TAFAE – TRFAE with Algebraic Data Types

1 INTRODUCTION

TAFAE is a toy language for the COSE212 course at Korea University. TAFAE stands for an extension of the TRFAE language with **algebraic data types**, 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**
- **algebraic data types** (`enum`) and **pattern matching** (`match`)

This document is the specification of TAFAE. 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 TAFAE.

2 CONCRETE SYNTAX

The concrete syntax of TAFAE 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> | <params> "=>" <expr>
         | "def" <id> <params> ":" <type> "=" <expr> ";"? <expr>
         | <expr> "(" ")" | <expr> "(" <expr> [ "," <expr> ]* ")"
         | "if" "(" <expr> ")" <expr> "else" <expr>
         | "enum" <id> "{" [ <variant> ";"? ]+ "}" ";"? <expr>
         | <expr> "match" "{" [ <mcase> ";"? ]+ "}"
```

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

// function parameters
<params> ::= "(" ")" | "(" <param> [ "," <param> ]* ")"
<param>  ::= <id> ":" <type>

// variants
<variant> ::= "case" <id> "(" ")"
            | "case" <id> "(" <type> [ "," <type> ]* ")"

// match cases
<mcase> ::= "case" <id> "(" ")" "=>" <expr>
          | "case" <id> "(" <id> [ "," <id> ]* ")" "=>" <expr>

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

For types, the arrow (=>) 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	
Pattern Matching	match	8	

3 ABSTRACT SYNTAX

The abstract syntax of TAFAE is defined as follows:

$$
\begin{array}{lrlllll}
\text{Expressions} & \mathbb{E} \ni e ::= & n & (\texttt{Num}) & \mid \texttt{val}\ x = e;\ e & (\texttt{Val}) \\
 & \mid & b & (\texttt{Bool}) & \mid x & (\texttt{Id}) \\
 & \mid & e + e & (\texttt{Add}) & \mid \lambda([x{:}\tau]^*).e & (\texttt{Fun}) \\
 & \mid & e \times e & (\texttt{Mul}) & \mid \texttt{def}\ x([x{:}\tau]^*){:}\tau = e;\ e & (\texttt{Rec}) \\
 & \mid & e \div e & (\texttt{Div}) & \mid e(e^*) & (\texttt{App}) \\
 & \mid & e\ \texttt{mod}\ e & (\texttt{Mod}) & \mid \texttt{if}\ (e)\ e\ \texttt{else}\ e & (\texttt{If}) \\
 & \mid & e = e & (\texttt{Eq}) & \mid \texttt{enum}\ t\ \{\ [\texttt{case}\ x(\tau^*)]^*\ \};\ e & (\texttt{TypeDef}) \\
 & \mid & e < e & (\texttt{Lt}) & \mid e\ \texttt{match}\ \{\ [\texttt{case}\ x(x^*)\ \texttt{=>}\ e]^*\ \} & (\texttt{Match})
\end{array}
$$

$$
\begin{array}{lrll}
\text{Types} & \mathbb{T} \ni \tau ::= & \texttt{num} & (\texttt{NumT}) \\
 & \mid & \texttt{bool} & (\texttt{BoolT}) \\
 & \mid & (\tau^*) \rightarrow \tau & (\texttt{ArrowT}) \\
 & \mid & t & (\texttt{VarT})
\end{array}
\qquad
\begin{array}{llll}
\text{Numbers} & n & \in \mathbb{Z} & (\texttt{BigInt}) \\
\text{Identifiers} & x & \in \mathbb{X} & (\texttt{String}) \\
\text{Booleans} & b & \in \mathbb{B} = \{\texttt{true}, \texttt{false}\} & (\texttt{Boolean})
\end{array}
$$

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

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

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

4 TYPE SYSTEM

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

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

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

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

