Lecture 5 – Identifiers (2)

COSE212: Programming Languages

Jihyeok Park

PLRG

2024 Fall

Recall

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

Recall

- **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

Contents

Contents

Evaluation with Environments

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
  }
}
```

Evaluation with Environments

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?

$$\vdash \mathtt{x} + \mathtt{y} \Rightarrow 3$$

Evaluation with Environments

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?

$$\vdash \texttt{x + y} \Rightarrow 3$$

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

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

Evaluation with Environments

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

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

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

Evaluation with Environments

```
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.

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

We read it as "*with the* **environment** σ*, the* **expression** e *evaluates to the* **number** n"

Evaluation with Environments

PLRG

```
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.

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

We read it as "*with the* **environment** σ, *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
```

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

Contents

Interpreter and Natural Semantics for VAE

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**

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

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

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

$$
\begin{array}{llll}
\text{Expressions} & e ::= & n & (\texttt{Num}) \\
 & \mid & e \ \texttt{+} \ e & (\texttt{Add}) \\
 & \mid & e \ \texttt{*} \ e & (\texttt{Mul}) \\
 & \mid & \texttt{val} \ x \ \texttt{=} \ e\texttt{;} \ e & (\texttt{Val}) \\
 & \mid & x & (\texttt{Id})
\end{array}
$$

where

$$
\begin{array}{lll}
\text{Environments} & \sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{Z} & (\texttt{Env}) \\
\text{Numbers} & n \in \mathbb{Z} & (\texttt{BigInt}) \\
\text{Identifiers} & x \in \mathbb{X} & (\texttt{String})
\end{array}
$$

Interpreter and Natural Semantics for VAE

```
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)          => ???
```

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

$$\textsc{Num}\ \frac{???}{\sigma \vdash n \Rightarrow ???}$$

$$\textsc{Add}\ \frac{???}{\sigma \vdash e_1\ \texttt{+}\ e_2 \Rightarrow ???} \qquad \textsc{Mul}\ \frac{???}{\sigma \vdash e_1\ \texttt{*}\ e_2 \Rightarrow ???}$$

$$\textsc{Val}\ \frac{???}{\sigma \vdash \texttt{val}\ x\ \texttt{=}\ e_1\texttt{;}\ e_2 \Rightarrow ???} \qquad \textsc{Id}\ \frac{???}{\sigma \vdash x \Rightarrow ???}$$

Numbers

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

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

$$\textsc{Num}\ \frac{???}{\sigma \vdash n \Rightarrow ???}$$

Numbers

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

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

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

With the **environment** σ, the **expression** n evaluates to the **number** n.

Addition

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

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

$$\textsc{Add}\ \frac{???}{\sigma \vdash e_1 \texttt{ + } e_2 \Rightarrow ???}$$

Addition

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

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

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

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

- With the **environment** σ, the **expression** e_1 evaluates to the **number** n_1.
- With the **environment** σ, the **expression** e_2 evaluates to the **number** n_2.

Multiplication

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

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

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

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

- With the **environment** σ, the **expression** e_1 evaluates to the **number** n_1.
- With the **environment** σ, the **expression** e_2 evaluates to the **number** n_2.

Variable Definition

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

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

$$\textsc{Val}\ \frac{\texttt{???}}{\sigma \vdash \texttt{val}\ x\ \texttt{=}\ e_1;\ e_2 \Rightarrow \texttt{???}}$$

Variable Definition

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

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

$$\textsc{Val}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \ldots}{\sigma \vdash \texttt{val}\ x\ \texttt{=}\ e_1\texttt{;}\ e_2 \Rightarrow \texttt{???}}$$

With the **environment** σ, the **expression** `val` x `=` e_1`;` e_2 evaluates to the **number** ??? when

1. With the **environment** σ, the **expression** e_1 evaluates to the **number** n_1.
2. ...

Variable Definition

PLRG

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

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

$$\textsc{Val}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma[x \mapsto n_1] \qquad \ldots}{\sigma \vdash \texttt{val}\ x\ \texttt{=}\ e_1\texttt{;}\ e_2 \Rightarrow \texttt{???}}$$

With the **environment** σ, the **expression** `val` x `=` e_1`;` e_2 evaluates to the **number** ??? when

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

Variable Definition

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

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

$$\textsc{Val}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma[x \mapsto n_1] \vdash e_2 \Rightarrow n_2}{\sigma \vdash \texttt{val}\ x\ \texttt{=}\ e_1\texttt{;}\ e_2 \Rightarrow n_2}$$

With the **environment** σ, the **expression** `val` x `=` e_1`;` e_2 evaluates to the **number** n_2 when

1. With the **environment** σ, 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.

Variable Lookup

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

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

$$\textsc{Id}\ \frac{???}{\sigma \vdash x \Rightarrow ???}$$

Variable Lookup

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

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

$$\textsc{Id}\ \frac{x \in \mathsf{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

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

1. The **variable** x is in the domain of the **environment** σ.

Contents

Example 1

$$
\textsc{Val}\;\dfrac{
\textsc{Num}\;\dfrac{}{\varnothing \vdash 1 \Rightarrow 1}
\qquad
\textsc{Add}\;\dfrac{
\textsc{Id}\;\dfrac{\mathtt{x} \in \mathsf{Domain}([\mathtt{x} \mapsto 1])}{[\mathtt{x} \mapsto 1] \vdash \mathtt{x} \Rightarrow 1}
\quad
\textsc{Num}\;\dfrac{}{[\mathtt{x} \mapsto 1] \vdash 2 \Rightarrow 2}
}{[\mathtt{x} \mapsto 1] \vdash \mathtt{x}\ \texttt{+}\ 2 \Rightarrow 3}
}{\varnothing \vdash \mathtt{val}\ \mathtt{x}\ \texttt{=}\ 1;\ \mathtt{x}\ \texttt{+}\ 2 \Rightarrow 3}
$$

