Fundamentals of Programming Languages

Evan Chang

Meeting 1: Welcome
CSCI 5535, Spring 2010
http://www.cs.colorado.edu/~bec/courses/csci5535-s10/

Introductions
- Who am I?
- About you?
  - What do you want to get out of this class?

Administrivia
- Website
  http://www.cs.colorado.edu/~bec/courses/csci5535-s10/
  - readings, slides, assignments, etc.
- Moodle
  - discussion forums, assignment submission
- Office hours
  - T 1-2, R 4:45-5:45?
  - and by appointment
  - ECOT 621 and on Gchat/Skype (see moodle)

Today
- Some historical context
- Goals for this course
- Requirements and grading
- Course summary
- Convince you that PL is useful

Meta-Level Information
- Please interrupt at any time!
- It's completely ok to say:
  - I don't understand. Please say it another way.
  - Slow down! Wait, I want to read that!
- Discussion, not lecture

"Isn't PL a solved problem?"
- PL is an old field within Computer Science
- 1920's: "computer" = "person"
- 1936: Church's Lambda Calculus (= PL)
- 1937: Shannon's digital circuit design
- 1940's: first digital computers
- 1950's: FORTRAN (= PL)
- 1958: LISP (= PL)
- 1960's: Unix
- 1972: C Programming Language
- 1981: TCP/IP
- 1985: Microsoft Windows
New and Better Compilers?

A Dismal View of PL Research

Programming Languages

- Touches most other areas of CS
  - Theory: DFAs, TMs, language theory (e.g., LALR)
  - Systems: system calls, memory management
  - Arch: compiler targets, optimizations, stack frames
  - Numerics: FORTRAN, IEEE FP, Matlab
  - AI: theorem proving, search
  - DB: SQL, transactions
  - Networking: packet filters, protocols
  - Graphics: OpenGL, LaTeX, PostScript
  - Security: buffer overruns, NET, bytecode, PCC, ...
  - Computational Biology: pathway models
  - Software Engineering: software quality, development tools
  - Human Computer Interaction: development tools
- Both theory (math) and practice (engineering)

Overarching Theme

- I assert (and shall convince you) that
  - PL is one of the most vibrant and active areas of CS research today
    - It is both theoretical and practical
    - It intersects most other CS areas
  - You will be able to use PL techniques in your own projects

Goals

Goal 1

Learn to use advanced PL techniques
No Useless Memorization

• I will not waste your time with useless memorization
• This course will cover complex subjects
• I will teach their details to help you understand them the first time
• But you will never have to memorize anything low-level
• Rather, learn to apply broad concepts

Goal 2

When (not if) you design a language, it will avoid the mistakes of the past, and you will be able to describe it formally

Discussion: Language Design

• Languages are adopted to fill a void
  - Enable a previously difficult/impossible application
  - Orthogonal to language design quality (almost)
• Training is the dominant adoption cost
  - Languages with many users are replaced rarely
  - But easy to start in a new niche. Examples:

Why so many languages?

• Many languages were created for specific applications
• Application domains have distinctive (and conflicting) needs
  - which leads to a proliferation of languages.
• Examples:
  - Artificial intelligence: symbolic computation (Lisp, Prolog)
  - Scientific Computing: high performance (Fortran)
  - Business: report generation (COBOL)
  - Systems Programming: low-level access (C)
  - Scripting (Perl, Python, TCL)
  - Distributed systems: mobile computation (Java)
  - Special purpose languages: …
Language Paradigms

Loose classification of languages.
- Imperative
  - Fortran, Algol, Cobol, C, Pascal
- Functional
  - Lisp, Scheme, ML, Haskell
- Object oriented
  - Smalltalk, Eiffel, Self, C++, Java, C#, Javascript
- Logic
  - Prolog
- Concurrent
  - CSP, dialects of the above languages
- Special purpose
  - TEX, Postscript, TrueType, sh, HTML, make

What makes a good language?

- No universally accepted metrics for design
- "A good language is one people use"?

What are good language features?

- Simplicity (syntax and semantics)
- Readability
- Safety
- Support for programming large systems
- Efficiency (of execution and compilation)
Designing good languages is hard

- Goals almost always conflict.
- Examples:
  - Safety checks cost something in either compilation or execution time.
  - Type systems restrict programming style in exchange for strong guarantees.

