Final Exam

COSE212: Programming Languages
2024 Fall

Instructor: Jihyeok Park

December 18, 2024. 18:30-21:00

- **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 10 pages and 10 questions.**
  (시험은 10 장으로 총 10 문제로 구성되어 있습니다.)
- **Syntax, semantics, and typing rules of languages are given in Appendix.**
  (언어의 문법, 의미, 타입 규칙은 부록에서 참조할 수 있습니다.)

Student ID	
Student Name	

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	10	15	5	10	10	15	10	10	10	5	100
Score:											

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

ad-hoc	continuation	intersection	recursive	subtype
algebraic	dynamic	let	sound	type inference
complete	first-class	parametric	static	union

- A type system is said to be [ ] if it guarantees that a well-typed program will never cause a type error at run-time.

- The [ ] algorithm automatically infers the types of expressions in a program without explicit type annotations.

- In a type system, polymorphism helps to use a single entity to represent different types. For example, [ ] polymorphism allows a value of a subtype to be used in place of a value of a supertype. On the other hand, [ ] polymorphism introduces type parameters that can be instantiated with given type arguments.

- A(n) [ ] is a representation of the remaining computation to be performed after a given computation and used to represent control flows, such as exceptions, generators, and coroutines.

2. [15 points] Consider a language KFAE defined with the following syntax and small-step operational semantics. It supports **first-class functions** and **first-class continuations**.

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

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

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

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

$$\mathcal{D}[\![\texttt{val}\ x\ \texttt{=}\ e_1;\ e_2]\!] = (\lambda x.\mathcal{D}[\![e_2]\!])(\mathcal{D}[\![e_1]\!])$$

(a) 10 points Consider the following KFAE expression.

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

What is the **evaluation result** of the given expression? ______.

The following reduction steps show the evaluation process of the given expression. **Complete the remaining reduction steps** by filling out the following boxes until the final evaluation result.

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

where $\sigma_0 =$ ______.

(b) 5 points Write the evaluation result of the following KFAE expression:

```
val f = { vcc x; x };
val g = {
   vcc y;
   val z = f(y) * 3;
   val x = z * 11;
   y(x * 2) + 1;
};
g(λx.x)(5) * 7
```

Result: ______

3. [5 points] Assume that we revised one of **typing rules** in TFAE from the left to the right:

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

Is the revised type system still **type sound**? If it is, explain why. If not, give a TFAE expression as a **counterexample** that passes the type-checking process but causes a run-time type error.

4. [10 points] Fill in the blanks in the **type derivation** (proof tree) according to the **typing rules** of TRFAE.

$$\frac{\boxed{\texttt{(A)}} \qquad \boxed{\texttt{(B)}}}{\varnothing \vdash \texttt{def f(x:bool):num = if(x) 42 else f(x); f(1 < 2) * 5} : \boxed{}}$$

(A) =

(B) =

You can use $\Gamma_0 = [\texttt{f} : \texttt{bool} \rightarrow \texttt{num}]$ and $\Gamma_1 = \Gamma_0[\texttt{x} : \texttt{bool}]$ in the type derivation.

5. [10 points] Write down the **evaluation results** and **types** of the following ATFAE expressions according to the given semantics and typing rules of the language.

- You should **write evaluation results** of expressions even if they are not well-typed.
- If the evaluation results in a run-time error, write error.
- If the evaluation result is a function value, write closure.
- If the given expression is not well-typed, write no type.
- You can get a score **only if both** evaluation results and types are correct.

(a) [2 points]
Result: ______
Type: ______

```
enum A {
  case B(num);
  case C(num → num);
};
B(5) match {
  case C(f) => f;
  case B(n) => λ(x : num).{ x + n };
}
```

(b) [2 points]
Result: ______
Type: ______

```
enum X { case X(num); };
def f(x : X) : num = x match {
  case X(n) => n;
  case X(m) => m + 1;
};
f(X(3))
```

(c) [3 points]
Result: ______
Type: ______

```
enum Color { case Orange(num); };
enum Fruit { case Orange(num); };
def f(color : Color) : num = color match { case Orange(y) => y * 2; };
f(Orange(7))
```

(d) [3 points]
Result: ______
Type: ______

```
val x = {
  enum A { case B(num); };
  val f = λ(a : A).{
    a match { case B(n) => n; }
  };
  f(B(2))
};
enum A { case B(num → num); };
B(λ(m : num).{ m * 3 }) match {
  case B(f) => f(5) + x;
}
```

6. [15 points] STFAE supports **subtype polymorphism** with the **subtype relation** (<:) between types.

