

Lecture 27 – Course Review

AAA551: Programming Language Theory

Jihyeok Park



2026 Spring

We started with the gap between “*what we think code does*” and “*what it actually does*”.

- JavaScript: `[] + []` is `""` but `[] - []` is `0`.
- Python: a lambda capturing a loop variable prints 12 three times, not 10, 11, and 12.
- Scala: `Child.y` prints 0, not 20, due to initialization order.
- We replaced fuzzy intuition with **mathematical precision** by giving programs **formal semantics** and **type systems**.

We viewed Programming Languages through two complementary lenses:

- **Semantics (The Meaning)**
 - **Operational:** *How* is a program executed?
 - **Denotational:** *What* is the mathematical object for a program?
 - **Axiomatic:** *Which properties* does a program satisfy?
- **Type Systems (The Guardrails)**
 - **Foundations:** Simply-Typed Lambda Calculus, Extensions
 - **Type Abstraction:** Universal, Subtyping, Existential, Operators
 - **Propositions as Types:** Curry-Howard, Dependent Types
 - **Modern Types:** Refinement, Flow-Sensitive, Substructural, Effects

- **Operational Semantics** – *How* a program runs.
 - Lec 2: **Big-Step** Operational Semantics
 - Lec 3: **Small-Step** Operational Semantics
 - Lec 4: **Evaluation Contexts** and **Lambda Calculus**
 - Lec 9: **Abstract Machines**

- **Denotational Semantics** – *What* a program means.
 - Lec 5: **Denotational** Semantics
 - Lec 6: **Fixed-Point Theory**
 - Lec 7: **Trace** Semantics
 - Lec 8: **Trace Properties** (safety and liveness)

- **Axiomatic Semantics** – *Which properties* a program satisfies.
 - Lec 10: **Axiomatic** Semantics (Hoare Logic)
 - Lec 11: **Systematic Program Proofs**
 - Lec 12–13: **Separation Logic**

- **Foundations**

- Lec 14: **Type System** (progress and preservation)
- Lec 15: **Simply-Typed Lambda Calculus**
- Lec 16: **Type Extensions**

- **Type Abstraction**

- Lec 17: **Universal Types** (parametric polymorphism)
- Lec 18: **Subtyping**
- Lec 19: **Existential Types** (data abstraction)
- Lec 20: **Type Operators** (higher-kinded types)

- **Propositions as Types**

- Lec 21: **Curry-Howard Isomorphism**
- Lec 22: **Dependent Types**

- **Modern Types**
 - Lec 23: **Refinement Types**
 - Lec 24: **Flow-Sensitive Types** (abstract interpretation)
 - Lec 25: **Substructural Types** (linear and affine types)
 - Lec 26: **Effect Systems**

- All of these are **static guardrails**: they reject ill-behaved programs *before* they ever run.

- Together with **semantics**, they let us **state** what a program should do and **prove** that it does.

- We learned how to give programs a **precise meaning** and how to **reason** about them with **mathematical rigor**.
- These foundations underlie real-world tools:
 - Compilers and optimizers (semantics-preserving transformations)
 - Type checkers and static analyzers
 - Program verifiers and proof assistants
- Thank you for your hard work throughout the semester!

- I hope you enjoyed the class!

Jihyeok Park

jihyeok_park@korea.ac.kr

<https://plrg.korea.ac.kr>