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

$$\tau\text{–}\texttt{Add}\ \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 + e_2 : \texttt{num}} \qquad \tau\text{–}\texttt{Mul}\ \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 \times e_2 : \texttt{num}}$$

$$\tau\text{–}\texttt{Div}\ \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 \div e_2 : \texttt{num}} \qquad \tau\text{–}\texttt{Mod}\ \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1\ \texttt{mod}\ e_2 : \texttt{num}}$$

$$\tau\text{–}\texttt{Eq}\ \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 = e_2 : \texttt{bool}} \qquad \tau\text{–}\texttt{Lt}\ \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 < e_2 : \texttt{bool}}$$

$$\tau\text{–}\texttt{Val}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma[x : \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{val}\ x = e_1;\ e_2 : \tau_2} \qquad \tau\text{–}\texttt{Id}\ \frac{x \in \text{Domain}(\Gamma)}{\Gamma \vdash x : \Gamma(x)}$$

$$\tau\text{–}\texttt{Fun}\ \frac{\Gamma \vdash \tau_1 \quad \dots \quad \Gamma \vdash \tau_n \qquad \Gamma[x_1 : \tau_1, \dots, x_n : \tau_n] \vdash e : \tau}{\Gamma \vdash \lambda(x_1 : \tau_1, \dots, x_n : \tau_n).e : (\tau_1, \dots, \tau_n) \to \tau}$$

$$\tau\text{–}\texttt{Rec}\ \frac{\begin{array}{c}\Gamma \vdash \tau_1 \qquad \dots \qquad \Gamma \vdash \tau_n \\ \Gamma[x_0 : (\tau_1, \dots, \tau_n) \to \tau, x_1 : \tau_1, \dots, x_n : \tau_n] \vdash e : \tau \qquad \Gamma[x_0 : (\tau_1, \dots, \tau_n) \to \tau] \vdash e' : \tau'\end{array}}{\Gamma \vdash \texttt{def}\ x_0(x_1 : \tau_1, \dots, x_n : \tau_n) : \tau = e;\ e' : \tau'}$$

$$\tau\text{–}\texttt{App}\ \frac{\Gamma \vdash e_0 : (\tau_1, \dots, \tau_n) \to \tau \qquad \Gamma \vdash e_1 : \tau_1 \qquad \dots \qquad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash e_0(e_1, \dots, e_n) : \tau}$$

$$\tau\text{–}\texttt{If}\ \frac{\Gamma \vdash e_0 : \texttt{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{if}\ (e_0)\ e_1\ \texttt{else}\ e_2 : \tau}$$

$$\tau\text{–}\mathtt{TypeDef}\ \frac{\Gamma' = \Gamma[t = x_1(\tau_{1,1},\ldots,\tau_{1,m_1}) + \ldots + x_n(\tau_{n,1},\ldots,\tau_{n,m_n})] \qquad \Gamma' \vdash \tau_{1,1} \quad \ldots \quad \Gamma' \vdash \tau_{n,m_n} \\ \Gamma'[x_1 : (\tau_{1,1},\ldots,\tau_{1,m_1}) \to t, \ldots, x_n : (\tau_{n,1},\ldots,\tau_{n,m_n}) \to t] \vdash e : \tau}{\Gamma \vdash \mathsf{enum}\ t\ \{\ \mathtt{case}\ x_1(\tau_{1,1},\ldots,\tau_{1,m_1});\ \ldots;\ \mathtt{case}\ x_n(\tau_{n,1},\ldots,\tau_{n,m_n})\ \};\ e : \tau}$$

$$\tau\text{–}\mathtt{Match}\ \frac{\Gamma \vdash e : t \qquad \Gamma(t) = x_1(\tau_{1,1},\ldots,\tau_{1,m_1}) + \ldots + x_n(\tau_{n,1},\ldots,\tau_{n,m_n}) \\ \Gamma[x_{1,1} : \tau_{1,1},\ldots,x_{1,m_1} : \tau_{1,m_1}] \vdash e_1 : \tau \qquad \ldots \qquad \Gamma[x_{n,1} : \tau_{n,1},\ldots,x_{n,m_n} : \tau_{n,m_n}] \vdash e_n : \tau}{\Gamma \vdash e\ \mathtt{match}\ \{\ \mathtt{case}\ x_1(x_{1,1},\ldots,x_{1,m_1})\ \texttt{=>}\ e_1;\ \ldots;\ \mathtt{case}\ x_n(x_{n,1},\ldots,x_{n,m_n})\ \texttt{=>}\ e_n\ \} : \tau}$$

with the following rules for well-formedness:

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

