Final Exam

COSE212: Programming Languages

2023 Fall

Instructor: Jihyeok Park

December 20, 2023. 13:30-14:45

- **If you are not good at English, please write your answers in Korean.**
  (영어가 익숙하지 않은 경우, 답안을 한글로 작성해 주세요.)
- **Write answers in good handwriting.**
  If we cannot recognize your answers, you will not get any points.
  (글씨를 알아보기 힘들면 점수를 드릴 수 없습니다. 답안을 읽기 좋게 작성해주세요.)
- **Write your answers in the boxes provided.**
  (답안을 제공된 박스 안에 작성해 주세요.)
- **There are 6 pages and 8 questions.**
  (시험은 6장으로 구성되어 있으며, 총 8개의 문제가 있습니다.)
- **Syntax and semantics of languages are given in Appendix.**
  (언어의 문법과 의미는 부록에서 참조할 수 있습니다.)

Student ID	
Student Name	

Question:	1	2	3	4	5	6	7	8	Total
Points:	10	20	10	10	15	10	10	15	100
Score:									

1. 10 points The following sentences explain the fundamental concepts of programming languages. Fill in the blanks with the following terms (**2pt per blank**):

ad-hoc	continuation	eager	memory	subtype
algebraic data type	continuation-passing style	lazy	parametric	type inference
complete	dynamic analysis	let	static analysis	type sound

- A(n) [ ] represents the rest of the computation, and it is used to describe the control flow of a program. [ ] is a programming style passing it as an explicit parameter to a function.
- A type system is [ ] if it guarantees that a well-typed program will never cause a type error at run-time.
- Polymorphism uses a single entity as multiple types, and there are various kinds of polymorphism. Among them,
  - [ ] polymorphism defines a subtype relation between types to support polymorphism.
  - [ ] polymorphism introduces type variables and ways to instantiate them with type arguments to support polymorphism.

2. Consider a language KFAE defined with the following syntax and small-step operational (reduction) semantics. It supports **first-class functions** and **first-class continuations**.

$$\text{Expressions} \quad \mathbb{E} \ni e ::= n \mid e \texttt{ + } e \mid e \texttt{ * } e \mid x \mid \lambda x.e \mid e(e) \mid \texttt{vcc } x;\ e$$

$$\boxed{\langle \kappa \mid\mid s \rangle \to \langle \kappa \mid\mid s \rangle}$$

$$\begin{array}{lcl}
\langle (\sigma \vdash n) :: \kappa \mid\mid s \rangle & \to & \langle \kappa \mid\mid n :: s \rangle \\
\langle (\sigma \vdash e_1 \texttt{ + } e_2) :: \kappa \mid\mid s \rangle & \to & \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (\texttt{+}) :: \kappa \mid\mid s \rangle \\
\langle (\texttt{+}) :: \kappa \mid\mid n_2 :: n_1 :: s \rangle & \to & \langle \kappa \mid\mid (n_1 \texttt{ + } n_2) :: s \rangle \\
\langle (\sigma \vdash e_1 \texttt{ * } e_2) :: \kappa \mid\mid s \rangle & \to & \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (\texttt{*}) :: \kappa \mid\mid s \rangle \\
\langle (\texttt{*}) :: \kappa \mid\mid n_2 :: n_1 :: s \rangle & \to & \langle \kappa \mid\mid (n_1 \texttt{ * } n_2) :: s \rangle \\
\langle (\sigma \vdash x) :: \kappa \mid\mid s \rangle & \to & \langle \kappa \mid\mid \sigma(x) :: s \rangle \\
\langle (\sigma \vdash \lambda x.e) :: \kappa \mid\mid s \rangle & \to & \langle \kappa \mid\mid \langle \lambda x.e, \sigma \rangle :: s \rangle \\
\langle (\sigma \vdash e_1(e_2)) :: \kappa \mid\mid s \rangle & \to & \langle (\sigma \vdash e_1) :: (\sigma \vdash e_2) :: (@) :: \kappa \mid\mid s \rangle \\
\langle (@) :: \kappa \mid\mid v_2 :: \langle \lambda x.e, \sigma \rangle :: s \rangle & \to & \langle (\sigma[x \mapsto v_2] \vdash e) :: \kappa \mid\mid s \rangle \\
\langle (@) :: \kappa \mid\mid v_2 :: \langle \kappa' \mid\mid s' \rangle :: s \rangle & \to & \langle \kappa' \mid\mid v_2 :: s' \rangle \\
\langle (\sigma \vdash \texttt{vcc } x;\ e) :: \kappa \mid\mid s \rangle & \to & \langle (\sigma[x \mapsto \langle \kappa \mid\mid s \rangle] \vdash e) :: \kappa \mid\mid s \rangle
\end{array}$$

