Lecture 11 – Mutable Variables
COSE212: Programming Languages

Jihyeok Park

PLRG

2025 Fall

- Mutation makes it possible to **change the state** of a program by **updating the contents** of a data structure or a variable.
    - **Mutable data structures**
    - **Mutable variables**

- Mutation makes it possible to **change the state** of a program by **updating the contents** of a data structure or a variable.
  - **Mutable data structures**
  - **Mutable variables**

- **Mutable Data Structures** – Mutable Boxes

- BFAE – FAE with Mutable Boxes
  - Evaluation with Memories

- Mutation makes it possible to **change the state** of a program by **updating the contents** of a data structure or a variable.
  - **Mutable data structures**
  - **Mutable variables**

- **Mutable Data Structures** – Mutable Boxes

- BFAE – FAE with Mutable Boxes
  - Evaluation with Memories

- In this lecture, we will learn **Mutable Variables**

- **MFAE** – FAE with Mutable Variables
  - Concrete and Abstract Syntax
  - Interpreter and Natural Semantics

Contents

Contents

Mutable Variables

A **mutable variable** is a variable whose value can be changed after its initialization.

Mutable Variables

A **mutable variable** is a variable whose value can be changed after its initialization.

Let's define mutable variables in Scala:

```scala
// A mutable variable `x` of type `Int` with 1
var x: Int = 1
x + 2            // 1 + 2 == 3 : Int

// We can reassign a mutable variable `x`
x = 2            // x == 2
x + 2            // 2 + 2 == 4 : Int

// The function `f` is impure because it uses a mutable variable `y`
var y: Int = 1
def f(x: Int): Int = x + y
f(5)             // 5 + 1 == 6 : Int
y = 3
f(5)             // 5 + 3 == 8 : Int
```

Contents

OPLRG

Now, let's extend FAE into MFAE to support **mutable variables**.

```
/* MFAE */
var x = 5;
x;          // 5
x = 8;
x           // 8
```

```
/* MFAE */
var y = 1;
var f = x => { x = x + y; x * x };
f(5);    // (5 + 1) * (5 + 1) = 36
y = 3;
f(5)     // (5 + 3) * (5 + 3) = 64
```

For MFAE, we need to extend **expressions** of FAE with

1. **mutable variables** (var) rather than immutable variables (val)
   (all variables, including parameters, are mutable in MFAE)

2. **assignment** (=)
   (right-associative: e.g., $x = y = e$ is equivalent to $x = (y = e)$)

3. **sequence** of expressions

Concrete Syntax

```
// expressions
<expr> ::= ...
            | "var" <id> "=" <expr> ";" <expr>
            | <id> "=" <expr>
            | <expr> ";" <expr>
```

For MFAE, we need to extend **expressions** of FAE with

1. **mutable variables** (var) rather than immutable variables (val)
   (all variables, including parameters, are mutable in MFAE)

2. **assignment** (=)
   (right-associative: e.g., $x = y = e$ is equivalent to $x = (y = e)$)

3. **sequence** of expressions

Abstract Syntax

Let's define the **abstract syntax** of MFAE in BNF:

$$\text{Expressions} \quad \mathbb{E} \ni e ::= \dots$$

$$
\begin{array}{ll}
| \text{ var } x = e;\ e & (\text{Var}) \\
| \ x = e & (\text{Assign}) \\
| \ e;\ e & (\text{Seq})
\end{array}
$$

```
enum Expr:
  ...
  // mutable variable definition
  case Var(name: String, init: Expr, body: Expr)
  // variable assignment
  case Assign(name: String, expr: Expr)
  // sequence
  case Seq(left: Expr, right: Expr)
```

Contents

We can represent mutable variables by assigning different **addresses** to each variable in the environment and storing their values in the **memory**.

We can represent mutable variables by assigning different **addresses** to each variable in the environment and storing their values in the **memory**.

