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**
- **basic arithmetic operators**: addition (`+`) and multiplication (`*`)
- **first-class functions**
- **immutable variables** (`val`)
- **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"
```

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
+	left	3

3 ABSTRACT SYNTAX

The abstract syntax of STFAE is defined as follows:

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

4 TYPE SYSTEM

This section explains type system of STFAE, 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:

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

$$\tau\text{–Num}\ \frac{}{\Gamma \vdash n : \texttt{num}} \qquad \tau\text{–Add}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \tau_1 <: \texttt{num} \qquad \Gamma \vdash e_2 : \tau_2 \qquad \tau_2 <: \texttt{num}}{\Gamma \vdash e_1 + e_2 : \texttt{num}}$$

$$\tau\text{–Mul}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \tau_1 <: \texttt{num} \qquad \Gamma \vdash e_2 : \tau_2 \qquad \tau_2 <: \texttt{num}}{\Gamma \vdash e_1 \times e_2 : \texttt{num}} \qquad \tau\text{–Id}\ \frac{x \in \text{Domain}(\Gamma)}{\Gamma \vdash x : \Gamma(x)}$$

$$\tau\text{–Val}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma[x : \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{val}\ x = e_1;\ e_2 : \tau_2} \qquad \tau\text{–Val}_\tau\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \tau_1 <: \tau_0 \qquad \Gamma[x : \tau_0] \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{val}\ x : \tau_0 = e_1;\ e_2 : \tau_2}$$

$$\tau\text{–Fun}\ \frac{\Gamma \vdash \tau \qquad \Gamma[x : \tau] \vdash e : \tau'}{\Gamma \vdash \lambda x{:}\tau.e : \tau \to \tau'} \qquad \tau\text{–App}\ \frac{\Gamma \vdash e_0 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_1 : \tau_3 \qquad \tau_3 <: \tau_1}{\Gamma \vdash e_0(e_1) : \tau_2}$$

$$\tau\text{–Record}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \ldots \qquad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash \{x_1 = e_1, \ldots, x_n = e_n\} : \{x_1 : \tau_1, \ldots, x_n : \tau_n\}}$$

$$\tau\text{–Access}\ \frac{\Gamma \vdash e : \{x_1 : \tau_1, \ldots, x_n : \tau_n\} \qquad 1 \le i \le n}{\Gamma \vdash e.x_i : \tau_i} \qquad \tau\text{–Exit}\ \frac{}{\Gamma \vdash \texttt{exit} : \bot}$$

the following rules for subtyping:

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

$$\frac{}{\bot <: \tau} \qquad \frac{}{\tau <: \top} \qquad \frac{}{\mathsf{num} <: \mathsf{num}} \qquad \frac{\tau_1' <: \tau_1 \qquad \tau_2 <: \tau_2'}{(\tau_1 \to \tau_2) <: (\tau_1' \to \tau_2')}$$

$$\frac{}{\{x_1 : \tau_1, \ldots, x_n : \tau_n, x : \tau\} <: \{x_1 : \tau_1, \ldots, x_n : \tau_n\}} \qquad \frac{\tau_1 <: \tau_1' \qquad \ldots \qquad \tau_n <: \tau_n'}{\{x_1 : \tau_1, \ldots, x_n : \tau_n\} <: \{x_1 : \tau_1', \ldots, x_n : \tau_n'\}}$$

$$\frac{\{x_1 : \tau_1, \ldots, x_n : \tau_n\} \text{ is a permutation of } \{x_1' : \tau_1', \ldots, x_n' : \tau_n'\}}{\{x_1 : \tau_1, \ldots, x_n : \tau_n\} <: \{x_1' : \tau_1', \ldots, x_n' : \tau_n'\}}$$

5 SEMANTICS

We use the following notations in the semantics:

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

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

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

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

$$\mathtt{Val}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \mathtt{val}\ x = e_1;\ e_2 \Rightarrow v_2} \qquad \mathtt{Val}_\tau\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \mathtt{val}\ x : \tau_0 = e_1;\ e_2 \Rightarrow v_2}$$

$$\mathtt{Fun}\ \frac{}{\sigma \vdash \lambda x{:}\tau.e \Rightarrow \langle \lambda x.e, \sigma \rangle}$$

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

$$\mathtt{Record}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \ldots \qquad \sigma \vdash e_n \Rightarrow v_n}{\sigma \vdash \{x_1 = e_1, \ldots, x_n = e_n\} \Rightarrow \{x_1 = v_1, \ldots, x_n = v_n\}}$$

$$\mathtt{Access}\ \frac{\sigma \vdash e \Rightarrow \{x_1 = v_1, \ldots, x_n = v_n\} \qquad 1 \le i \le n}{\sigma \vdash e.x_i \Rightarrow v_i}$$

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