BFAE – FAE with Mutable Boxes

1 INTRODUCTION

BFAE is a toy language for the COSE212 course at Korea University. BFAE stands for an extension of the FAE language with **mutable boxes**, and it supports the following features:

- **number (integer) values** (`0`, `1`, `-1`, `2`, `-2`, `3`, `-3`, ...)
- **arithmetic operators**: addition (`+`) and multiplication (`*`)
- **immutable variable definitions** (`val`)
- **first-class functions** (`=>`)
- **mutable boxes** (`Box`)
- **box operations**: get (`get`) and set (`set`)
- **sequences** (`;`)

This document is the specification of BFAE. 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 BFAE.

2 CONCRETE SYNTAX

The concrete syntax of BFAE 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>  ::= "Box" | "val"
<id>       ::= <idstart> <idcont>* butnot <keyword>

// expressions
<expr> ::= <number> | <expr> "+" <expr> | <expr> "*" <expr>
         | "(" <expr> ")" | "{" <expr> "}"
         | "val" <id> "=" <expr> ";" <expr> | <id>
         | <id> "=>" <expr> | <expr> "(" <expr> ")"
         | "Box" "(" <expr> ")"
         | <expr> "." "get" | <expr> "." "set" "(" <expr> ")"
         | <expr> ";" <expr>
```

The precedence and associativity of operators are defined as follows:

Description	Operator	Precedence	Associativity
Multiplicative	`*`	1	left
Additive	`+`	2	

3 ABSTRACT SYNTAX

The abstract syntax of BFAE is defined as follows:

$$
\begin{array}{llll}
\text{Expressions} \quad \mathbb{E} \ni e ::= n & (\texttt{Num}) & \mid \texttt{Box}(e) & (\texttt{NewBox}) \\
\quad\quad\quad\quad\quad\quad\quad \mid e + e & (\texttt{Add}) & \mid e\texttt{.get} & (\texttt{GetBox}) \\
\quad\quad\quad\quad\quad\quad\quad \mid e * e & (\texttt{Mul}) & \mid e\texttt{.set}(e) & (\texttt{SetBox}) \\
\quad\quad\quad\quad\quad\quad\quad \mid x & (\texttt{Id}) & \mid e;\ e & (\texttt{Seq}) \\
\quad\quad\quad\quad\quad\quad\quad \mid \lambda x.e & (\texttt{Fun}) & & \\
\quad\quad\quad\quad\quad\quad\quad \mid e(e) & (\texttt{App}) & &
\end{array}
$$

where

$$
\text{Numbers} \quad n \in \mathbb{Z} \quad (\texttt{BigInt}) \qquad\qquad \text{Identifiers} \quad x \in \mathbb{X} \quad (\texttt{String})
$$

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

$$
\mathcal{D}[\![\texttt{val}\ x = e_1;\ e_2]\!] = (\lambda x.\mathcal{D}[\![e_2]\!])(\mathcal{D}[\![e_1]\!])
$$

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

4 SEMANTICS

We use the following notations in the semantics:

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

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

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

$$
\texttt{Num}\ \frac{}{\sigma, M \vdash n \Rightarrow n, M} \qquad \texttt{Add}\ \frac{\sigma, M \vdash e_1 \Rightarrow n_1, M_1 \qquad \sigma, M_1 \vdash e_2 \Rightarrow n_2, M_2}{\sigma, M \vdash e_1 + e_2 \Rightarrow n_1 + n_2, M_2}
$$

$$
\texttt{Mul}\ \frac{\sigma, M \vdash e_1 \Rightarrow n_1, M_1 \qquad \sigma, M_1 \vdash e_2 \Rightarrow n_2, M_2}{\sigma, M \vdash e_1 * e_2 \Rightarrow n_1 \times n_2, M_2}
$$

$$
\texttt{Id}\ \frac{x \in \text{Domain}(\sigma)}{\sigma, M \vdash x \Rightarrow \sigma(x), M} \qquad \texttt{Fun}\ \frac{}{\sigma, M \vdash \lambda x.e \Rightarrow \langle \lambda x.e, \sigma \rangle, M}
$$

$$
\texttt{App}\ \frac{\sigma, M \vdash e_1 \Rightarrow \langle \lambda x.e_3, \sigma' \rangle, M_1 \qquad \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2 \qquad \sigma'[x \mapsto v_2], M_2 \vdash e_3 \Rightarrow v_3, M_3}{\sigma, M \vdash e_1(e_2) \Rightarrow v_3, M_3}
$$

$$
\texttt{NewBox}\ \frac{\sigma, M \vdash e \Rightarrow v, M_1 \qquad a \notin \text{Domain}(M_1)}{\sigma, M \vdash \texttt{Box}(e) \Rightarrow a, M_1[a \mapsto v]}
$$

$$
\texttt{GetBox}\ \frac{\sigma, M \vdash e \Rightarrow a, M_1}{\sigma, M \vdash e\texttt{.get} \Rightarrow M_1(a), M_1} \qquad \texttt{SetBox}\ \frac{\sigma, M \vdash e_1 \Rightarrow a, M_1 \qquad \sigma, M_1 \vdash e_2 \Rightarrow v, M_2}{\sigma, M \vdash e_1\texttt{.set}(e_2) \Rightarrow v, M_2[a \mapsto v]}
$$

$$
\texttt{Seq}\ \frac{\sigma, M \vdash e_1 \Rightarrow \_, M_1 \qquad \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash e_1;\ e_2 \Rightarrow v_2, M_2}
$$