$$\begin{array}{lll}
\text{Values} & \mathbb{V} \ni v ::= n & \text{Continuations} \quad \mathbb{K} \ni \kappa ::= \square \\
& \mid \langle \lambda x.e, \sigma \rangle & \mid (\sigma \vdash e) :: \kappa \\
& \mid \langle \kappa \mid\mid s \rangle & \mid (+) :: \kappa \\
\text{Environments} & \sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{V} & \mid (\times) :: \kappa \\
\text{Value Stacks} & \mathbb{S} \ni s ::= \blacksquare \mid v :: s & \mid (@) :: \kappa
\end{array}$$

The desugaring function $\mathcal{D}[\![-]\!]$ is defined as follows, and recursive cases omitted.

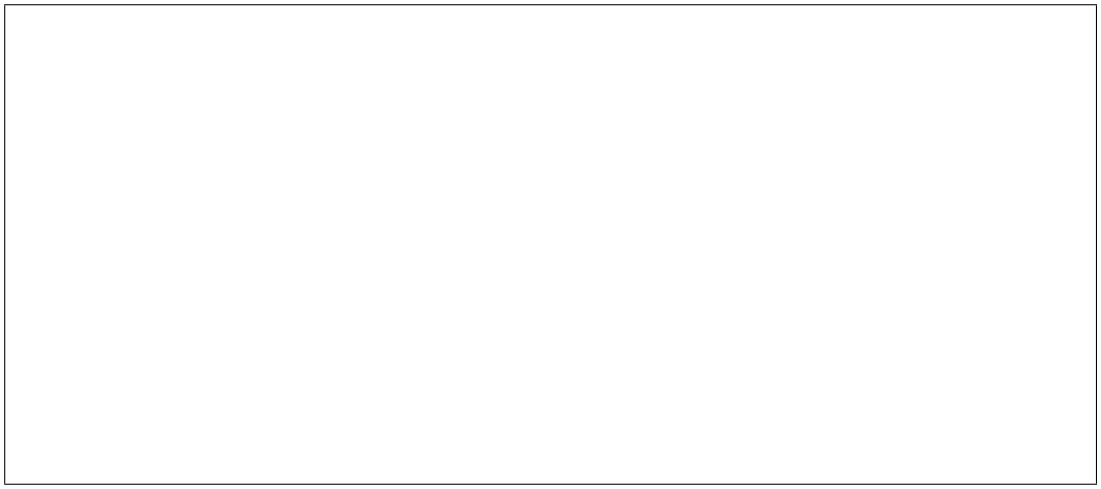
$$\mathcal{D}[\![\texttt{val } x \texttt{ = } e;\ e']\!] = (\lambda x.\mathcal{D}[\![e']\!])(\mathcal{D}[\![e]\!])$$

(a) [15 points] Consider the following KFAE expression:

$$2 * \{\ \texttt{vcc x};\ 3 + \texttt{x}(5)\ \}$$

Complete the following **reduction steps** of the expression.

