

# PTFAE – TFAE with Parametric Polymorphism

## 1 INTRODUCTION

PTFAE is a toy language for the [COSE212](#) course at Korea University. PTFAE stands for an extension of the [TFAE](#) language with **parametric polymorphism**, and it supports the following features:

- **number (integer) values**
- **basic arithmetic operators**: addition (+) and multiplication (\*)
- **first-class functions**
- **immutable variables** (val)
- **parametric polymorphism** (forall)
- **static type checking**

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

## 2 CONCRETE SYNTAX

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

// expressions
<expr>       ::= <number> | <expr> "+" <expr> | <expr> "*" <expr>
               | "(" <expr> ")" | "{" <expr> "}"
               | "val" <id> "=" <expr> ";"? <expr> | <id>
               | "(" <id> ":" <type> ")" "=>" <expr> | <expr> "(" <expr> ")"
               | "forall" "[" <type> "]" <expr> | <expr> "[" <type> "]"

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

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

Operator	Associativity	Precedence
*	left	1
+	left	2

### 3 ABSTRACT SYNTAX

The abstract syntax of PTFAE is defined as follows:

Expressions	$\mathbb{E} \ni e ::= n$	(Num)	Numbers	$n \in \mathbb{Z}$	(BigInt)
	$e + e$	(Add)	Identifiers	$x \in \mathbb{X}$	(String)
	$e \times e$	(Mul)	Type Variables	$\alpha \in \mathbb{X}_\alpha$	(String)
	$\text{val } x = e; e$	(Val)			
	$x$	(Id)	Types	$\mathbb{T} \ni \tau ::= \text{num}$	(NumT)
	$\lambda x : \tau. e$	(Fun)		$\tau \rightarrow \tau$	(ArrowT)
	$e(e)$	(App)		$\alpha$	(VarT)
	$\forall \alpha. e$	(TypeAbs)		$\forall \alpha. \tau$	(PolyT)
	$e[\tau]$	(TypeApp)			

### 4 TYPE SYSTEM

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

$$\text{Type Environments} \quad \Gamma \in (\mathbb{X} \xrightarrow{\text{fin}} \mathbb{T}) \times \mathcal{P}(\mathbb{X}_\alpha) \quad (\text{TypeEnv})$$

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

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

$$\begin{array}{c}
\tau\text{-Num} \frac{}{\Gamma \vdash n : \text{num}} \quad \tau\text{-Add} \frac{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 + e_2 : \text{num}} \quad \tau\text{-Mul} \frac{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 \times e_2 : \text{num}} \\
\\
\tau\text{-Val} \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma[x : \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \text{val } x = e_1; e_2 : \tau_2} \quad \tau\text{-Id} \frac{x \in \text{Domain}(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \\
\\
\tau\text{-Fun} \frac{\Gamma \vdash \tau \quad \Gamma[x : \tau] \vdash e : \tau'}{\Gamma \vdash \lambda x : \tau. e : \tau \rightarrow \tau'} \quad \tau\text{-App} \frac{\Gamma \vdash e_0 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_1 : \tau_3 \quad \tau_1 \equiv \tau_3}{\Gamma \vdash e_0(e_1) : \tau_2} \\
\\
\tau\text{-TypeAbs} \frac{\alpha \notin \text{Domain}(\Gamma) \quad \Gamma[\alpha] \vdash e : \tau}{\Gamma \vdash \forall \alpha. e : \forall \alpha. \tau} \quad \tau\text{-TypeApp} \frac{\Gamma \vdash \tau \quad \Gamma \vdash e : \forall \alpha. \tau'}{\Gamma \vdash e[\tau] : \tau'[\alpha \leftarrow \tau]}
\end{array}$$

the following rules for well-formedness of types:

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

$$\begin{array}{c}
\frac{}{\Gamma \vdash \text{num}} \quad \frac{\Gamma \vdash \tau \quad \Gamma \vdash \tau'}{\Gamma \vdash \tau \rightarrow \tau'} \quad \frac{\alpha \in \text{Domain}(\Gamma)}{\Gamma \vdash \alpha} \quad \frac{\Gamma[\alpha] \vdash \tau}{\Gamma \vdash \forall \alpha. \tau}
\end{array}$$

and the following rules for type equivalence:

$$\boxed{\tau \equiv \tau}$$

$$\begin{array}{c}
\frac{}{\text{num} \equiv \text{num}} \quad \frac{\tau_1 \equiv \tau'_1 \quad \tau_2 \equiv \tau'_2}{(\tau_1 \rightarrow \tau_2) \equiv (\tau'_1 \rightarrow \tau'_2)} \quad \frac{}{\alpha \equiv \alpha} \quad \frac{\tau \equiv \tau'[\alpha' \leftarrow \alpha]}{\forall \alpha. \tau \equiv \forall \alpha'. \tau'}
\end{array}$$

## 5 SEMANTICS

We use the following notations in the semantics:

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

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

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

$$\begin{array}{c}
 \text{Num} \frac{}{\sigma \vdash n \Rightarrow n} \quad \text{Add} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \quad \text{Mul} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 \times e_2 \Rightarrow n_1 \times n_2} \\
 \\
 \text{Val} \frac{\sigma \vdash e_1 \Rightarrow v_1 \quad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \text{val } x=e_1; e_2 \Rightarrow v_2} \quad \text{Fun} \frac{}{\sigma \vdash \lambda x:\tau.e \Rightarrow \langle \lambda x.e, \sigma \rangle} \\
 \\
 \text{Id} \frac{x \in \text{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)} \quad \text{App} \frac{\sigma \vdash e_0 \Rightarrow \langle \lambda x.e_2, \sigma' \rangle \quad \sigma \vdash e_1 \Rightarrow v_1 \quad \sigma'[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash e_0(e_1) \Rightarrow v_2} \\
 \\
 \text{TypeAbs} \frac{}{\sigma \vdash \forall \alpha.e \Rightarrow \langle \forall \alpha.e, \sigma \rangle} \quad \text{TypeApp} \frac{\sigma \vdash e \Rightarrow \langle \forall \alpha.e', \sigma' \rangle \quad \sigma' \vdash e' \Rightarrow v}{\Gamma \vdash e[\tau] : v}
 \end{array}$$