(a) [7 points] Fill in the blanks with <:, :>, or X according to the **subtyping rules** of STFAE. Note that X means that they do not have a subtype relation. (**1 point per blank**)

$\{\ \texttt{a} : \texttt{num},\ \texttt{b} : \texttt{num}\ \}$	☐	$\{\ \texttt{a} : \texttt{num}\ \}$	$\texttt{num} \to \texttt{num}$	☐	$\top \to \texttt{num}$
$\{\ \texttt{a} : \top,\ \texttt{b} : \texttt{num}\ \}$	☐	$\{\ \texttt{b} : \texttt{num},\ \texttt{a} : \texttt{num}\ \}$	$\texttt{num} \to (\texttt{num} \to \texttt{num})$	☐	$\bot \to (\top \to \top)$
$\{\ \texttt{a} : \top,\ \texttt{b} : \texttt{num}\ \}$	☐	$\{\ \texttt{a} : \texttt{num}\ \}$	$(\bot \to \top) \to \bot$	☐	$(\texttt{num} \to \texttt{num}) \to \texttt{num}$

$\{\ \texttt{a} : (\texttt{num} \to \top) \to \texttt{num}\ \} \to \{\ \texttt{c} : \top\ \}$ ☐ $\{\ \texttt{a} : (\top \to \texttt{num}) \to \top\ \} \to \{\ \texttt{b} : \texttt{num},\ \texttt{c} : \texttt{num}\ \}$

(b) [5 points] Using the subtype relation (<:) in STFAE, we can define a **join** ($\vee$) operation between two types satisfying the following properties for any types τ and τ':

- $\tau <: (\tau \vee \tau')$
- $\tau' <: (\tau \vee \tau')$
- $\forall \tau'' \in \mathbb{T}.\ ((\tau <: \tau'') \wedge (\tau' <: \tau'')) \Rightarrow ((\tau \vee \tau') <: \tau'')$

In other words, $\tau \vee \tau'$ is the **least upper bound** of τ and τ' in the subtype relation. Fill in the blanks with the result of the join operation satisfying the above properties. If there is no possible result, write X to indicate that the join operation is undefined. (**1 point per blank**)

$\texttt{num}$	$\vee\ \texttt{bool}$	=	☐
$(\{\ \texttt{a} : \texttt{num},\ \texttt{b} : \texttt{num}\ \})$	$\vee\ (\{\ \texttt{a} : \texttt{bool}\ \})$	=	☐
$(\texttt{num} \to \texttt{num})$	$\vee\ (\texttt{bool} \to \texttt{bool})$	=	☐
$((\texttt{num} \to \top) \to \texttt{bool})$	$\vee\ ((\bot \to \texttt{bool}) \to \texttt{num})$	=	☐
$((\{\ \texttt{a} : \top,\ \texttt{b} : \texttt{bool}\ \}) \to \texttt{num})$	$\vee\ ((\{\ \texttt{a} : \texttt{num}\ \}) \to \texttt{bool})$	=	☐

(c) [3 points] The following **subsumption rule** in STFAE is a key typing rule for supporting **subtype polymorphism**. It allows a value of a subtype to be used in place of a value of a supertype.

$$\frac{\Gamma \vdash e : \tau \qquad \tau <: \tau'}{\Gamma \vdash e : \tau'}$$

However, it makes type system **non-algorithmic** because it does not syntax-directed. In this question, you need to **revise** the **typing rule** of **conditional expressions** (`if-else`) to make the following expression well-typed **without** the subsumption rule. (Hint: You can use the join ($\vee$) operation.)

```
val f = λ(x : bool). {
  if(x) { a = 1, b = true }
  else { a = 2; }
};
f(true).a * f(false).a
```

☐

7. [10 points] ATFAE supports algebraic data types, **recursive** sum types of product types. The following typing rule defines the type-checking of algebraic data types:

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

(a) [4 points] **Revise** the above typing rule to **forbid recursive** data type definitions, and **explain why** the revised rule can prevent recursive data type definitions.

