IPPL (CS 7400)
Lecture 0: Welcome!

Chris Martens
Northeastern University
Fall 2024

First things first

- Important links:
  - Class website - hub with all of these links
  - Piazza
  - Gradescope
  - (Canvas)
- Assignment 0 (Due Thursday, Sep 12)
  - Introduce yourself on Piazza
  - Complete the intake survey

Course scope

“Principles of Programming Languages”

- Programming Languages (PL) as a field of research
- Theoretical foundations of that research (back to 1930s!)
- Informed by how those foundations are relevant today

Not a tour of “language paradigms”

Course scope

“Intensive”?

- PhD level (expects a BS in CS background)
- Might feel fast paced if you haven’t seen the intake-survey material before
- Will touch on present-day active areas of research
- *Not* an unusually heavy workload (at least, not my intent!)
- Let me know if you have any concerns!

What is PL as a field of study?

Study of the relationship between **syntax** and **semantics**

```
function main() {
  print “hello world”;
  let x = 41;
  x++;
  print (toString x);
}
```

```
$ ./main

hello world
42
```

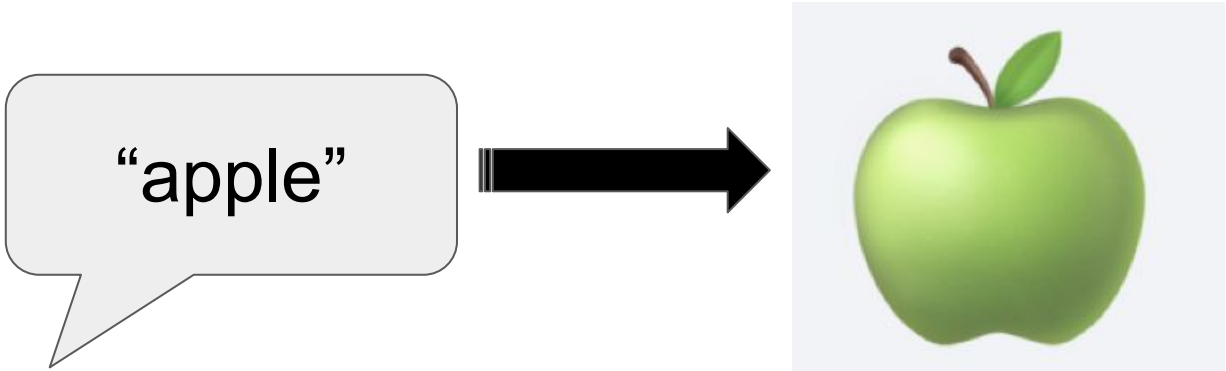
Syntax (program)

Semantics (program behavior)

What is PL as a field of study?

Study of the relationship between **syntax** and **semantics**

Also describes *linguistics…*



Syntax (utterance)

Semantics
(real-world object)

What is PL as a field of study?

Study of the relationship between **syntax** and **computational semantics**

What is PL as a field of study?

Study of the relationship between **syntax** and **computational semantics**

In other words, *what do programs mean?*

What is PL as a field of study?

Study of the relationship between **syntax** and **computational semantics**

In other words, *what do programs mean?*

A1: What they mean is what they do (when you run them)

- Operational (or dynamic) semantics

A2: What they mean is how I can reason about them abstractly

- Static semantics

Defining a programming language

A PL, in the abstract, is:

- A syntax
- At least one semantic definition for that syntax

Defining a programming language

A PL, in the abstract, is:

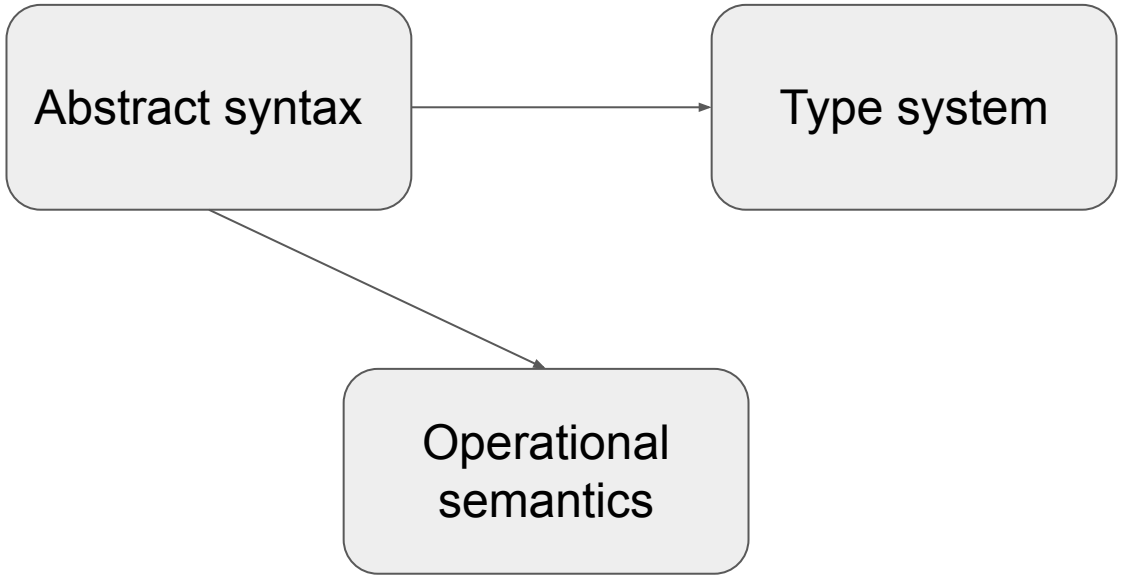
- A syntax
- At least one ***implementation-independent*** semantic definition for that syntax

