

# Lecture 5 – Identifiers (2)

## COSE212: Programming Languages

Jihyeok Park



2024 Fall

- **Identifiers**
  - Bound identifiers
  - Free identifiers
  - Shadowing
- **VAE – AE with variables**
  - Concrete syntax
  - Abstract syntax

- **Identifiers**
  - Bound identifiers
  - Free identifiers
  - Shadowing
- **VAE – AE with variables**
  - Concrete syntax
  - Abstract syntax
- In this lecture, we will
  - implement the **interpreter** for VAE
  - define the **natural semantics** for VAE

## 1. Evaluation with Environments

## 2. Interpreter and Natural Semantics for VAE

- Numbers

- Addition and Multiplication

- Variable Definition

- Variable Lookup

## 3. Examples

## 1. Evaluation with Environments

## 2. Interpreter and Natural Semantics for VAE

Numbers

Addition and Multiplication

Variable Definition

Variable Lookup

## 3. Examples

Let's evaluate the following VAE expressions:

```
/* VAE */  
val x = 1; {           // [ x -> 1 ]  
  val y = 2; {         // [ x -> 1, y -> 2 ]  
    x + y              // x + y = 1 + 2 = 3  
  }  
}
```

Let's evaluate the following VAE expressions:

```
/* VAE */  
val x = 1; {           // [ x -> 1 ]  
  val y = 2; {         // [ x -> 1, y -> 2 ]  
    x + y              // x + y = 1 + 2 = 3  
  }  
}
```

How to evaluate the expression  $x + y$  into the value 3?

$$\vdash x + y \Rightarrow 3$$

Let's evaluate the following VAE expressions:

```
/* VAE */  
val x = 1; {           // [ x -> 1 ]  
  val y = 2; {         // [ x -> 1, y -> 2 ]  
    x + y              // x + y = 1 + 2 = 3  
  }  
}
```

How to evaluate the expression  $x + y$  into the value 3?

$$\vdash x + y \Rightarrow 3$$

We need to keep track of the **environment** that maps identifiers to values:

$$[x \mapsto 1, y \mapsto 2] \vdash x + y \Rightarrow 3$$



```
type Value = BigInt           // values
def interp(expr: Expr): Value = ... // interpreter
```

For AE, the interpreter takes an expression and returns a number.

$$\vdash e \Rightarrow n$$

```
type Value = BigInt                                // values
type Env = Map[String, Value]                     // environments
def interp(expr: Expr, env: Env): Value = ...    // interpreter
```

For VAE, we extend the interpreter to take an **environment** as well.

$$\sigma \vdash e \Rightarrow n$$

We read it as “*with the **environment**  $\sigma$ , the **expression**  $e$  evaluates to the **number**  $n$* ”

```
type Value = BigInt                                // values
type Env = Map[String, Value]                     // environments
def interp(expr: Expr, env: Env): Value = ...    // interpreter
```

For VAE, we extend the interpreter to take an **environment** as well.

$$\sigma \vdash e \Rightarrow n$$

We read it as “*with the **environment**  $\sigma$ , the **expression**  $e$  evaluates to the **number**  $n$* ”

For example, the interpreter should be able to evaluate like this:

```
val env : Env    = Map("x" -> 1, "y" -> 2)    // [ x -> 1, y -> 2 ]
val expr: Expr   = Expr("x + y")              // Add(Id("x"), Id("y"))
val v    : Value = interp(expr, env)           // 3
```

$$[x \mapsto 1, y \mapsto 2] \vdash x + y \Rightarrow 3$$

## 1. Evaluation with Environments

## 2. Interpreter and Natural Semantics for VAE

- Numbers

- Addition and Multiplication

- Variable Definition

- Variable Lookup

## 3. Examples

For VAE, we need to 1) implement the **interpreter** with **environments**

```
def interp(expr: Expr, env: Env): Value = ???
```

For VAE, we need to 1) implement the **interpreter** with **environments**

```
def interp(expr: Expr, env: Env): Value = ???
```

and 2) define the **natural semantics** with **environments**.

$$\sigma \vdash e \Rightarrow n$$

|             |                        |       |
|-------------|------------------------|-------|
| Expressions | $e ::= n$              | (Num) |
|             | $e + e$                | (Add) |
|             | $e * e$                | (Mul) |
|             | $\text{val } x = e; e$ | (Val) |
|             | $x$                    | (Id)  |

where

|              |   |          |
|--------------|---|----------|
| Environments | $\sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{Z}$ | (Env)    |
| Integers     | $n \in \mathbb{Z}$  | (BigInt) |
| Identifiers  | $x \in \mathbb{X}$  | (String) |

```
def interp(expr: Expr, env: Env): Value = expr match
  case Num(n)          => ???
  case Add(l, r)        => ???
  case Mul(l, r)        => ???
  case Val(x, e, b)     => ???
  case Id(x)            => ???
```

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

$$\text{NUM} \frac{\text{???}}{\sigma \vdash n \Rightarrow \text{???}}$$

$$\text{ADD} \frac{\text{???}}{\sigma \vdash e_1 + e_2 \Rightarrow \text{???}}$$

$$\text{MUL} \frac{\text{???}}{\sigma \vdash e_1 * e_2 \Rightarrow \text{???}}$$

$$\text{VAL} \frac{\text{???}}{\sigma \vdash \text{val } x = e_1; e_2 \Rightarrow \text{???}}$$

$$\text{ID} \frac{\text{???}}{\sigma \vdash x \Rightarrow \text{???}}$$

```
def interp(expr: Expr, env: Env): Value = expr match
  case Num(n)      => ???
  ...
```

$$\sigma \vdash e \Rightarrow n$$

$$\text{NUM} \frac{\text{???}}{\sigma \vdash n \Rightarrow \text{???}}$$



```
def interp(expr: Expr, env: Env): Value = expr match
  case Num(n)      => n
  ...
```

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

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

With the **environment**  $\sigma$ , the **expression**  $n$  evaluates to the **number**  $n$ .

```
def interp(expr: Expr, env: Env): Value = expr match
  ...
  case Add(l, r)    => ???
  ...
```

$$\sigma \vdash e \Rightarrow n$$

$$\text{ADD} \frac{\text{???}}{\sigma \vdash e_1 + e_2 \Rightarrow \text{???}}$$

```
def interp(expr: Expr, env: Env): Value = expr match
...
case Add(l, r)    => interp(l, env) + interp(r, env)
...
```

$$\sigma \vdash e \Rightarrow n$$

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

With the **environment**  $\sigma$ , the **expression**  $e_1 + e_2$  evaluates to the **number**  $n_1 + n_2$  when

- With the **environment**  $\sigma$ , the **expression**  $e_1$  evaluates to the **number**  $n_1$ .
- With the **environment**  $\sigma$ , the **expression**  $e_2$  evaluates to the **number**  $n_2$ .

```
def interp(expr: Expr, env: Env): Value = expr match
...
case Mul(l, r)    => interp(l, env) * interp(r, env)
...
```

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

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

With the **environment**  $\sigma$ , the **expression**  $e_1 * e_2$  evaluates to the **number**  $n_1 \times n_2$  when

- With the **environment**  $\sigma$ , the **expression**  $e_1$  evaluates to the **number**  $n_1$ .
- With the **environment**  $\sigma$ , the **expression**  $e_2$  evaluates to the **number**  $n_2$ .

```
def interp(expr: Expr, env: Env): Value = expr match
...
case Val(x, e, b) => ???
...
```

$$\sigma \vdash e \Rightarrow n$$

$$\text{VAL} \frac{\text{???}}{\sigma \vdash \text{val } x = e_1; e_2 \Rightarrow \text{???}}$$

```
def interp(expr: Expr, env: Env): Value = expr match
  ...
  case Val(x, e, b) => ... interp(e, env) ...
  ...
```

$$\sigma \vdash e \Rightarrow n$$

$$\text{VAL} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \dots}{\sigma \vdash \text{val } x = e_1; e_2 \Rightarrow ???}$$

With the **environment**  $\sigma$ , the **expression**  $\text{val } x = e_1; e_2$  evaluates to the **number**  $???$  when

- ① With the **environment**  $\sigma$ , the **expression**  $e_1$  evaluates to the **number**  $n_1$ .
- ② ...

```
def interp(expr: Expr, env: Env): Value = expr match
  ...
  case Val(x, e, b) => ... env + (x -> interp(e, env)) ...
  ...
```

$$\sigma \vdash e \Rightarrow n$$

$$\text{VAL} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma[x \mapsto n_1] \quad \dots}{\sigma \vdash \text{val } x = e_1; e_2 \Rightarrow ???}$$