Let's see how to evaluate the following MFAE expression:

```
/* MFAE */                    *
var x = 5;
x;
x = 8;
x
```

$$\sigma = [$$

$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \ldots$$

$$M \quad = \quad \boxed{\quad | \quad | \quad | \quad | \quad \ldots}$$

We can represent mutable variables by assigning different **addresses** to
each variable in the environment and storing their values in the **memory**.

Let's see how to evaluate the following MFAE expression:

```
/* MFAE */
var x = 5;          *
x;
x = 8;
x
```

$$\sigma = [$$
$$\quad \mathtt{x} \mapsto a_0$$
$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \ldots$$
$$M \quad = \quad \boxed{5} \quad \boxed{} \quad \boxed{} \quad \boxed{} \quad \ldots$$

We can represent mutable variables by assigning different **addresses** to each variable in the environment and storing their values in the **memory**.

Let's see how to evaluate the following MFAE expression:

```
/* MFAE */
var x = 5;
x;          /* 5 */    *
x = 8;
x
```

$$\sigma = [$$
$$\quad \mathtt{x} \mapsto a_0$$
$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \ldots$$
$$M \quad = \quad \boxed{5 \quad \quad \quad \quad \ldots}$$

OPLRG

We can represent mutable variables by assigning different **addresses** to
each variable in the environment and storing their values in the **memory**.

Let's see how to evaluate the following MFAE expression:

```
/* MFAE */
var x = 5;
x;          /* 5 */
x = 8;                        *
x
```

$$\sigma = [\\ \quad \texttt{x} \mapsto a_0 \\ ]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \dots$$

$$M \quad = \quad \boxed{8} \;\;|\;\;|\;\;|\;\; \dots$$

PLRG

We can represent mutable variables by assigning different **addresses** to each variable in the environment and storing their values in the **memory**.

Let's see how to evaluate the following MFAE expression:

```
/* MFAE */
var x = 5;
x;          /* 5 */
x = 8;
x           /* 8 */      *
```

$$\sigma = [$$
$$\quad x \mapsto a_0$$
$$]$$

$$
\begin{array}{ccccccc}
\mathbb{A} & : & a_0 & a_1 & a_2 & a_3 & \dots \\
M & = & \boxed{8} & & & & \dots
\end{array}
$$

◆ PLRG

Here is another MFAE expression:

```
/* MFAE */                     *
var y = 1;
var f = x => {
  x = x + y;
  x * x
};
f(5);
y = 3;
f(5)
```

$$\sigma = [$$

$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \dots$$

$$M \quad = \quad \boxed{\,|\,\,|\,\,|\,} \dots$$

PLRG

Here is another MFAE expression:

```
/* MFAE */
var y = 1;                    *
var f = x => {
  x = x + y;
  x * x
};
f(5);
y = 3;
f(5)
```

$$\sigma = [$$
$$\text{y} \mapsto a_0$$

$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \ldots$$
$$M \quad = \quad \boxed{1} \quad \boxed{} \quad \boxed{} \quad \boxed{} \quad \ldots$$

Example

Here is another MFAE expression:

```
/* MFAE */
var y = 1;
var f = x => {
  x = x + y;
  x * x
};                                    *
f(5);
y = 3;
f(5)
```

$$\sigma = [$$
$$\quad \text{y} \mapsto a_0$$
$$\quad \text{f} \mapsto a_1$$
$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \ldots$$

$M \quad =$	1	v			$\ldots$

where $v = \langle \lambda \text{x}.(\text{x = x + y; x * x}), [\text{y} \mapsto a_0] \rangle$

PLRG

Here is another MFAE expression:

```
/* MFAE */
var y = 1;
var f = x => {          *
  x = x + y;
  x * x
};
f(5);
y = 3;
f(5)
```

$$\sigma = [$$
$$\quad \text{y} \mapsto a_0$$
$$\quad \text{x} \mapsto a_2$$
$$]$$

$$
\begin{array}{ccccc}
\mathbb{A} & : & a_0 & a_1 & a_2 & a_3 & \dots \\
M & = & \boxed{1} & v & 5 & & \dots
\end{array}
$$

where $v = \langle \lambda\text{x}.(\text{x = x + y; x * x}), [\text{y} \mapsto a_0] \rangle$

