ATFAE – TRFAE with Algebraic Data Types

1 INTRODUCTION

ATFAE is a toy language for the COSE212 course at Korea University. ATFAE stands for an extension of the TRFAE language with **algebraic data types**, 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 (!)
- **algebraic data types** (enum)
- **pattern matching** (match)
- **static type checking**

This document is the specification of ATFAE. 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 ATFAE.

2 CONCRETE SYNTAX

The concrete syntax of ATFAE 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.

```
<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" | "enum"
             | "case" | "match" | "Number" | "Boolean"
<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" | <id>
         | "(" ")" "=>" <type> | <type> "=>" <type>
         | "(" <type> ["," <type>]* ")" "=>" <type>
```

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	
Additive	+, -	3	
Relational	<, <=, >, >=	4	
Equality	==, !=	5	left
Logical Conjunction	&&	6	
Logical Disjunction	||	7	
Pattern Matching	match	8	

3 ABSTRACT SYNTAX

The abstract syntax of ATFAE is defined as follows:

Expressions $\mathbb{E} \ni e ::= n$ (Num) $\quad | \text{ val } x = e; e$ (Val)
$| b$ (Bool) $\quad | x$ (Id)
$| e + e$ (Add) $\quad | \lambda([x:\tau]^*).e$ (Fun)
$| e * e$ (Mul) $\quad | \text{def } x([x:\tau]^*):\tau = e; e$ (Rec)
$| e / e$ (Div) $\quad | e(e^*)$ (App)
$| e \% e$ (Mod) $\quad | \text{if } (e) \; e \text{ else } e$ (If)
$| e == e$ (Eq) $\quad | \text{enum } t \; \{ [\text{case } x(\tau^*)]^+ \}; e$ (TypeDef)
$| e < e$ (Lt) $\quad | e \text{ match } \{ [\text{case } x(x^*) => e]^+ \}$ (Match)

Types $\mathbb{T} \ni \tau ::= \text{num}$ (NumT) $\quad$ Numbers $\quad n \in \mathbb{Z}$ (BigInt)
$| \text{bool}$ (BoolT) $\quad$ Identifiers $\quad x \in \mathbb{X}$ (String)
$| (\tau^*) \rightarrow \tau$ (ArrowT) $\quad$ Booleans $\quad b \in \mathbb{B} = \{\text{true}, \text{false}\}$ (Boolean)
$| t$ (NameT) $\quad$ Type Names $\quad t \in \mathbb{X}_t$ (String)

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

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

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

4 TYPE SYSTEM

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

$$\text{Type Environments} \quad \Gamma \;\in\; (\mathbb{X} \xrightarrow{\text{fin}} \mathbb{T}) \times (\mathbb{X}_t \xrightarrow{\text{fin}} (\mathbb{X} \xrightarrow{\text{fin}} \mathbb{T}^*)) \quad (\text{TypeEnv})$$

The right part of the type environment is a mapping from each type name to its variants, where each variant is a mapping from a constructor to its argument types. Note that the order of variants is not significant. For example,

$$A = B(\text{bool}) + C(\text{num}) \qquad \text{equivalent to} \qquad A = C(\text{num}) + B(\text{bool})$$

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

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

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

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

$$\tau\text{–Div} \; \frac{\Gamma \vdash e_1 : \text{num} \qquad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 \,/\, e_2 : \text{num}} \qquad \tau\text{–Mod} \; \frac{\Gamma \vdash e_1 : \text{num} \qquad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 \,\% \, e_2 : \text{num}}$$

