

# CSE 505: Concepts of Programming Languages

Dan Grossman

Fall 2006

Lecture 1— Course Introduction

# Today

---

- Administrative stuff
- Course motivation and goals
  - A Java example
- Course overview
- Course pitfalls
- Caml tutorial, part 1
  - Advice: play with it between now and Tuesday (e.g., hw1, problem 1)

## Course facts

---

- Dan Grossman, CSE556, djg
- TA: Anna Gringauze, CSE362, annagrinn
- Office hours: (Tuesday 2-3 plus appt plus stop by...)
- Web page for mailing list, “homework 0”, and homework 1 (fairly carefully pipelined with first lectures; do not wait to do it all)

# Coursework

---

- 5 homeworks
  - “paper/pencil” (L<sup>A</sup>T<sub>E</sub>X recommended?)
  - programming (Caml required)
  - where you’ll probably learn the most
  - do extra credit if you *want* but not technically “extra”
- 2 exams
  - “my” reference sheet plus “your” reference sheet
- Lecture notes usually available online
- Textbook: mostly for “middle of course”
  - won’t follow it much

# Academic integrity

---

- If you violate the rules, I will enforce the maximum penalty allowed
  - and I'll be personally offended
  - far more important than your grade
- Rough guidelines
  - can sketch idea together
  - cannot look at code solutions
- Ask questions and always describe what you did
- Please *do* work together and learn from each other...

# Graduate-School Success

---

- Success in 505 (a graduate course) comes from:
  - Learning and enjoying the material
  - Challenging yourself
  - Managing the “big picture” and the details
- Success has nothing to do with:
  - Scrounging for grading points
  - “Doing better than the person next to you”
- The person next to you is your colleague for the next 5–50 years.

# Programming-language concepts

---

Focus on *semantic* concepts:

What do programs mean (do/compute/produce/represent)?

How to define a language *precisely*?

English is a poor *metalanguage*

Aspects of meaning:

equivalence, termination, determinism, type, ...

## Does it matter?

---

Novices write programs that “work as expected,” so why be rigorous/precise/pedantic?

- The world runs on software

Web-servers and nuclear reactors don't “seem to work”

- You buy language implementations—what do they do?
- Software is buggy—semantics assigns blame
- Never say “nobody would write that”

Also: Rigor is a hallmark of quality research

## Java example

---

```
class A { int f() { return 0; } }
class B {
    int g(A x) {
        try { return x.f(); }
        finally { s }
    }
}
```

For all  $s$ , is it equivalent for  $g$ 's body to be "return 0;"?

Motivation: code optimizer, code maintainer, ...

# Punch-line

---

Not equivalent:

- Extend A
- `x` could be `null`
- `s` could modify global state, *diverge*, throw, ...
- `s` could return

A silly example, but:

- PL makes you a good adversary, programmer
- PL gives you the tools to argue equivalence (hard!)

# Course goals

---

1. Learn intellectual tools for describing program behavior
2. Investigate concepts essential to most languages
  - mutation and iteration
  - scope and functions
  - objects
  - threads
3. Write programs to “connect theory with the code”
4. Sketch applicability to “real” languages
5. Provide background for current PL research  
(less important for most of you)

# Course nongoals

---

- Study syntax; learn to specify grammars, parsers
  - Transforming  $3 + 4$  or  $(+ 3 4)$  or  $+(3, 4)$  to “application of plus operator to constants three and four”
  - stop me when I get too sloppy
- Learn specific programming languages (but some ML)
- Denotational and axiomatic semantics
  - Would include them if I had 25 weeks
  - Will explain what they are later

## What we will do

---

- Define really small languages
  - Usually Turing complete
  - Always unsuitable for real programming
- Study them rigorously via *operational models*
- Extend them to realistic languages less rigorously
- Digress for cool results (this is fun!?!)
- Do programming assignments in Caml...

# Caml

---

- Caml is an awesome, high-level language
- We will use a tiny core subset of it that is well-suited for manipulating recursive data structures (like programs!)
- You have to learn it outside of class (start today!)
  - But feel free to ask me for advice
  - Even after the course
- Resources on course webpage
- I am not a language zealot, but knowing ML makes you a better programmer

# Pitfalls

---

How to hate this course and get the wrong idea:

- Forget that we made simple models to focus on essentials
- Don't quite get inductive definitions and proofs
- Don't try other ways to model/prove the idea
  - You'll probably be wrong
  - And therefore you'll learn more
- Think PL people focus on only obvious facts (need to start there)

# Final Metacomment

---

Acknowledging others is crucial...

This course will draw heavily on:

- Previous versions of the course (Borning, Chambers)
- Similar courses elsewhere (Felleisen, Flatt, Harper, Morrisett, Myers, Pierce, Rugina, Walker, ...)
- Texts (Pierce, Wynskel, ...)

This is a course, not my work.

# Caml tutorial

---

- “Let go of Java/C”
- If you have seen SML, Haskell, Scheme, Lisp, etc. this will feel more familiar
- Give us some small code snippets so we have a common experience we can talk about.
- Also see me use the tools.
- A note later on Seminal.

## Seminal

---

- This is optional, but Ben Lerner and I would be ever-so grateful for your informed feedback.
- An additional, complementary style of type-error message.
- No other change to compiler (parsing, code-generation, etc.)
- See “Running Caml locally” on the course website.