

TRFAE – RFAE with Type System

1 INTRODUCTION

TRFAE is a toy language for the [COSE212](#) course at Korea University. TRFAE stands for an extension of the [RFAE](#) language with **type system**, and it supports the following features:

- **number (integer) values**
- **boolean values** (true and false)
- **arithmetic operators**: negation (-), addition (+), subtraction (-), multiplication (*), division (/), and modulo (%)
- **arithmetic comparison operators**: equality (== and !=) and relational (<, >, <=, and >=)
- **first-class functions**
- **recursive functions** (def)
- **conditionals** (if-else)
- **logical operators**: conjunction (&&), disjunction (||), and negation (!)
- **static type checking**

This document is the specification of TRFAE. First, Section 2 describes the concrete syntax, and Section 3 describes the abstract syntax with the desugaring rules. Then, Section 4 describes the type system. Finally, Section 5 describes the big-step operational (natural) semantics of TRFAE.

2 CONCRETE SYNTAX

The concrete syntax of TRFAE 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>  ::= "true" | "false" | "def" | "if" | "else"
<id>       ::= <idstart> <idcont>* butnot <keyword>

// expressions
<expr> ::= <number> | "true" | "false"
        | <uop> <expr>
        | <expr> <bop> <expr>
        | "(" <expr> ")" | "{" <expr> "}"
        | "val" <id> "=" <expr> ";" <expr> | <id>
        | "(" <id> ":" <type> ")" "=" <expr>
        | "def" <id> "(" <id> ":" <type> ")" ":" <type> "=" <expr> ";" <expr>
        | <expr> "(" <expr> ")"
        | "if" "(" <expr> ")" <expr> "else" <expr>
```

```
// operators
<uop>  ::= "-" | "!"
<bop>  ::= "+" | "-" | "*" | "/" | "%" | "&&" | "||"
        | "==" | "!=" | "<" | "<=" | ">" | ">="

