

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	1
+	left	2

3 ABSTRACT SYNTAX

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

Expressions	$\mathbb{E} \ni e ::= n$	(Num)		
	$e + e$	(Add)		
	$e * e$	(Mul)		
	x	(Id)	where	Numbers $n \in \mathbb{Z}$ (BigInt)
	$\lambda x.e$	(Fun)		Identifiers $x \in \mathbb{X}$ (String)
	$e(e)$	(App)		

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:

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

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

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

Num	$\langle (\sigma \vdash n) :: \kappa \parallel s \rangle$	$\rightarrow \langle \kappa \parallel n :: s \rangle$
Add ₁	$\langle (\sigma \vdash e_1 + e_2) :: \kappa \parallel s \rangle$	$\rightarrow \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (+) :: \kappa \parallel s \rangle$
Add ₂	$\langle (+) :: \kappa \parallel n_2 :: n_1 :: s \rangle$	$\rightarrow \langle \kappa \parallel (n_1 + n_2) :: s \rangle$
Mul ₁	$\langle (\sigma \vdash e_1 * e_2) :: \kappa \parallel s \rangle$	$\rightarrow \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (*) :: \kappa \parallel s \rangle$
Mul ₂	$\langle (*) :: \kappa \parallel n_2 :: n_1 :: s \rangle$	$\rightarrow \langle \kappa \parallel (n_1 \times n_2) :: s \rangle$
Id	$\langle (\sigma \vdash x) :: \kappa \parallel s \rangle$	$\rightarrow \langle \kappa \parallel \sigma(x) :: s \rangle$
Fun	$\langle (\sigma \vdash \lambda x.e) :: \kappa \parallel s \rangle$	$\rightarrow \langle \kappa \parallel \langle \lambda x.e, \sigma \rangle :: s \rangle$
App ₁	$\langle (\sigma \vdash e_1(e_2)) :: \kappa \parallel s \rangle$	$\rightarrow \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (@) :: \kappa \parallel s \rangle$
App ₂	$\langle (@) :: \kappa \parallel v_2 :: \langle \lambda x.e, \sigma \rangle :: s \rangle$	$\rightarrow \langle (\sigma[x \mapsto v_2] \vdash e) :: \kappa \parallel s \rangle$

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

$$\begin{array}{c} \langle \kappa \parallel s \rangle \rightarrow^* \langle \kappa \parallel s \rangle \\ \hline \frac{\langle \kappa \parallel s \rangle \rightarrow^* \langle \kappa' \parallel s' \rangle \quad \langle \kappa' \parallel s' \rangle \rightarrow \langle \kappa'' \parallel s'' \rangle}{\langle \kappa \parallel s \rangle \rightarrow^* \langle \kappa'' \parallel s'' \rangle} \end{array}$$

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

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