

STFAE – TFAE with Subtype Polymorphism

1 INTRODUCTION

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

- **number (integer) values** (0, 1, -1, 2, -2, 3, -3, ...)
- **arithmetic operators**: addition (+) and multiplication (*)
- **immutable variable definitions** (val)
- **first-class functions** (=>)
- **records** ({...}) and **record access** (.)
- **subtype polymorphism**
- **bottom type** (Bot) and **top type** (Top)
- **exit** (exit)
- **static type checking**

This document is the specification of STFAE. 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 STFAE.

2 CONCRETE SYNTAX

The concrete syntax of STFAE 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. The notation `+{A}` or `*{A}` denotes the same as `+` or `*`, respectively, but the elements are separated by the element `A`. 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" | "exit" | "Number" | "Bot" | "Top"
<id>       ::= <idstart> <idcont>* butnot <keyword>

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

// types
<type>     ::= "(" <type> ")" | "Number" | <type> "==" <type>
              | "{" <id> ":" <type>*{" , " } "}" | "Bot" | "Top"
```

Duplicate field names are not allowed in record expressions and record types. For types, the arrow (\Rightarrow) operator is right-associative. For expressions, the precedence and associativity of operators are defined as follows:

Operator	Associativity	Precedence
.	left	1
*	left	2
+	left	3

3 ABSTRACT SYNTAX

The abstract syntax of STFAE is defined as follows:

Expressions	$\mathbb{E} \ni e ::= n$	(Num)	$\lambda x : \tau. e$	(Fun)
	$ e + e$	(Add)	$ e(e)$	(App)
	$ e * e$	(Mul)	$ \{[x = e]^*\}$	(Record)
	$ \text{val } x [: \tau]^? = e; e$	(Val)	$ e.x$	(Access)
	$ x$	(Id)	$ \text{exit}$	(Exit)
Types	$\mathbb{T} \ni \tau ::= \text{num}$	(NumT)	$ \perp$	(BotT)
	$ \tau \rightarrow \tau$	(ArrowT)	$ \top$	(TopT)
	$ \{[x : \tau]^*\}$	(RecordT)		
Numbers	$n \in \mathbb{Z}$	(BigInt)	Identifiers	$x \in \mathbb{X}$ (String)

4 TYPE SYSTEM

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

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

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

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

$$\begin{array}{c}
\tau\text{-Num} \frac{}{\Gamma \vdash n : \text{num}} \quad \tau\text{-Add} \frac{\Gamma \vdash e_1 : \tau_1 \quad \tau_1 <: \text{num} \quad \Gamma \vdash e_2 : \tau_2 \quad \tau_2 <: \text{num}}{\Gamma \vdash e_1 + e_2 : \text{num}} \\
\\
\tau\text{-Mul} \frac{\Gamma \vdash e_1 : \tau_1 \quad \tau_1 <: \text{num} \quad \Gamma \vdash e_2 : \tau_2 \quad \tau_2 <: \text{num}}{\Gamma \vdash e_1 * e_2 : \text{num}} \quad \tau\text{-Id} \frac{x \in \text{Domain}(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \\
\\
\tau\text{-Val} \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma[x : \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \text{val } x = e_1; e_2 : \tau_2} \quad \tau\text{-Val}_\tau \frac{\Gamma \vdash e_1 : \tau_1 \quad \tau_1 <: \tau_0 \quad \Gamma[x : \tau_0] \vdash e_2 : \tau_2}{\Gamma \vdash \text{val } x : \tau_0 = e_1; e_2 : \tau_2} \\
\\
