

TIFAE – TRFAE with Type Inference

1 INTRODUCTION

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

- **number (integer) values** (0, 1, -1, 2, -2, 3, -3, ...)
- **arithmetic operators**: negation (-), addition (+), subtraction (-), multiplication (*), division (/), and modulo (%)
- **first-class functions** (=>)
- **recursive functions** (def)
- **conditionals** (if-else)
- **boolean values** (true and false)
- **comparison operators**: equality (== and !=) and relational (<, >, <=, and >=)
- **logical operators**: conjunction (&&), disjunction (||), and negation (!)
- **static type checking with type inference**

This document is the specification of TIFAE. 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 TIFAE.

2 CONCRETE SYNTAX

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

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

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

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 TIFAE 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. e$ | (Fun) |
| | $ e * e$ | (Mul) | $ \text{def } x(x) = e; e$ | (Rec) |
| | $ e / e$ | (Div) | $ e(e)$ | (App) |
| | $ e \% e$ | (Mod) | $ \text{if } (e) e \text{ else } e$ | (If) |
| | $ e == e$ | (Eq) | | |
| | $ e < e$ | (Lt) | | |

| | | |
|-------------|--|-----------|
| 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 TIFAE, and we use the following notations:

| | | |
|-------------------|--|---|
| Type Environments | $\Gamma \in \Gamma = \mathbb{X} \xrightarrow{\text{fin}} \mathbb{T}^\vee$ | (TypeEnv) |
| Solution | $\psi \in \Psi = \mathbb{X}_\alpha \xrightarrow{\text{fin}} (\mathbb{T} \uplus \{\bullet\})$ | (Solution) |
| Types | $\mathbb{T} \ni \tau ::= \text{num}$ bool $\tau \rightarrow \tau$ α | (NumT) (BoolT) (ArrowT) (VarT) |
| Type Schemes | $\tau^\vee = \forall \alpha^*. \tau \in \mathbb{T}^\vee = \mathbb{X}_\alpha^* \times \mathbb{T}$ | (TypeScheme) |
| Type Variables | $\alpha \in \mathbb{X}_\alpha$ | (Int) |

We skip the \forall -quantifier in type schemes if they have no type variables. In the type system, type checking and type inference is defined with the following typing rules:

| | | |
|-------------------|--|---|
| | $\boxed{\Gamma, \psi \vdash e : \tau, \psi}$ | |
| | $\tau\text{-Num} \frac{}{\Gamma, \psi \vdash n : \text{num}, \psi}$ | $\tau\text{-Bool} \frac{}{\Gamma, \psi \vdash b : \text{bool}, \psi}$ |
| $\tau\text{-Add}$ | $\frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \text{unify}(\tau_1, \text{num}, \psi_2) = \psi_3 \quad \text{unify}(\tau_2, \text{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 + e_2 : \text{num}, \psi_4}$ | |
| $\tau\text{-Mul}$ | $\frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \text{unify}(\tau_1, \text{num}, \psi_2) = \psi_3 \quad \text{unify}(\tau_2, \text{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 * e_2 : \text{num}, \psi_4}$ | |
| $\tau\text{-Div}$ | $\frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \text{unify}(\tau_1, \text{num}, \psi_2) = \psi_3 \quad \text{unify}(\tau_2, \text{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 / e_2 : \text{num}, \psi_4}$ | |
| $\tau\text{-Mod}$ | $\frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \text{unify}(\tau_1, \text{num}, \psi_2) = \psi_3 \quad \text{unify}(\tau_2, \text{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 \% e_2 : \text{num}, \psi_4}$ | |
| $\tau\text{-Eq}$ | $\frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \text{unify}(\tau_1, \text{num}, \psi_2) = \psi_3 \quad \text{unify}(\tau_2, \text{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 == e_2 : \text{bool}, \psi_4}$ | |
| $\tau\text{-Lt}$ | $\frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \text{unify}(\tau_1, \text{num}, \psi_2) = \psi_3 \quad \text{unify}(\tau_2, \text{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 < e_2 : \text{bool}, \psi_4}$ | |