$$\begin{aligned}
&\langle && (\varnothing \vdash 2 * \{\ \texttt{vcc x};\ 3 + \texttt{x}(5)\ \}) :: \square \ ||\ && \blacksquare \rangle \\
\rightarrow\ &\langle && (\varnothing \vdash 2) :: (\varnothing \vdash \texttt{vcc x};\ 3 + \texttt{x}(5)) :: (*) :: \square \ ||\ && \blacksquare \rangle \\
\rightarrow\ &\langle && (\varnothing \vdash \texttt{vcc x};\ 3 + \texttt{x}(5)) :: (*) :: \square \ ||\ && 2 :: \blacksquare \rangle \\
\rightarrow\ &\langle && (\sigma_0 \vdash 3 + \texttt{x}(5)) :: (*) :: \square \ ||\ && 2 :: \blacksquare \rangle \\
\rightarrow\ &\langle && (\sigma_0 \vdash 3) :: (\sigma_0 \vdash \texttt{x}(5)) :: (+) :: (*) :: \square \ ||\ && 2 :: \blacksquare \rangle \\
\rightarrow\ &\langle && (\sigma_0 \vdash \texttt{x}(5)) :: (+) :: (*) :: \square \ ||\ && 3 :: 2 :: \blacksquare \rangle
\end{aligned}$$



where $\sigma_0 = [\texttt{x} \mapsto \langle (*) :: \square \ ||\ 2 :: \blacksquare \rangle]$

(b) [5 points] Write the result of evaluating the following KFAE expression:

```
2 * {
  vcc a;
  val f = { vcc b; 3 * b(a) };
  val x = 5 * f(7);
  x * 11
}
```

Result:

3. [10 points] **True/False questions.** Answer O for True and X for False.
(Each question is worth **2 points**, but you will get **-2 points** for each wrong answer.)

1. In a continuation-passing style, every function call is in a tail position. ☐
2. The type system is sound if all normally terminating programs are well-typed. ☐
3. Typically, a type system defined with a subsumption rule is algorithmic. ☐
4. Compilers for functional languages often utilize the continuations. ☐
5. A general algebraic data type is a possibly recursive sum type of product types. ☐

4. Assume that we revised one of **typing rules** in TFAE from the left to the right:

$$\frac{\Gamma[x_1 : \tau_1, \ldots, x_n : \tau_n] \vdash e : \tau}{\Gamma \vdash \lambda(x_1 : \tau_1, \ldots, x_n : \tau_n).e : (\tau_1, \ldots, \tau_n) \to \tau} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \lambda(x_1 : \tau_1, \ldots, x_n : \tau_n).e : (\tau_1, \ldots, \tau_n) \to \tau}$$

In the revised typing rule, parameter types $x_1 : \tau_1, \ldots, x_n : \tau_n$ are no longer passed when checking the type of the function body e.

(a) [3 points] Give an example of a TFAE expression that is **well-typed** under the **original (left) rule** but **ill-typed** under the **revised (right) rule**.

(b) [7 points] If the type system with the above **revised (right) rule** is **type sound**, explain why it is. Otherwise, give an example of a TFAE expression that passes the type checking but causes a run-time type error.

5. [15 points] Fill in the blanks in the **type derivation** (proof tree) according to the **typing rules** of TRFAE. If it is impossible to derive a tree, write **"impossible"** in the blank.

$$\varnothing \vdash \texttt{def f(x:num):num = f(x) * 7; f} : \boxed{\phantom{\texttt{num} \to \texttt{num}}}$$

where
$$\begin{cases} \Gamma_0 = [\texttt{f} : \texttt{num} \to \texttt{num}] & \Gamma_3 = [\texttt{f} : \texttt{num} \to \texttt{bool}] \\ \Gamma_1 = [\texttt{x} : \texttt{num}] & \Gamma_4 = [\texttt{x} : \texttt{bool}] \\ \Gamma_2 = [\texttt{f} : \texttt{num} \to \texttt{num}, \texttt{x} : \texttt{num}] & \Gamma_5 = [\texttt{f} : \texttt{num} \to \texttt{bool}, \texttt{x} : \texttt{num}] \end{cases}$$

6. 10 points The following Scala code is an excerpt from the **type checker** for ATFAE. Fill the blank according to the **typing rules** of ATFAE (**5pt per blank**).

```
def mustValid(ty: Type, tenv: TypeEnv): Type = ...
def typeCheck(expr: Expr, tenv: TypeEnv): Type = expr match
  ...
  case TypeDef(tname, ws, body) =>
    if (tenv.tys.contains(tname)) error(s"already defined type: $tname")
    val tenv1 = tenv.addType(tname, ws.map(w => w.name -> w.ptys).toMap)
    val tenv2 =
      tenv1.addVars(ws.map(w => w.name -> ArrowT(w.ptys, NameT(tname))))
    for (w <- ws; pty <- w.ptys) { (A) }
    (B)
```

Note that the `mustValid` function is implements the **well-formedness of types** according to the rules defined in the form $\boxed{\Gamma \vdash \tau}$.

