

MFAE – FAE with Mutable Variables

1 INTRODUCTION

MFAE is a toy language for the [COSE212](#) course at Korea University. MFAE stands for an extension of the [FAE](#) language with **mutable variables**, and it supports the following features:

- **number (integer) values** (0, 1, -1, 2, -2, 3, -3, ...)
- **arithmetic operators**: addition (+) and multiplication (*)
- **mutable variable definitions** (var)
- **first-class functions** (=>)
- **assignment** (=)
- **augmented assignment**: additive (+=) and multiplicative (*=)
- **sequences** (;)

This document is the specification of MFAE. First, Section 2 describes the concrete syntax, and Section 3 describes the abstract syntax with the desugaring rules. Then, Section 4 describes the big-step operational (natural) semantics of MFAE.

2 CONCRETE SYNTAX

The concrete syntax of MFAE is written in a variant of the extended Backus–Naur form (EBNF). The notation `<nt>` denotes a nonterminal, and `"t"` denotes a terminal. We use `?` to denote an optional element and `+` (or `*`) to denote one or more (or zero or more) repetitions of the preceding element. We use **butnot** to denote a set difference to exclude some strings from a producible set of strings. We omit some obvious terminals using the ellipsis (`...`) notation.

```
// basic elements
<digit>      ::= "0" | "1" | "2" | ... | "9"
<number>     ::= "-"? <digit>+
<alphabet>   ::= "A" | "B" | "C" | ... | "Z" | "a" | "b" | "c" | ... | "z"
<idstart>    ::= <alphabet> | "_"
<idcont>     ::= <alphabet> | "_" | <digit>
<keyword>    ::= "var"
<id>         ::= <idstart> <idcont>* butnot <keyword>

// expressions
<expr> ::= <number> | <expr> "+" <expr> | <expr> "*" <expr>
        | "(" <expr> ")" | "{" <expr> "}"
        | "var" <id> "=" <expr> ";" <expr> | <id>
        | <id> "=" <expr> | <expr> "(" <expr> ")"
        | <id> "=" <expr> | <id> "+=" <expr> | <id> "*=" <expr>
        | <expr> ";" <expr>
```

The precedence and associativity of operators are defined as follows:

Description	Operator	Precedence	Associativity
Multiplicative	*	1	left
Additive	+	2	
Assignment	=, +=, *=	3	right

3 ABSTRACT SYNTAX

The abstract syntax of MFAE is defined as follows:

Expressions	$\mathbb{E} \ni e ::= n$	(Num)	$ \ \lambda x.e$	(Fun)
	$ \ e + e$	(Add)	$ \ e(e)$	(App)
	$ \ e * e$	(Mul)	$ \ x = e$	(Assign)
	$ \ \text{var } x = e; e$	(Var)	$ \ e; e$	(Seq)
	$ \ x$	(Id)		

where

Numbers	$n \in \mathbb{Z}$ (BigInt)	Identifiers	$x \in \mathbb{X}$ (String)
---------	-----------------------------	-------------	-----------------------------

The semantics of the remaining cases are defined with the following desugaring rules:

$$\mathcal{D}[\![x += e]\!] = x = x + \mathcal{D}[\![e]\!] \quad \mathcal{D}[\![x * = e]\!] = x = x * \mathcal{D}[\![e]\!]$$

The omitted cases recursively apply the desugaring rule to sub-expressions.

4 SEMANTICS

We use the following notations in the semantics:

Environments	$\sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{A}$ (Env)	Memories	$M \in \mathbb{A} \xrightarrow{\text{fin}} \mathbb{V}$ (Mem)
Values	$\mathbb{V} \ni v ::= n$ (NumV)	Addresses	$a \in \mathbb{A}$ (Addr)
	$ \ \langle \lambda x.e, \sigma \rangle$ (CloV)		

The big-step operational (natural) semantics of MFAE is defined as follows:

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

$$\begin{array}{c}
 \text{Num} \frac{}{\sigma, M \vdash n \Rightarrow n, M} \quad \text{Add} \frac{\sigma, M \vdash e_1 \Rightarrow n_1, M_1 \quad \sigma, M_1 \vdash e_2 \Rightarrow n_2, M_2}{\sigma, M \vdash e_1 + e_2 \Rightarrow n_1 + n_2, M_2} \\
 \\
 \text{Mul} \frac{\sigma, M \vdash e_1 \Rightarrow n_1, M_1 \quad \sigma, M_1 \vdash e_2 \Rightarrow n_2, M_2}{\sigma, M \vdash e_1 * e_2 \Rightarrow n_1 \times n_2, M_2} \quad \text{Id} \frac{x \in \text{Domain}(\sigma)}{\sigma, M \vdash x \Rightarrow M(\sigma(x)), M} \\
 \\
 \text{Var} \frac{\sigma, M \vdash e_1 \Rightarrow v_1, M_1 \quad a \notin \text{Domain}(M_1) \quad \sigma[x \mapsto a], M_1[a \mapsto v_1] \vdash e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash \text{var } x = e_1; e_2 \Rightarrow v_2, M_2} \\
 \\
 \text{Fun} \frac{}{\sigma, M \vdash \lambda x.e \Rightarrow \langle \lambda x.e, \sigma \rangle, M} \\
 \\
 \text{App} \frac{\sigma, M \vdash e_1 \Rightarrow \langle \lambda x.e_3, \sigma' \rangle, M_1 \quad \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2 \quad a \notin \text{Domain}(M_2) \quad \sigma'[x \mapsto a], M_2[a \mapsto v_2] \vdash e_3 \Rightarrow v_3, M_3}{\sigma, M \vdash e_1(e_2) \Rightarrow v_3, M_3} \\
 \\
 \text{Assign} \frac{\sigma, M \vdash e \Rightarrow v, M' \quad x \in \text{Domain}(\sigma)}{\sigma, M \vdash x = e \Rightarrow v, M'[\sigma(x) \mapsto v]} \\
 \\
 \text{Seq} \frac{\sigma, M \vdash e_1 \Rightarrow _, M_1 \quad \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash e_1; e_2 \Rightarrow v_2, M_2}
 \end{array}$$

4.1 Call-By-Reference (CBR) semantics

The above semantics is defined with **call-by-value (CBV)** evaluation strategy. We can augment it with **call-by-reference (CBR)** evaluation strategy by replacing the rule for function application with the following two rules:

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