FAE-cps – FAE with Continuation-Passing Style

1 INTRODUCTION

FAE-cps is a toy language for the COSE212 course at Korea University. FAE-cps stands for the FAE language with the **continuation-passing style (CPS)**. Since it has the same syntax and semantics as FAE, 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** (`=>`)

This document is the specification of FAE-cps. While it has the same syntax and semantics as FAE, Section 2 and Section 3 repeat the concrete and abstract syntax parts for completeness, respectively, with the desugaring rules. Section 4 redefines the same semantics in a small-step operational (reduction) semantics style rather than a big-step style.

2 CONCRETE SYNTAX

The concrete syntax of FAE-cps 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"
<id>       ::= <idstart> <idcont>* butnot <keyword>

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

The precedence and associativity of operators are defined as follows:

Operator	Associativity	Precedence
*	left	2
+	left	1

3 ABSTRACT SYNTAX

The abstract syntax of FAE-cps is defined as follows:

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

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:

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

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

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

$$\begin{array}{llll} \texttt{Num} & \langle (\sigma \vdash n) :: \kappa \mid\mid s \rangle & \rightarrow & \langle \kappa \mid\mid n :: s \rangle \\ \\ \texttt{Add}_1 & \langle (\sigma \vdash e_1 + e_2) :: \kappa \mid\mid s \rangle & \rightarrow & \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (+) :: \kappa \mid\mid s \rangle \\ \texttt{Add}_2 & \langle (+) :: \kappa \mid\mid n_2 :: n_1 :: s \rangle & \rightarrow & \langle \kappa \mid\mid (n_1 + n_2) :: s \rangle \\ \\ \texttt{Mul}_1 & \langle (\sigma \vdash e_1 * e_2) :: \kappa \mid\mid s \rangle & \rightarrow & \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (*) :: \kappa \mid\mid s \rangle \\ \texttt{Mul}_2 & \langle (*) :: \kappa \mid\mid n_2 :: n_1 :: s \rangle & \rightarrow & \langle \kappa \mid\mid (n_1 \times n_2) :: s \rangle \\ \\ \texttt{Id} & \langle (\sigma \vdash x) :: \kappa \mid\mid s \rangle & \rightarrow & \langle \kappa \mid\mid \sigma(x) :: s \rangle \\ \\ \texttt{Fun} & \langle (\sigma \vdash \lambda x.e) :: \kappa \mid\mid s \rangle & \rightarrow & \langle \kappa \mid\mid \langle \lambda x.e, \sigma \rangle :: s \rangle \\ \\ \texttt{App}_1 & \langle (\sigma \vdash e_1(e_2)) :: \kappa \mid\mid s \rangle & \rightarrow & \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (@) :: \kappa \mid\mid s \rangle \\ \texttt{App}_2 & \langle (@) :: \kappa \mid\mid v_2 :: \langle \lambda x.e, \sigma \rangle :: s \rangle & \rightarrow & \langle (\sigma[x \mapsto v_2] \vdash e) :: \kappa \mid\mid s \rangle \end{array}$$

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

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

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

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

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