CS 565: Programming Languages

August 16, 2017

Instructor

Lecturer
Prof. Suresh Jagannathan
Room 3154J
Lawson Computer Science Building
Ph: x4-0971
suresh@cs.purdue.edu

Office Hours:
Tu, Th. 4-5 PM

Course Overview

This course will examine the design and implementation of programming languages from a rigorous, foundational, and principled perspective. Our goal will be to develop and use tools that enable formal reasoning about the design and specification of new language features, that allow precise understanding of the rationale for existing features in modern languages, that give us insight on how design decisions can impact implementations, and that equip us with mechanisms to prove important properties about language features and their implementations.

This course will *not* be a survey of existing languages or a taxonomy of language constructs. Neither will it be a course on programming *per se*. Instead, we will explore ways to understand programming languages generally that help us to answer questions such as the following:

1. What is a program *specification* and what role do specifications play in program construction and reliability?
2. What does *program verification* mean?
3. What are sensible and tractable notions of program correctness? What are the conditions under which we can assert that a program is "safe"?
4. How do we qualify the "expressive power" of a language feature? How do we relate different features found in different languages?
5. How do we understand notions of data abstraction and types found in modern programming languages? Can we formalize the relationship between types and computation?
6. How do we rigorously express the meaning of program structures, and *prove* useful properties about them?
7. How do we reason about the equivalence of programs, or programs and their compiled translation?
8. What tools can we bring to bear to help automate correctness proofs?

To help answer these questions, the course is designed around several interleaved themes: (1) the role of logic and related mathematical formalisms in programming language specification and design; (2) formal reasoning devices that precisely explain the meaning of programming language features and program executions; (3) the use of types to define and specify safety conditions on program executions, and to enrich language expressivity; (4) formalizing notions of correctness and devising mechanisms to verify that programs are correct with respect to their specification; (4) the use of automated proof assistants to help validate important theorems that describe useful properties based on the structure of (2) and (3), using techniques enabled by (1).

At the end of this class, students should be comfortable with modern notions of language semantics and specification, have developed sufficient mathematical maturity to formally prove interesting properties about a language features, type systems, implementations, or complex computation structures generally, and be capable of understanding current research trends in the field.

The course will be divided into roughly four parts:

1. The first part of the course will introduce students to mechanized proof assistants and functional programming. We will use Coq, an expressive and powerful programming language and proof assistant that enables interactive construction of formal constructive proofs.
2. The second part of the course will introduce student to program logics, inductive reasoning principles, the relationship between proofs and programming, and related proof methods.
3. The third part of the course will provide an overview of different ways of capturing a language's semantics, discuss different styles of specifying program behavior, and formalize notions of program equivalence.
4. The fourth part of the course will consider a number of case studies. There will be particular emphasis on the typed lambda-calculus, a core language that can be used to study a variety of existing language features and designs. Time permitting, we will examine recent languages like Rust and Go, as well as some more esoteric language (and language feature) proposals.

Prerequistes

It is assumed that students taking this class would have had some exposure to an undergraduate programming languages class (equivalent to CS 465), and are comfortable with basic mathematical concepts (e.g., basic logic, graph and set theory).

Programming Assignments

There will be two classes of assignments. Collaboration is allowed for both. *Exercises* will be given every week. These are sample problems given in the text that are meant to reinforce the material discussed in class. While you are required to hand in your solutions, they will not be graded for correctness, only completion. *Problem Sets* will be given every few weeks. A problem set will typically consist of a single problem, with multiple sub-problems that must be solved to realize a complete solution. As a result, problem sets will be somewhat more challenging. Problem sets will be graded for correctness. Both exercises and problem sets will take the form of programming exercises in Coq and will entail developing mechanized proofs whose correctness is automically checked.

Grading

Grading for this class will be based as follows:

Exercises	15%
Programming Sets	25%
Midterm	20%
Final	40%

The final examination will also serve as the Ph.D qualifying examination for this subject.

Texts

The main text for the class is **Software Foundations**:

`https://softwarefoundations.cis.upenn.edu/current/index.html`

The text uses *literate programming* - the entire text is an executable Coq program with concepts explained in comments, and concretely instantiated as definitions, theorems, and proofs that can be processed by Coq.

Besides the main text, there are a number of other texts that cover material discussed in the class. These include:

1. **Certified Programming with Dependent Types** by Adam Chilpala (MIT Press, 2013).
2. **Types and Programming Languages**, by Benjamin C. Pierce (MIT Press, 2002).
3. **Practical Foundations of Programming Languages**, by Robert Harper (Cambridge University Press, 2013)
4. **Formal Semantics**, Glynn Winskel, MIT Press 1993.

Resources

There are a number of tutorials and detailed documentation available on Coq:

1. The Coq homepage (`http://coq.inria.fr`) provides an explanation of Coq syntax, its commands, and its extensive collection of supported tactics.
2. Adam Chilpala has a draft text on formal reasoning for programming that explores a number of verification methods such as model checking and abstract interpretation (see `http://adam.chlipala.net/frap`)
3. A reference text on Coq (**Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions**) by Bertot and Casternan, Springer-Verlag, 2004 describes the calculus that underlies Coq's proof engine.

Web Pages and Newsgroups

Copies of the lectures and handouts will be available through the course web page:

`http://www.cs.purdue.edu/homes/suresh/565-Fall2017/index.html`.

The course will leverage piazza (`http://piazza.com`), an interactive Q&A newsgroup messaging system that allows students, faculty, and TAs to post questions (and answers) to problems and issues related to the course.

Tentative Syllabus

The course will roughly follow the breakdown given below.

Subject	*Specific Topics*
Foundations	Introduction Functional Programming Polymorphism and Higher-Order Programming Dependent Types Propositions, Evidence, and Relations
Semantics	Imperative Programs Program Equivalence Hoare Logic and Axiomatic Semantics Operational Semantics
Types	Lambda Calculus Typing Judgments, Contexts, and Relations Record Subtyping State References
Case Studies	Recent language (and feature) proposals

These topics are subject to change based on class interest, pace, and time availability. Time permitting, we may discuss other advanced topics including logical relations, abstract interpretation, concurrency, theorem proving, etc.