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

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

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

$$\tau\text{–Rec} \; \frac{\Gamma \vdash \tau_1 \quad \dots \quad \Gamma \vdash \tau_n \qquad \Gamma \vdash \tau \qquad\qquad}{\Gamma \vdash \text{def } x_0(x_1{:}\tau_1, \dots, x_n{:}\tau_n){:}\tau = e; \; e' : \tau'}$$

with the premises $\Gamma[x_0 : (\tau_1, \dots, \tau_n) \to \tau, x_1 : \tau_1, \dots, x_n : \tau_n] \vdash e : \tau$ and $\Gamma[x_0 : (\tau_1, \dots, \tau_n) \to \tau] \vdash e' : \tau'$

$$\tau\text{--App} \quad \frac{\Gamma \vdash e_0 : (\tau_1, \ldots, \tau_n) \to \tau \qquad \Gamma \vdash e_1 : \tau_1 \qquad \ldots \qquad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash e_0(e_1, \ldots, e_n) : \tau}$$

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

$$\tau\text{--TypeDef} \quad \frac{\begin{array}{c} \Gamma' = \Gamma[t = x_1(\tau_{1,1}, \ldots, \tau_{1,m_1}) + \ldots + x_n(\tau_{n,1}, \ldots, \tau_{n,m_n})] \\ t \notin \text{Domain}(\Gamma) \qquad \Gamma' \vdash \tau_{1,1} \quad \ldots \quad \Gamma' \vdash \tau_{n,m_n} \\ \Gamma'[x_1 : (\tau_{1,1}, \ldots, \tau_{1,m_1}) \to t, \ldots, x_n : (\tau_{n,1}, \ldots, \tau_{n,m_n}) \to t] \vdash e : \tau \qquad \Gamma \vdash \tau \end{array}}{\Gamma \vdash \texttt{enum } t \;\{\; \texttt{case } x_1(\tau_{1,1}, \ldots, \tau_{1,m_1}); \; \ldots; \; \texttt{case } x_n(\tau_{n,1}, \ldots, \tau_{n,m_n}) \;\}; \; e : \tau}$$

$$\tau\text{--Match} \quad \frac{\begin{array}{c} \Gamma \vdash e : t \qquad \Gamma(t) = x_1(\tau_{1,1}, \ldots, \tau_{1,m_1}) + \ldots + x_n(\tau_{n,1}, \ldots, \tau_{n,m_n}) \\ \forall 1 \le i \le n.\; \Gamma_i = \Gamma[x_{i,1} : \tau_{i,1}, \ldots, x_{i,m_i} : \tau_{i,m_i}] \qquad \Gamma_1 \vdash e_1 : \tau \qquad \ldots \qquad \Gamma_n \vdash e_n : \tau \end{array}}{\Gamma \vdash e \texttt{ match } \{\; \texttt{case } x_1(x_{1,1}, \ldots, x_{1,m_1}) \texttt{ => } e_1; \; \ldots; \; \texttt{case } x_n(x_{n,1}, \ldots, x_{n,m_n}) \texttt{ => } e_n \;\} : \tau}$$

and the following rules for well-formedness of types:

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

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

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

5 SEMANTICS

We use the following notations in the semantics:

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

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

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

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

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

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

$$\text{Div} \; \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2 \qquad n_2 \ne 0}{\sigma \vdash e_1 \texttt{ / } e_2 \Rightarrow n_1 \div n_2} \qquad \text{Mod} \; \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2 \qquad n_2 \ne 0}{\sigma \vdash e_1 \texttt{ \% } e_2 \Rightarrow n_1 \% n_2}$$

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

$$\text{Lt} \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 < e_2 \Rightarrow n_1 < n_2}$$

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

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

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

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

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

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

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

$$\text{Match} \frac{1 \le i \le n \\ \sigma \vdash e \Rightarrow x_i(v_1, \ldots, v_{m_i}) \qquad \forall j < i.\ x_j \ne x_i \qquad \sigma[x_{i,1} \mapsto v_1, \ldots, x_{i,m_i} \mapsto v_{m_i}] \vdash e_i \Rightarrow v}{\sigma \vdash e\ \mathtt{match}\ \{\ \mathtt{case}\ x_1(x_{1,1}, \ldots, x_{1,m_1}) \Rightarrow e_1;\ \ldots;\ \mathtt{case}\ x_n(x_{n,1}, \ldots, x_{n,m_n}) \Rightarrow e_n\ \} \Rightarrow v}$$