With the **environment**  $\sigma$ , the **expression**  $\text{val } x = e_1; e_2$  evaluates to the **number**  $???$  when

- ① With the **environment**  $\sigma$ , the **expression**  $e_1$  evaluates to the **number**  $n_1$ .
- ② With the **environment**  $\sigma[x \mapsto n_1]$ , ...

```
def interp(expr: Expr, env: Env): Value = expr match
  ...
  case Val(x, e, b) => interp(b, env + (x -> interp(e, env)))
  ...
```

$$\sigma \vdash e \Rightarrow n$$

$$\text{VAL} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma[x \mapsto n_1] \vdash e_2 \Rightarrow n_2}{\sigma \vdash \text{val } x = e_1; e_2 \Rightarrow n_2}$$

With the **environment**  $\sigma$ , the **expression**  $\text{val } x = e_1; e_2$  evaluates to the **number**  $n_2$  when

- ① With the **environment**  $\sigma$ , the **expression**  $e_1$  evaluates to the **number**  $n_1$ .
- ② With the **environment**  $\sigma[x \mapsto n_1]$ , the **expression**  $e_2$  evaluates to the **number**  $n_2$ .



```
def interp(expr: Expr, env: Env): Value = expr match
...
case Id(x)           => ???
...
```

$$\sigma \vdash e \Rightarrow n$$

$$\text{ID} \frac{\text{???}}{\sigma \vdash x \Rightarrow \text{???}}$$

```
def interp(expr: Expr, env: Env): Value = expr match
  ...
  case Id(x)          => env.getOrElse(x, error(s"free identifier: $x"))
  ...
```

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

$$\text{ID} \frac{x \in \text{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

With the **environment**  $\sigma$ , the **expression**  $x$  evaluates to the **number**  $\sigma(x)$  when

- 1 The **variable**  $x$  is in the domain of the **environment**  $\sigma$ .

## 1. Evaluation with Environments

## 2. Interpreter and Natural Semantics for VAE

Numbers

Addition and Multiplication

Variable Definition

Variable Lookup

## 3. Examples

# Example 1

$$\begin{array}{c} \text{NUM} \frac{}{\emptyset \vdash 1 \Rightarrow 1} \quad \text{ID} \frac{x \in \text{Domain}([x \mapsto 1])}{[x \mapsto 1] \vdash x \Rightarrow 1} \quad \text{NUM} \frac{}{[x \mapsto 1] \vdash 2 \Rightarrow 2} \\ \text{VAL} \frac{\text{ADD} \frac{[x \mapsto 1] \vdash x \Rightarrow 1 \quad [x \mapsto 1] \vdash 2 \Rightarrow 2}{[x \mapsto 1] \vdash x + 2 \Rightarrow 3}}{\emptyset \vdash \text{val } x = 1; x + 2 \Rightarrow 3} \end{array}$$

$$\begin{array}{c}
 \text{NUM} \frac{}{\emptyset \vdash 1 \Rightarrow 1} \quad \text{ID} \frac{x \in \text{Domain}([x \mapsto 1])}{[x \mapsto 1] \vdash x \Rightarrow 1} \quad \text{NUM} \frac{}{[x \mapsto 1] \vdash 2 \Rightarrow 2} \\
 \text{VAL} \frac{\text{ADD} \frac{}{[x \mapsto 1] \vdash x + 2 \Rightarrow 3}}{\emptyset \vdash \text{val } x = 1; x + 2 \Rightarrow 3}
 \end{array}$$

We can name environments  $\sigma_i$  to make the derivation tree concise.

