

F1VAE – VAE with First-Order Functions

1 INTRODUCTION

F1VAE is a toy language for the [COSE212](#) course at Korea University. F1VAE stands for an extension of the [VAE](#) language with **first-order functions**, and it supports the following features:

- **integers**
- **basic arithmetic operators**: addition (+) and multiplication (*)
- **immutable variables** (val)
- **first-order functions** (def)

This document is the specification of F1VAE. First, Section 2 describes the concrete syntax, and Section 3 describes the abstract syntax. Then, Section 4 describes the big-step operational (natural) semantics of F1VAE.

2 CONCRETE SYNTAX

The concrete syntax of F1VAE 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" | "def"
<id>         ::= <idstart> <idcont>* butnot <keyword>

// programs
<program>    ::= <fdef>* <expr>

// function definitions
<fdef>       ::= "def" <id> "(" <id> ")" "=" <expr> ";"

// expressions
<expr>       ::= <number> | <expr> "+" <expr> | <expr> "*" <expr>
               | "(" <expr> ")" | "{" <expr> "}"
               | "val" <id> "=" <expr> ";" <expr> | <id>
               | <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 F1VAE is defined as follows:

| | | |
|----------------------|---|-----------|
| Programs | $\mathbb{P} \ni p ::= f^* e$ | (Program) |
| Function Definitions | $\mathbb{F} \ni f ::= \text{def } x(x)=e$ | (FunDef) |
| Expressions | $\mathbb{E} \ni e ::= n$ | (Num) |
| | $ e + e$ | (Add) |
| | $ e \times e$ | (Mul) |
| | $ \text{val } x=e; e$ | (Val) |
| | $ x$ | (Id) |
| | $ x(e)$ | (App) |

where

| | | |
|-----------------------|--|----------|
| Environments | $\sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{Z}$ | (Env) |
| Function Environments | $\Lambda \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{F}$ | (FEnv) |
| Integers | $n \in \mathbb{Z}$ | (BigInt) |
| Identifiers | $x \in \mathbb{X}$ | (String) |

4 SEMANTICS

For a given program $p = f^* e$, the initial function environment Λ is constructed from f^* by mapping each function name to its definition. We assume that there is no duplicate function name in f^* . The big-step operational (natural) semantics of F1VAE is defined as follows:

$$\boxed{\sigma, \Lambda \vdash e \Rightarrow n}$$

$$\text{Num} \frac{}{\sigma, \Lambda \vdash n \Rightarrow n}$$

$$\text{Add} \frac{\sigma, \Lambda \vdash e_1 \Rightarrow n_1 \quad \sigma, \Lambda \vdash e_2 \Rightarrow n_2}{\sigma, \Lambda \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \quad \text{Mul} \frac{\sigma, \Lambda \vdash e_1 \Rightarrow n_1 \quad \sigma, \Lambda \vdash e_2 \Rightarrow n_2}{\sigma, \Lambda \vdash e_1 \times e_2 \Rightarrow n_1 \times n_2}$$

$$\text{Val} \frac{\sigma, \Lambda \vdash e_1 \Rightarrow n_1 \quad \sigma[x \mapsto n_1], \Lambda \vdash e_2 \Rightarrow n_2}{\sigma, \Lambda \vdash \text{val } x=e_1; e_2 \Rightarrow n_2} \quad \text{Id} \frac{x \in \text{Domain}(\sigma)}{\sigma, \Lambda \vdash x \Rightarrow \sigma(x)}$$

$$\text{App} \frac{\sigma, \Lambda \vdash e_1 \Rightarrow n_1 \quad x_0 \in \text{Domain}(\Lambda) \quad \Lambda(x_0) = \text{def } x_0(x_1)=e_2 \quad [x_1 \mapsto n_1], \Lambda \vdash e_2 \Rightarrow n_2}{\sigma, \Lambda \vdash x_0(e_1) \Rightarrow n_2}$$

4.1 Dynamic Scoping

The above semantics is defined with **static scoping (or lexical scoping)**. We can augment it with **dynamic scoping** by changing the rule for function application as follows:

$$\text{App} \frac{\sigma, \Lambda \vdash e_1 \Rightarrow n_1 \quad x_0 \in \text{Domain}(\Lambda) \quad \Lambda(x_0) = \text{def } x_0(x_1)=e_2 \quad \sigma[x_1 \mapsto n_1], \Lambda \vdash e_2 \Rightarrow n_2}{\sigma, \Lambda \vdash x_0(e_1) \Rightarrow n_2}$$