# CS 4110

# Programming Languages & Logics

# Lecture 1 Course Overview

24 <del>26</del> August <del>2014</del> 2016

# Programming Languages

One of the oldest fields in Computer Science	
• $\lambda$ -calculus – Church	(1936)
• FORTRAN – Backus	(1957)
• LISP – McCarthy	(1958)
<ul> <li>ALGOL 60 – Backus, Naur, Perlis, &amp; others</li> </ul>	(1960)
• Pascal – Wirth	(1970)
• C – Ritchie	(1972)
<ul> <li>Smalltalk – Kay &amp; others</li> </ul>	(1972)
<ul> <li>ML – Milner and others</li> </ul>	(1978)
• C++ – Stroustrup	(1982)
<ul> <li>Haskell – Hudak, Peyton Jones, Wadler, &amp; others</li> </ul>	(1989)
• Java – Gosling	(1995)
• C# – Microsoft	(2001)
• Scala – Odersky	(2003)
• F# – Syme	(2005)

# Programming Languages

...and one of the most vibrant areas today!

PL intersects with many other areas of computing

### Current trends

- Domain-specific languages
- Static analysis and types
- Language-based security
- Verification and model checking
- Concurrency

Both theoretically and practically "meaty"

Syllabus

### Course Staff

#### Instructor

Nate Foster Adrian Sampson Office: Gates 432 411A Hours: Mon 4-5pm and Friday 11am 12pm TBA

Teaching Assistant Fran Mota Office: Hours: TBA

#### Web Page

http://www.cs.cornell.edu/Courses/cs4110/<del>2014fa</del> 2016fa

#### Discussion

```
http://www.piazza.com
```

# Course Goals

- Techniques for modeling programs\* mathematically
  - Operational, axiomatic, and denotational semantics
  - Examples with advanced features
  - Reasoning principles (induction, co-induction)
- Explore applications of these techniques
  - Optimization
  - Type systems
  - Verification
- Gain experience implementing languages
  - Interpreters
  - Program transformations
  - Analysis tools
- PhD students: cover material for PL qualifying exam
- Have fun :-)

\*and whole languages!

### Prerequisites

### Mathematical Maturity

- Much of this class will involve formal reasoning
- Set theory, formal proofs, induction
- Most challenging topic: denotational semantics

### Programming Experience

- Comfortable using a functional language
- For undergrads: CS 3110 or equivalent

Interest (having fun is a goal! :-)

If you don't meet these prerequisites, get in touch

Programming Languages and Logics MWF 9:05-9:55 Gates G01

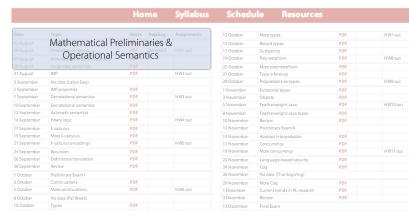


Cornell University Department of Computer Science

	н		me	Syllabus	Schedule Resource				
Date	Topic	Notes	Reading	Assignments	12 October	More types	PDF	HW7 out	
22 August	Introduction	PDF	Winskel 1		15 October	Record types	PDF		
24 August	Small-step semantics	PDF	Winskel 2	HW1 out	17 October	Subtyping	PDF		
27 August	Inductive definitions and proofs	PDF			19 October	Polymorphism	PDF	HW8 out	
29 August	Large-step semantics	PDF			25 October	More polymorphism	PDF		
31 August	IMP	PDF		HW2 out	27 October	Type inference	PDF		
3 September	No class (Labor Day)				29 October	Propositions-as-types	PDF	HW9 out	
5 September	IMP properties	PDF			1 November	Existential types	PDF		
7 September	Denotational semantics	PDF		HW3 out	3 November	Objects	PDF		
10 September	Denotational semantics	PDF			5 November	Featherweight Java	PDF	HW10 out	
12 September	Axiomatic semantics	PDF			8 November	Featherweight Java types	PDF		
14 September	Hoare logic	PDF		HW4 out	10 November	Review	PDF		
17 September	λ-calculus	PDF			12 November	Preliminary Exam II			
19 September	More $\lambda$ -calculus	PDF			15 November	Abstract interpretation	PDF		
21 September	λ-calculus encodings	PDF		HWS out	17 November	Concurrency	PDF		
24 September	Recursion	PDF			19 November	More concurrency	PDF	HW11 out	
26 September	Definitional translation	PDF			22 November	Language-based security	PDF		
28 September	Review	PDF			24 November	Cog	PDF		
1 October	Preliminary Exam I			26 November	No dass (Thanksgiving)				
3 October	Continuations	PDF			29 November	More Cog	PDF		
5 October	More continuations	PDF		HW6 out	1 December	Current trends in PL research	PDF		
8 October	No class (Fall Break)				3 December	Review	PDF		
10 October	Types	PDF			13 December	Final Exam			