$$\begin{array}{c}
\tau\text{-Val} \frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \text{gen}(\tau_1, \Gamma, \psi_1) = \tau_1^\forall \quad \Gamma[x : \tau_1^\forall], \psi_1 \vdash e_2 : \tau_2, \psi_2}{\Gamma, \psi_0 \vdash \text{val } x = e_1; e_2 : \tau_2, \psi_2} \\
\\
\tau\text{-Id} \frac{\Gamma(x) = \tau^\forall \quad \text{inst}(\tau^\forall, \psi) = (\tau, \psi')}{\Gamma, \psi \vdash x : \tau, \psi'} \quad \tau\text{-Fun} \frac{\alpha_p \notin \psi \quad \Gamma[x : \alpha_p], \psi[\alpha_p \mapsto \bullet] \vdash e : \tau, \psi'}{\Gamma, \psi \vdash \lambda x. e : \alpha_p \rightarrow \tau, \psi'} \\
\\
\tau\text{-App} \frac{\Gamma, \psi \vdash e_f : \tau_f, \psi_f \quad \Gamma, \psi_f \vdash e_a : \tau_a, \psi_a \quad \alpha_r \notin \psi_a \quad \text{unify}(\tau_a \rightarrow \alpha_r, \tau_f, \psi_a[\alpha_r \mapsto \bullet]) = \psi'}{\Gamma, \psi \vdash e_f(e_a) : \alpha_r, \psi'} \\
\\
\tau\text{-Rec} \frac{\alpha_p, \alpha_r \notin \psi \quad \alpha_p \neq \alpha_r \quad \Gamma_1 = \Gamma[x_f \mapsto (\alpha_p \rightarrow \alpha_r)] \quad \Gamma_2 = \Gamma_1[x_p \mapsto \alpha_p] \quad \Gamma_2, \psi[\alpha_p \mapsto \bullet, \alpha_r \mapsto \bullet] \vdash e_b : \tau_b, \psi_b \quad \text{unify}(\tau_b, \alpha_r, \psi_b) = \psi_r \quad \Gamma_1, \psi_r \vdash e_s : \tau_s, \psi_s}{\Gamma, \psi \vdash \text{def } x_f(x_p) = e_b; e_s : \tau_s, \psi_s} \\
\\
\tau\text{-If} \frac{\Gamma, \psi_t \vdash e_e : \tau_e, \psi_e \quad \Gamma, \psi \vdash e_c : \tau_c, \psi_c \quad \Gamma, \psi_c \vdash e_t : \tau_t, \psi_t \quad \text{unify}(\tau_c, \text{bool}, \psi_c) = \psi' \quad \text{unify}(\tau_t, \tau_e, \psi') = \psi''}{\Gamma, \psi \vdash \text{if } (e_c) e_t \text{ else } e_e : \tau_t, \psi''}
\end{array}$$

type unification is defined as a partial function:

$$\boxed{\text{unify} : (\mathbb{T} \times \mathbb{T} \times \Psi) \rightarrow \Psi}$$

$$\text{unify}(\tau_1, \tau_2, \psi) = \begin{cases} \psi & \text{if } \tau'_1 = \text{num} \wedge \tau'_2 = \text{num} \\ \psi & \text{if } \tau'_1 = \text{bool} \wedge \tau'_2 = \text{bool} \\ \text{unify}(\tau_{1,r}, \tau_{2,r}, \text{unify}(\tau_{1,p}, \tau_{2,p}, \psi)) & \text{if } \tau'_1 = (\tau_{1,p} \rightarrow \tau_{1,r}) \wedge \tau'_2 = (\tau_{2,p} \rightarrow \tau_{2,r}) \\ \psi & \text{if } \tau'_1 = \alpha = \tau'_2 \\ \psi[\alpha \mapsto \tau'_2] & \text{if } \tau'_1 = \alpha \wedge \neg \text{occur}(\alpha, \tau'_2, \psi) \\ \psi[\alpha \mapsto \tau'_1] & \text{if } \tau'_2 = \alpha \wedge \neg \text{occur}(\alpha, \tau'_1, \psi) \end{cases}$$

where $\tau'_1 = \text{resolve}(\tau_1, \psi)$ and $\tau'_2 = \text{resolve}(\tau_2, \psi)$.