For example, the following expression should be ill-typed with the revised rule:

```
enum List { case Nil(); case Cons(num,List); }; 42
```

(b) [6 points] FAE is an **untyped** version of TFAE without type annotations. The following untyped FAE expression defines the `mkRec` function, which constructs a recursive function using a fixed point combinator.

```
val mkRec = λf.{
  val g = λx.{
    val h = λv.x(x)(v); f(h)
  }; g(g)
};
val sum = mkRec(λsum.λn.{ if(n < 1) 0 else n + sum(n + -1) }); sum(10)
```

Using **recursive** data types, you can define its typed version. Fill in the blanks in the following ATFAE expression to make it well-typed and produce the same result as the above FAE expression:

```
enum T { ______ };
val mkRec = λ(f : ______).{
  val g = λ(x : ______).{
    val h = λ(v : ______).______;
    f(h)
  };
  g(______)
};
val sum = mkRec(λ(sum : num → num).λ(n : num).{ if(n < 1) 0 else n + sum(n + -1) }); sum(10)
```

8. 10 points The following language is a **typed language** defined with **lists** (i.e., `nil` and ::) and **list operations** (i.e., `head` and `tail`) and supports **type inference** without type annotations. Note that the notation $\langle\langle\tau\rangle\rangle$ represents the type of lists with elements of type τ.

$$
\begin{array}{llll}
\text{Expressions} & \mathbb{E} \ni e & ::= & n \mid e \texttt{ + } e \mid e \texttt{ * } e \mid \texttt{val } x \texttt{ = } e;\ e \mid x \mid \lambda x.e \mid e(e) \\
& & & \mid \texttt{nil} \mid e :: e \mid e.\texttt{head} \mid e.\texttt{tail} \\
\text{Types} & \mathbb{T} \ni \tau & ::= & \texttt{num} \mid \tau \to \tau \mid \alpha \mid \langle\langle\tau\rangle\rangle \\
\text{Type Environments} & \Gamma & \in & \mathbb{X} \xrightarrow{\text{fin}} \mathbb{T} \\
\text{Solutions} & \psi & \in & \Psi = \mathbb{X}_\alpha \xrightarrow{\text{fin}} (\mathbb{T} \uplus \{\bullet\})
\end{array}
$$

The following is an excerpt of the Scala implementation of the type checker for the above language.

```
enum Expr:
  ...
  case App(fexpr: Expr, aexpr: Expr)
  case Nil
  case Cons(head: Expr, tail: Expr)
  case Head(list: Expr)
  case Tail(list: Expr)
enum Type:
  case NumT
  case ArrowT(pty: Type, rty: Type)
  case VarT(k: Int)
  case ListT(elem: Type)
type TypeEnv = Map[String, Type]
type Solution = Map[Int, Option[Type]]
// Unification algorithm
def unify(lty: Type, rty: Type, sol: Solution): Solution = ...
// Generate a new type variable
def newTypeVar(sol: Solution): (Type, Solution) = ...
// Type-checking procedure
def typeCheck(
  expr: Expr,
  tenv: TypeEnv,
  sol: Solution,
): (Type, Solution) = expr match
  ...
  case App(f, a) =>
    val (fty, sol1) = typeCheck(f, tenv, sol)
    val (aty, sol2) = typeCheck(a, tenv, sol1)
    val (rty, sol3) = newTypeVar(sol2)
    val sol4 = unify(ArrowT(aty, rty), fty, sol3)
    (rty, sol4)
  case Nil =>
    val (ety, sol1) = newTypeVar(sol)
    (ListT(ety), sol1)
  case Cons(h, t) =>
    val (hty, sol1) = typeCheck(h, tenv, sol)
    val (tty, sol2) = typeCheck(t, tenv, sol1)
    val sol3 = unify(ListT(hty), tty, sol2)
    (tty, sol3)
  case Head(l) => ...
  case Tail(l) => ...
```

The **typing rules** of this language are defined in the following way. For example, the typing rule for function applications $e(e)$ (i.e., `App`) is defined as follows:

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

$$\frac{\Gamma, \psi \vdash e_f : \tau_f, \psi_f \qquad \Gamma, \psi_f \vdash e_a : \tau_a, \psi_a \qquad \alpha_r \notin \psi_a \qquad \texttt{unify}(\tau_a \to \alpha_r, \tau_f, \psi_a[\alpha_r \mapsto \bullet]) = \psi'}{\Gamma, \psi \vdash e_f(e_a) : \alpha_r, \psi'}$$

where $\texttt{unify} : (\mathbb{T} \times \mathbb{T} \times \Psi) \rightharpoonup \Psi$ is a function that unifies two types and updates a given solution, and it corresponds to the `unify` function in the Scala implementation.

