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 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** 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 (=>) operator is right-associative. For expressions, the precedence and associativity of operators are defined as follows:

<table>
<tr><th>Description</th><th>Operator</th><th>Precedence</th><th>Associativity</th></tr>
<tr><td>Unary</td><td>-, !</td><td>1</td><td>right</td></tr>
<tr><td>Multiplicative</td><td>*, /, %</td><td>2</td><td rowspan="6">left</td></tr>
<tr><td>Additive</td><td>+, -</td><td>3</td></tr>
<tr><td>Relational</td><td><, <=, >, >=</td><td>4</td></tr>
<tr><td>Equality</td><td>==, !=</td><td>5</td></tr>
<tr><td>Logical Conjunction</td><td>&&</td><td>6</td></tr>
<tr><td>Logical Disjunction</td><td>||</td><td>7</td></tr>
</table>

3 ABSTRACT SYNTAX

The abstract syntax of TIFAE is defined as follows:

$$
\begin{array}{llll|ll}
\text{Expressions} & \mathbb{E} \ni e ::= n & (\texttt{Num}) & & \mid \texttt{val}\ x = e;\ e & (\texttt{Val}) \\
& \mid b & (\texttt{Bool}) & & \mid x & (\texttt{Id}) \\
& \mid e + e & (\texttt{Add}) & & \mid \lambda x.e & (\texttt{Fun}) \\
& \mid e * e & (\texttt{Mul}) & & \mid \texttt{def}\ x(x) = e;\ e & (\texttt{Rec}) \\
& \mid e \mathbin{/} e & (\texttt{Div}) & & \mid e(e) & (\texttt{App}) \\
& \mid e \mathbin{\%} e & (\texttt{Mod}) & & \mid \texttt{if}\ (e)\ e\ \texttt{else}\ e & (\texttt{If}) \\
& \mid e = e & (\texttt{Eq}) & & & \\
& \mid e < e & (\texttt{Lt}) & & &
\end{array}
$$

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

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

$$
\begin{array}{ll}
\mathcal{D}\llbracket -\ e \rrbracket = \mathcal{D}\llbracket e \rrbracket * \texttt{(-1)} & \mathcal{D}\llbracket e_1 \texttt{ != } e_2 \rrbracket = \mathcal{D}\llbracket \texttt{!}\ (e_1 \texttt{ == } e_2) \rrbracket \\
\mathcal{D}\llbracket \texttt{!}\ e \rrbracket = \texttt{if}\ (\mathcal{D}\llbracket e \rrbracket)\ \texttt{false else true} & \mathcal{D}\llbracket e_1 \texttt{ <= } e_2 \rrbracket = \mathcal{D}\llbracket (e_1 < e_2) \texttt{ || } (e_1 \texttt{ == } e_2) \rrbracket \\
\mathcal{D}\llbracket e_1 - e_2 \rrbracket = \mathcal{D}\llbracket e_1 \rrbracket + \mathcal{D}\llbracket -\ e_2 \rrbracket & \mathcal{D}\llbracket e_1 > e_2 \rrbracket = \mathcal{D}\llbracket \texttt{!}\ (e_1 \texttt{ <= } e_2) \rrbracket \\
\mathcal{D}\llbracket e_1 \texttt{ \&\& } e_2 \rrbracket = \texttt{if}\ (\mathcal{D}\llbracket e_1 \rrbracket)\ \mathcal{D}\llbracket e_2 \rrbracket\ \texttt{else false} & \mathcal{D}\llbracket e_1 \texttt{ >= } e_2 \rrbracket = \mathcal{D}\llbracket \texttt{!}\ (e_1 < e_2) \rrbracket \\
\mathcal{D}\llbracket e_1 \texttt{ || } e_2 \rrbracket = \texttt{if}\ (\mathcal{D}\llbracket e_1 \rrbracket)\ \texttt{true else}\ \mathcal{D}\llbracket e_2 \rrbracket &
\end{array}
$$

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		$\Gamma \in \mathbb{\Gamma} = \mathbb{X} \xrightarrow{\text{fin}} \mathbb{T}^{\forall}$	(TypeEnv)
Solution		$\psi \in \Psi = \mathbb{X}_{\alpha} \xrightarrow{\text{fin}} (\mathbb{T} \uplus \{\bot\})$	(Solution)
Types	$\mathbb{T} \ni$	$\tau ::= \mathsf{num}$	(NumT)
		$\mid \mathsf{bool}$	(BoolT)
		$\mid \tau \to \tau$	(ArrowT)
		$\mid \alpha$	(VarT)
Polymorphic Types	$\tau^{\forall} = \forall \alpha^{*}.\tau$	$\in \mathbb{T}^{\forall} = \mathbb{X}_{\alpha}^{*} \times \mathbb{T}$	(TypeEnv)
Type Variables		$\alpha \in \mathbb{X}_{\alpha}$	(String)

We skip the ∀-quantifier in polymorphic types if they have zero type variables. In the type system, type checking and type inference is defined with the following typing rules:

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

$$\tau\text{–Num}\ \frac{}{\Gamma, \psi \vdash n : \mathsf{num}, \psi} \qquad \tau\text{–Bool}\ \frac{}{\Gamma, \psi \vdash b : \mathsf{bool}, \psi}$$

$$\tau\text{–Add}\ \frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \mathtt{unify}(\tau_1, \mathsf{num}, \psi_2) = \psi_3 \quad \mathtt{unify}(\tau_2, \mathsf{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 + e_2 : \mathsf{num}, \psi_4}$$

$$\tau\text{–Mul}\ \frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \mathtt{unify}(\tau_1, \mathsf{num}, \psi_2) = \psi_3 \quad \mathtt{unify}(\tau_2, \mathsf{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 * e_2 : \mathsf{num}, \psi_4}$$

$$\tau\text{–Div}\ \frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \mathtt{unify}(\tau_1, \mathsf{num}, \psi_2) = \psi_3 \quad \mathtt{unify}(\tau_2, \mathsf{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1\ /\ e_2 : \mathsf{num}, \psi_4}$$

$$\tau\text{–Mod}\ \frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \mathtt{unify}(\tau_1, \mathsf{num}, \psi_2) = \psi_3 \quad \mathtt{unify}(\tau_2, \mathsf{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 \,\%\, e_2 : \mathsf{num}, \psi_4}$$

$$\tau\text{–Eq}\ \frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \mathtt{unify}(\tau_1, \mathsf{num}, \psi_2) = \psi_3 \quad \mathtt{unify}(\tau_2, \mathsf{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 = e_2 : \mathsf{bool}, \psi_4}$$

$$\tau\text{–Lt}\ \frac{\Gamma, \psi_0 \vdash e_1 : \tau_1, \psi_1 \quad \Gamma, \psi_1 \vdash e_2 : \tau_2, \psi_2 \quad \mathtt{unify}(\tau_1, \mathsf{num}, \psi_2) = \psi_3 \quad \mathtt{unify}(\tau_2, \mathsf{num}, \psi_3) = \psi_4}{\Gamma, \psi_0 \vdash e_1 < e_2 : \mathsf{bool}, \psi_4}$$

$$\tau\text{−Val}\;\frac{\Gamma,\psi_0 \vdash e_1 : \tau_1,\psi_1 \qquad \mathtt{gen}(\tau_1,\Gamma,\psi_1) = \tau_1^{\forall} \qquad \Gamma[x : \tau_1^{\forall}] \vdash e_2 : \tau_2,\psi_2}{\Gamma,\psi_0 \vdash \mathtt{val}\ x = e_1;\ e_2 : \tau_2,\psi_2}$$

$$\tau\text{−Id}\;\frac{\Gamma(x) = \tau^{\forall} \qquad \mathtt{inst}(\tau^{\forall},\psi) = (\tau,\psi')}{\Gamma,\psi \vdash x : \tau,\psi'} \qquad \tau\text{−Fun}\;\frac{\alpha_p \notin \psi \qquad \Gamma[x : \alpha_p],\psi[\alpha_p \mapsto \bot] \vdash e : \tau,\psi'}{\Gamma,\psi \vdash \lambda x.e : \alpha_p \to \tau,\psi'}$$

$$\tau\text{−App}\;\frac{\begin{array}{c}\Gamma,\psi_0 \vdash e_1 : \tau_1,\psi_1 \qquad \Gamma,\psi_1 \vdash e_2 : \tau_2,\psi_2 \\ \alpha_r \notin \psi_2 \qquad \mathtt{unify}(\tau_2 \to \alpha_r,\tau_1,\psi_2[\alpha_r \mapsto \bot]) = \psi_3\end{array}}{\Gamma,\psi_0 \vdash e_1(e_2) : \alpha_r,\psi_3}$$

$$\tau\text{−Rec}\;\frac{\begin{array}{c}\alpha_p,\alpha_r \notin \psi \qquad \alpha_p \neq \alpha_r \qquad \Gamma_1 = \Gamma[x_f \mapsto (\alpha_p \to \alpha_r)] \qquad \Gamma_2 = \Gamma_1[x_p \mapsto \alpha_p] \\ \Gamma_2,\psi[\alpha \mapsto \bot,\alpha' \mapsto \bot] \vdash e_b : \tau_b,\psi_b \qquad \mathtt{unify}(\tau_b,\alpha_r,\psi_b) = \psi_r \qquad \Gamma_1,\psi_r \vdash e_s : \tau_s,\psi_s\end{array}}{\Gamma,\psi \vdash \mathtt{def}\ x_f(x_p) = e_b;\ e_s : \tau_s,\psi_s}$$

$$\tau\text{−If}\;\frac{\begin{array}{c}\Gamma,\psi \vdash e_c : \mathsf{bool},\psi_c \qquad \Gamma,\psi_c \vdash e_t : \tau_t,\psi_t \\ \Gamma,\psi_t \vdash e_e : \tau_e,\psi_e \qquad \mathtt{unify}(\tau_c,\mathsf{bool},\psi_e) = \psi' \qquad \mathtt{unify}(\tau_t,\tau_e,\psi') = \psi''\end{array}}{\Gamma,\psi \vdash \mathtt{if}\ (e_c)\ e_t\ \mathtt{else}\ e_e : \tau_t,\psi''}$$

type unification is defined as a partial function:

$$\boxed{\mathtt{unify} : (\mathbb{T} \times \mathbb{T} \times \Psi) \rightharpoonup \Psi}$$