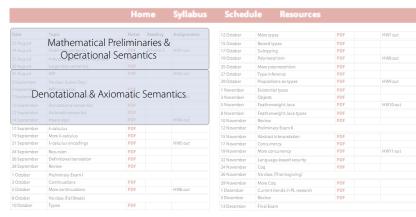
Your schedule with probably look something like this

Programming Languages and Logics MWF 9:05-9:55 Gates G01



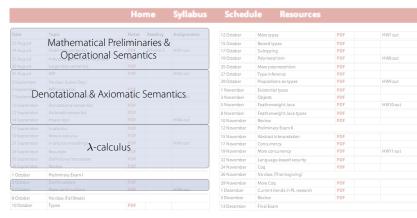


Programming Languages and Logics MWF 9:05-9:55 Gates G01



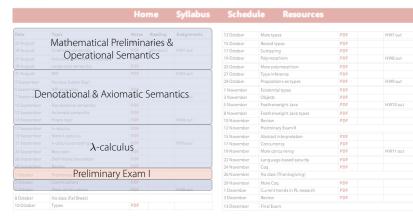


Programming Languages and Logics MWF 9:05-9:55 Gates G01



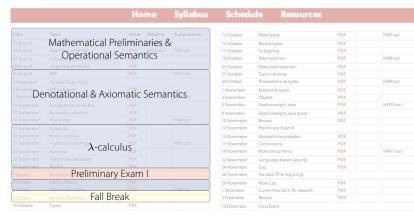


Programming Languages and Logics MWF 9:05-9:55 Gates G01



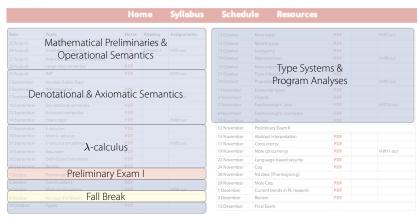


Programming Languages and Logics MWF 9:05-9:55 Gates G01





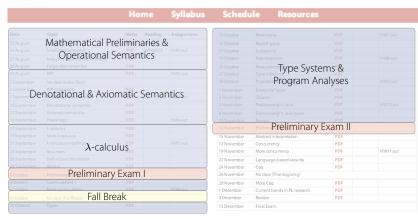
Programming Languages and Logics MWF 9:05-9:55 Gates G01



Department of

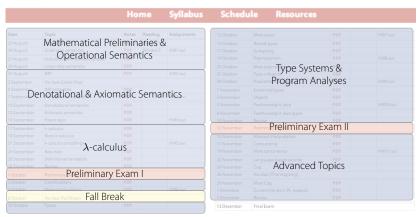
Computer Science

Programming Languages and Logics MWF 9:05-9:55 Gates G01



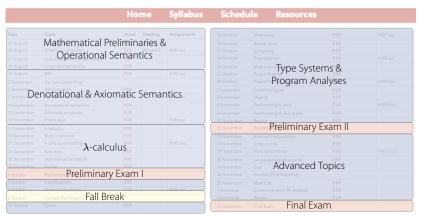


Programming Languages and Logics MWF 9:05-9:55 Gates G01





Programming Languages and Logics MWF 9:05-9:55





Department of Computer Science

Programming Languages and Logics MWF 9:05-9:55 Gates G01





# Course Work

### Participation (5%)

- Lectures
- Office hours
- Online discussions
- Homework (40%)
- 10 assignments, roughly one per week
- Mix of theory and practice
- Can work with one partner
- No late submissions
- Two slip days and lowest score discarded

Preliminary Exams (15% each)

- October 6th Not scheduled by the registrar-
- November 14th probably in-class

Final Exam (25%)

• Date TBD

### Some simple requests:

- 1. You are here as members of an academic community. Conduct yourself with integrity.
- 2. Problem sets must be completed with your partner, and only your partner. You must *not* consult other students, alums, friends, Google, GitHub, StackExchange, Course Hero, etc.!
- 3. If you aren't sure what is allowed and what isn't, please ask.

- I will provide reasonable accommodations to students with documented disabilities (e.g., physical, learning, psychiatric, vision, hearing, or systemic).
- If you are experiencing undue personal or academic stress at any time during the semester (or if you notice that a fellow student is), contact me, Engineering Advising, or Gannett.

# Language Specification

# Language Specification

Formal Semantics: what do programs mean?

Three Approaches

- Operational
  - Models program by its execution on abstract machine
  - Useful for implementing compilers and interpreters
- Axiomatic
  - Models program by the logical formulas it obeys
  - Useful for proving program correctness
- Denotational
  - Models program literally as mathematical objects
  - Useful for theoretical foundations

# Language Specification

Formal Semantics: what do programs mean?

Three Approaches

- Operational
  - Models program by its execution on abstract machine
  - Useful for implementing compilers and interpreters
- Axiomatic
  - Models program by the logical formulas it obeys
  - Useful for proving program correctness
- Denotational
  - Models program literally as mathematical objects
  - Useful for theoretical foundations

Question: few languages have a formal semantics. Why?

# Formal Semantics

### Too Hard?

- Modeling a real-world language is hard
- Notation can gets very dense
- Sometimes requires developing new mathematics
- Not yet cost-effective for everyday use

### Overly General?

- Explains the behavior of a program on *every* input
- Most programmers are content knowing the behavior of their program on *this* input (or these inputs)

Okay, so who needs semantics?

## A Tricky Example

Question #1: is the following Java program legal?

Question #2: if yes, what does it do?

class A { static int a = B.b + 1; }

class B { static int b = A.a + 1; }

# Who Needs Semantics?

### Unambiguous Description

- Anyone who wants to design a new feature
- Basis for most formal arguments
- Standard tool in PL research

### Exhaustive Reasoning

- Sometimes have to know behavior on all inputs
- Compilers and interpreters
- Static analysis tools
- Program transformation tools
- Critical software

# Language Design

# Design Desiderata

Question: What makes a good programming language?

# Design Desiderata

Question: What makes a good programming language?

One answer: "a good language is one people use"

Question: What makes a good programming language?

One answer: "a good language is one people use"

Wrong! Are COBOL and JavaScript the best languages?

Question: What makes a good programming language?

One answer: "a good language is one people use"

Wrong! Are COBOL and JavaScript the best languages?

Some good features:

- Simplicity (clean, orthogonal constructs)
- Readability (elegant syntax)
- Safety (guarantees that programs won't "go wrong")
- Support for programming in the large (modularity)
- Efficiency (good execution model and tools)

Unfortunately these goals almost always conflict.

- Types provide strong guarantees but restrict expressiveness.
- Safety checks eliminate errors but have a cost—either at compile time or run time.
- Some verification tools are so complicated, you essentially need a PhD to use them!

Unfortunately these goals almost always conflict.

- Types provide strong guarantees but restrict expressiveness.
- Safety checks eliminate errors but have a cost—either at compile time or run time.
- Some verification tools are so complicated, you essentially need a PhD to use them!

A lot of research in programming languages is about discovering ways to gain without (too much) pain.