(a) 4 points **Define** the typing rules for the list expressions `nil` (i.e., `Nil`) and $e :: e$ (i.e., `Cons`) according to the given Scala implementation. (**2 points per rule**)

(b) 4 points **Define** the typing rules for the list operations e.`head` (i.e., `Head`) and e.`tail` (i.e., `Tail`). Note that there is no given Scala implementation for these operations. (**2 points per rule**)

You need to define the typing rules to support the type inference of these operations to make the following expression **well-typed**.

```
val f = λx. {
  val y = x.head(42);
  val z = x.tail;
  z.head(y)
};
f
```

(c) 2 points Write the **type** of the following well-typed expression according to the typing rules you defined. (Note that you need to **replace** all **type variables** with their **solutions** in the final type.)

Type:

9. [10 points] This question extends STFAE into BP-STFAE to support not only subtype polymorphism but also **bounded parametric polymorphism**, which is a variant of parametric polymorphism with **bounded quantification** based on the subtype relation.

$$\begin{array}{lll} \text{Expressions} & \mathbb{E} \ni e ::= \ldots \mid \forall[\alpha <: \tau].e \mid e[\tau] \\ \text{Types} & \mathbb{T} \ni \tau ::= \ldots \mid \alpha \mid \forall[\alpha <: \tau].\tau \\ \text{Type Variables} & \quad \alpha \in \mathbb{X}_\alpha \end{array}$$

Syntax is extended with the **bounded type abstraction** ($\forall[\alpha <: \tau].e$) and **type application** ($e[\tau]$) expressions. Types are extended with **type variables** α and **bounded polymorphic types** ($\forall[\alpha <: \tau].\tau$).

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

The **type environment** Γ is extended with another mapping $\mathbb{X}_\alpha \xrightarrow{\text{fin}} \mathbb{T}$ from type variables to types to store the upper bounds of introduced type variables. You can use $\Gamma[\alpha <: \tau]$ to update the upper bound of a type variable, $\alpha \in \text{Domain}(\Gamma)$ to check if a type variable exists, and $\Gamma(\alpha)$ or $\alpha <: \tau \in \Gamma$ to look up the upper bound of a type variable in the type environment.

Values and **operational semantic** rules are extended as follows.

$$\text{Values} \quad \mathbb{V} \ni v ::= \ldots \mid \langle \forall[\alpha <: \tau].e, \sigma \rangle$$

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

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

The goal of this question is to complete the type system of BP-STFAE. You need to fill in the blanks to make the following BP-STFAE expression **well-typed**:

```
val f = ∀[α <: { a : ⊤ }].λ(x : α).{ x.a };
val x = f[{ a : num → num, b : bool }]({ a = λ(z : num).{ z + 1 }, b = true });
val y = f[{ a : num }]({ a = 2 });
val g = λ(z : ∀[α <: num].α → num).z[num]
g(∀[α <: ⊤].{ λ(z : α).z })
```

but the following expressions should be **ill-typed** in the completed type system of BP-STFAE:

- $\lambda(\mathtt{x} : \forall[\alpha <: \beta].\alpha).\mathtt{x}$
- $\lambda(\mathtt{x} : \forall[\alpha <: \mathtt{num}].\alpha).\mathtt{x}[\beta]$
- $\lambda(\mathtt{x} : \{\ \mathtt{a} : \alpha\ \}).\mathtt{x}$
- $\forall[\alpha <: \beta].42$
- $\forall[\alpha <: \mathtt{num}].\forall[\alpha <: \mathtt{num}].42$
- `val f = λ(x : ∀[α <: ⊤].num).x; f(∀[α <: num].42)`

(a) [2 points] Complete the **well-formedness** rules of types for **record types**.

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

$$\frac{}{\Gamma \vdash \mathtt{num}} \qquad \frac{}{\Gamma \vdash \mathtt{bool}} \qquad \frac{\Gamma \vdash \tau \qquad \Gamma \vdash \tau'}{\Gamma \vdash \tau \to \tau'} \qquad \frac{\alpha <: \tau \in \Gamma}{\Gamma \vdash \alpha}$$

$$\frac{\boxed{}}{\Gamma \vdash \{x_1 : \tau_1, \ldots, x_n : \tau_n\}} \qquad \frac{\boxed{}}{\Gamma \vdash \forall[\alpha <: \tau].\tau'}$$

(b) [4 points] Complete the **subtype relation** for **type variables** and **bounded polymorphic types**. (Note that it is defined with a type environment Γ.)

