

# Lecture 2 – Big-Step Operational Semantics

AAA551: Programming Language Theory

Jihyeok Park



2026 Spring

- Before entering the world of PL, we learned the basics of **Scala** language in the previous lecture.
  - Basic Features of Scala
  - User-Defined Data Types
  - First-Class Functions
  - Immutable Collections

## 1. Programming Languages

## 2. Mathematical Preliminaries

Binary Relations and Functions

Inductively Defined Sets

## 3. Big-Step Operational Semantics

Example: VAE

Scala Implementation

Derivation

## 4. Simple Imperative Language – IMP

Big-Step Operational Semantics of IMP

Equivalence of Statements

Determinism

## 1. Programming Languages

## 2. Mathematical Preliminaries

Binary Relations and Functions

Inductively Defined Sets

## 3. Big-Step Operational Semantics

Example: VAE

Scala Implementation

Derivation

## 4. Simple Imperative Language – IMP

Big-Step Operational Semantics of IMP

Equivalence of Statements

Determinism

## Definition (Programming Language)

A **programming language** is defined by

- **Syntax**: a grammar that defines the **structure** of programs
- **Semantics**: a set of rules that defines the **meaning** of programs

For example, VAE is a simple programming language for arithmetic expressions with variables.

The following are valid VAE programs and their results:

- $1 + 2$  evaluates to 3
- $1 + 2 * 3$  evaluates to 7
- $x := 1; x + 2$  evaluates to 3
- $x := 2; y := 3; x * y$  evaluates to 6

We use a variant of the **extended Backus-Naur form (EBNF)** to define the **abstract syntax** of programming languages.

For example, let's define the abstract syntax of VAE in BNF:

$$e ::= n \mid e + e \mid e * e \mid x := e; e \mid x$$

where  $n \in \mathbb{Z}$  denotes an integer and  $x \in \mathbb{X}$  denotes a variable.

Its Scala implementation is as follows:

```
enum Expr:  
  case Num(number: BigInt)  
  case Add(left: Expr, right: Expr)  
  case Mul(left: Expr, right: Expr)  
  case Val(name: String, init: Expr, body: Expr)  
  case Id(name: String)
```

Three approaches to **formal semantics**:

- **Operational**

$$\sigma \vdash e \Rightarrow v$$

- *How* is a program executed?
- Useful for implementation of compilers and interpreters.

$$\frac{\vdash e_1 \Rightarrow n_1 \quad \vdash e_2 \Rightarrow n_2}{\vdash e_1 + e_2 \Rightarrow n_1 + n_2}$$

- **Denotational**

$$\llbracket e \rrbracket$$

- *What* is the mathematical object for a program?
- Useful for theoretical foundations.

$$\llbracket e_1 + e_2 \rrbracket = \llbracket e_1 \rrbracket + \llbracket e_2 \rrbracket$$

- **Axiomatic**

$$\vdash \{ \phi \} e \{ \phi' \}$$

- *Which properties* does a program satisfy?
- Useful for proving program properties and correctness.

$$\vdash \{ x < n \wedge y < m \} z = x + y \{ z < n + m \}$$

## 1. Programming Languages

## 2. Mathematical Preliminaries

Binary Relations and Functions

Inductively Defined Sets

## 3. Big-Step Operational Semantics

Example: VAE

Scala Implementation

Derivation

## 4. Simple Imperative Language – IMP

Big-Step Operational Semantics of IMP

Equivalence of Statements

Determinism

- The **product** of two sets  $A$  and  $B$  is the set of all ordered pairs  $(a, b)$  where  $a \in A$  and  $b \in B$ :

$$A \times B \triangleq \{(a, b) \mid a \in A, b \in B\}$$

- A **binary relation**  $R$  between  $A$  and  $B$  is a subset of  $A \times B$ .

$$R \subseteq A \times B$$

- Examples of binary relations:
  - empty relation:  $\emptyset$
  - total relation:  $A \times B$
  - identity relation:  $\{(a, a) \mid a \in A\}$

- A (total) **function**  $f : A \rightarrow B$  from a set  $A$  to a set  $B$  is a binary relation that satisfies the following properties:
  - Totality –  $\forall a \in A, \exists b \in B. (a, b) \in f$
  - Uniqueness –  $\forall a \in A, \forall b, b' \in B. ((a, b) \in f \wedge (a, b') \in f) \implies b = b'$
- A **partial function**  $f : A \rightarrow B$  from a set  $A$  to a set  $B$  is a binary relation only satisfying the uniqueness property.
- The **domain** of a function  $f$  is the set of all  $a \in A$  such that  $\exists b \in B. (a, b) \in f$ :

$$\text{dom}(f) \triangleq \{a \in A \mid \exists b \in B. (a, b) \in f\}$$

- Given two functions  $f : A \rightarrow B$  and  $g : B \rightarrow C$ , their **composition**  $g \circ f : A \rightarrow C$  is defined as follows:

$$\forall a \in A, (g \circ f)(a) = g(f(a))$$

We can define sets inductively by specifying a set of **inference rules**.

