CS 312 Lecture 1
Course overview

Andrew Myers
Cornell University Computer Science
Spring 2007

What this course is about

Helping you become expert software system designers and programmers

1) Programming paradigms

Programming language concepts and constructs

2) Reasoning about programs

- Correctness
- Performance
- Designing for reasoning

3) Tools

Data structures and algorithms

Course staff

- Prof. Andrew Myers
- Two TAs:
  - Xin Zheng
  - Olga Belomestnykh
- Consultants:
  - Tyler Steele
  - Ben Weber
  - Edward McTighe
  - Kareem Amin
  - Bob Albright
  - Paul Lewellen
  - Andrew Owens
- Office, consulting hours posted on web
- One hour of consulting Sun-Wed evening
- TAs, instructor have office hours: use them!

3

Course meetings

- Lectures Tues, Thurs: Thurston 203
- Recitations Monday, Wednesday
  - Olin Hall 245, at 2:30pm
  - Olin Hall 245, at 3:35pm
  - Possible third section
- New material is presented in lecture **and** recitation
- Attendance is expected at lecture **and** recitation
- Participation counts

4

Course web site

http://www.cs.cornell.edu/courses/cs312

- Announcements
- Lecture notes
- Assignments
- Course software
- ML documentation
- Other resources

5

Course newsgroup

cornell.class.cs312

- A great place to ask questions!
- A great place to see if your question has already been asked
- A place to discuss course ideas
  - But don't solve assignments for other people

6

Readings

- Course material in lecture notes on website
  - But also responsible for in-class material…
- Some other useful texts:
  - *Elements of ML Programming*, Ullman
  - *ML for the working programmer*, Paulson
  - *Programming in Standard ML*, Harper (on-line)
  - *Notes on Programming in SML*, Pucella (on-line)
  - *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*. Liskov, Guttag.
    - Material on abstraction and specification, but in Java

7

Assignments

- 6 problem sets
  - PS1 assigned today: “SML Warmup”
- Mix of programming, written problems
- Submitted electronically via CMS
- Three single-person assignments (1-3)
- Three two-person assignments (4-6)

8

Exams

- Exams test material from lectures, written problems, assume you have done assignments
- Prelim 1: March 8
- Prelim 2: April 17
- Final exam May 14, 9-11:30AM
- Makeup exams must be scheduled within the first two weeks of class
  - Check your schedule and let the instructor know

9

Academic integrity

- Strictly and carefully enforced
- Please don't make us waste time on this
- Start assignments early and get help from course staff!

10

What this course is about

Goal: help you develop as expert programmers and system designers

1) Programming paradigms

Programming language concepts and constructs

2) Reasoning about programs

- Correctness
- Performance
- Designing for reasoning

3) Tools

Data structures and algorithms

11

Why do you need to know this?

- Science and craft of programming
- You'll acquire skills that will help you become better programmers
  - 10x difference in productivity, fun, …
- Needed in many upper level courses
- Needed for any serious programming task
- Needed for managing programming projects

12

1) Programming Paradigms

- Functional programming
- Polymorphism
- Pattern matching
- Modular programming
- Concurrent programming
- Type inference
- Garbage collection

- We'll use ML to convey these concepts
  - The important part are the concepts, not the ML syntax!

13

2) Programming Techniques

- *Design and reasoning: critical to robust, trustworthy software systems*.
- Design and planning:
  - Modular programming
  - Data abstraction
  - Specifications, interfaces
- Reasoning about programs
  - Program execution models
  - Reasoning about program correctness
  - Reasoning about performance via asymptotic complexity
  - Using induction to reason about program behavior
- Testing

14

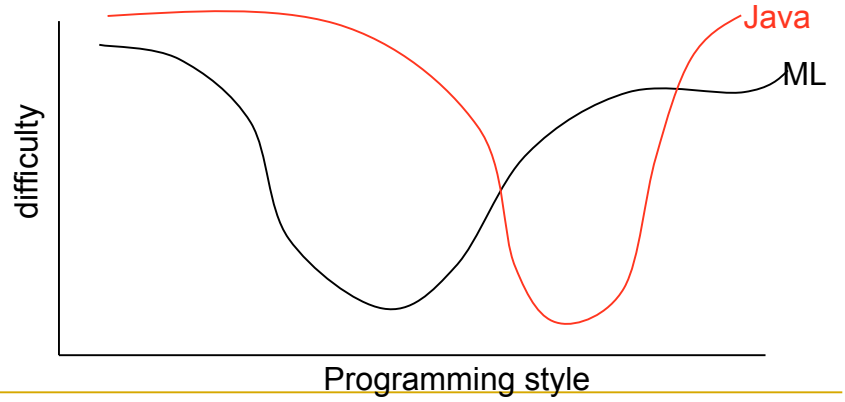
3) Data Structures & Algorithms

- Standard structures: lists, trees, stacks, graphs, etc.
  - Functional versions of these structures
- Advanced structures:
  - Balanced trees: AVL, Red-Black, B-trees, splay trees
  - Hash tables
  - Binary heaps
- Algorithms on these data structures

