

CMPSCI 630: Programming Languages
Summary
Spring 2009
(with thanks to Robert Harper)

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
- Real(istic) languages: Scala, Java (Featherweight),

1

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!

2

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.

3

PL's as Types

A programming language is "just" a collection of types!

- Modular arithmetic: `int`.
- IEEE f.p. arithmetic: `float`.
- Tuples/structures/records: $\tau_1 \times \tau_2$.
- Procedures/functions: $\tau_1 \rightarrow \tau_2$.

4

- Variants: $\tau_1 + \tau_2$.
- Recursive types: $\mu t. \tau$.
- Generics/polymorphism: $\forall t. \tau$.
- Abstract types: $\exists t. \tau$.
- And so on!

Type-Based Approach to Programming Languages

- **Language constructs** arise as **introductory and eliminatory forms** associated with types.
- **Static semantics** specify how constructs may be combined in well-formed programs.
- **Dynamic semantics** specify how constructs may be executed, 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

5

Propositions as Types

Why does type theory work so well as a foundation for programming?

- Wigner: The Unreasonable Effectiveness of Mathematics in Physics. Why is it that mathematics is so effective in modelling physical phenomena?
- Vardi, *et. al.*: The Unusual Effectiveness of Logic in Computer Science. Why is logic so effective as a tool for computer science?

6

Propositions as Types

Type theory "works" because of **deep connections** with logic!

- A **type** is a **specification** of behavior.
- A **program** of a type is a **proof** of that specification.

As type systems become more expressive, types become as powerful as general mathematics.

Eventually, computer science and mathematics merge into a unified theory of computation (based on type theory).

7

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.

8

Why Formalisms?

Why are formalisms so important for studying language concepts?

- **Discipline**: subject vague ideas to rigorous analysis.
- **Specification**: formalisms are the best form of documentation.
- **Mechanization**: machines understand formalisms.

9

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!

10

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.