Course Overview

What is "software foundations"?

Software foundations (or "theory of programming languages") 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)
  - Important in some domains (safety-critical systems, hardware design, security protocols, inner loops of key algorithms, ...), but still quite difficult and expensive
- 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 understand language features (and their interactions) deeply and develop principles for better language design *(PL is the "materials science" of computer science...)*

What you can expect to 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
- Deep intuitions about key language properties such as type safety
- Powerful tools for language design, description, and analysis

Most software designers are language designers!

Approaches to Program Meaning

- *Denotational semantics* and *domain theory* view programs as simple mathematical objects, abstracting away their flow of control and concentrating on their input-output behavior.
- *Program logics* such as *Hoare logic* and *dependent type theories* focus on logical rules for reasoning about programs.
- *Operational semantics* describes program behaviors by means of *abstract machines*. This approach is somewhat lower-level than the others, but is extremely flexible.
- *Process calculi* focus on the communication and synchronization behaviors of complex concurrent systems.
- *Type systems* describe *approximations* of program behaviors, concentrating on the shapes of the values passed between different parts of the program.