CS 565: Programming Languages

Handout 1 August 10, 2011

Instructor

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

Office Hours:
Tu, Th 2-3 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 reasoning about both 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 fundamental properties. This course will *not* be a survey of existing languages or language features. 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 are sensible and tractable notions of program correctness?
3. How do we qualify the "expressive power" of a language feature?
4. How do we qualify notions of abstraction and types found in modern programming languages? Can we formalize the relationship between types and computation?
5. How do we rigorously express the meaning of program structures, and *prove* useful properties about them?
6. What tools can we bring to bear to help automate such proofs?

To help answer these questions, the course is designed around several interleaved themes: (1) the role of logic and proofs in programming language design; (2) formal reasoning devices (aka operational and axiomatic semantics) 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) 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, 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. We will use Coq, an expressive and powerful 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 semantic models (both axiomatic and operational), will introduce different notions of interpretation and evaluation, discuss program specification techniques, and formalize notions of program equivalence.
4. The fourth part of the course will focus on types, with special emphasis on the typed lambda-calculus, a core language within which different type systems (e.g., simple types, subtyping, polymorphism) can be precisely studied.

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., graph and set theory).

Programming Assignments

There will be assignments given every week. These assignments will take the form of programming exercises in Coq; most of these exercises will entail developing mechanized proofs whose correctness is automically checked by Coq. Thus, These exercises will require students to define various concepts, and prove important theorems about the topics discussed during the lecture, using Coq. Students are encouraged to work together on these assignments.

Project

There will be a semester-long project that will involve using Coq to define a formal system of a computing artifact (e.g., a type system, a concurrency abstraction, a security policy, etc.), develop theorems and proofs about existing systems, explore program analysis definitions and correctness, etc. Students can work teams or individually. A proposal indicating the project topic, justification, and methodology must be submitted and approved by Oct. 1. Projects will be due at the end of the semester.

Grading

Grading for this class will be based as follows:

Programming Assignments	30%
Project	40%
Final	30%

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

Texts

The course will follow Pierce's Software Foundations course notes (available in `.html` format from the course web-page.

Useful supplemental material can also be found in **Types and Programming Languages**, by Benjamin C. Pierce (MIT Press 2002). Some material will also be drawn from **Principles of Program Analysis**, by Nielson, Nielson, and Hankin (Springer, 2005).

In addition, students might also find the following texts useful:

- **Formal Semantics**, Glynn Winskel, MIT Press 1993.
- **Essentials of Programming Languages**, Daniel Frideman, Mitchell Wand, and Christopher Haynes, MIT Press 2001.
- **Foundations of Programming Languages**, John Mitchell, MIT Press, 1996.
- **The Structure of Typed Programming Languages**, David Schmidt, MIT Press, 1994.

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 proof engineering using Coq (see: `http://adam.chlipala.net/cpdt/html/toc.h`
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-Fall2011/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
Program Semantics	Imperative Programs Program Equivalence Hoare Logic and Axiomatic Semantics Operational Semantics
Types	Simply-Typed Lambda Calculus Typing Judgments, Contexts, and Relations Record Subtyping
More on Types	Existential Types Recursive Types System F

Time permitting, we will also discuss other advanced topics including contracts, hybrid typechecking, linear types, regions, automated theorem proving, types for security, and concurrency. The additional material covered will depend upon class interest.