An **inference rule** consists of multiple **premises** and one **conclusion**:

$$\frac{\textit{premise}_1 \quad \textit{premise}_2 \quad \dots \quad \textit{premise}_n}{\textit{conclusion}}$$

meaning that “*if all the premises are true, then the conclusion is true*”:

$$\textit{premise}_1 \wedge \textit{premise}_2 \wedge \dots \wedge \textit{premise}_n \implies \textit{conclusion}$$

For example, consider the following BNF grammar for VAE:

$$e ::= n \mid e + e \mid e * e \mid x := e; e \mid x$$

It can be translated into the following inference rules:

$$\frac{}{n \in \mathbb{E}} \quad \frac{e_1 \in \mathbb{E} \quad e_2 \in \mathbb{E}}{e_1 + e_2 \in \mathbb{E}} \quad \frac{e_1 \in \mathbb{E} \quad e_2 \in \mathbb{E}}{e_1 * e_2 \in \mathbb{E}}$$
$$\frac{e_1 \in \mathbb{E} \quad e_2 \in \mathbb{E}}{x := e_1; e_2 \in \mathbb{E}} \quad \frac{}{x \in \mathbb{E}}$$

Another example is the set of natural numbers  $\mathbb{N}$ :

$$\frac{}{0 \in \mathbb{N}} \quad \frac{n \in \mathbb{N}}{S(n) \in \mathbb{N}}$$

where  $S$  is the successor function.

Thus, 3 (i.e.,  $S(S(S(0)))$ ) is a natural number because:

$$\frac{\frac{\frac{\frac{}{0 \in \mathbb{N}}}{S(0) \in \mathbb{N}}}{S(S(0)) \in \mathbb{N}}}{S(S(S(0))) \in \mathbb{N}}$$

## 1. Programming Languages

## 2. Mathematical Preliminaries

Binary Relations and Functions

Inductively Defined Sets

## 3. Big-Step Operational Semantics

Example: VAE

Scala Implementation

Derivation

## 4. Simple Imperative Language – IMP

Big-Step Operational Semantics of IMP

Equivalence of Statements

Determinism

There are two main styles of operational semantics:

- **Big-Step Operational Semantics** defines the meaning of a program by specifying how it executes on a machine in **one big step**.

$$\frac{\dots}{\vdash e \Rightarrow n}$$

(An expression  $e$  evaluates to  $n$ .)

- **Small-Step Operational Semantics** defines the meaning of a program by specifying how it executes on a machine **step-by-step**.

$$e \rightarrow e' \rightarrow e'' \rightarrow \dots \rightarrow n$$

(An expression  $e$  is reduced to  $e'$ , then to  $e''$ , and so on until  $n$ .)

Let's define the **big-step operational semantics** of VAE:

$$e ::= n \mid e + e \mid e * e \mid x := e; e \mid x$$

It means “*under environment  $\sigma$ , an expression  $e$  evaluates to integer  $n$* ”:

$$\boxed{\sigma \vdash e \Rightarrow n}$$

where an environment  $\sigma : \mathbb{X} \rightarrow \mathbb{Z}$  stores integers bound to variables.

$$\frac{}{\sigma \vdash n \Rightarrow n}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \qquad \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 * e_2 \Rightarrow n_1 \times n_2}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma[x \mapsto n_1] \vdash e_2 \Rightarrow n_2}{\sigma \vdash x := e_1; e_2 \Rightarrow n_2} \qquad \frac{x \in \text{dom}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

$$\overline{\sigma \vdash n \Rightarrow n}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \qquad \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 * e_2 \Rightarrow n_1 \times n_2}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma[x \mapsto n_1] \vdash e_2 \Rightarrow n_2}{\sigma \vdash x := e_1; e_2 \Rightarrow n_2} \qquad \frac{x \in \text{dom}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

The following is the interpreter of VAE in Scala, which directly corresponds to the big-step operational semantics defined in the previous slide:

```
def interp(expr: Expr, env: Env): Value = expr match
  case Num(n)      => n
  case Add(l, r)   => interp(l, env) + interp(r, env)
  case Mul(l, r)   => interp(l, env) * interp(r, env)
  case Val(x, i, b) => interp(b, env + (x -> interp(i, env)))
  case Id(x)       => env.getOrElse(x, error(s"free identifier: $x"))
```

$$\overline{\sigma \vdash n \Rightarrow n}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \quad \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 * e_2 \Rightarrow n_1 \times n_2}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma[x \mapsto n_1] \vdash e_2 \Rightarrow n_2}{\sigma \vdash x := e_1; e_2 \Rightarrow n_2} \quad \frac{x \in \text{dom}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