\tau\text{-Fun} \frac{\Gamma \vdash \tau \quad \Gamma[x : \tau] \vdash e : \tau'}{\Gamma \vdash \lambda x : \tau. e : \tau \rightarrow \tau'} \quad \tau\text{-App} \frac{\Gamma \vdash e_0 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_1 : \tau_3 \quad \tau_3 <: \tau_1}{\Gamma \vdash e_0(e_1) : \tau_2} \\
\\
\tau\text{-Record} \frac{\Gamma \vdash e_1 : \tau_1 \quad \dots \quad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash \{x_1 = e_1, \dots, x_n = e_n\} : \{x_1 : \tau_1, \dots, x_n : \tau_n\}} \\
\\
\tau\text{-Access} \frac{\Gamma \vdash e : \{x_1 : \tau_1, \dots, x_n : \tau_n\} \quad 1 \leq i \leq n}{\Gamma \vdash e.x_i : \tau_i} \quad \tau\text{-Exit} \frac{}{\Gamma \vdash \text{exit} : \perp}
\end{array}$$

the following rules for subtyping:

$$\boxed{\tau <: \tau}$$

$$\frac{}{\perp <: \tau} \quad \frac{}{\tau <: \top} \quad \frac{}{\tau <: \tau} \quad \frac{\tau_1 :> \tau'_1 \quad \tau_2 <: \tau'_2}{(\tau_1 \rightarrow \tau_2) <: (\tau'_1 \rightarrow \tau'_2)}$$

$$\frac{}{\{x_1 : \tau_1, \dots, x_n : \tau_n, x : \tau\} <: \{x_1 : \tau_1, \dots, x_n : \tau_n\}} \quad \frac{\tau_1 <: \tau'_1 \quad \dots \quad \tau_n <: \tau'_n}{\{x_1 : \tau_1, \dots, x_n : \tau_n\} <: \{x_1 : \tau'_1, \dots, x_n : \tau'_n\}}$$

$$\frac{\{x_1 : \tau_1, \dots, x_n : \tau_n\} \text{ is a permutation of } \{x'_1 : \tau'_1, \dots, x'_n : \tau'_n\}}{\{x_1 : \tau_1, \dots, x_n : \tau_n\} <: \{x'_1 : \tau'_1, \dots, x'_n : \tau'_n\}}$$

5 SEMANTICS

We use the following notations in the semantics:

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

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

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

$$\begin{array}{l}
 \text{Num} \frac{}{\sigma \vdash n \Rightarrow n} \quad \text{Add} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \quad \text{Mul} \frac{\sigma \vdash e_1 \Rightarrow n_1 \quad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 * e_2 \Rightarrow n_1 \times n_2} \\
 \\
 \text{Val} \frac{\sigma \vdash e_1 \Rightarrow v_1 \quad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \text{val } x = e_1; e_2 \Rightarrow v_2} \quad \text{Val}_\tau \frac{\sigma \vdash e_1 \Rightarrow v_1 \quad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \text{val } x : \tau_0 = e_1; e_2 \Rightarrow v_2} \\
 \\
 \text{Fun} \frac{}{\sigma \vdash \lambda x : \tau. e \Rightarrow \langle \lambda x. e, \sigma \rangle} \\
 \\
 \text{Id} \frac{x \in \text{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)} \quad \text{App} \frac{\sigma \vdash e_0 \Rightarrow \langle \lambda x. e_2, \sigma' \rangle \quad \sigma \vdash e_1 \Rightarrow v_1 \quad \sigma'[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash e_0(e_1) \Rightarrow v_2} \\
 \\
 \text{Record} \frac{\sigma \vdash e_1 \Rightarrow v_1 \quad \dots \quad \sigma \vdash e_n \Rightarrow v_n}{\sigma \vdash \{x_1 = e_1, \dots, x_n = e_n\} \Rightarrow \{x_1 = v_1, \dots, x_n = v_n\}} \\
 \\
 \text{Access} \frac{\sigma \vdash e \Rightarrow \{x_1 = v_1, \dots, x_n = v_n\} \quad 1 \leq i \leq n}{\sigma \vdash e.x_i \Rightarrow v_i}
 \end{array}$$

Note that there is no rule for `exit` because it cannot produce any value.