KFAE – FAE with First-Class Continuations

1 INTRODUCTION

KFAE is a toy language for the COSE212 course at Korea University. KFAE stands for the KFAE language with the **first-class continuations**, 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** (=>)
- **first-class continuations** (vcc)

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

2 CONCRETE SYNTAX

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

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

The precedence and associativity of operators are defined as follows:

Operator	Associativity	Precedence
*	left	1
+	left	2

3 ABSTRACT SYNTAX

The abstract syntax of KFAE is defined as follows:

$$
\begin{array}{llll}
\text{Expressions} & \mathbb{E} \ni e ::= & n & \text{(Num)} \\
& | & e + e & \text{(Add)} \\
& | & e * e & \text{(Mul)} \\
& | & x & \text{(Id)} \\
& | & \lambda x.e & \text{(Fun)} \\
& | & e(e) & \text{(App)} \\
& | & \text{vcc } x;\ e & \text{(Vcc)}
\end{array}
\qquad \text{where} \qquad
\begin{array}{lll}
\text{Numbers} & n \in \mathbb{Z} & \text{(BigInt)} \\
\text{Identifiers} & x \in \mathbb{X} & \text{(String)}
\end{array}
$$

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

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

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

4 SEMANTICS

We use the following notations in the semantics:

$$
\begin{array}{llll}
\text{Values} & \mathbb{V} \ni v ::= & n & \text{(NumV)} \\
& | & \langle \lambda x.e, \sigma \rangle & \text{(CloV)} \\
& | & \langle \kappa \mid\mid s \rangle & \text{(ContV)} \\
\text{Environments} & \sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{V} & & \text{(Env)} \\
\text{Continuations} & \mathbb{K} \ni \kappa ::= & \square & \text{(EmptyK)} \\
& | & (\sigma \vdash e) :: \kappa & \text{(EvalK)} \\
& | & (+) :: \kappa & \text{(AddK)} \\
& | & (\times) :: \kappa & \text{(MulK)} \\
& | & (@) :: \kappa & \text{(AppK)} \\
\text{Value Stacks} & \mathbb{S} \ni s ::= \blacksquare \mid v :: s & & \text{(List[Value])}
\end{array}
$$

The small-step operational (reduction) semantics of KFAE is defined as follows:

$$
\boxed{\langle \kappa \mid\mid s \rangle \to \langle \kappa \mid\mid s \rangle}
$$

$$
\begin{array}{llll}
\text{Num} & \langle (\sigma \vdash n) :: \kappa \mid\mid s \rangle & \to & \langle \kappa \mid\mid n :: s \rangle \\[6pt]
\text{Add}_1 & \langle (\sigma \vdash e_1 + e_2) :: \kappa \mid\mid s \rangle & \to & \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (+) :: \kappa \mid\mid s \rangle \\
\text{Add}_2 & \langle (+) :: \kappa \mid\mid n_2 :: n_1 :: s \rangle & \to & \langle \kappa \mid\mid (n_1 + n_2) :: s \rangle \\[6pt]
\text{Mul}_1 & \langle (\sigma \vdash e_1 * e_2) :: \kappa \mid\mid s \rangle & \to & \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (\times) :: \kappa \mid\mid s \rangle \\
\text{Mul}_2 & \langle (\times) :: \kappa \mid\mid n_2 :: n_1 :: s \rangle & \to & \langle \kappa \mid\mid (n_1 \times n_2) :: s \rangle \\[6pt]
\text{Id} & \langle (\sigma \vdash x) :: \kappa \mid\mid s \rangle & \to & \langle \kappa \mid\mid \sigma(x) :: s \rangle \\[6pt]
\text{Fun} & \langle (\sigma \vdash \lambda x.e) :: \kappa \mid\mid s \rangle & \to & \langle \kappa \mid\mid \langle \lambda x.e, \sigma \rangle :: s \rangle \\[6pt]
\text{App}_1 & \langle (\sigma \vdash e_1(e_2)) :: \kappa \mid\mid s \rangle & \to & \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (@) :: \kappa \mid\mid s \rangle \\
\text{App}_{2,\lambda} & \langle (@) :: \kappa \mid\mid v_2 :: \langle \lambda x.e, \sigma \rangle :: s \rangle & \to & \langle (\sigma[x \mapsto v_2] \vdash e) :: \kappa \mid\mid s \rangle \\
\text{App}_{2,\kappa} & \langle (@) :: \kappa \mid\mid v_2 :: \langle \kappa' \mid\mid s' \rangle :: s \rangle & \to & \langle \kappa' \mid\mid v_2 :: s' \rangle \\
\text{Vcc} & \langle (\sigma \vdash \text{vcc } x;\ e) :: \kappa \mid\mid s \rangle & \to & \langle (\sigma[x \mapsto \langle \kappa \mid\mid s \rangle] \vdash e) :: \kappa \mid\mid s \rangle
\end{array}
$$

where $\rightarrow^*$ is the reflexive-transitive closure of $\rightarrow$ and denotes the repeated reduction:

$$\langle \kappa \parallel s \rangle \rightarrow^* \langle \kappa \parallel s \rangle$$

$$\frac{\langle \kappa \parallel s \rangle \rightarrow^* \langle \kappa' \parallel s' \rangle \qquad \langle \kappa' \parallel s' \rangle \rightarrow \langle \kappa'' \parallel s'' \rangle}{\langle \kappa \parallel s \rangle \rightarrow^* \langle \kappa'' \parallel s'' \rangle}$$

The evaluation result of an expression e is the value v if

$$\langle (\varnothing \vdash e) :: \square \parallel \blacksquare \rangle \rightarrow^* \langle \square \parallel v :: \blacksquare \rangle$$