The Course: Summary

Four (more or less) equal sized units:

- Basics: inductive definitions, syntax, semantics, type safety
- Extensions: functions, product/sum/recursive types, continuations/exceptions (abstract machines), parallelism/concurrency
- Advanced features: polymorphism, data abstraction, dynamic typing, references/monads, subtypes/inheritance
- Realistic languages: Scala, Java (Featherweight),

The Moral

There is a scientific theory of programming languages that motivates and is motivated by the practical problems of programming.

- Don’t settle for less!
- It’s a very beautiful theory!
- There’s lots more to be done!

The GUT of PL’s

Type theory is the grand unified theory of programming languages.

- Rigorous framework for specifying dynamic and static semantics.
- Supports reasoning about languages, e.g. type safety.
- Supports reasoning about programs, e.g. time complexity.

PL’s as Types

A programming language is “just” a collection of types!

- Modular arithmetic: int.
- Tuples/structures/records: \( \tau_1 \times \tau_2 \).
- Procedures/functions: \( \tau_1 \rightarrow \tau_2 \).
- Variants: \( \tau_1 + \tau_2 \).
- Recursive types: \( \mu \tau. \).
- Generics/polymorphism: \( \forall \tau. \).
- Abstract types: \( \exists \tau. \).
- And so on!
Type-Based Approach to Programming Languages

- **Language constructs** arise as introductory and elimina-
tory forms associated with types.

- **Static semantics** specify how constructs may be combined
  in well-formed programs.

- **Dynamic semantics** specify how constructs may be exe-
cuted, subject to type safety.

- **Type safety** is a **conservation principle**: introductory forms
  are values of the type and elimination forms are **inverse** to
  the introductory forms.

Propositions as Types

Why does type theory work so well as a foundation for program-
mng?

- **Wigner**: The Unreasonable Effectiveness of Mathematics in
  Physics. Why is it that mathematics is so effective in mod-
eling physical phenomena?

- **Vardi, et. al.**: The Unusual Effectiveness of Logic in Com-
  puter Science. Why is logic so effective as a tool for com-
  puter science?

Where Does This End?

No one knows! It’s the subject of active research!

- **Modal** logic codifies run-time code generation and meta-
  programming!

- **Linear** logic codifies state change and concurrency.

- **Quantifier** logic codifies reasoning about array bounds.

Various research efforts are concerned with working out these
beautiful connections between logic and computer science.

Why Formalisms?

Why are formalisms so important for studying language con-
cepts?

- **Discipline**: subject vague ideas to rigorous analysis.

- **Specification**: formalisms are the best form of documenta-
tion.

- **Mechanization**: machines understand formalisms.

Formalization

What good is all this formalization?

- Facilitates prototyping and experimentation.

- One type checker for all languages.

- One interpreter for all languages.

The catch: it’s not always easy to formalize existing languages!
Summary

Logic, semantics, type theory are fundamental to the systematic study of programming.

Using them effectively leads to new solutions to practical problems.

These solutions could never have been achieved without these tools.