Here is another MFAE expression:

```
/* MFAE */
var y = 1;
var f = x => {
  x = x + y; /* 5 + 1 */    *
  x * x
};
f(5);
y = 3;
f(5)
```

$$\sigma = [$$
$$\quad \text{y} \mapsto a_0$$
$$\quad \text{x} \mapsto a_2$$
$$]$$

$\mathbb{A}$	:	a_0	a_1	a_2	a_3	...
M	=	1	v	6		...

where $v = \langle \lambda \text{x}.(\text{x = x + y; } \text{x * x}), [\text{y} \mapsto a_0] \rangle$

Here is another MFAE expression:

```
/* MFAE */
var y = 1;
var f = x => {
  x = x + y; /* 5 + 1 */
  x * x     /* 6 * 6 */   *
};
f(5);
y = 3;
f(5)
```

$$\sigma = [$$
$$\quad y \mapsto a_0$$
$$\quad x \mapsto a_2$$
$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \dots$$

$$M = \boxed{\begin{array}{|c|c|c|c|c|} \hline 1 & v & 6 & & \dots \\ \hline \end{array}}$$

where $v = \langle \lambda x.(x = x + y; \ x * x), [y \mapsto a_0] \rangle$

PLRG

Here is another MFAE expression:

```
/* MFAE */
var y = 1;
var f = x => {
  x = x + y;
  x * x
};
f(5);        /* 36 */    *
y = 3;
f(5)
```

$$\sigma = [$$
$$\quad \texttt{y} \mapsto a_0$$
$$\quad \texttt{f} \mapsto a_1$$
$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \ldots$$

$$M \quad = \quad \boxed{\begin{array}{|c|c|c|c|c|} \hline 1 & v & 6 & & \ldots \\ \hline \end{array}}$$

where $v = \langle \lambda \texttt{x}.(\texttt{x = x + y; x * x}), [\texttt{y} \mapsto a_0] \rangle$

Here is another MFAE expression:

```
/* MFAE */
var y = 1;
var f = x => {
  x = x + y;
  x * x
};
f(5);          /* 36 */
y = 3;
f(5)
```

$$\sigma = [$$
$$\quad \texttt{y} \mapsto a_0$$
$$\quad \texttt{f} \mapsto a_1$$
$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \dots$$
$$M \quad = \quad \boxed{3} \quad v \quad 6 \quad \boxed{} \quad \dots$$

where $v = \langle \lambda\texttt{x}.(\texttt{x = x + y; x * x}), [\texttt{y} \mapsto a_0] \rangle$

Example

Here is another MFAE expression:

```
/* MFAE */
var y = 1;
var f = x => {          *
  x = x + y;
  x * x
};
f(5);        /* 36 */
y = 3;
f(5)
```

$$\sigma = [$$
$$\quad \text{y} \mapsto a_0$$
$$\quad \text{x} \mapsto a_3$$
$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \ldots$$
$$M \quad = \quad \boxed{\begin{array}{|c|c|c|c|c|} \hline 3 & v & 6 & 5 & \ldots \\ \hline \end{array}}$$

where $v = \langle \lambda \text{x}.(\text{x = x + y; x * x}), [\text{y} \mapsto a_0] \rangle$

Example

Here is another MFAE expression:

```
/* MFAE */
var y = 1;
var f = x => {
  x = x + y; /* 5 + 3 */   *
  x * x
};
f(5);        /* 36 */
y = 3;
f(5)
```

$$\sigma = [$$
$$\quad \text{y} \mapsto a_0$$
$$\quad \text{x} \mapsto a_3$$
$$]$$

$$\mathbb{A} \;:\; a_0 \quad a_1 \quad a_2 \quad a_3 \;\ldots$$

$$M \;=\; \boxed{3 \mid v \mid 6 \mid 8 \mid \ldots}$$

where $v = \langle \lambda \text{x}.(\text{x = x + y; x * x}), [\text{y} \mapsto a_0] \rangle$