type resolving and occurrence checking are defined as following functions:

$$\boxed{\text{resolve} : (\mathbb{T} \times \Psi) \rightarrow \mathbb{T}}$$

$$\text{resolve}(\tau, \psi) = \begin{cases} \text{resolve}(\tau', \psi) & \text{if } \tau = \alpha \wedge \psi(\alpha) = \tau' \\ \tau & \text{otherwise} \end{cases}$$

$$\boxed{\text{occur} : (\mathbb{X}_\alpha \times \mathbb{T} \times \Psi) \rightarrow \text{bool}}$$

$$\text{occur}(\alpha, \tau, \psi) = \begin{cases} \text{true} & \text{if } \tau = \alpha \\ \text{occur}(\alpha, \tau_p, \psi) \vee \text{occur}(\alpha, \tau_r, \psi) & \text{if } \tau = (\tau_p \rightarrow \tau_r) \\ \text{false} & \text{otherwise} \end{cases}$$

type generalization and instantiation are defined as following functions:

$$\boxed{\text{gen} : (\mathbb{T} \times \Gamma \times \Psi) \rightarrow \mathbb{T}^\forall}$$

$$\text{gen}(\tau, \Gamma, \psi) = \forall \alpha_1, \dots, \alpha_m. \tau \quad \text{where} \quad \text{free}_\tau(\tau, \psi) \setminus \text{free}_\Gamma(\Gamma, \psi) = \{\alpha_1, \dots, \alpha_m\}$$

$$\boxed{\text{inst} : (\mathbb{T}^\forall \times \Psi) \rightarrow (\mathbb{T} \times \Psi)}$$

$$\text{inst}(\forall \alpha_1, \dots, \alpha_m. \tau, \psi) = (\text{subst}(\tau, \psi[\alpha_1 \mapsto \alpha'_1, \dots, \alpha_m \mapsto \alpha'_m]), \psi[\alpha'_1 \mapsto \bullet, \dots, \alpha'_m \mapsto \bullet])$$

$$\text{where} \quad \alpha'_1, \dots, \alpha'_m \notin \psi \wedge \forall 1 \leq i < j \leq m. \alpha'_i \neq \alpha'_j$$

free type variables and substitution are defined as following functions:

$$\boxed{\text{free}_\tau : (\mathbb{T} \times \Psi) \rightarrow \mathcal{P}(\mathbb{X}_\alpha)}$$

$$\text{free}_\tau(\tau, \psi) = \begin{cases} \text{free}_\tau(\tau', \psi) & \text{if } \tau = \alpha \wedge \psi(\alpha) = \tau' \\ \{\alpha\} & \text{if } \tau = \alpha \wedge \psi(\alpha) = \bullet \\ \text{free}_\tau(\tau_p, \psi) \cup \text{free}_\tau(\tau_r, \psi) & \text{if } \tau = (\tau_p \rightarrow \tau_r) \\ \emptyset & \text{otherwise} \end{cases}$$

$$\boxed{\text{free}_{\tau^\forall} : (\mathbb{T}^\forall \times \Psi) \rightarrow \mathcal{P}(\mathbb{X}_\alpha)}$$

$$\text{free}_{\tau^\forall}(\forall \alpha_1, \dots, \alpha_m. \tau, \psi) = \text{free}_\tau(\tau, \psi) \setminus \{\alpha_1, \dots, \alpha_m\}$$

$$\boxed{\text{free}_\Gamma : (\Gamma \times \Psi) \rightarrow \mathcal{P}(\mathbb{X}_\alpha)}$$

$$\text{free}_\Gamma([x_1 : \tau_1^\forall, \dots, x_n : \tau_n^\forall], \psi) = \text{free}_{\tau_1^\forall}(\tau_1^\forall, \psi) \cup \dots \cup \text{free}_{\tau_n^\forall}(\tau_n^\forall, \psi)$$

$$\boxed{\text{subst} : (\mathbb{T} \times \Psi) \rightarrow \mathbb{T}}$$

$$\text{subst}(\tau, \psi) = \begin{cases} \text{subst}(\tau', \psi) & \text{if } \tau = \alpha \wedge \psi(\alpha) = \tau' \\ \text{subst}(\tau_p, \psi) \rightarrow \text{subst}(\tau_r, \psi) & \text{if } \tau = (\tau_p \rightarrow \tau_r) \\ \tau & \text{otherwise} \end{cases}$$

5 SEMANTICS

We use the following notations in the semantics:

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

The big-step operational (natural) semantics of TIFAE 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 + num_2} \qquad \text{Mul} \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 \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 / 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 \% e_2 \Rightarrow n_1 \% 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) = 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}$$