$$\begin{array}{c}
 \text{NUM} \frac{}{\emptyset \vdash 1 \Rightarrow 1} \quad \text{ID} \frac{x \in \text{Domain}(\sigma_0)}{\sigma_0 \vdash x \Rightarrow 1} \quad \text{NUM} \frac{}{\sigma_0 \vdash 2 \Rightarrow 2} \\
 \text{VAL} \frac{\text{ADD} \frac{}{\sigma_0 \vdash x + 2 \Rightarrow 3}}{\emptyset \vdash \text{val } x = 1; x + 2 \Rightarrow 3}
 \end{array}$$

where

$$\sigma_0 = [x \mapsto 1]$$

## Example 2

VAL  $\frac{}{\emptyset \vdash \text{val } x = 1; \{\text{val } y = 2; x + y\} \Rightarrow}$

where

$$\begin{array}{c}
 \text{NUM} \frac{}{\emptyset \vdash 1 \Rightarrow 1} \quad \text{VAL} \frac{\text{NUM} \frac{}{\sigma_0 \vdash 2 \Rightarrow 2} \quad \text{ADD} \frac{\text{ID} \frac{x \in \text{Domain}(\sigma_1)}{\sigma_1 \vdash x \Rightarrow 1} \quad \text{ID} \frac{y \in \text{Domain}(\sigma_1)}{\sigma_1 \vdash y \Rightarrow 2}}{\sigma_1 \vdash x + y \Rightarrow 3}}{\sigma_0 \vdash \text{val } y = 2; x + y \Rightarrow 3}} \\
 \text{VAL} \frac{}{\emptyset \vdash \text{val } x = 1; \{\text{val } y = 2; x + y\} \Rightarrow 3}
 \end{array}$$

where

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

## Example 3

VAL  $\frac{}{\emptyset \vdash \text{val } x = 1; \{ \text{val } x = 2; x \} + x \Rightarrow}$

where



## Example 3

$$\begin{array}{c}
 \text{NUM} \frac{}{\emptyset \vdash 1 \Rightarrow 1} \quad \text{VAL} \frac{}{\emptyset \vdash 1 \Rightarrow 1} \quad \text{ADD} \frac{\text{NUM} \frac{}{\sigma_0 \vdash 2 \Rightarrow 2} \quad \text{ID} \frac{x \in \text{Domain}(\sigma_1)}{\sigma_1 \vdash x \Rightarrow 2}}{\sigma_0 \vdash \text{val } x = 2; x \Rightarrow 2} \quad \text{ID} \frac{x \in \text{Domain}(\sigma_0)}{\sigma_0 \vdash x \Rightarrow 1}}{\sigma_0 \vdash \{\text{val } x = 2; x\} + x \Rightarrow 3} \\
 \text{VAL} \frac{}{\emptyset \vdash \text{val } x = 1; \{\text{val } x = 2; x\} + x \Rightarrow 3}
 \end{array}$$

where

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

$$\begin{array}{c}
 \text{NUM} \frac{}{\emptyset \vdash 1 \Rightarrow 1} \quad \text{ID} \frac{x \in \text{Domain}(\sigma_0)}{\sigma_0 \vdash x \Rightarrow 1} \\
 \text{VAL} \frac{}{\emptyset \vdash \text{val } x = 1; x \Rightarrow 1} \quad \text{ID} \frac{x \notin \text{Domain}(\emptyset)}{\emptyset \vdash x \Rightarrow \text{FAIL}} \\
 \text{ADD} \frac{}{\emptyset \vdash \{\text{val } x = 1; x\} + x \Rightarrow \text{FAIL}}
 \end{array}$$

where

$$\sigma_0 = [x \mapsto 1]$$

We cannot draw the derivation tree for this example because of the **free variable**  $x$  in the right-hand side of the addition.

```
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, e, b)     => interp(b, env + (x -> interp(e, env)))
  case Id(x)            => env.getOrElse(x, error(s"free identifier: $x"))
```

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

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

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

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

$$\text{VAL} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma[x \mapsto n_1] \vdash e_2 \Rightarrow n_2}{\sigma \vdash \text{val } x = e_1; e_2 \Rightarrow n_2}$$

$$\text{ID} \frac{x \in \text{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

<https://github.com/ku-plrg-classroom/docs/tree/main/cose212/vae>

- Please see above document on GitHub:
  - Implement `interp` function.
  - Implement `freeIds` function.
  - Implement `bindingIds` function.
  - Implement `boundIds` function.
  - Implement `shadowedIds` function.
- It is just an exercise, and you **don't need to submit** anything.
- However, some exam questions might be related to this exercise.

- First-Order Functions

Jihyeok Park

[jihyeok\\_park@korea.ac.kr](mailto:jihyeok_park@korea.ac.kr)

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