Here is another MFAE expression:

```
/* MFAE */
var y = 1;
var f = x => {
  x = x + y; /* 5 + 3 */
  x * x     /* 8 * 8 */   *
};
f(5);       /* 36 */
y = 3;
f(5)
```

$$\sigma = [$$
$$\quad y \mapsto a_0$$
$$\quad x \mapsto a_3$$
$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \dots$$

M	=	3	v	6	8	$\dots$

where $v = \langle \lambda\text{x}.(\text{x = x + y; x * x}), [\text{y} \mapsto a_0] \rangle$

PLRG

Here is another MFAE expression:

```
/* MFAE */
var y = 1;
var f = x => {
  x = x + y;
  x * x
};
f(5);        /* 36 */
y = 3;
f(5)         /* 64 */   *
```

$$\sigma = [$$
$$\quad \text{y} \mapsto a_0$$
$$\quad \text{f} \mapsto a_1$$
$$]$$

$$\mathbb{A} \quad : \quad a_0 \quad a_1 \quad a_2 \quad a_3 \quad \dots$$
$$M \quad = \quad \boxed{\begin{array}{|c|c|c|c|c|} 3 & v & 6 & 8 & \dots \end{array}}$$

where $v = \langle \lambda \text{x}.(\text{x = x + y; x * x}), [\text{y} \mapsto a_0] \rangle$

Interpreter and Natural Semantics

For MFAE, we need to 1) implement the **interpreter** with environments and **memories** by passing the updated memory in the result:

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

```
type Env  = Map[String, Addr]
type Addr = Int
type Mem  = Map[Addr, Value]
```

```
enum Value:
  case NumV(n: BigInt)
  case CloV(p: String, b: Expr, e: Env)
```

For MFAE, we need to 1) implement the **interpreter** with environments and **memories** by passing the updated memory in the result:

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

```scala
type Env  = Map[String, Addr]
type Addr = Int
type Mem  = Map[Addr, Value]
```

```scala
enum Value:
  case NumV(n: BigInt)
  case CloV(p: String, b: Expr, e: Env)
```

and 2) define the **natural semantics** with environments and **memories** by passing the updated memory in the result:

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

Environments	σ	$\in$	$\mathbb{X} \xrightarrow{\mathsf{fin}} \mathbb{A}$	(Env)
Addresses	a	$\in$	$\mathbb{A}$	(Addr)
Memories	M	$\in$	$\mathbb{A} \xrightarrow{\mathsf{fin}} \mathbb{V}$	(Mem)

Mutable Variable

```
def interp(expr: Expr, env: Env, mem: Mem): (Value, Mem) = expr match
  ...
  case Var(name, init, body) =>
    val (iv, imem) = interp(init, env, mem)
    val addr = malloc(imem)
    interp(body, env + (name -> addr), imem + (addr -> iv))
```

$$\sigma, M \vdash e \Rightarrow v, M$$

$$\text{Var} \ \frac{\sigma, M \vdash e_1 \Rightarrow v_1, M_1 \qquad a \notin \text{Domain}(M_1) \qquad \sigma[x \mapsto a], M_1[a \mapsto v_1] \vdash e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash \texttt{var} \ x = e_1; \ e_2 \Rightarrow v_2, M_2}$$