This course’s focus: type systems

A theoretical (and practical!) lens for understanding programming languages

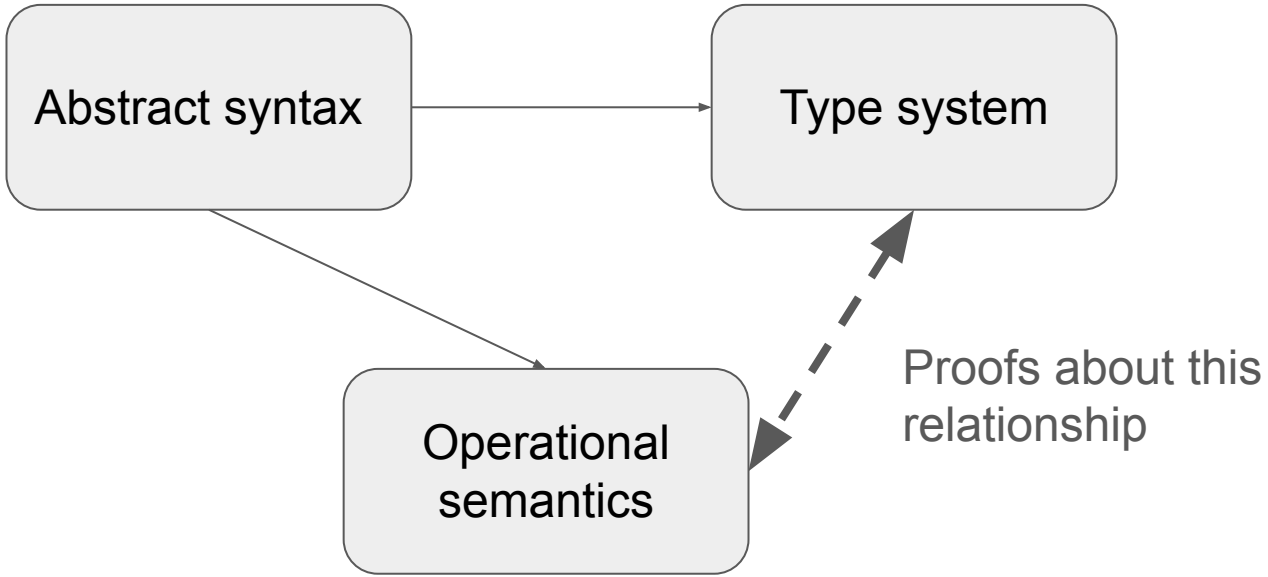
Defining a programming language

A formula we'll follow in this class:



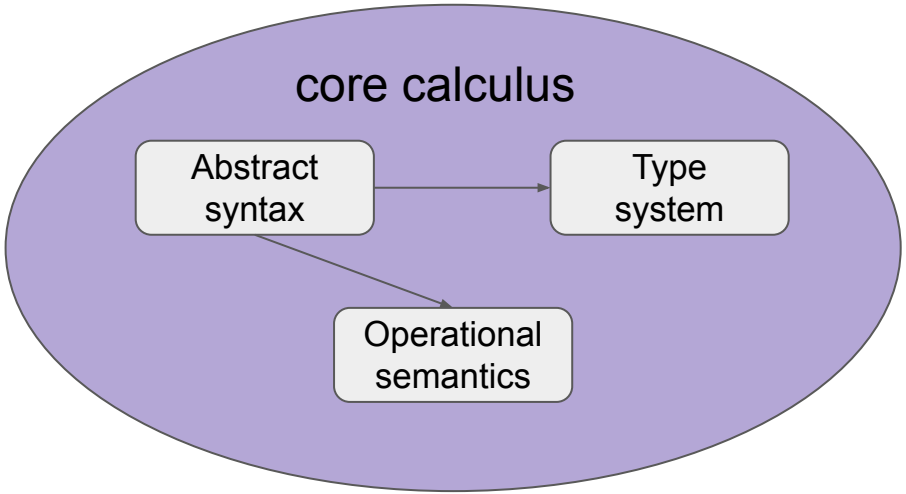
Defining a programming language

A formula we'll follow in this class:



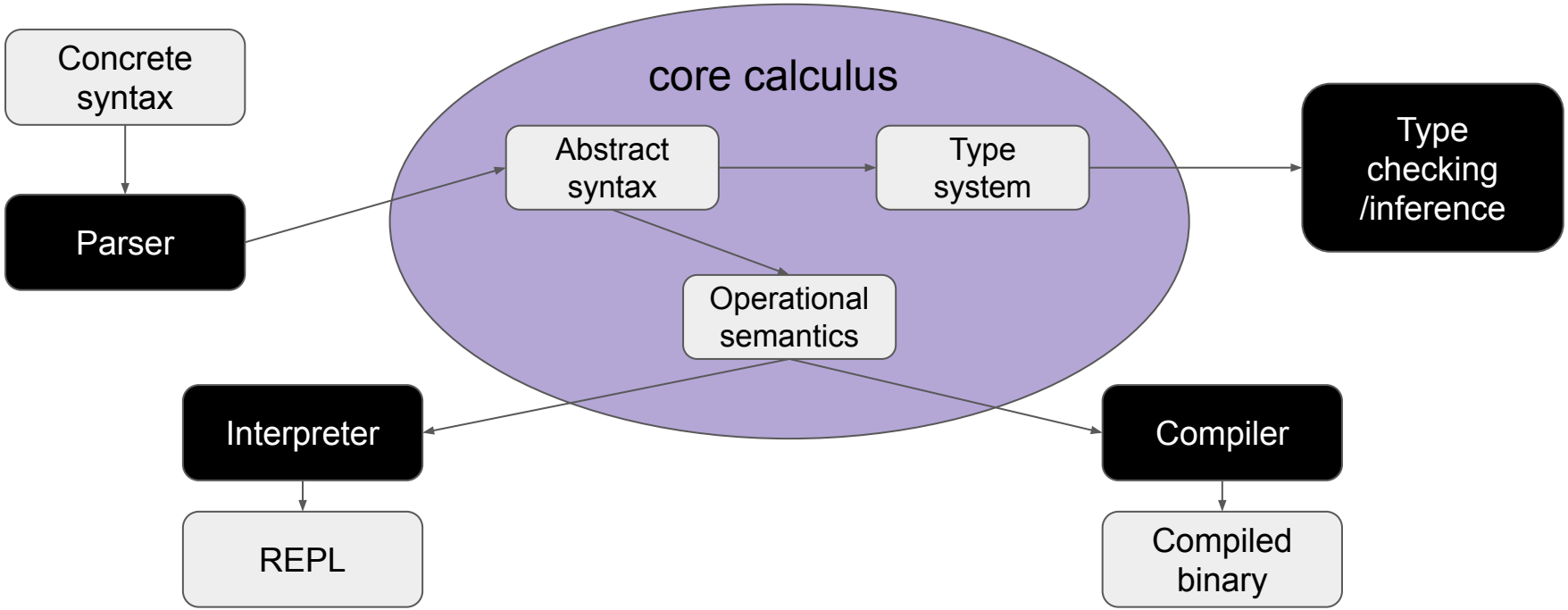
Defining a programming language

Just a tiny kernel of a PL in practice!



Defining a programming language

Just a tiny kernel of a PL in practice!



What is PL as a field of study?

- **Theory:** Discovering and organizing the fundamental nature and principles of computation; proving properties of languages (like (non)termination)
- **Design:** exploring and understanding the possibility space of PLs; establishing how their features interact
- **Engineering:** implementing languages to be fast on real hardware; ensuring preservation of high-level semantics

This course will emphasize:

- **Common structures** that appear in many languages over specific PLs
- **Core calculi**: minimal formal structures that express an idea or feature
- **Compositionality**: how can the meaning of complex programs be expressed in terms of the meaning of its parts? How do language features interact with each other?
- **Metatheory**: What formal properties do PL designs admit? What properties do we want, and how can we design PLs to have them?
- **Expressivity**: how can we capture rich computational ideas (e.g. concurrency, interaction, resource usage) in ways that admit desirable these properties?

Learning objectives

1. Foundations
- Formal language definitions
  - Inference rules
  - Derivation trees
  - Type systems
  - Operational semantics
- Functional programming
  - Lambda calculus
  - Higher-order functions
  - Inductive definitions and pattern matching
  - Type checking and inference
- Propositions-as-types, proofs-as-programs

Note! This is not a chronological list of topics that will be taught.

These objectives will be repeated & interleaved throughout the course schedule of topics.

Learning objectives

2. Core calculi: the classics

- STLC: 1, +, x
  - Propositions as types
  - Lazy/eager evaluation
- Fixed point/recursive functions
- Recursive types
- Polymorphism
- Mutable state/effects

Note! This is not a chronological list of topics that will be taught.

These objectives will be repeated & interleaved throughout the course schedule of topics.

Learning objectives

3. Metatheoretic techniques

- Proof by structural and rule induction
- Proofs of type soundness (e.g., progress and preservation)
- Proofs of translation soundness / completeness
- Proofs of parametricity/representation independence
- Understanding how to repair a language definition when a proof fails, and how to revise a theorem statements to describe stronger and weaker properties

Note! This is not a chronological list of topics that will be taught.

These objectives will be repeated & interleaved throughout the course schedule of topics.

Potential advanced topics

Depending on student interest/pace of learning:

- Bidirectional typing
- Substructural type systems (e.g. linear, affine)
- Imperative programming (and translations to/from functional)
- Monadic computation and effects
- Asynchronous/reactive/interactive computation
- Polarized type systems
- Dependent types and proof assistants
- Languages for concurrent and distributed computation
- Cost semantics
- Information flow type systems

Class format

Mostly whiteboard lectures + in-class exercises

Occasional code demos or slides

I'll aim to post notes (pdf) within a day or two of lecture; taking your own notes is strongly recommended

Alternative options:

- collaborative notes repo for the class
- assigned scribes

Video recordings available in exceptional circumstances if you miss class

Student work

No exams

5 written assignments (due biweekly)

- Typeset in LaTeX
- Occasional “programming” in mini-languages

Mini project: explore an idea from class more deeply. Examples:

- Mechanize a theorem in a proof assistant
- Write an elaborator/interpreter/compiler for one of our languages
- Extend a language definition and prove corresponding theorems

You may use your language of choice for the mini-project.

Tentative assignment topics

1. Extending a tiny language definition (Tracery)
2. Untyped and simply-typed lambda calculus
3. Recursive types and functions
4. Polymorphism and parametricity
5. Asynchronous control flow (e.g. continuations, effect handlers)

My goal for coursework difficulty

Low floor, high ceiling

- Some of you are brand new to this way of thinking about PL: you should expect to productively struggle with your misunderstandings and need to consult with peers/instructors to overcome them, but you should still be able to complete assignments in a reasonable amount of time.
- Some of you know most of this material already: assignments might go faster for you, but each one should include at least some material that stretches the limits of your understanding/abilities

Let me know as we go if the assignments miss this target.

Course Policies

- No late hand-ins accepted (except in exceptional circumstances with written permission)
- 1 free re-grade per assignment resubmitted within 1 week after it’s returned

Consequently:

- Turn in whatever you have by the deadline. Don’t leave questions blank, but a sketch of how you’d approach the problem (sans solution) is ok, or even “I have no idea how to start.”
- Write down as much as possible about your understanding of the problem/solution so we can help identify and correct misconceptions

Course Policies

Collaboration:

- Goal: a mutually supportive classroom, not a competitive one
- Learn from your peers; be generous with your knowledge
- But give everyone (including yourself!) a chance to arrive at knowledge in your own way
- “Whiteboard rule”: you can discuss problems together at the whiteboard, but you **may not write notes to take away**. Written work is **individual**.
- Projects can be individual or in pairs. Pairs must clearly document and co-sign the contributions of each member.

Course Policies

Student well-being:

- Never prioritize your coursework over your (mental or physical) well-being.
  - Let me know if the work I'm asking you to do for this class becomes an obstacle to your health.
- If you feel sick, stay at home. No need for doctor's note. Ask me and I will give you the lecture recording.
- Tips to protect yourself and others from contagious illness:
  - Get updated flu/covid shots as soon as possible
  - Keep a stock of tests and let contacts know if you test positive
  - Limit large indoor gatherings without good ventilation
  - Wear face coverings (KN95 or N95) when feasible

Course Policies

Accommodations:

If you need learning support that isn't provided by default:

- Getting an official accommodation request through DAS is the best route
- But if you're having trouble with that route, ask me anyway

Course Policies

Respecting and supporting your classmates:

- Use stated names and pronouns
- Be generous with your knowledge (but…)
- Check yourself before assuming someone knows less than you
  - No “feigned surprise”
  - No “um, actually…”s
- Help me create an environment where no one feels afraid to ask “stupid questions” or like they don’t belong here
- There *are* wrong answers – hold each other to a high standard of rigor
- But **everyone here can meet or exceed that standard**.

First things last

- Important links:
  - Class website - hub with all of these links
  - Piazza
  - Gradescope
  - (Canvas)
- Assignment 0 (Due Thursday, Sep 12)
  - Introduce yourself on Piazza
  - Complete the intake survey