

- Non-determinism: concurrent threads are accessing shared mutable state
- We can encapsulate state in actors or transactions, but the fundamental problem is the same

```plaintext
var x = 0;
thread {
  x = 1;
  x = x + 1;
}
thread {
  x = x * 2;
}
```

Value of x finally: 2, 3, 4

(assignments are atomic)

Non-determinism = parallel processing + mutable state
 Functional Programming

- To get deterministic processing, avoid the mutable state
- Avoid mutable state means programming functionally
- Rebirth of interest in functional programming triggered by multi-core hardware
Functional Programming

- To get deterministic processing, avoid the mutable state
- Avoid mutable state means programming functionally
- Rebirth of interest in functional programming triggered by multi-core hardware

- No mutable state: variables are immutable
- No assignment statement
- Functions are first-class values
- Functional program: collection of mathematical functions