We learned one way to implement malloc in the previous lecture:

```
def malloc(mem: Mem): Addr = mem.keySet.maxOption.fold(0)(_ + 1)
```

```
def interp(expr: Expr, env: Env, mem: Mem): (Value, Mem) = expr match
  ...
  case Id(name) => (mem(lookupId(env, name)), mem)

def lookupId(env: Env, name: String): Addr =
  env.getOrElse(name, error(s"free identifier: $name"))
```

$$\sigma, M \vdash e \Rightarrow v, M$$

$$\text{Id} \; \frac{x \in \mathsf{Domain}(\sigma)}{\sigma, M \vdash x \Rightarrow M(\sigma(x)), M}$$

Function Application

```scala
def interp(expr: Expr, env: Env, mem: Mem): (Value, Mem) = expr match
  ...
  case App(fun, arg) =>
    val (fv, fmem) = interp(fun, env, mem)
    fv match
      case CloV(param, body, fenv) =>
        val (av, amem) = interp(arg, env, fmem)
        val addr = malloc(amem)
        interp(body, fenv + (param -> addr), amem + (addr -> av))
      case _ =>
        error(s"not a function: ${fv.str}")
```

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

$$\text{App} \frac{\begin{array}{ll} \sigma, M \vdash e_1 \Rightarrow \langle \lambda x.e_3, \sigma' \rangle, M_1 & \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2 \\ a \notin \mathsf{Domain}(M_2) & \sigma'[x \mapsto a], M_2[a \mapsto v_2] \vdash e_3 \Rightarrow v_3, M_3 \end{array}}{\sigma, M \vdash e_1(e_2) \Rightarrow v_3, M_3}$$

Assignment

```
def interp(expr: Expr, env: Env, mem: Mem): (Value, Mem) = expr match
  ...
  case Assign(name, expr) =>
    val (ev, emem) = interp(expr, env, mem)
    (ev, emem + (lookupId(env, name) -> ev))
```

$$\sigma, M \vdash e \Rightarrow v, M$$

$$\text{Assign} \frac{\sigma, M \vdash e \Rightarrow v, M' \qquad x \in \mathsf{Domain}(\sigma)}{\sigma, M \vdash x = e \Rightarrow v, M'[\sigma(x) \mapsto v]}$$

Contents

The current semantics of MFAE is based on the **call-by-value (CBV)** evaluation strategy, because the argument expression is always evaluated and the result **value** is passed to the parameter.

Call-by-Value vs. Call-by-Reference

The current semantics of MFAE is based on the **call-by-value (CBV)** evaluation strategy, because the argument expression is always evaluated and the result **value** is passed to the parameter.

However, we can define the semantics of MFAE in another way by using the **call-by-reference (CBR)** evaluation strategy instead; if the argument expression is an identifier, the parameter points to its **address**.

Call-by-Value vs. Call-by-Reference

The current semantics of MFAE is based on the **call-by-value (CBV)** evaluation strategy, because the argument expression is always evaluated and the result **value** is passed to the parameter.

However, we can define the semantics of MFAE in another way by using the **call-by-reference (CBR)** evaluation strategy instead; if the argument expression is an identifier, the parameter points to its **address**.

$$\boxed{\text{CBV}}$$

$$\sigma = [$$
$$\quad \texttt{f} \mapsto a_0,$$
$$\quad \texttt{a} \mapsto a_1,$$
$$\quad \texttt{b} \mapsto a_2,$$
$$]$$

a_0	a_1	a_2	a_3	a_4	a_5
$\langle \lambda \ldots \rangle$	1	2			

```
/* MFAE */
var f = x => y => {
  var t = x;
  x = y;
  y = t;
};
var a = 1;
var b = 2;      *
f(a)(b); a; b
```

$$\boxed{\text{CBR}}$$

$$\sigma = [$$
$$\quad \texttt{f} \mapsto a_0,$$
$$\quad \texttt{a} \mapsto a_1,$$
$$\quad \texttt{b} \mapsto a_2,$$
$$]$$

a_0	a_1	a_2	a_3
$\langle \lambda \ldots \rangle$	1	2	

The current semantics of MFAE is based on the **call-by-value (CBV)** evaluation strategy, because the argument expression is always evaluated and the result **value** is passed to the parameter.