Story: The Clash of Two Features

- Real story about bad programming language design
- Cast includes famous scientists
- ML (82) functional language with polymorphism and monomorphic references (i.e., pointers)
- Standard ML (85) innovates by adding polymorphic references
- It took 10 years to fix the "innovation"

Polymorphism (Informal)

- Code that works uniformly on various types of data
- Examples of function signatures:
  - length: α list → int (takes an argument of type "list of α", returns an integer, for any type α)
  - head: α list → α
- Type inference:
  - generalize all elements of the input type that are not used by the computation

References in Standard ML

- Like "updatable pointers" in C
- Type constructor: τ ref
  - x: int ref "x is a pointer to an integer"
- Expressions:
  - ref: τ → τ ref (allocate a cell to store a τ, like malloc)
  - !e: when e: τ ref (read through a pointer, like *e)
  - e := e': when e: τ ref and e': τ (write through a pointer, like *e = e')
- Works just as you might expect

Polymorphic References: A Major Pain

Consider the following program fragment:

<table>
<thead>
<tr>
<th>Code</th>
<th>Type inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>fun id(x) = x</td>
<td>id: α → α (for any α)</td>
</tr>
<tr>
<td>val c = ref id</td>
<td>c: (α → α) ref (for any α)</td>
</tr>
<tr>
<td>fun inc(x) = x + 1</td>
<td>inc: int → int</td>
</tr>
<tr>
<td>c := inc</td>
<td>Ok, since c: (int → int) ref</td>
</tr>
<tr>
<td>(lc) (true)</td>
<td>Ok, since c: (bool → bool) ref</td>
</tr>
</tbody>
</table>

Reconciling Polymorphism and References

- Type system fails to prevent a type error!
- Commonly accepted solution today:
  - value restriction: generalize only the type of values!
    - easy to use, simple proof of soundness
  - many "failed fixes"
- To see what went wrong we need to understand semantics, type systems, polymorphism and references
Story: Java Bytecode Subroutines

• Java bytecode programs contain subroutines (jsr) that run in the caller's stack frame (why?)
• jsr complicates the formal semantics of bytecodes
  - Several verifier bugs were in code implementing jsr
  - 30% of typing rules, 50% of soundness proof due to jsr
• It is not worth it:
  - In 650K lines of Java code, 230 subroutines, saving 2427 bytes, or 0.02%
  - 13 times more space could be saved by renaming the language back to Oak

Recall Goal 2

When (not if) you design a language, it will avoid the mistakes of the past, and you will be able to describe it formally

Goal 3

Understand current PL research (POPL, PLDI, OOPSLA, TOPLAS, ...)

Most Important Goal

Have Lots of Fun!

Prerequisites

• "Programming experience"
  - exposure to various language constructs and their meaning (e.g., CSCI 3155)
  - ideal: undergraduate compilers (e.g., CSCI 4555)
• "Mathematical maturity"
  - we'll use formal notation to describe the meaning of programs
• If you are an undergraduate or from another department, please see me.
Assignments

• Reading and participation (each meeting)
• Weekly homework (for half semester)
• Take-home midterm exam
• Final project

Reading and Participation

• ~2 papers/book chapter, each meeting
  - Spark class discussion, post/bring questions

• Online discussion forum
  - Post ≥1 substantive comment, question, or answer for each lecture
  - On moodle.cs.colorado.edu
  - Due before the next meeting
  - Distance students participate more online!

What is “substantive”?

• May be less than a blog post but more than a tweet.
• Some examples:
  - Questions
  - Thoughtful answers
  - Clarification of some point
  - What you think is the main point in the reading set.
  - An idea of how some work could be improved
  - Comments on a related web resource related
• Intent: take a moment to reflect on the day's reading/discussion (not to go scour the web)

Homework and Exam

• Homework/Problem Sets
  - You have one week to do each one
  - First half of the semester only
  - Some material will be "mathy"
  - Collaborate with peers (but acknowledge!)

• Take-Home Midterm Exam
  - Like a longer homework

Final Project

• Options:
  - Research project
  - Literature survey
  - Implementation project
• Write a ~5-8 page paper (conference-like)
• Give a ~15-20 minute presentation
• On a topic of your choice
  - Ideal: integrate PL with your research
• Pair projects (indiv/3-person possible)

Course Summary
Course At-A-Glance

- Part I: Language Specification
  - Semantics = Describing programs
  - Evaluation strategies, imperative languages
- Part II: Language Design
  - Types = Classifying programs
  - Typed $\lambda$-calculus, functional languages
- Part III: Applications

Core Topics

- Semantics
  - Operational semantics
    - rules for execution on an abstract machine
    - useful for implementing a compiler or interpreter
  - Axiomatic semantics
    - logical rules for reasoning about the behavior of a program
    - useful for proving program correctness
  - Abstract interpretation
    - application: program analysis
- Types
  - $\lambda$-calculus
    - tiny language to study core issues in isolation

Possible Special Topics

- Software model checking
- Object-oriented languages
- Types for low-level languages
- Types for resource management
- Shape analysis

What do you want to hear about?

First Topic: Model Checking

- Verify properties or find bugs in software
- Take an important program (e.g., a device driver)
- Merge it with a property (e.g., no deadlocks)
- Transform the result into a boolean program
- Use a model checker to exhaustively explore the resulting state space
  - Result 1: program provably satisfies property
  - Result 2: program violates property "right here on line 92,376"

For Next Time

- Join the course moodle and introduce yourself (forum discussion for today)
  - Write a few sentences on why you are taking this course
- Read the two articles on SLAM
  - see the website under "Schedule"