

What To Teach About Programming Languages

Robert Harper

Carnegie Mellon University

ACM SIGCSE 2009

The Traditional Approach

A **survey** of well-known programming languages.

- A taste of Java, Lisp, Prolog,
- Selection based on current fads.

A **taxonomy** of “paradigms.”

- x -oriented programming languages, for various values of x .
- “declarative” versus “non-declarative” languages.
- “scripting” languages, “domain-specific” languages.
- . . . and various other distinctions according to taste.

There are **many** textbooks written in these styles.

What's Wrong With This Approach

Students accumulate empty vitae items, but little understanding.

- How to transfer ideas to a new setting?
- Fad-driven, quickly obsolete.
- There is no such thing as a “programming paradigm.”

More importantly, it is **unscientific**.

- Based on superficial morphology (Gould's Zebra).
- No theory of what is going on.
- No permanent results, but a lot of opinions.

The field is slipping from science to vocation

What We're Advocating

A **scientific theory** of programming languages based on **abstract models of computation**.

- Supports precise definitions and rigorous analysis.
- Transfers to new problem domains.
- Separates abstraction from implementation.
- Enables verification.
- Emphasizes enduring principles, not current trends.

The field of programming languages has changed **dramatically** over 25 years!

A Formalism-Based Approach

A linguistic model is specified by

- ① A **static semantics** defining the structure of the model.
- ② A **dynamic semantics** defining the execution steps.

The key is that such a model is

- ① **abstract**: essence, not accident.
- ② **analyzable**: supports rigorous proof.

Example: Parallel Computing

Theme: parallelism is about **performance**, not concurrency.

- **Deterministic**: same meaning as sequential.
- **Efficient**: makes better use of processors.

Two ingredients:

- Abstract **cost semantics**: **sequential** and **parallel** cost measures.
- Concrete **realization**: communication and scheduling costs.

Example: Parallel Computing

Sequential execution:

$$\frac{e_1 \mapsto_s e'_1}{e_1 + e_2 \mapsto_s e'_1 + e_2} \quad \frac{e_2 \mapsto_s e'_2}{n_1 + e_2 \mapsto_s n_1 + e'_2} \quad \frac{}{17 + 18 \mapsto_s 35}$$

Idealized parallel execution:

$$\frac{e_1 \mapsto_p e'_1 \quad e_2 \mapsto_p e'_2}{e_1 + e_2 \mapsto_p e'_1 + e'_2} \quad \frac{}{17 + 18 \mapsto_p 35}$$

Theorem [[Implicit Parallelism](#)]: $e \mapsto_s^* n$ iff $e \mapsto_p^* n$.

Example: Parallelism

Time complexity = number of steps for sequential and parallel semantics.

- **Sequential:** $T_s(e) = k$ s.t. $e \mapsto_s^{(k)} n$.
- **Parallel:** $T_p(e) = k$ s.t. $e \mapsto_p^{(k)} n$.

Essential for making precise statements about **efficiency**, which is the essence of parallelism.

Theorem [**Efficient Implementation**] If parallel complexity is d and sequential complexity is w , then runs on a p -processor RAM in time $O(w/p + d)$.

A Selection of Topics

- Abstract and concrete syntax, binding and scope.
- Semantic specification of models, statics and dynamics.
- Finite and infinite data structures.
- Modularity, genericity, and data abstraction.
- Time and space complexity, cost semantics.
- Laziness, speculative parallelism.
- Deterministic parallelism.
- Indeterminacy and concurrency.
- Mutable storage.
- Continuations, exceptions, coroutines.
- Run-time systems, storage management, scheduling.

The Bottom Line

PL's are about **linguistic models of computation**.

- An important tool for any computer scientist.
- Fundamental to modern software development techniques.

Every CS student should learn to use these tools!

- Languages abound.
- Directly applicable to implementation.
- Principles, not fashions.