(A) =

(B) =

7. 10 points Fill in the blanks with **types** to make the following PATFAE expression pass the type-checking process according to the typing rules of PATFAE.
(**2pt for (A), (B), and (C)** and **4pt for (D)**).
(Note that this language has no definition of **type equivalence**.)

```
val f = ∀α. {
  enum t { case a(); case b(α, t) };
  λ(g:(A)). λ(x0:α, x1:α, x2:α, x3:α). {
    def h(y:(B)):(C) = y match {
      case a() => false
      case b(c, d) => if (g(c)) true else h(d)
    };
    h(b(x0, b(x1, b(x2, b(x3, a())))))
  }
};
val x0 = f[num](λ(n:num).{ n < 5 })(5, 6, 4, 7);
val x1 = f[num](λ(n:num).{ n < 3 })(5, 6, 4, 7);
val x2 = f[bool](λ(b:bool).{ b })(true, false, true, false);
val x3 = f[bool](λ(b:bool).{ false })(true, false, true, false);
(λ(g:(D)).{ g })(f)
```

(A) = (B) = (C) =

(D) =

8. The following Scala code is a **type checker** for SETFAE, an extension of TFAE with the **subtype polymorphism** and the **even number type** (EvenT) for the set of even numbers.

```
enum Expr:
  case Num(number: BigInt)
  case Add(left: Expr, right: Expr)
  case Mul(left: Expr, right: Expr)
  case Val(name: String, init: Expr, body: Expr)
  case Id(name: String)
  case Fun(params: List[String], tys: List[Type], body: Expr)
  case App(fun: Expr, args: List[Expr])
enum Type:
  case NumT
  case EvenT /* newly added even number types */
  case ArrowT(paramTys: List[Type], retTy: Type)
type TypeEnv = Map[String, Type]
import Expr.*, Type.*;
def err(): Nothing = ...

def mustSub(l: Type, r: Type): Unit = if (!isSub(l, r)) err()
def isSub(l: Type, r: Type): Boolean = (l, r) match
  case (EvenT, EvenT) | (EvenT, NumT) | (NumT, NumT) => true
  case (ArrowT(lps, lr), ArrowT(rps, rr)) => [    (A)    ]
  case _ => false

def typeCheck(expr: Expr, tenv: TypeEnv = Map.empty): Type = expr match
  case Num(n)       => if (n % 2 == 0) EvenT else NumT
  case Add(l, r)    => ...
  case Mul(l, r)    => [    (B)    ]
  case Val(x, e, b) => typeCheck(b, tenv + (x -> typeCheck(e, tenv)))
  case Id(x) => tenv.getOrElse(x, err())
  case Fun(ps, ts, b) => ArrowT(ts, typeCheck(b, tenv ++ (ps zip ts)))
  case App(f, as) => typeCheck(f, tenv) match
    case ArrowT(pts, rt) if pts.length == as.length =>
      for ((a, p) <- (as.map(typeCheck(_, tenv)) zip pts)) mustSub(a, p)
      rt
    case _ => err()
```

The followings are given **tests** for the type checker implementation:

```
def parse(str: String): Expr = ...
def tytest(s: String, t: Type): Unit = if (typeCheck(parse(s)) != t) err()

tytest("1 + 1", NumT);  tytest("1 + 2", NumT); tytest("2 + 1", NumT);
tytest("2 + 2", EvenT); tytest("1 * 1", NumT); tytest("1 * 2", EvenT);
tytest("2 * 1", EvenT); tytest("2 * 2", EvenT);
tytest(check("""
  val g = (x: Number, y: Number) => x * y * 2
  val h = (f: (Even, Even) => Number) => f(4, 6)
  h(g)
"""), NumT)
```

(a) [5 points] Fill in the blank `(A)` in the Scala code to **pass all the given tests**.