Example 1

$$\textsc{Val}\ \dfrac{\textsc{Num}\ \dfrac{}{\varnothing \vdash 1 \Rightarrow 1} \quad \textsc{Add}\ \dfrac{\textsc{Id}\ \dfrac{\texttt{x} \in \mathsf{Domain}([\texttt{x} \mapsto 1])}{[\texttt{x} \mapsto 1] \vdash \texttt{x} \Rightarrow 1} \quad \textsc{Num}\ \dfrac{}{[\texttt{x} \mapsto 1] \vdash 2 \Rightarrow 2}}{[\texttt{x} \mapsto 1] \vdash \texttt{x + } 2 \Rightarrow 3}}{\varnothing \vdash \texttt{val x = } 1\texttt{; x + } 2 \Rightarrow 3}$$

We can name environments σ_i to make the derivation tree concise.

$$\textsc{Val}\ \dfrac{\textsc{Num}\ \dfrac{}{\varnothing \vdash 1 \Rightarrow 1} \quad \textsc{Add}\ \dfrac{\textsc{Id}\ \dfrac{\texttt{x} \in \mathsf{Domain}(\sigma_0)}{\sigma_0 \vdash \texttt{x} \Rightarrow 1} \quad \textsc{Num}\ \dfrac{}{\sigma_0 \vdash 2 \Rightarrow 2}}{\sigma_0 \vdash \texttt{x + } 2 \Rightarrow 3}}{\varnothing \vdash \texttt{val x = } 1\texttt{; x + } 2 \Rightarrow 3}$$

where

$$\sigma_0 = [\texttt{x} \mapsto 1]$$

Example 2

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

where

Example 2

$$
\dfrac{\textsc{Num}\;\dfrac{}{\varnothing \vdash 1 \Rightarrow 1} \quad \textsc{Val}\;\dfrac{\textsc{Num}\;\dfrac{}{\sigma_0 \vdash 2 \Rightarrow 2} \quad \textsc{Add}\;\dfrac{\textsc{Id}\;\dfrac{\texttt{x} \in \mathsf{Domain}(\sigma_1)}{\sigma_1 \vdash \texttt{x} \Rightarrow 1} \quad \textsc{Id}\;\dfrac{y \in \mathsf{Domain}(\sigma_1)}{\sigma_1 \vdash y \Rightarrow 2}}{\sigma_1 \vdash \texttt{x + } y \Rightarrow 3}}{\sigma_0 \vdash \texttt{val } y \texttt{ = } 2\texttt{; x + } y \Rightarrow 3}}{\textsc{Val}\quad \varnothing \vdash \texttt{val x = } 1\texttt{; } \{\texttt{val } y \texttt{ = } 2\texttt{; x + } y\} \Rightarrow 3}
$$

where

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

Example 3

$$\textsc{Val}\ \frac{}{\varnothing \vdash \texttt{val x = } 1;\ \{\texttt{val x = } 2;\ \texttt{x}\} \texttt{ + x} \Rightarrow}$$

where

Example 3

$$
\textsc{Val}\ \dfrac{\textsc{Num}\ \dfrac{}{\varnothing \vdash 1 \Rightarrow 1} \quad \textsc{Add}\ \dfrac{\textsc{Val}\ \dfrac{\textsc{Num}\ \dfrac{}{\sigma_0 \vdash 2 \Rightarrow 2} \quad \textsc{Id}\ \dfrac{\texttt{x} \in \mathsf{Domain}(\sigma_1)}{\sigma_1 \vdash \texttt{x} \Rightarrow 2}}{\sigma_0 \vdash \texttt{val x = } 2\texttt{; x} \Rightarrow 2} \quad \textsc{Id}\ \dfrac{\texttt{x} \in \mathsf{Domain}(\sigma_0)}{\sigma_0 \vdash \texttt{x} \Rightarrow 1}}{\sigma_0 \vdash \{\texttt{val x = } 2\texttt{; x}\}\ \texttt{+ x} \Rightarrow 3}}{\varnothing \vdash \texttt{val x = } 1\texttt{; } \{\texttt{val x = } 2\texttt{; x}\}\ \texttt{+ x} \Rightarrow 3}
$$

where

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

Example 4

$$
\textsc{Add}\ \dfrac{\textsc{Val}\ \dfrac{\textsc{Num}\ \dfrac{}{\varnothing \vdash 1 \Rightarrow 1} \quad \textsc{Id}\ \dfrac{\texttt{x} \in \mathsf{Domain}(\sigma_0)}{\sigma_0 \vdash \texttt{x} \Rightarrow 1}}{\varnothing \vdash \texttt{val x = } 1\texttt{; x} \Rightarrow 1} \quad \textsc{Id}\ \dfrac{\color{red}{\texttt{x} \notin \mathsf{Domain}(\varnothing)}}{\varnothing \vdash \texttt{x} \Rightarrow \color{red}{\mathsf{FAIL}}}}{\varnothing \vdash \{\texttt{val x = } 1\texttt{; x}\}\texttt{ + x} \Rightarrow \color{red}{\mathsf{FAIL}}}
$$

where

$$\sigma_0 \quad = \quad [\texttt{x} \mapsto 1]$$

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

Summary

```scala
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"))
```

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

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

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

$$\textsc{Val}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma[x \mapsto n_1] \vdash e_2 \Rightarrow n_2}{\sigma \vdash \texttt{val}\ x \texttt{ = } e_1\texttt{;}\ e_2 \Rightarrow n_2} \qquad \textsc{Id}\ \frac{x \in \mathsf{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

Exercise #2

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.

Next Lecture

- First-Order Functions

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