### Story: Unexpected Interactions

A real story illustrating the perils of language design

Cast of characters includes famous computer scientists

### Timeline:

- 1982: ML is a functional language with type inference, polymorphism (generics), and monomorphic references (pointers)
- 1985: Standard ML innovates by adding polymorphic references
   → unsoundness
- 1995: The "innovation" fixed

Polymorphism: allows code to be used at different types

Examples:

- List.length :  $\forall \alpha. \ \alpha \text{ list} \rightarrow \text{int}$
- List.hd :  $\forall \alpha. \ \alpha \text{ list} \rightarrow \alpha$

Type Inference:  $e \rightsquigarrow \tau$ 

- e.g., let *id*  $(x) = x \rightsquigarrow \forall \alpha. \ \alpha \to \alpha$
- Generalize types not constrainted by the program
- Instantiate types at use *id* (true) → bool

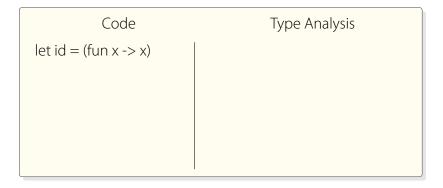
By default, values in ML are immutable.

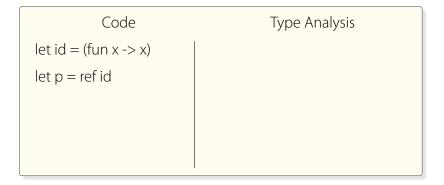
But we can easily extend the language with imperative features.

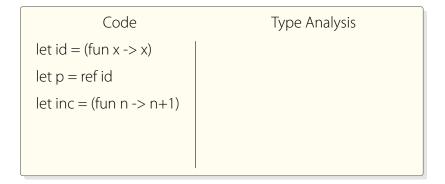
Add reference types of the form  $\tau$  ref

Add expressions of the formref  $e: \tau$  refwhere  $e: \tau$ (allocate) $!e: \tau$ where  $e: \tau$  ref(dereference) $e_1 := e_2 :$  unitwhere  $e_1 : \tau$  ref and  $e_2 : \tau$ (assign)

Works as you'd expect (like pointers in C).







Code	Type Analysis	
let id = (fun x -> x)		
let $p = ref id$		
let inc = (fun n -> n+1)		
p := inc;		
(!p) true		

Type Analysis
$id: \alpha \to \alpha$

Code	Type Analysis
let id = (fun x -> x)	$id:\alpha\to\alpha$
let $p = ref id$	p : ( $\alpha \rightarrow \alpha$ ) ref
let inc = (fun n -> n+1)	
p := inc;	
(!p) true	

Code	Type Analysis
let id = (fun x -> x)	$id:\alpha\to\alpha$
let $p = ref id$	p : ( $lpha  ightarrow lpha$ ) ref
let inc = (fun n -> n+1)	inc : int $\rightarrow$ int
p := inc;	
(!p) true	

Code	Type Analysis
let id = (fun x -> x)	$id:\alpha\to\alpha$
let p = ref id	p : ( $\alpha \rightarrow \alpha$ ) ref
let inc = (fun n -> n+1)	inc : int $\rightarrow$ int
p := inc;	OK since p : (int $ ightarrow$ int) ref
(!p) true	OK since p : (bool $ ightarrow$ bool) ref

#### Problem

- Type system is not sound
- Well-typed program  $\rightarrow^*$  type error!

#### Problem

- Type system is not sound
- Well-typed program  $\rightarrow^*$  type error!

**Proposed Solutions** 

- 1. "Weak" type variables
  - Can only be instantiated in restricted ways
  - But type exposes functional vs. imperative
  - Difficult to use

#### Problem

- Type system is not sound
- Well-typed program  $\rightarrow^*$  type error!

### **Proposed Solutions**

- 1. "Weak" type variables
  - Can only be instantiated in restricted ways
  - But type exposes functional vs. imperative
  - Difficult to use
- 2. Value restriction
  - Only generalize types of values
  - Most ML programs already obey it
  - Simple proof of type soundness

### Lessons Learned

- Features often interact in unexpected ways
- The design space is huge
- Good designs are sparse and don't happen by accident
- Simplicity is rare: *n* features  $\rightarrow n^2$  interactions
- Most PL researchers work with small languages (e.g.,  $\lambda$ -calculus) to study core issues in isolation
- But must pay attention to whole languages too