---


$$\emptyset \vdash x := 1; \{y := 2; x + y\} \Rightarrow$$

where

$$\overline{\sigma \vdash n \Rightarrow n}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \quad \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 * e_2 \Rightarrow n_1 \times n_2}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma[x \mapsto n_1] \vdash e_2 \Rightarrow n_2}{\sigma \vdash x := e_1; e_2 \Rightarrow n_2} \quad \frac{x \in \text{dom}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

$$\frac{\overline{\emptyset \vdash 1 \Rightarrow 1} \quad \frac{\overline{\sigma_0 \vdash 2 \Rightarrow 2} \quad \frac{\frac{x \in \text{dom}(\sigma_1)}{\sigma_1 \vdash x \Rightarrow 1} \quad \frac{y \in \text{dom}(\sigma_1)}{\sigma_1 \vdash y \Rightarrow 2}}{\sigma_1 \vdash x + y \Rightarrow 3}}{\sigma_0 \vdash y := 2; x + y \Rightarrow 3}}{\emptyset \vdash x := 1; \{y := 2; x + y\} \Rightarrow 3}$$

where

$$\begin{aligned} \sigma_0 &= [x \mapsto 1] \\ \sigma_1 &= [x \mapsto 1, y \mapsto 2] \end{aligned}$$

## 1. Programming Languages

## 2. Mathematical Preliminaries

Binary Relations and Functions

Inductively Defined Sets

## 3. Big-Step Operational Semantics

Example: VAE

Scala Implementation

Derivation

## 4. Simple Imperative Language – IMP

Big-Step Operational Semantics of IMP

Equivalence of Statements

Determinism

Let's construct a simple imperative programming language IMP:

Expressions  $e ::= n \mid x \mid e + e \mid e * e \mid e < e \mid \text{true} \mid \text{false}$

Statements  $s ::= \text{skip}$   
                   $\mid x := e$   
                   $\mid s; s$   
                   $\mid \text{if } e \text{ then } s \text{ else } s$   
                   $\mid \text{while } e \text{ do } s$

Values  $v ::= n \mid \text{true} \mid \text{false}$

where  $n \in \mathbb{Z}$ , and  $x \in \mathbb{X}$ .

We will define two forms of big-step operational semantics for IMP:

$$\boxed{\sigma \vdash e \Rightarrow v} \quad \boxed{\sigma \vdash s \Rightarrow \sigma}$$

where  $\sigma : \mathbb{X} \rightarrow \mathbb{V}$ .

$$\boxed{\sigma \vdash e \Rightarrow v}$$

$$\sigma \vdash n \Rightarrow n \qquad \frac{x \in \text{dom}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \qquad \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 * e_2 \Rightarrow n_1 \times n_2}$$

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 < e_2 \Rightarrow n_1 < n_2}$$

$$\sigma \vdash \text{true} \Rightarrow \text{true} \qquad \sigma \vdash \text{false} \Rightarrow \text{false}$$

$$\boxed{\sigma \vdash s \Rightarrow \sigma}$$

$$\sigma \vdash \text{skip} \Rightarrow \sigma$$

$$\frac{\sigma \vdash e \Rightarrow v}{\sigma \vdash x := e \Rightarrow \sigma[x \mapsto v]}$$

$$\frac{\sigma \vdash s_1 \Rightarrow \sigma_1 \quad \sigma_1 \vdash s_2 \Rightarrow \sigma_2}{\sigma \vdash s_1; s_2 \Rightarrow \sigma_2}$$

$$\boxed{\sigma \vdash s \Rightarrow \sigma}$$

$$\frac{\sigma \vdash e \Rightarrow \text{true} \quad \sigma \vdash s_1 \Rightarrow \sigma'}{\sigma \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \Rightarrow \sigma'} \quad \frac{\sigma \vdash e \Rightarrow \text{false} \quad \sigma \vdash s_2 \Rightarrow \sigma'}{\sigma \vdash \text{if } e \text{ then } s_1 \text{ else } s_2 \Rightarrow \sigma'}$$

$$\frac{\sigma \vdash e \Rightarrow \text{true} \quad \sigma \vdash s \Rightarrow \sigma' \quad \sigma' \vdash \text{while } e \text{ do } s \Rightarrow \sigma''}{\sigma \vdash \text{while } e \text{ do } s \Rightarrow \sigma''}$$

$$\frac{\sigma \vdash e \Rightarrow \text{false}}{\sigma \vdash \text{while } e \text{ do } s \Rightarrow \sigma}$$

