Lecture 2 – Big-Step Operational Semantics

AAA551: Programming Language Theory

Jihyeok Park

2026 Spring

Recall

- 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

Contents

Contents

Programming Languages

PLRG

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

Programming Languages

PLRG

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
- $\texttt{x}$:= 1; $\texttt{x}$ + 2 evaluates to 3
- $\texttt{x}$:= 2; $\texttt{y}$:= 3; $\texttt{x}$ * $\texttt{y}$ evaluates to 6

Syntax

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:

$$e ::= n \mid e \texttt{ + } e \mid e \texttt{ * } e \mid x \texttt{ := } e; \ e \mid x$$

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

Syntax

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:

$$e ::= n \mid e \texttt{ + } e \mid e \texttt{ * } e \mid x \texttt{ := } e\texttt{; } e \mid x$$

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)
```

Semantics

Three approaches to **formal semantics**:

- **Operational** $\sigma \vdash e \Rightarrow v$
  - *How* is a program executed?
  - Useful for implementation of compilers and interpreters.

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

- **Denotational** $[\![e]\!]$
  - *What* is the mathematical object for a program?
  - Useful for theoretical foundations.

$$[\![e_1 \texttt{ + } e_2]\!] = [\![e_1]\!] + [\![e_2]\!]$$

- **Axiomatic** $\vdash \{\phi\}\ e\ \{\phi'\}$
  - *Which properties* does a program satisfy?
  - Useful for proving program properties and correctness.

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

Contents

Binary Relations

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

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

Binary Relations

- 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 $A \times B$.

$$R \subseteq A \times B$$

Binary Relations

- 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: $\varnothing$
  - total relation: $A \times B$
  - identity relation: $\{(a, a) \mid a \in A\}$

Function

- A (total) **function** $f : A \to B$ 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 – $\forall a \in A, \forall b, b' \in B.((a, b) \in f \land (a, b') \in f) \implies b = b'$

Function

- A (total) **function** $f : A \to B$ 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 – $\forall a \in A, \forall b, b' \in B.((a, b) \in f \land (a, b') \in f) \implies b = b'$

- A **partial function** $f : A \rightharpoonup B$ from a set A to a set B is a binary relation only satisfying the uniqueness property.

Function

- A (total) **function** $f : A \to B$ 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 – $\forall a \in A, \forall b, b' \in B.((a, b) \in f \land (a, b') \in f) \implies b = b'$

- A **partial function** $f : A \rightharpoonup B$ 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 $\exists b \in B.(a, b) \in f$:

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

Function

- A (total) **function** $f : A \to B$ 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 – $\forall a \in A, \forall b, b' \in B.((a, b) \in f \land (a, b') \in f) \implies b = b'$

- A **partial function** $f : A \rightharpoonup B$ 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 $\exists b \in B.(a, b) \in f$:

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

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

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

Inductively Defined Sets

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

Inductively Defined Sets

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

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

$$\frac{\mathit{premise}_1 \qquad \mathit{premise}_2 \qquad \ldots \qquad \mathit{premise}_n}{\mathit{conclusion}}$$

Inductively Defined Sets

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

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

$$\frac{\textit{premise}_1 \qquad \textit{premise}_2 \qquad \ldots \qquad \textit{premise}_n}{\textit{conclusion}}$$

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

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

Inductively Defined Sets

PLRG

For example, consider the following BNF grammar for VAE:

$$e ::= n \mid e \texttt{ + } e \mid e \texttt{ * } e \mid x \texttt{ := } e\texttt{; } e \mid x$$

It can be translated into the following inference rules:

$$\frac{}{n \in \mathbb{E}} \qquad \frac{e_1 \in \mathbb{E} \qquad e_2 \in \mathbb{E}}{e_1 \texttt{ + } e_2 \in \mathbb{E}} \qquad \frac{e_1 \in \mathbb{E} \qquad e_2 \in \mathbb{E}}{e_1 \texttt{ * } e_2 \in \mathbb{E}}$$

$$\frac{e_1 \in \mathbb{E} \qquad e_2 \in \mathbb{E}}{x \texttt{ := } e_1\texttt{; } e_2 \in \mathbb{E}} \qquad \frac{}{x \in \mathbb{E}}$$

Inductively Defined Sets

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

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

where S is the successor function.

Inductively Defined Sets

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

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

where S is the successor function.

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

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

Contents

Operational Semantics

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**.

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

(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**.

$$e \to e' \to e'' \to \ldots \to n$$

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

Big-Step Operational Semantics – VAE

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

$$e ::= n \mid e \texttt{ + } e \mid e \texttt{ * } e \mid x \texttt{ := } e; \ e \mid x$$

Big-Step Operational Semantics – VAE

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

$$e ::= n \mid e \texttt{ + } e \mid e \texttt{ * } e \mid x \texttt{ := } e; \ e \mid x$$

It means "*under* **environment** σ, *an* **expression** e *evaluates to* **integer** n":

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

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

Big-Step Operational Semantics – VAE

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

$$e ::= n \mid e \texttt{ + } e \mid e \texttt{ * } e \mid x \texttt{ := } e; \ e \mid x$$

It means "*under* **environment** σ, *an* **expression** e *evaluates to* **integer** n":

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

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

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

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

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

Scala Implementation

PLRG

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

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

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

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"))
```

