Computation & Interaction
Precisely expressed by PL
Dependence means need for analysis

- Safety
- Security
- Performance

Goal: Techniques for PL/Program Analysis

1. Semantics: What does a program mean/do?
   - Intellectual tools for reasoning about behavior
   - Algorithmic tools for analyzing program behavior

2. Design: Analyzing features and constructs
   - Precisely modeling PL constructs
   - Understanding how features interact

3. Applications: Intro to current PL research
   - Safe low-level languages for systems programming
   - Checking cryptographic protocols
   - Generating worm signatures
   - Finding bugs in Systems Code
Mechanics

www.cs.ucsd.edu/classes/wi10/cse230/

Nothing printed, everything on Web

Meetings:
• Lectures: Ranjit Jhala
• Tu-Th 11:00-12:30pm @ CSE 2154

Lecture Rescheduling

No lectures on Jan 12, 14, 19, 21

Make-up Lectures:
Fri, Jan 8: 12:30 – 3:00pm
Mon, Jan 25: 12:30 – 3:00pm
Location TBD

Material

Outline:
1. Operational Semantics (3 lectures)
2. Axiomatic Semantics (3 lectures)
3. $\lambda$-calculus (3 lectures)
4. Type Systems (4 lectures)
5. Abstract Interpretation (2 lectures)
6. Projects (Recent Work) (2 lectures)

Recommended Texts (Slides, Papers on Web)
• {1,2} Winskel: “The Formal Semantics of PL”
• {3,4} Pierce: “Types and PL”
• {5,6} Papers

Material

General Plan:
1. Define a tiny PL
2. Isolate a few features
3. Rigorously analyze aspects of PL
   - Key properties
   - How to reason about them
4. Implement PL constructs, analyses
5. Project: Pick a topic, study in depth
Requirements and Grading

1. [40%] HW assignments (4-6):
   - Part theoretical (i.e. “pencil-paper”)
   - Part programming - learn Ocaml
   - Done in pairs

2. [30%] Project...

3. [30%] Final (take home)

Prerequisites

• Interest in the material

• Basic discrete math
  - Notation to describe programs, meaning
  - Set theory, logic
  - Induction

• Programming experience
  - Exposure to languages, paradigms, constructs
  - Bonus points if you know ML

• If you have concerns, talk to me

Project (done in pairs)

• Survey (read, present some recent PL research)
  or,

• Research (work on a concrete problem)

• Either: done in pairs

Timeline

• Feb 5: 1/2 page proposal due
• Feb 19: First midway meeting
• Mar 5: Second midway meeting
• Mar 11/13: 20 minute talk (No Report)

Goal: Techniques for PL/Program Analysis

1. Semantics: What does a program mean/do?
   - Intellectual tools for reasoning about behavior
   - Algorithmic tools for analyzing program behavior

2. Design: Analyzing features and constructs
   - Precisely modeling PL constructs
   - Understanding how features interact

3. Applications: Intro to current PL research
   - Safe low-level languages for systems programming
   - Programming and Concurrency
   - Checking cryptographic protocols
   - Automatically Testing Code
   - Securing AJAX Web Applications
Topics: Semantics

- Three pedigreed approaches:
  - **Operational** semantics
    - rules for execution on an abstract machine
    - useful for implementing a compiler or interpreter
  - **Axiomatic** semantics
    - represent program behavior using logical predicates
    - useful for proving program correctness
  - **Denotational** semantics
    - meaning = function from programs to elements of a domain

- Why isn’t semantics used on a mass scale?

Who needs semantics?

- Those who want unambiguous descriptions of:
  - Languages
  - Language features

Who needs semantics?

- Heavyweight and not (yet) cost-effective
  - For everyday (and everyone’s) use
  - Notation is sometimes dense

- Semantics is general
  - For all possible inputs x,
    The output is y and the state changes …
  - Most programmers want:
    What output for my 5½ test inputs?

Explosion of Languages Due to Web

“On the internet, nobody knows you’re a dog...”

...a Ruby Program
...a Scala Program
...an Erlang Program

"On the Internet, nobody knows you're a dog."
Who needs semantics?

- Those who want **unambiguous** descriptions of:
  - Languages
  - Language features
- For writing programs must **work for all inputs:**
  - Compilers and interpreters
  - Program Transformation and Instrumentation tools
  - Software Engineering tools
  - Programs dealing with complex data
  - Critical software (e.g. for nuclear reactors)
- Basis for **reasoning** about programs
- Basis for formal arguments in **PL research**

Topics: Design

- Languages are adopted for many reasons
  - Cost of training programmers
  - Similarity to old languages helps ...
  - Libraries
  - Documentation
  - What itch is scratched? PLs become entrenched
  - Orthogonal to language design quality
- Why bother? PL inside **every** extensible system:
  - Emacs: E-Lisp
  - Word, Powerpoint: VBScript
  - SQL, Renderman, LaTeX, XML, make, ...

Good Language Properties

- Simplicity (syntax and semantics)
- Readability
- Safety
- Support for programming large systems
- Efficiency (of execution and compilation)

Note: “**Goodness**” is not something we will study

Language Properties Conflict

- Dynamic safety checks
  - Cost compilation and/or execution time
- **Static types give strong guarantees**
  - But restrict programming style
- PL research = How to gain without pain?
  - More expressive type systems
  - Modern compilers for modern languages
- MSR Singularity: OS written in **C#**
  - Static checking makes protection rings redundant
  - Speeds up execution (!)
Story: The clash of two features

- Real story about bad PL design
  - By “sound” PL academics

**ML** ('82) functional language:
- polymorphism & monomorphic pointers

**Standard ML** ('85):
- innovates with polymorphic pointers

- It took 10 years to fix the “innovation”

Polymorphism (informal)

- Code that works uniformly on various types of data

- Examples:
  - `length : α list → int` (for any type `α`, takes arg of type list of `α`, returns `int`)
  - `head : α list → α` (for any type `α`, takes arg of type list of `α`, returns `α`)

- Type inference:
  - generalize elements of input type that are not used
  - instantiation: if `e : τ` then `e : [τ'/α]τ` (substitute `τ'` for `α` in `τ`)

References in Standard ML

Like “updatable pointers” in C

- Type constructor: `ptr τ`

- Expressions:
  - `alloc : τ → ptr τ` (allocate a cell to store a `τ`)
  - `*e : τ` when `e : ptr τ` (read through a pointer)
  - `*e := e'` with `e : ptr τ` and `e' : τ` (write through a pointer)

- Works just as you might expect

Polymorphic References: A Major Pain

- The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

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

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

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

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!

The type system fails to prevent a run-time type error!
Language Design Lessons

• Good language design is hard
  - Simplicity is rare in practice
  - Never by accident

• Real PLs = isolated points in huge design space

• PL research considers tiny languages (e.g., λ-calc)
  - Separate and study core issues in isolation
  - In practice, must pay attention to whole language

• Key is finding the right definitions
  - Greatly simplifies subsequent analysis

Topics: Applications

• Techniques from PL research will crop up in a variety of places

• Sample applications:
  - Safely executing untrusted code
  - Checking properties of systems code
  - Designing concurrent languages
  - Checking Crypto Protocols
  - Building a “formally verified” compiler, OS,…

Proof-Carrying Code

• Problem:
  - It is difficult to understand low-level, optimized code
  - We need to understand mobile-code to protect ourselves

• Idea:
  - Establish a safety policy that must be enforced
  - Ask the producer to prove that the code obeys the policy
  - Use a simple proof checker to validate the proof
  - Then run the code at full speed

• Better than:
  - Safe (and slow) interpreter
  - Complex JIT that inserts run-time checks

Proof-Carrying Code (PCC)

- untrusted client
- application code
- Certifying compiler
- automated support in a familiar tool
- host
- optimized native code
- proof
- small & tamper-proof
- formal safety policy
- Flexible, customizable safety policies
- safety with maximum performance
- CPU
- simple, small, & fast
Checking vs. Generating a Proof

Definition: A maze is “safe” if there is a path through it!

PCC Builds on Semantics and Type Theory

- PCC ensures safety indirectly
  - If exists a proof matching code then code is safe
  - Soundness of PCC: needs tools from semantics

- PCC proofs represented w/ “dependent” types
  - proof is correct iff it is well-typed
  - i.e. proof-checking = type-checking
  - Soundness of proof-checking: tools from type theory

How to generate proofs?

“An attempt to re-acquire an acquired lock or release a released lock will cause a deadlock.”

Safety Policy:
Calls to lock and unlock must alternate.

How to prove property?
Infinite configurations inputs!

- Axiomatic Semantics
- Abstract Interpretation
Designing Concurrent PLs

- MapReduce: [Old-school Functional PLs]
- Dryad/LINQ: Generalizes MapReduce
  - Based on Language Integrated Queries which allow .Net code to query a DB based on ideas from Functional PLs
- Software Transactions: [Haskell]
- Erlang, Clojure, Scala, X10, Fortress, ...

Formally Verified Compilers

[X. Leroy 06 POPL 2006]
- Formalize semantics of C code
- Formalize semantics of PowerPC
- Compiler: C -> PowerPC

Theorem: For all programs P,
Semantics_C(P) = Semantics_{ppc}(Compiler(P))

Proof is checked by machine (tiny TCB)

Formally Verified OS Kernels

[Klein et. al. best paper at SOSP 2009]
- Formalize semantics of C and Haskell
- Describe ideal OS Kernel [Haskell program]
- Describe real OS Kernel [C Program]

Theorem: For all programs P,
Semantics_{Haskell}(Ideal) = Semantics_C(Real)

Proof is checked by machine (tiny TCB)

That’s the sales pitch

- For Thu read papers posted on Web
- First HW assignment is up (due 21st)
- We’ll begin at the beginning ...
  ...with a tiny imperative language