CS 565: Programming Languages

Handout 1

August 20, 2010

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
M, W 10 - 11AM, Room 3162, Dr. Nicholas Kidd

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 characteristic properties. This course will not be a survey of existing languages or language features, nor will it emphasize language implementations or analyses. 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 useful and foundational models of program execution?
3. What are sensible and tractable notions of program correctness and safety?
4. How do we precisely characterize the "expressive power" of a language feature?
5. How do we qualify and compare notions of 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. 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 can be used to precisely explain the meaning of programming language features; (3) the use of types to define and specify safety conditions on program executions, and to enrich language expressivity;
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 specific language feature, 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 the typed lambda-calculus, a core language within which different type systems (e.g., simple types, subtyping, polymorphism) can be precisely studied.

Prerequisites

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

Each lecture will be accompanied by a number of programming exercises in Coq. 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.

Grading

Grading for this class will be based as follows:

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Assignments</td>
<td>65%</td>
</tr>
<tr>
<td>Final</td>
<td>35%</td>
</tr>
</tbody>
</table>

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), and also available at: http://cis.upenn.edu/bcpierce/sf.

Useful supplemental material can also be found in **Types and Programming Languages**, by Benjamin C. Pierce (MIT Press 2002).

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


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 Chlipala has a draft text on proof engineering using Coq; see: http://adam.chlipala.net/cpdt/html/toc.html


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-Fall2010/index.html.

There is a CS mailing list: cs565-fall2010@cs.purdue.edu that we’ll use to broadcast important messages (e.g., errata) about the lectures, homeworks, and programming assignments. You should add yourself to this list using the command: mailer add me to cs565-fall2010.

There is also a web blog at http://cs565fall2010.wordpress.com that can be used as a general question and discussion forum site. Students are free to post responses to any questions posed.
# Tentative Syllabus

The course will roughly follow the breakdown given below.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Specific Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coq Foundations</td>
<td>Introduction</td>
</tr>
<tr>
<td></td>
<td>Functional Programming</td>
</tr>
<tr>
<td></td>
<td>Polymorphism and Higher-Order Programming</td>
</tr>
<tr>
<td></td>
<td>Propositions, Evidence, and Relations</td>
</tr>
<tr>
<td>Program Semantics</td>
<td>Untyped Lambda-Calculus</td>
</tr>
<tr>
<td></td>
<td>Imperative Programs</td>
</tr>
<tr>
<td></td>
<td>Program Equivalence</td>
</tr>
<tr>
<td></td>
<td>An Introduction to Hoare Logic</td>
</tr>
<tr>
<td></td>
<td>Deriving Verification Conditions</td>
</tr>
<tr>
<td></td>
<td>Operational Semantics</td>
</tr>
<tr>
<td>Types</td>
<td>Simply-Typed Lambda Calculus</td>
</tr>
<tr>
<td></td>
<td>Typing Judgments, Contexts, and Relations</td>
</tr>
<tr>
<td></td>
<td>Record Subtyping</td>
</tr>
<tr>
<td></td>
<td>Object-Oriented Programming</td>
</tr>
<tr>
<td></td>
<td>Strong Normalization</td>
</tr>
</tbody>
</table>

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.