CIS 500
Software Foundations

Spring 2012

Course Overview

What is "software foundations"?

Software foundations is the mathematical study of the **meaning** of programs.

The goal is finding ways to describe program behaviors that are both **precise** and **abstract**:

- **Precise** so that we can use mathematical tools to formalize and check interesting properties.
- **Abstract** so that properties of interest can be discussed clearly, without getting bogged down in low-level details.

Why study software foundations?

- To prove specific properties of particular programs (i.e., program verification)
  - Still fairly labor-intensive
  - But very important in some domains (safety-critical systems, hardware design, security protocols, inner loops of key algorithms, ...)
- To develop intuitions for *informal* reasoning about programs
- To prove general facts about all the programs in a given programming language (e.g., safety or isolation properties)
- To deeply understand language features (and their interactions) and develop principles for better language design

Questions this course could help answer

- "I'm designing a new web scripting language that's going to take over the world. How can I specify it so that different people implementing compilers will know how it is supposed to behave?"
- I'm writing a compiler and I'd like to add this optimization, but I'm not convinced it's always correct. How can I be sure?"
- "I want to write a program analysis tool that examines PHP-based web sites and checks for possible command injection attacks. How can I know I haven't missed any?"

What you'll get out of the course

- A more sophisticated perspective on programs, programming languages, and the activity of programming
  - How to view programs and whole languages as formal, mathematical objects
  - How to make and prove rigorous claims about them
  - Detailed study of a range of basic language features
- Powerful mathematical tools for software specification and analysis
  - Constructive logic
  - Inductive methods of definition and proof
  - Hoare logic
  - Type systems
- Expertise using a cutting-edge mechanical proof assistant

Syllabus

1. Foundations
   - 1.1 Functional programming
   - 1.2 Inductive definitions and proof techniques
   - 1.3 Constructive logic
   - 1.4 The Coq proof assistant
2. Reasoning about programs
   - 2.1 Specifying a simple imperative language
   - 2.2 Operational semantics
   - 2.3 Hoare Logic
   - 2.4 Information-flow security
3. Type systems
   - 3.1 Simply typed lambda-calculus
   - 3.2 Type safety
   - 3.3 Subtyping

What this course is not

- An introduction to programming (see CIT 591)
- A course on functional programming (though we'll be doing some functional programming along the way)
- A course on compilers (you should ideally already have basic concepts such as lexical analysis, parsing, abstract syntax, and scope under your belt)

Administrative Stuff

Personnel

Instructor:	Benjamin Pierce Levine 562
Teaching Assistants:	Loris D'Antoni Catalin Hritcu Mukund Raghothaman
Administrative Assistant:	Brittany Binler Levine 311

Information

Webpage:	`http://www.seas.upenn.edu/~cis500`
Textbook:	Software Foundations (see web page)
Collaboration tool:	Piazza (If you are not already signed up, please do so at `piazza.com`)

Exams

- Two in-class midterms
  - Wednesday, Feb 15
  - Wednesday, Mar 28
- Final exam
  - Thursday, May 3, 9-11AM

Clickers

This semester we'll be experimenting with using "clicker" technology to improve the interactivity of lectures.
By next week, please stop by the bookstore and buy yourself a "clicker" ($40, can be sold back at the end of the semester for $25)
Bring it to every lecture during the semester

Grading

Course grades will be computed as follows:

- Homework: 20%
- 2 midterms: 20% each
- Final: 40%

(Lack of) extra Credit

1. Grade improvements can *only* be obtained by sitting in on the course next year and turning in all homeworks and exams. (If you are doing this now to improve your grade from last year, please drop me an email so I know who you are.)
2. There will be no opportunity for extra credit projects, either during the semester or after the course ends. Concentrate your efforts on this course, now.

Collaboration

- Collaboration on homework is *strongly encouraged*
- Studying with other people is the best way to internalize the material
- Form study groups!
  - 2 people is the ideal size.
  - 3 is OK.
  - 4 is too many.

Homework

- Small part of your grade, but a large part of your understanding — impossible to perform well on exams without seriously grappling with the homework
- Submit one assignment per study group
- On written parts of homeworks, we will grade a semi-random subset of the problems on each assignment
- Late policy: Late homeworks lose 25% of their value for each day (or partial day) after the announced deadline

First Homework Assignment

- The first homework assignment is **due next Wednesday by noon**.
- You will need:
  - An account on a machine where Coq is installed (you can also install Coq on your own machine if you like)
  - Instructions on running Coq (available on the course web page)
  - The files `Preface.v` and `Basics.v` from the course web page

No Class Monday

Next Monday is Martin Luther King day.

The Coq Proof Assistant

What is a Proof?

- A proof is an indisputable argument for the truth of some mathematical assertion
- Proofs are ubiquitous in most branches of computer science
- However, unlike proofs in pure mathematics, many CS proofs are very long, shallow, and boring
- Can computers help?

What is a Proof Assistant?

Different ways of proving theorems with a computer:

- **Automatic theorem provers** find complete proofs on their own
  - Huge amount of work, beginning in the AI community in the 50s to 80s
  - In limited domains, extremely successful (e.g., hardware model checking)
  - In general, though, this approach is just too hard
- **Proof checkers** simply *verify* proofs that they are given
  - These proofs must be presented in an extremely detailed, low-level form
- **Proof assistants** are hybrid tools
  - "Hard steps" of proofs (the ones requiring deep insight) are provided by a human
  - "Easy parts" are filled in automatically

Some Proof Assistants

There are now a number of mature, sophisticated, and widely used proof assistants...

- Mizar (Poland)
- PVS (SRI)
- ACL2 (U. Texas)
- Isabelle (Cambridge / Munich)
- Twelf (CMU)
- Coq (INRIA)
- ...

The Coq Proof Assistant

- Developed at the INRIA research lab near Paris over the past 20 years
- Based on an extremely expressive logical foundation
  - The *Calculus of Inductive Constructions*
  - (hence the name!)
- Has been used to check a wide range of significant mathematical results
  - E.g., Gonthier's fully verified proof of the four-color theorem

Why Use Coq in This Course

- Rigor
  - using Coq forces us to be completely precise about the things we define and the claims we make about them
  - Coq's core notions of inductive definition and proof are a good match for the fundamental ways we define and reason about programming languages
- Interactivity
  - Instant feedback on homework
  - Easy to experiment with consequences of changing definitions, different reasoning techniques, etc.
- Useful background
  - Proof assistants are being used more and more widely in industry and academia
- Fun!
  - Coq is pretty addictive...