However, we can define the semantics of MFAE in another way by using the **call-by-reference (CBR)** evaluation strategy instead; if the argument expression is an identifier, the parameter points to its **address**.

CBV		
$\sigma = [$		
$\quad x \mapsto a_3,$		
$\quad y \mapsto a_4,$		
$]$		

```
/* MFAE */
var f = x => y => { *
  var t = x;
  x = y;
  y = t;
};
var a = 1;
var b = 2;
f(a)(b); a; b
```

CBR		
$\sigma = [$		
$\quad x \mapsto a_1,$		
$\quad y \mapsto a_2,$		
$]$		

a_0	a_1	a_2	a_3	a_4	a_5
$\langle\lambda\ldots\rangle$	1	2	1	2	

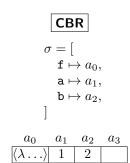| a_0	a_1	a_2	a_3
$\langle\lambda\ldots\rangle$	1	2	

The current semantics of MFAE is based on the **call-by-value (CBV)** evaluation strategy, because the argument expression is always evaluated and the result **value** is passed to the parameter.

However, we can define the semantics of MFAE in another way by using the **call-by-reference (CBR)** evaluation strategy instead; if the argument expression is an identifier, the parameter points to its **address**.

CBV

$$\sigma = [$$
$$\quad \texttt{x} \mapsto a_3,$$
$$\quad \texttt{y} \mapsto a_4,$$
$$\quad \texttt{t} \mapsto a_5,$$
$$]$$

```
/* MFAE */
var f = x => y => {
  var t = x;          *
  x = y;
  y = t;
};
var a = 1;
var b = 2;
f(a)(b); a; b
```

CBR

$$\sigma = [$$
$$\quad \texttt{x} \mapsto a_1,$$
$$\quad \texttt{y} \mapsto a_2,$$
$$\quad \texttt{t} \mapsto a_3,$$
$$]$$

a_0	a_1	a_2	a_3	a_4	a_5
$\langle\lambda\ldots\rangle$	1	2	1	2	1

a_0	a_1	a_2	a_3
$\langle\lambda\ldots\rangle$	1	2	1

Call-by-Value vs. Call-by-Reference

The current semantics of MFAE is based on the **call-by-value (CBV)** evaluation strategy, because the argument expression is always evaluated and the result **value** is passed to the parameter.
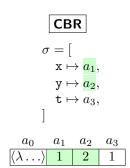
However, we can define the semantics of MFAE in another way by using the **call-by-reference (CBR)** evaluation strategy instead; if the argument expression is an identifier, the parameter points to its **address**.

CBV		MFAE		CBR

$$\sigma = [$$
$$\quad \text{x} \mapsto a_3,$$
$$\quad \text{y} \mapsto a_4,$$
$$\quad \text{t} \mapsto a_5,$$
$$]$$

```
/* MFAE */
var f = x => y => {
  var t = x;
  x = y;          *
  y = t;
};
var a = 1;
var b = 2;
f(a)(b); a; b
```

$$\sigma = [$$
$$\quad \text{x} \mapsto a_1,$$
$$\quad \text{y} \mapsto a_2,$$
$$\quad \text{t} \mapsto a_3,$$
$$]$$

a_0	a_1	a_2	a_3	a_4	a_5
$\langle \lambda \ldots \rangle$	1	2	2	2	1

a_0	a_1	a_2	a_3
$\langle \lambda \ldots \rangle$	2	2	1

Call-by-Value vs. Call-by-Reference

The current semantics of MFAE is based on the **call-by-value (CBV)** evaluation strategy, because the argument expression is always evaluated and the result **value** is passed to the parameter.
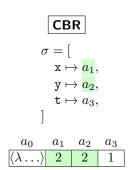
However, we can define the semantics of MFAE in another way by using the **call-by-reference (CBR)** evaluation strategy instead; if the argument expression is an identifier, the parameter points to its **address**.