// types
<type> ::= "(" <type> ")" | "Number" | "Boolean" | <type> ">" <type>
```

For types, the arrow (\Rightarrow) operator is right-associative. For expressions, the precedence and associativity of operators are defined as follows:

Description	Operator	Precedence	Associativity
Unary	$-$, $!$	1	right
Multiplicative	$*$, $/$, $\%$	2	left
Additive	$+$, $-$	3	
Relational	$<$, $<=$, $>$, $>=$	4	
Equality	$==$, $!=$	5	
Logical Conjunction	$\&\&$	6	
Logical Disjunction	$ $	7	

3 ABSTRACT SYNTAX

The abstract syntax of TRFAE is defined as follows:

Expressions $\mathbb{E} \ni e ::= n$ (Num) | $\text{val } x = e; e$ (Val)
| b (Bool) | x (Id)
| $e + e$ (Add) | $\lambda x : \tau. e$ (Fun)
| $e \times e$ (Mul) | $\text{def } x(x : \tau) : \tau = e; e$ (Rec)
| $e \div e$ (Div) | $e(e)$ (App)
| $e \bmod e$ (Mod) | $\text{if } (e) e \text{ else } e$ (If)
| $e = e$ (Eq)
| $e < e$ (Lt)

Types $\mathbb{T} \ni \tau ::= \text{num}$ (NumT)
| bool (BoolT)
| $\tau \rightarrow \tau$ (ArrowT)

Numbers $n \in \mathbb{Z}$ (BigInt)
Identifiers $x \in \mathbb{X}$ (String)
Booleans $b \in \mathbb{B} = \{\text{true}, \text{false}\}$ (Boolean)

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

$\mathcal{D}[-e] = \mathcal{D}[e] * (-1)$ $\mathcal{D}[e_1 != e_2] = \mathcal{D}[!(e_1 == e_2)]$
 $\mathcal{D}[!e] = \text{if } (\mathcal{D}[e]) \text{ false else true}$ $\mathcal{D}[e_1 <= e_2] = \mathcal{D}[(e_1 < e_2) || (e_1 == e_2)]$
 $\mathcal{D}[e_1 - e_2] = \mathcal{D}[e_1] + \mathcal{D}[-e_2]$ $\mathcal{D}[e_1 > e_2] = \mathcal{D}[!(e_1 <= e_2)]$
 $\mathcal{D}[e_1 \&\& e_2] = \text{if } (\mathcal{D}[e_1]) \mathcal{D}[e_2] \text{ else false}$ $\mathcal{D}[e_1 >= e_2] = \mathcal{D}[!(e_1 < e_2)]$
 $\mathcal{D}[e_1 || e_2] = \text{if } (\mathcal{D}[e_1]) \text{ true else } \mathcal{D}[e_2]$

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

4 TYPE SYSTEM

This section explains type system of TRFAE, and we use the following notations:

$$\text{Type Environments} \quad \Gamma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{T} \quad (\text{TypeEnv})$$

In the type system, type checking is defined with the following typing rules:

$$\boxed{\Gamma \vdash e : \tau}$$

$$\begin{array}{c}
 \tau\text{-Num} \frac{}{\Gamma \vdash n : \text{num}} \quad \tau\text{-Bool} \frac{}{\Gamma \vdash b : \text{bool}} \\
 \\
 \tau\text{-Add} \frac{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 + e_2 : \text{num}} \quad \tau\text{-Mul} \frac{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 \times e_2 : \text{num}} \\
 \\
 \tau\text{-Div} \frac{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 \div e_2 : \text{num}} \quad \tau\text{-Mod} \frac{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 \bmod e_2 : \text{num}} \\
 \\
 \tau\text{-Eq} \frac{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 = e_2 : \text{bool}} \quad \tau\text{-Lt} \frac{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 < e_2 : \text{bool}} \\
 \\
 \tau\text{-Val} \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma[x : \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \text{val } x = e_1; e_2 : \tau_2} \quad \tau\text{-Id} \frac{x \in \text{Domain}(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \\
 \\
 \tau\text{-Fun} \frac{\Gamma[x : \tau] \vdash e : \tau'}{\Gamma \vdash \lambda x : \tau. e : \tau \rightarrow \tau'} \quad \tau\text{-App} \frac{\Gamma \vdash e_0 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash e_0(e_1) : \tau_2} \\
 \\
 \tau\text{-Rec} \frac{\Gamma[x_0 : \tau_1 \rightarrow \tau_2, x_1 : \tau_1] \vdash e_2 : \tau_2 \quad \Gamma[x_0 : \tau_1 \rightarrow \tau_2] \vdash e_3 : \tau_3}{\Gamma \vdash \text{def } x_0(x_1 : \tau_1) : \tau_2 = e_2; e_3 : \tau_3} \\
 \\
 \tau\text{-If} \frac{\Gamma \vdash e_0 : \text{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \text{if } (e_0) e_1 \text{ else } e_2 : \tau}
 \end{array}$$

5 SEMANTICS

We use the following notations in the semantics:

$$\begin{array}{lll}
 \text{Values} & \mathbb{V} \ni v ::= n & (\text{NumV}) \\
 & | b & (\text{BoolV}) \\
 & | \langle \lambda x. e, \sigma \rangle & (\text{CloV}) \\
 \text{Environments} & \sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{V} & (\text{Env})
 \end{array}$$

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

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

$$\begin{array}{c}
 \text{Num} \frac{}{\sigma \vdash n \Rightarrow n} \qquad \text{Bool} \frac{}{\sigma \vdash b \Rightarrow b} \\
 \\
 \text{Add} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \qquad \text{Mul} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 \times e_2 \Rightarrow n_1 \times n_2} \\
 \\
 \text{Div} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2 \quad n_2 \neq 0}{\sigma \vdash e_1 \div e_2 \Rightarrow n_1 \div n_2} \qquad \text{Mod} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2 \quad n_2 \neq 0}{\sigma \vdash e_1 \bmod e_2 \Rightarrow n_1 \bmod n_2} \\
 \\
 \text{Eq} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 = e_2 \Rightarrow n_1 = n_2} \qquad \text{Lt} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 < e_2 \Rightarrow n_1 < n_2} \\
 \\
 \text{Val} \frac{\sigma \vdash e_1 \Rightarrow v_1 \quad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \text{val } x = e_1; e_2 \Rightarrow v_2} \qquad \text{Id} \frac{x \in \text{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)} \\
 \\
 \text{Fun} \frac{}{\sigma \vdash \lambda x. e \Rightarrow \langle \lambda x. e, \sigma \rangle} \qquad \text{Rec} \frac{\sigma' = \sigma[x_0 \mapsto \langle \lambda x_1. e_2, \sigma' \rangle] \quad \sigma' \vdash e_3 \Rightarrow v_3}{\sigma \vdash \text{def } x_0(x_1 : \tau_1) : \tau_2 = e_2; e_3 \Rightarrow v_3} \\
 \\
 \text{App} \frac{\sigma \vdash e_0 \Rightarrow \langle \lambda x. e_2, \sigma' \rangle \quad \sigma \vdash e_1 \Rightarrow v_1 \quad \sigma'[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash e_0(e_1) \Rightarrow v_2} \\
 \\
 \text{If}_T \frac{\sigma \vdash e_0 \Rightarrow \text{true} \quad \sigma \vdash e_1 \Rightarrow v_1}{\sigma \vdash \text{if } (e_0) e_1 \text{ else } e_2 \Rightarrow v_1} \qquad \text{If}_F \frac{\sigma \vdash e_0 \Rightarrow \text{false} \quad \sigma \vdash e_2 \Rightarrow v_2}{\sigma \vdash \text{if } (e_0) e_1 \text{ else } e_2 \Rightarrow v_2}
 \end{array}$$