## Definition (Equivalence of Statements)

Two statements  $s_1$  and  $s_2$  are **equivalent**, denoted by  $s_1 \equiv s_2$ , if they produce the same final environment for any initial environment:

$$\forall \sigma, \sigma' \quad \sigma \vdash s_1 \Rightarrow \sigma' \iff \sigma \vdash s_2 \Rightarrow \sigma'$$

For example, the following two statements are equivalent:

$$\text{while } e \text{ do } s \quad \equiv \quad \text{if } e \text{ then } (s; \text{while } e \text{ do } s) \text{ else skip}$$

For a given environment  $\sigma$ ,

- Case 1:  $\sigma \vdash e \Rightarrow \text{true}$ , then both statements evaluate  $s$  and then evaluate the while loop again.

$$\frac{\frac{\dots}{\sigma \vdash e \Rightarrow \text{true}} \quad \frac{\dots}{\sigma \vdash s \Rightarrow \sigma'} \quad \frac{\dots}{\sigma' \vdash \text{while } e \text{ do } s \Rightarrow \sigma''}}{\sigma \vdash \text{while } e \text{ do } s \Rightarrow \sigma''}$$

$$\frac{\frac{\dots}{\sigma \vdash e \Rightarrow \text{true}} \quad \frac{\frac{\dots}{\sigma \vdash s \Rightarrow \sigma'} \quad \frac{\dots}{\sigma' \vdash \text{while } e \text{ do } s \Rightarrow \sigma''}}{\sigma \vdash s; \text{while } e \text{ do } s \Rightarrow \sigma''}}{\sigma \vdash \text{if } e \text{ then } (s; \text{while } e \text{ do } s) \text{ else skip} \Rightarrow \sigma''}$$

For example, the following two statements are equivalent:

$$\text{while } e \text{ do } s \quad \equiv \quad \text{if } e \text{ then } (s; \text{while } e \text{ do } s) \text{ else skip}$$

For a given environment  $\sigma$ ,

- Case 2:  $\sigma \vdash e \Rightarrow \text{false}$ , then both statements result in  $\sigma$ .

$$\frac{\dots}{\sigma \vdash e \Rightarrow \text{false}} \\ \sigma \vdash \text{while } e \text{ do } s \Rightarrow \sigma$$

$$\frac{\dots \quad \dots}{\sigma \vdash e \Rightarrow \text{false} \quad \sigma \vdash \text{skip} \Rightarrow \sigma} \\ \sigma \vdash \text{if } e \text{ then } (s; \text{while } e \text{ do } s) \text{ else skip} \Rightarrow \sigma$$

## Theorem (Determinism of IMP)

$$\forall s, \sigma, \sigma', \sigma''. \sigma \vdash s \Rightarrow \sigma' \wedge \sigma \vdash s \Rightarrow \sigma'' \implies \sigma' = \sigma''$$

We can prove this theorem by induction on the inference rules of IMP.

For base cases, we can easily show that there is only one possible evaluation for `skip` and `x := e`.

For inductive cases, we need to use the induction hypothesis to show that there is only one possible evaluation.

For example, consider the case of `while e do s` and  $\sigma \vdash e \Rightarrow \text{true}$ :

$$\frac{\frac{\dots}{\sigma \vdash e \Rightarrow \text{true}} \quad \frac{\dots}{\sigma \vdash s \Rightarrow \sigma_1} \quad \frac{\dots}{\sigma_1 \vdash \text{while } e \text{ do } s \Rightarrow \sigma'_1}}{\sigma \vdash \text{while } e \text{ do } s \Rightarrow \sigma'_1}$$

$$\frac{\frac{\dots}{\sigma \vdash e \Rightarrow \text{true}} \quad \frac{\dots}{\sigma \vdash s \Rightarrow \sigma_2} \quad \frac{\dots}{\sigma_2 \vdash \text{while } e \text{ do } s \Rightarrow \sigma'_2}}{\sigma \vdash \text{while } e \text{ do } s \Rightarrow \sigma'_2}$$

By the induction hypothesis, we have  $\sigma_1 = \sigma_2$ . Then, by the induction hypothesis again, we have  $\sigma'_1 = \sigma'_2$ . Thus,  $\sigma'_1 = \sigma'_2$ .

Since we applied induction on the inference rules, we can use the induction hypothesis on the same statement  $s$  without any problem.

The other cases can be proved similarly.

- Small-Step Operational Semantics

Jihyeok Park  
jihyeok\_park@korea.ac.kr  
<https://plrg.korea.ac.kr>