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

where $\tau_1' = \mathtt{resolve}(\tau_1,\psi)$ and $\tau_2' = \mathtt{resolve}(\tau_2,\psi)$.

type resolving and occurrence checking are defined as following functions:

$$\boxed{\mathtt{resolve} : (\mathbb{T} \times \Psi) \to \mathbb{T}}$$

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

$$\boxed{\mathtt{occur} : (\mathbb{X}_\alpha \times \mathbb{T} \times \Psi) \to \mathsf{bool}}$$

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

type generalization and instantiation are defined as following functions:

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

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

$$\boxed{\texttt{inst} : (\mathbb{T}^{\forall} \times \Psi) \to (\mathbb{T} \times \Psi)}$$

$$\texttt{inst}(\forall \alpha_1, \ldots, \alpha_m.\tau, \psi) = (\texttt{subst}(\tau, \psi[\alpha_1 \mapsto \alpha'_1, \ldots, \alpha_m \mapsto \alpha'_m]), \psi[\alpha'_1 \mapsto \bot, \ldots, \alpha'_m \mapsto \bot])$$

$$\text{where} \qquad \alpha'_1, \ldots, \alpha'_m \notin \psi \wedge \forall 1 \le i < j \le m.\ \alpha'_i \ne \alpha'_j$$

free type variables and substitution are defined as following functions:

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

$$\texttt{free}_{\tau}(\tau, \psi) = \begin{cases} \texttt{free}_{\tau'}(\tau', \psi) & \text{if } \tau = \alpha \wedge \psi(\alpha) = \tau' \\ \{\alpha\} & \text{if } \tau = \alpha \wedge \psi(\alpha) = \bot \\ \texttt{free}_{\tau_p}(\tau_p, \psi) \cup \texttt{free}_{\tau_r}(\tau_r, \psi) & \text{if } \tau = (\tau_p \to \tau_r) \\ \varnothing & \text{otherwise} \end{cases}$$

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

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

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

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

$$\boxed{\texttt{subst} : (\mathbb{T} \times \Psi) \to \mathbb{T}}$$

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

5 SEMANTICS

We use the following notations in the semantics:

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

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

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

$$\texttt{Num}\ \frac{}{\sigma \vdash n \Rightarrow n} \qquad \texttt{Bool}\ \frac{}{\sigma \vdash b \Rightarrow b}$$

$$\texttt{Add}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1\ \texttt{+}\ e_2 \Rightarrow n_1 + num_2} \qquad \texttt{Mul}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1\ \texttt{*}\ e_2 \Rightarrow n_1 \times n_2}$$

$$\texttt{Div}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2 \qquad n_2 \neq 0}{\sigma \vdash e_1\ \texttt{/}\ e_2 \Rightarrow n_1 / n_2} \qquad \texttt{Mod}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2 \qquad n_2 \neq 0}{\sigma \vdash e_1\ \texttt{\%}\ e_2 \Rightarrow n_1\ \%\ n_2}$$

$$\texttt{Eq}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1\ \texttt{==}\ e_2 \Rightarrow n_1 = n_2} \qquad \texttt{Lt}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1\ \texttt{<}\ e_2 \Rightarrow n_1 < n_2}$$

$$\texttt{Val}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \texttt{val}\ x\ \texttt{=}\ e_1\texttt{;}\ e_2 \Rightarrow v_2} \qquad \texttt{Id}\ \frac{x \in \mathrm{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

$$\texttt{Fun}\ \frac{}{\sigma \vdash \lambda x.e \Rightarrow \langle \lambda x.e, \sigma \rangle} \qquad \texttt{Rec}\ \frac{\sigma' = \sigma[x_0 \mapsto \langle \lambda x_1.e_2, \sigma' \rangle] \qquad \sigma' \vdash e_3 \Rightarrow v_3}{\sigma \vdash \texttt{def}\ x_0(x_1)\ \texttt{=}\ e_2\texttt{;}\ e_3 \Rightarrow v_3}$$

$$\texttt{App}\ \frac{\sigma \vdash e_0 \Rightarrow \langle \lambda x.e_2, \sigma' \rangle \qquad \sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma'[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash e_0(e_1) \Rightarrow v_2}$$

$$\texttt{If}_T\ \frac{\sigma \vdash e_0 \Rightarrow \texttt{true} \qquad \sigma \vdash e_1 \Rightarrow v_1}{\sigma \vdash \texttt{if}\ (e_0)\ e_1\ \texttt{else}\ e_2 \Rightarrow v_1} \qquad \texttt{If}_F\ \frac{\sigma \vdash e_0 \Rightarrow \texttt{false} \qquad \sigma \vdash e_2 \Rightarrow v_2}{\sigma \vdash \texttt{if}\ (e_0)\ e_1\ \texttt{else}\ e_2 \Rightarrow v_2}$$