`(A)` =

(b) [5 points] Fill in the blank `(B)` in the Scala code to **pass all the given tests**.

`(B)` =

(c) [5 points] Write the **algorithmic type rules** representing the type-checking algorithm for function applications (`App`) according to the given Scala code.
(You can use the notation $\tau <: \tau'$ to represent that τ is a **subtype** of τ'.)

$$\Gamma \vdash e_0(e_1, \ldots, e_n) : \square$$

This is the last page.
I hope that your tests went well!

Appendix

TFAE – Typed Functions and Arithmetic Expressions

Expressions $\mathbb{E} \ni e ::= n \mid e \texttt{ + } e \mid e \texttt{ * } e \mid \texttt{val } x \texttt{ = } e\texttt{; } e \mid x \mid \lambda([x\texttt{:}\tau]^*).e \mid e(e^*)$
Types $\mathbb{T} \ni \tau ::= \texttt{num} \mid (\tau^*) \to \tau$ Numbers $n \in \mathbb{Z}$ Identifiers $x \in \mathbb{X}$

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

$$\frac{}{\Gamma \vdash n : \texttt{num}} \qquad \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 \texttt{ + } e_2 : \texttt{num}} \qquad \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 \texttt{ * } e_2 : \texttt{num}}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma[x : \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{val } x \texttt{ = } e_1\texttt{; } e_2 : \tau_2} \qquad \frac{\Gamma[x_1 : \tau_1, \ldots, x_n : \tau_n] \vdash e : \tau}{\Gamma \vdash \lambda(x_1\texttt{:}\tau_1, \ldots, x_n\texttt{:}\tau_n).e : (\tau_1, \ldots, \tau_n) \to \tau}$$

$$\frac{x \in \text{Domain}(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma \vdash e_0 : (\tau_1, \ldots, \tau_n) \to \tau \qquad \Gamma \vdash e_1 : \tau_1 \qquad \ldots \qquad \Gamma \vdash e_n : \tau_n}{\Gamma \vdash e_0(e_1, \ldots, e_n) : \tau}$$

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

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

$$\frac{}{\sigma \vdash n \Rightarrow n} \qquad \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 \texttt{ + } e_2 \Rightarrow n_1 \texttt{ + } n_2} \qquad \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 \texttt{ * } e_2 \Rightarrow n_1 \texttt{ * } n_2}$$

$$\frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \texttt{val } x \texttt{ = } e_1\texttt{; } e_2 \Rightarrow v_2} \qquad \frac{}{\sigma \vdash \lambda(x_1\texttt{:}\tau_1, \ldots, x_n\texttt{:}\tau_n).e \Rightarrow \langle \lambda(x_1, \ldots, x_n).e, \sigma \rangle}$$

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

Values $\mathbb{V} \ni v ::= n \mid \langle \lambda(x_1, \ldots, x_n).e, \sigma \rangle$ Environments $\sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{V}$

TRFAE – TFAE with Recursion and Conditionals

Expressions $\mathbb{E} \ni e ::= \ldots \mid b \mid e \texttt{ < } e \mid \texttt{if } (e)\ e \texttt{ else } e \mid \texttt{def } x([x\texttt{:}\tau]^*)\texttt{:}\tau \texttt{ = } e\texttt{; } e$
Types $\mathbb{T} \ni \tau ::= \ldots \mid \texttt{bool}$ Booleans $b \in \mathbb{B}$

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

$$\ldots \quad \frac{}{\Gamma \vdash b : \texttt{bool}} \qquad \frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 \texttt{ < } e_2 : \texttt{bool}} \qquad \frac{\Gamma \vdash e_0 : \texttt{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{if } (e_0)\ e_1 \texttt{ else } e_2 : \tau}$$

$$\frac{\Gamma[x_0 : (\tau_1, \ldots, \tau_n) \to \tau, x_1 : \tau_1, \ldots, x_n : \tau_n] \vdash e : \tau \qquad \Gamma[x_0 : (\tau_1, \ldots, \tau_n) \to \tau] \vdash e' : \tau'}{\Gamma \vdash \texttt{def } x_0(x_1\texttt{:}\tau_1, \ldots, x_n\texttt{:}\tau_n)\texttt{:}\tau \texttt{ = } e\texttt{; } e' : \tau'}$$