Derivation

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

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

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

$$\frac{}{\varnothing \vdash \texttt{x := 1;}\ \{\texttt{y := 2;}\ \texttt{x + y}\} \Rightarrow}$$

where

Derivation

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

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

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

$$\dfrac{\dfrac{}{\varnothing \vdash 1 \Rightarrow 1} \qquad \dfrac{\dfrac{}{\sigma_0 \vdash 2 \Rightarrow 2} \qquad \dfrac{\dfrac{\texttt{x} \in \mathsf{dom}(\sigma_1)}{\sigma_1 \vdash \texttt{x} \Rightarrow 1} \qquad \dfrac{y \in \mathsf{dom}(\sigma_1)}{\sigma_1 \vdash y \Rightarrow 2}}{\sigma_1 \vdash \texttt{x + } y \Rightarrow 3}}{\sigma_0 \vdash y \texttt{ := } 2;\ \texttt{x + } y \Rightarrow 3}}{\varnothing \vdash \texttt{x := } 1;\ \{y \texttt{ := } 2;\ \texttt{x + } y\} \Rightarrow 3}$$

where

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

Contents

Simple Imperative Language – IMP

Let's construct a simple imperative programming language IMP:

$$
\begin{array}{lll}
\text{Expressions} & e & ::= n \mid x \mid e \texttt{ + } e \mid e \texttt{ * } e \mid e \texttt{ < } e \mid \texttt{true} \mid \texttt{false} \\
\text{Statements} & s & ::= \texttt{skip} \\
 & & \mid x \texttt{ := } e \\
 & & \mid s \texttt{;} s \\
 & & \mid \texttt{if } e \texttt{ then } s \texttt{ else } s \\
 & & \mid \texttt{while } e \texttt{ do } s \\
\text{Values} & v & ::= n \mid \texttt{true} \mid \texttt{false}
\end{array}
$$

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

Simple Imperative Language – IMP

Let's construct a simple imperative programming language IMP:

$$
\begin{array}{lll}
\text{Expressions} & e & ::= n \mid x \mid e \texttt{ + } e \mid e \texttt{ * } e \mid e \texttt{ < } e \mid \texttt{true} \mid \texttt{false} \\
\text{Statements} & s & ::= \texttt{skip} \\
 & & \mid x \texttt{ := } e \\
 & & \mid s \texttt{;} s \\
 & & \mid \texttt{if } e \texttt{ then } s \texttt{ else } s \\
 & & \mid \texttt{while } e \texttt{ do } s \\
\text{Values} & v & ::= n \mid \texttt{true} \mid \texttt{false}
\end{array}
$$

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

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

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

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

IMP – Expressions

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

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

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

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

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

IMP – Statements (1)

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

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

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

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

IMP – Statements (2)

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

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

$$\frac{\sigma \vdash e \Rightarrow \texttt{true} \qquad \sigma \vdash s \Rightarrow \sigma' \qquad \sigma' \vdash \texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma''}{\sigma \vdash \texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma''}$$

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

Equivalence of Statements

PLRG

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:

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

Equivalence of Statements

PLRG

For example, the following two statements are equivalent:

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

Equivalence of Statements

PLRG

For example, the following two statements are equivalent:

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

For a given environment σ,

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

$$\frac{\dfrac{\cdots}{\sigma \vdash e \Rightarrow \texttt{true}} \qquad \dfrac{\cdots}{\sigma \vdash s \Rightarrow \sigma'} \qquad \dfrac{\cdots}{\sigma' \vdash \texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma''}}{\sigma \vdash \texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma''}$$

$$\frac{\dfrac{\cdots}{\sigma \vdash e \Rightarrow \texttt{true}} \qquad \dfrac{\dfrac{\cdots}{\sigma \vdash s \Rightarrow \sigma'} \qquad \dfrac{\cdots}{\sigma' \vdash \texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma''}}{\sigma \vdash s;\texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma''}}{\sigma \vdash \texttt{if}\ e\ \texttt{then}\ (s;\texttt{while}\ e\ \texttt{do}\ s)\ \texttt{else skip} \Rightarrow \sigma''}$$

Equivalence of Statements

PLRG

For example, the following two statements are equivalent:

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

For a given environment σ,

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

$$\frac{\dfrac{\cdots}{\sigma \vdash e \Rightarrow \texttt{false}}}{\sigma \vdash \texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma}$$

$$\frac{\dfrac{\cdots}{\sigma \vdash e \Rightarrow \texttt{false}} \qquad \dfrac{}{\sigma \vdash \texttt{skip} \Rightarrow \sigma}}{\sigma \vdash \texttt{if}\ e\ \texttt{then}\ (s;\texttt{while}\ e\ \texttt{do}\ s)\ \texttt{else skip} \Rightarrow \sigma}$$

Determinism

Theorem (Determinism of IMP)

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

Determinism

PLRG

Theorem (Determinism of IMP)

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

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.

Determinism – Proof

PLRG

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

$$\frac{\dots \qquad \dots \qquad \dots}{\sigma \vdash e \Rightarrow \texttt{true} \qquad \sigma \vdash s \Rightarrow \sigma_1 \qquad \sigma_1 \vdash \texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma_1'}{\sigma \vdash \texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma_1'}$$

$$\frac{\dots \qquad \dots \qquad \dots}{\sigma \vdash e \Rightarrow \texttt{true} \qquad \sigma \vdash s \Rightarrow \sigma_2 \qquad \sigma_2 \vdash \texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma_2'}{\sigma \vdash \texttt{while}\ e\ \texttt{do}\ s \Rightarrow \sigma_2'}$$

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.

Next Lecture

- Small-Step Operational Semantics

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr