CS 510: Programming Languages

Introduction

David Walker

Fall, 2003

Why Study Programming Languages?

Programming is an **explanatory** activity.

- To yourself, now and in the future.
- To other developers and maintainers.
- To the machine!

Why Study Programming Languages?

Therefore the **language** we use matters enormously.

- How to write a program to solve a problem?
- How to express your assumptions and guarantees?
- How to document the structure of a system?

Why Study Programming Languages?

There is (or ought to be) a close relationship between

- the **code**, which is executable, and

- its **properties**, which are descriptions of its behavior.

Good languages make it easier to **establish**, **verify**, and **maintain** the relationship between code and its properties.

The Science of Programming Languages

You may have heard that all programming languages are equal.

- All familiar languages are **Turing equivalent** — they express the same set of **computable functions** as each other.

- Therefore (?) there's no difference between them. It's all a matter of taste.

The Science of Programming Languages

Yet we all know that some languages are **more equal** than others.

- Do you want to build web pages in assembly language?

- Do you want to build a device driver in Perl?

- Do you want to use C as your markup language?

The Science of Programming Languages

There **is** an irreducible subjective element.

- Surface syntax: $3{+}2$ *vs.* $+(3, 2)$ or $(+\ 3\ 2)$ or $(3, 2)+$.

- Resemblance to familiar languages and notation.

- Development environment: structure editors *vs.* text editors.

The Science of Programming Languages

There is also a **scientific basis** for programming languages whose primary tools are

- **Type theory**. Techniques for structuring languages to ensure **safety** and **modularity** of programs.

- **Operational semantics**. Techniques for describing the execution behavior of programs, at various levels of abstraction.

- **Mathematical logic**. Techniques for **specifying** and **verifying** programs.

The Goal Of This Course

The goal of 510 is to introduce the **fundamental principles** of programming language design and implementation.

- Emphasis on **rigor** and **elegance**.

- Emphasis on both **theory** and **practice**.

We will model a wide variety of programming concepts in the framework of type theory and operational semantics.

Some Topics We'll Discuss

Theoretical foundations.

- Inductive definitions.
- Structural induction.

Some Topics We'll Discuss

Syntactic structure.

- Concrete syntax: the strings you type.

- Abstract syntax: the "deep structure" of a language.

  1. First-order: tree structure.

  2. Higher-order: binding and scope.

Some Topics We'll Discuss

Semantics of languages.

- Static semantics, or type systems.
- Dynamic semantics, or execution rules.
- What is a safe language?

Some Topics We'll Discuss

Language concepts:

- Higher-order functions.
- Abstract machines.
- Imperative programming.

Some Topics We'll Discuss

More language concepts:

- Continuations and concurrency.

- Dynamic typing.

- Lazy evaluation.

- Parallelism.

Some Topics We'll Discuss

Yet more language concepts:

- Moduarlity and data abstraction.
- Polymorphism and parametricity.
- Inheritance and subtyping.

Some Things We Won't Discuss

Taxonomy.

- Not a trip to the zoo.
- Not a survey of the "Top Ten" PL's.
- Not a way to collect C.V. items.

What's 510 Like?

It will stress both **mathematical** skills and **implementation** skills.

- Prove theorems, disprove conjectures.
- Build programs to illustrate concepts.

Emphasis on rigor, clarity, and elegance:

"If it isn't beautiful, it isn't right."

Acknowledgment

- Acknowledging ideas, insights, code, writing, engineering, and explanations of others is a necessary part of carrying out new research, writing, programming, teaching or engineering of your own.

- I would like to thank Robert Harper from Carnegie Mellon University for supplying me with many, if not most of the lecture notes that I will use in this course as well as ideas for assignments.

- I would also like to thank Princeton's Andrew Appel for test-driving this course several times and for his input in terms of course structure, lecture notes and assignment ideas.