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

$$\ldots \quad \frac{}{\sigma \vdash b \Rightarrow b} \qquad \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 \texttt{ < } e_2 \Rightarrow n_1 \texttt{ < } n_2} \qquad \frac{\sigma \vdash e_0 \Rightarrow \texttt{true} \qquad \sigma \vdash e_1 \Rightarrow v_1}{\sigma \vdash \texttt{if } (e_0)\ e_1 \texttt{ else } e_2 \Rightarrow v_1}$$

$$\frac{\sigma \vdash e_0 \Rightarrow \texttt{false} \qquad \sigma \vdash e_2 \Rightarrow v_2}{\sigma \vdash \texttt{if } (e_0)\ e_1 \texttt{ else } e_2 \Rightarrow v_2} \qquad \frac{\sigma' = \sigma[x_0 \mapsto \langle \lambda(x_1, \ldots, x_n).e, \sigma' \rangle] \qquad \sigma' \vdash e' \Rightarrow v'}{\sigma \vdash \texttt{def } x_0(x_1\texttt{:}\tau_1, \ldots, x_n\texttt{:}\tau_n)\texttt{:}\tau \texttt{ = } e\texttt{; } e' \Rightarrow v'}$$

Values $\mathbb{V} \ni v ::= \ldots \mid b$

ATFAE – TRFAE with Algebraic Data Types

Expressions $\mathbb{E} \ni e ::= \ldots \mid \texttt{enum}\ t\ \{\ [\texttt{case}\ x(\tau^*)]^*\ \};\ e \mid e\ \texttt{match}\ \{\ [\texttt{case}\ x(x^*)\ \texttt{=>}\ e]^*\ \}$
Types $\mathbb{T} \ni \tau ::= \ldots \mid t$ Type Names $t \in \mathbb{X}_t$

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

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

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

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

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

$$\frac{}{\Gamma \vdash \texttt{num}} \qquad \frac{}{\Gamma \vdash \texttt{bool}} \qquad \frac{\Gamma \vdash \tau_1 \quad \ldots \quad \Gamma \vdash \tau_n \qquad \Gamma \vdash \tau}{\Gamma \vdash (\tau_1, \ldots, \tau_n) \to \tau} \qquad \frac{\Gamma(t) = x_1(\ldots) + \ldots + x_n(\ldots)}{\Gamma \vdash t}$$

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

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

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

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

Values $\mathbb{V} \ni v ::= \ldots \mid \langle x \rangle \mid x(v^*)$

PATFAE – ATFAE with Parametric Polymorphism

Expressions $\mathbb{E} \ni e ::= \ldots \mid \forall\alpha.e \mid e[\tau]$
Types $\mathbb{T} \ni \tau ::= \ldots \mid \forall\alpha.\tau \mid \alpha$ Type Variables $\alpha \in \mathbb{X}_\alpha$

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

$$\ldots \quad \frac{\alpha \notin \text{Domain}(\Gamma) \qquad \Gamma[\alpha] \vdash e : \tau}{\Gamma \vdash \forall\alpha.e : \forall\alpha.\tau} \qquad \frac{\Gamma \vdash \tau \qquad \Gamma \vdash e : \forall\alpha.\tau'}{\Gamma \vdash e[\tau] : \tau'[\alpha \leftarrow \tau]}$$

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

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

$$\ldots \quad \frac{\Gamma[\alpha] \vdash \tau}{\Gamma \vdash \forall\alpha.\tau} \qquad \frac{\alpha \in \text{Domain}(\Gamma)}{\Gamma \vdash \alpha}$$

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

$$\ldots \quad \frac{}{\sigma \vdash \forall\alpha.e \Rightarrow \langle \forall\alpha.e, \sigma \rangle} \qquad \frac{\sigma \vdash e \Rightarrow \langle \forall\alpha.e', \sigma' \rangle \qquad \sigma' \vdash e' \Rightarrow v'}{\sigma \vdash e[\tau] : v'}$$

Values $\mathbb{V} \ni v ::= \ldots \mid \langle \forall\alpha.e, \sigma \rangle$