15

Language and programming style

- Sapir-Whorf hypothesis: language influences how we think
  - In CS: language influences how we design software



16

Imperative style

- Program uses **commands** (a.k.a **statements**) that *do* things to the **state** of the system:
  - x = x + 1;
  - p.next = p.next.next;
- Functions/methods can have **side effects**
  - int wheels(Vehicle v) { v.size++; return v.numw; }
- **Problem:** Difficult to reason about how state changes during program execution
  - Intertwined state across module boundaries
  - Complex object graphs

17

Functional style

- **Idea:** program without side effects
  - Effect of a function abstraction is *only* to return a result value
- Program is an **expression** that **evaluates** to produce a **value** (e.g., 4)
  - E.g., 2+2
  - Works like mathematical expressions
- Allows **equational reasoning** to show programs work:
  - if x = y, replacing y with x has no effect:
  - let val x = f(0) in x+x vs. f(0) + f(0)
- A good match to staged computation
- Information has tree-like structure (no cycles)

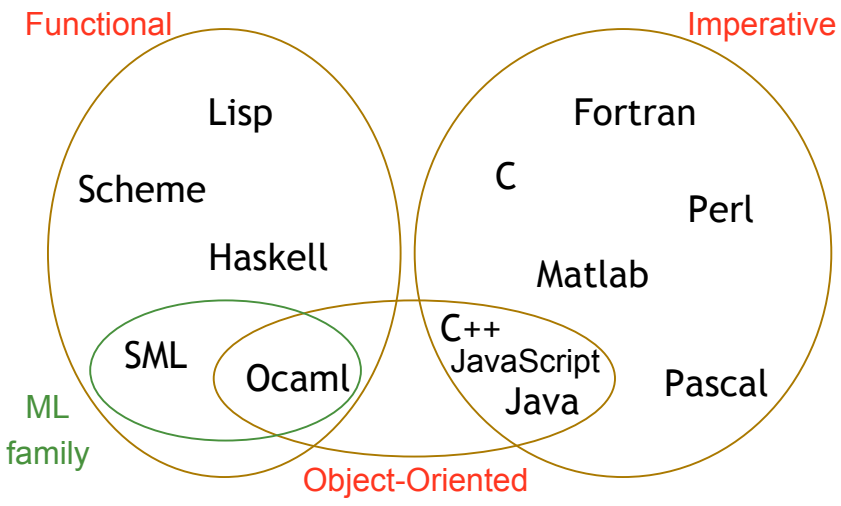
18

Imperative vs. functional

- ML: a *functional* programming language
  - Encourages building code out of functions
  - Like mathematical functions; f(x) always gives the same result
- Functional style usable in ML, Java, C, …
  - No side effects: easier to reason about what happens
  - Equational reasoning

19

Programming Languages Map



20

Imperative vs. functional

- Functional languages:
  - Higher level of abstraction
  - Closer to specification
  - Easier to develop robust software
- Imperative languages:
  - Lower level of abstraction
  - Sometimes more efficient
  - More difficult to maintain, debug
  - More error-prone

21

Example 1: Sum

```
y = 0;
for (x = 1; x <= n; x++) {
  y = y + x*x;
}
```

22

Example 1: Sum

```
int sum(int n) {
    y = 0;
    for (x = 1; x <= n; x++) {
        y += x*x;
    }
    return n;
}
fun sum(n: int): int =
    if n=0 then 0
        else n*n + sum(n-1)
```

23

Example 2: Reverse

```
List reverse(List x) {
  List y = null;
  while (x != null) {
    List t = x.next;
    x.next = y;
    y = x;
    x = t;
  }
  return y;
}
```

24

Example 2: Reverse

```
fun reverse(l : int list) : int list =
  case l of
     [] => []
   | h :: t => reverse(t) @ [h]
```


Why ML?

- ML is not used much in industry. But:
- ML embodies important ideas much better than Java, C++
  - These ideas have Java, C++ manifestations
- Learning a very different language will give you more flexibility down the road
  - New languages are constantly emerging: Java and C++ will be obsolete soon
  - Principles and concepts beat syntax
  - Ideas in ML will probably be in next gen languages
- Cred among the right people!

Rough schedule

- Introduction to functional programming (5)
- Specs and modular programming (4)
- Reasoning about programs (4)
- ***Prelim 1***
- Data structure case studies (2)
- ***Spring break***
- Language semantics and implementation (4)
- ***Prelim 2***
- Advanced data structures (4)
- Concurrency and event-driven programming (3)
- ***Final exam***

27

Announcements

- Problem set 1 released today
  - Due January 31, at 11:59pm
  - Posted on the course web site and CMS
- Consulting starts today
- **Help session:** getting started with SML + Emacs: Thursday, Upson B7, 7pm
- Send mail to Xin (xz83) if you do not have CMS access for 312

28