$$\frac{}{\Gamma \vdash \mathsf{num}} \qquad \frac{}{\Gamma \vdash \mathsf{bool}} \qquad \frac{\Gamma \vdash \tau_1 \quad \ldots \quad \Gamma \vdash \tau_n \qquad \Gamma \vdash \tau}{\Gamma \vdash (\tau_1,\ldots,\tau_n) \to \tau}$$

$$\frac{\Gamma(t) = x_1(\tau_{1,1},\ldots,\tau_{1,m_1}) + \ldots + x_n(\tau_{n,1},\ldots,\tau_{n,m_n})}{\Gamma \vdash t}$$

5 SEMANTICS

We use the following notations in the semantics:

$$\text{Environments} \qquad \sigma \ \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{V} \quad (\mathtt{Env})$$

$$\begin{array}{llll} \text{Values} \quad \mathbb{V} \ni v ::= n & (\mathtt{NumV}) & \mid \langle x \rangle & (\mathtt{ConstrV}) \\ \qquad\qquad \mid b & (\mathtt{BoolV}) & \mid x(v^*) & (\mathtt{VariantV}) \\ \qquad\qquad \mid \langle \lambda x.(e,\ldots,e), \sigma \rangle & (\mathtt{CloV}) & & \end{array}$$

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

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

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

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

$$\mathtt{Div}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2 \qquad n_2 \neq 0}{\sigma \vdash e_1 \div e_2 \Rightarrow n_1 \div n_2} \qquad \mathtt{Mod}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2 \qquad n_2 \neq 0}{\sigma \vdash e_1\ \mathsf{mod}\ e_2 \Rightarrow n_1\ \mathsf{mod}\ n_2}$$

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

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

$$\textsf{Fun}\ \frac{}{\sigma \vdash \lambda(x_1 : \tau_1, \ldots, x_n : \tau_n).e \Rightarrow \langle \lambda(x_1, \ldots, x_n).e, \sigma \rangle}$$

$$\textsf{Rec}\ \frac{\sigma' = \sigma[x_0 \mapsto \langle \lambda(x_1, \ldots, x_n).e, \sigma' \rangle] \qquad \sigma' \vdash e' \Rightarrow v'}{\sigma \vdash \texttt{def}\ x_0(x_1 : \tau_1, \ldots, x_n : \tau_n) : \tau = e;\ e' \Rightarrow v'}$$

$$\textsf{App}_\lambda\ \frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow \langle \lambda(x_1, \ldots, x_n).e, \sigma' \rangle \\ \sigma \vdash e_1 \Rightarrow v_1 \quad \ldots \quad \sigma \vdash e_n \Rightarrow v_n \qquad \sigma'[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n] \vdash e \Rightarrow v\end{array}}{\sigma \vdash e_0(e_1, \ldots, e_n) \Rightarrow v}$$

$$\textsf{App}_{\langle - \rangle}\ \frac{\sigma \vdash e_0 \Rightarrow \langle x \rangle \qquad \sigma \vdash e_1 \Rightarrow v_1 \qquad \ldots \qquad \sigma \vdash e_n \Rightarrow v_n}{\sigma \vdash e_0(e_1, \ldots, e_n) \Rightarrow x(v_1, \ldots, v_n)}$$

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

$$\texttt{TypeDef}\ \frac{\sigma[x_1 \mapsto \langle x_1 \rangle, \ldots, x_n \mapsto \langle x_n \rangle] \vdash e \Rightarrow v}{\sigma \vdash \texttt{enum}\ t\ \{\ \texttt{case}\ x_1(\tau_{1,1}, \ldots, \tau_{1,m_1});\ \ldots;\ \texttt{case}\ x_n(\tau_{n,1}, \ldots, \tau_{n,m_n})\ \};\ e \Rightarrow v}$$

$$\texttt{Match}\ \frac{\begin{array}{c}1 \le i \le n \\ \sigma \vdash e \Rightarrow x_i(v_1, \ldots, v_{m_i}) \qquad \forall j < i.\ x_j \neq x_i \qquad \sigma[x_{i,1} \mapsto v_1, \ldots, x_{i,m_i} \mapsto v_{m_i}] \vdash e_i \Rightarrow v\end{array}}{\sigma \vdash e\ \texttt{match}\ \{\ \texttt{case}\ x_1(x_{1,1}, \ldots, x_{1,m_1})\ \texttt{=>}\ e_1;\ \ldots;\ \texttt{case}\ x_n(x_{n,1}, \ldots, x_{n,m_n})\ \texttt{=>}\ e_n\ \} \Rightarrow v}$$