CS152: Programming Languages

Lecture 1 — Course Introduction

Dan Grossman
Spring 2011

Today

- Administrative stuff
- Introducing myself
    - Expanded version because I'm a visitor
- Course motivation and goals
    - A Java example
- Course overview
    - Expanded version because you're shopping
- Course pitfalls
- Start on Caml tutorial (most of Thursday)
    - Advice: start playing with it soon to learn and/or remember (e.g., hw1, problem 1)

Course facts

- Dan Grossman, Maxwell Dworkin 233, grossman@seas.harvard.edu
- TF: Paul Govereau, Maxwell Dworkin 309, govereau@cs.harvard.edu
- Office hours to-be-determined (see web page)
  - Also encouraged to make appointments with me or even just stop by
- Web page for:
  - "homework 0"
  - homework 1, fairly carefully pipelined with first lectures
    - Do not wait to do it all

Coursework

- 6 homework assignments [almost surely]
  - "Paper/pencil" (LaTeX recommended?)
  - Programming (Caml required)
  - Where you'll probably learn the most
  - Do challenge problems if you *want* but not technically "extra"

- One "introduction/summary" to a published research paper
  - More details in a few weeks; high work/length ratio

- 2 exams
  - my reference sheet plus your reference sheet; samples provided

- No textbook
  - But several books on reserve (see web page and ask)
  - Will post slides and will *try* to write lecture notes
  - Lecture notes from CS152 Spring 2010 may prove useful
    - 80%+ same material, but somewhat different order/style

Academic integrity

- Don't cheat in my class
  - I'll be personally offended
  - Being honest is 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

Logistical Advice

- Take notes:
  - Slides posted, but they are enough to teach from not to learn from
  - Will often work through examples by hand
- Arrive on time:
  - Unlike many CS people, I start and end punctually (10:07–11:30)
  - Missing the first N minutes is so much less efficient than missing the last N minutes
  - I *know* you can get here on time (cf. exam days)

Talking about myself

I'm a "visiting faculty member" just for this semester

- Normally at the University of Washington in Seattle
- This should *not* scare you away from taking this course
- Let me compensate for you not being able to look up my evaluations or ask your friends about me...

What will this guy be like?

- Last year's CS152 is a reasonable approximation
- I've taught this material [mostly to graduate students] 8 times
  - Planning about 15% new stuff to keep things fresh/improving and because the term is longer
- I love teaching and I love the material in this course
  - Hopefully "Lecture 1" is the most boring one?
  - Most professors don't teach while on sabbatical

Student Evaluations

Evaluations from last time I taught a similar course (Fall 2009)

	Excellent	Very Good	Good	Fair	Poor	Very Poor
Course as a whole	62%	29%	8%	0%	0%	0%
Course content	50%	33%	17%	0%	0%	0%
Instructor's contribution	83%	12%	4%	0%	0%	0%
Instructor's effectiveness	79%	12%	8%	0%	0%	0%
Instructor's interest	75%	12%	12%	0%	0%	0%
Amount learned	54%	17%	25%	4%	0%	0%
Grading techniques	42%	42%	12%	4%	0%	0%

More about me

Saving you a Google search:

- http://www.cs.washington.edu/homes/djg
- http://www.facebook.com/profile.php?id=10717335

Professional life story:

- St. Louis suburbs → Rice → Cornell → UW
  - UW universally “top-10” CS and arguably #5
  - But try to convince my grandma
  - Seriously, if looking at grad school, we should talk
- Programming languages from theory to practice
  - Morrisett was my Ph.D. advisor; Chong was an office-mate
  - I’m here to refresh, collaborate, learn, and teach — and have fun

Other: Ice hockey, cycling and running, non-fiction, my nephew, beer, ...

What could go wrong?

So this is sort of like “study abroad” for the professor instead of the students

- Please don’t get too upset when I mess up the jargon, but correct me
  - TF, semester, concentration, ...
- Different logistics than I’m used to
  - web page, grades, photocopier, ...
  - will probably all settle down after this week
- Help me if you see me lost on campus :-)

More importantly, we may have to work together on the pace

- But based on last year’s CS152, I think we’ll be fine

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, . . .

This course does *not* gives superficial exposure to N weird PLs

- More like CS121 than CS51, but not really like either
- But it will help you learn new languages via foundations

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
- Real languages have many features: building them from well-understood foundations is good engineering
- Never say "nobody would write that" (surprising interactions)

Is this Really about PL?

Building a rigorous and precise model is a hallmark of deep understanding.

The value of a model is in its:

- Fidelity
- Convenience for establishing (proving) properties
- Revealing alternatives and design decisions
- Ability to communicate ideas concisely

Why we mostly do it for programming languages:

- Elegant things we all use
- Remarkably complicated (need rigor)

I believe this “theory” makes you a better computer scientist

- Focus on the model-building, not just the PL features

APIs

Like almost anything in computing, we can describe the course in terms of designing an API.

Many APIs have 1000s of functions with simple inputs

- Kernel calls take a struct or two and return an int

A typical language implementation more or less has just

- $\boldsymbol{typecheck : program \rightarrow bool}$
- $\boldsymbol{compile : program \rightarrow (string \rightarrow value)}$

But defining $\boldsymbol{program}$ and these functions is subtle, hard

- Conversely, "a data structure is just a really dumb PL"
- Every extensible system ends up defining a PL (game engines, editors, web browsers, CAD tools, ...)

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
   - types
   - 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 $\mathbf{3 + 4}$ or $\mathbf{(+\ 3\ 4)}$ or $\mathbf{+(3, 4)}$ to "application of plus operator to constants three and four"
  - Stop me when I get too sloppy
- Learn specific programming languages (but some ML)

What we will do

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

Plenty of Theory

Hard to give a taste of what the "theory" will look like, but here is some cut-and-paste from topics we will cover in the next few weeks

Lectures 3–5

$$\frac{H \; ; \; e \Downarrow c}{H \; ; \; x := e \rightarrow H, x \mapsto c \; ; \; \mathsf{skip}}$$

$$\frac{H \; ; \; e \Downarrow c \qquad c > 0}{H \; ; \; \mathsf{if} \; e \; s_1 \; s_2 \rightarrow H \; ; \; s_1} \qquad \frac{H \; ; \; e \Downarrow c \qquad c \leq 0}{H \; ; \; \mathsf{if} \; e \; s_1 \; s_2 \rightarrow H \; ; \; s_2}$$

Lectures 7–10

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.\; e : \tau_1 \rightarrow \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \rightarrow \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 \; e_2 : \tau_1}$$

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 mostly have to learn it outside of class
  - Don't procrastinate
  - Don't hesitate to ask questions
- 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 the essence
- Don't quite get inductive definitions and proofs when introduced
- 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 draws heavily on pedagogic ideas from at least: Chambers, Chong, Felleisen, Flatt, Fluet, Harper, Morrisett, Myers, Pierce, Rugina, Walker

And material covered in texts from Pierce, Wynskel, and others

(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
- If you have seen Caml, focus here on "how I say things" and what subset will be most useful to us in studying PL
- Give us some small code snippets so we have a common experience we can talk about
- Also see me use the tools