CBV

$$\sigma = [$$
$$\quad \text{x} \mapsto a_3,$$
$$\quad \text{y} \mapsto a_4,$$
$$\quad \text{t} \mapsto a_5,$$
$$]$$

```
/* MFAE */
var f = x => y => {
  var t = x;
  x = y;
  y = t;        *
};
var a = 1;
var b = 2;
f(a)(b); a; b
```

CBR

$$\sigma = [$$
$$\quad \text{x} \mapsto a_1,$$
$$\quad \text{y} \mapsto a_2,$$
$$\quad \text{t} \mapsto a_3,$$
$$]$$

a_0	a_1	a_2	a_3	a_4	a_5
$\langle\lambda\ldots\rangle$	1	2	2	1	1

a_0	a_1	a_2	a_3
$\langle\lambda\ldots\rangle$	2	1	1

Call-by-Value vs. Call-by-Reference

The current semantics of MFAE is based on the **call-by-value (CBV)** evaluation strategy, because the argument expression is always evaluated and the result **value** is passed to the parameter.
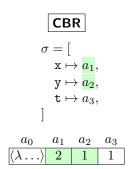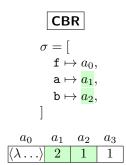
However, we can define the semantics of MFAE in another way by using the **call-by-reference (CBR)** evaluation strategy instead; if the argument expression is an identifier, the parameter points to its **address**.

CBV					

$$\sigma = [$$
$$\quad \texttt{f} \mapsto a_0,$$
$$\quad \texttt{a} \mapsto a_1,$$
$$\quad \texttt{b} \mapsto a_2,$$
$$]$$

a_0	a_1	a_2	a_3	a_4	a_5
$\langle \lambda \ldots \rangle$	1	2	2	1	1

```
/* MFAE */
var f = x => y => {
  var t = x;
  x = y;
  y = t;
};
var a = 1;
var b = 2;
f(a)(b); a; b    *
```

CBR			

$$\sigma = [$$
$$\quad \texttt{f} \mapsto a_0,$$
$$\quad \texttt{a} \mapsto a_1,$$
$$\quad \texttt{b} \mapsto a_2,$$
$$]$$

a_0	a_1	a_2	a_3
$\langle \lambda \ldots \rangle$	2	1	1

Function Application (Call-by-Reference)

We can define the semantics of MFAE with the **call-by-reference (CBR)** evaluation strategy by adding the following case:

```scala
def interp(expr: Expr, env: Env, mem: Mem): (Value, Mem) = expr match
  ...
  case App(fun, arg) =>
    val (fv, fmem) = interp(fun, env, mem)
    fv match
      case CloV(param, body, fenv) => arg match
        case Id(name) =>
          val addr = lookupId(env, name)
          interp(body, fenv + (param -> addr), fmem)
        case _ => ...
      case _ => error(s"not a function: ${fv.str}")
  ...
```

$$\text{App}_x \frac{\sigma, M \vdash e_1 \Rightarrow \langle \lambda x'.e_2, \sigma' \rangle, M_1 \quad x \in \text{Domain}(\sigma) \quad \sigma'[x' \mapsto \sigma(x)], M_1 \vdash e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash e_1(x) \Rightarrow v_2, M_2}$$

https://github.com/ku-plrg-classroom/docs/tree/main/cose212/mfae

- Please see above document on GitHub:
  - Implement interp function.
  - Implement interpCBR function.

- It is just an exercise, and you **don't need to submit** anything.

- However, some exam questions might be related to this exercise.

Summary

1. Mutable Variables

2. MFAE – FAE with Mutable Variables
    Concrete Syntax
    Abstract Syntax

3. Interpreter and Natural Semantics for MFAE
    Evaluation with Memories
    Interpreter and Natural Semantics
    Mutable Variable
    Identifier Lookup
    Function Application
    Assignment

4. Call-by-Value vs. Call-by-Reference

- Garbage Collection

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