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

$$\ldots$$

$$\frac{\boxed{}}{\Gamma \vdash \alpha <: \tau} \qquad \frac{\boxed{}}{\Gamma \vdash (\forall[\alpha <: \tau_1].\tau_2) <: (\forall[\alpha <: \tau_1'].\tau_2')}$$

(c) [4 points] Complete the **typing rules** for **type abstraction** and **type application**.

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

$$\cdots$$

$$\frac{\boxed{}}{\Gamma \vdash \forall[\alpha <: \tau].e : \boxed{}}$$

$$\frac{\boxed{}}{\Gamma \vdash e[\tau] : \boxed{}}$$

10. [5 points] Church encoding is a way to represent data structures and operations in the λ-calculus. For example, we can represent a **pair** data structure with two values and **first**/**second** operations to extract each value in the untyped language FAE:

```
val pair = λx.λy.{ λf.{ f(x)(y) } };
val fst = λp.p(λx.λy.x);
val snd = λp.p(λx.λy.y);
val p = pair(1)(λx.{ x < 2 });
val a = fst(p);
val b = snd(p);
b(a)
```

In this question, you need to define the above Church encoding in the typed language PTFAE using the **parametric polymorphism** feature. Fill in the blanks to make the following PTFAE expression well-typed according to the typing rules of PTFAE.

```
val pair = ∀α.∀β.λ(x:α).λ(y:β).{ [ (A) ] };
val fst = ∀α.∀β.λ(p : [ (B) ]).[ (C) ](λ(x:α).λ(y:β).x);
val snd = ∀α.∀β.λ(p : [ (B) ]).[ (D) ](λ(x:α).λ(y:β).y);
val p = pair[num][num → bool](1)(λ(x:num).{ x < 2 });
val a = fst[num][num → bool](p);
val b = snd[num][num → bool](p);
b(a)
```

(A) = [ ]

(B) = [ ]

(C) = [ ]

(D) = [ ]

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

Appendix

TFAE – Typed Functions and Arithmetic Conditional Expressions

Expressions $\mathbb{E} \ni e ::= n \mid b \mid x \mid e \texttt{ + } e \mid e \texttt{ * } e \mid e \texttt{ < } e \mid \texttt{val } x \texttt{ = } e\texttt{; } e \mid \lambda([x{:}\tau]^*).e \mid e(e^*) \mid \texttt{if } (e)\ e \texttt{ else } e$

Types $\mathbb{T} \ni \tau ::= \texttt{num} \mid \texttt{bool} \mid (\tau^*) \to \tau$ Booleans $b \in \mathbb{B}$ Numbers $n \in \mathbb{Z}$ Identifiers $x \in \mathbb{X}$

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

$$\frac{}{\sigma \vdash n \Rightarrow n} \qquad \frac{}{\sigma \vdash b \Rightarrow b} \qquad \frac{x \in \text{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}$$

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

$$\frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 \texttt{ < } e_2 \Rightarrow n_1 \texttt{ < } n_2} \qquad \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \texttt{val } x \texttt{ = } e_1\texttt{; } e_2 \Rightarrow v_2}$$

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

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow \langle \lambda(x_1, \ldots, x_n).e, \sigma' \rangle \\ \sigma \vdash e_1 \Rightarrow v_1 \qquad \ldots \qquad \sigma \vdash e_n \Rightarrow v_n \qquad \sigma'[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n] \vdash e \Rightarrow v\end{array}}{\sigma \vdash e_0(e_1, \ldots, e_n) \Rightarrow v}$$

$$\frac{\sigma \vdash e_0 \Rightarrow \texttt{true} \qquad \sigma \vdash e_1 \Rightarrow v_1}{\sigma \vdash \texttt{if } (e_0)\ e_1 \texttt{ else } e_2 \Rightarrow v_1} \qquad \frac{\sigma \vdash e_0 \Rightarrow \texttt{false} \qquad \sigma \vdash e_2 \Rightarrow v_2}{\sigma \vdash \texttt{if } (e_0)\ e_1 \texttt{ else } e_2 \Rightarrow v_2}$$

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

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

$$\frac{}{\Gamma \vdash n : \texttt{num}} \qquad \frac{}{\Gamma \vdash b : \texttt{bool}} \qquad \frac{x \in \text{Domain}(\Gamma)}{\Gamma \vdash x : \Gamma(x)}$$

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

$$\frac{\Gamma \vdash e_1 : \texttt{num} \qquad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash e_1 \texttt{ < } e_2 : \texttt{bool}} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma[x : \tau_1] \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{val } x \texttt{ = } e_1\texttt{; } e_2 : \tau_2}$$

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

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

$$\frac{\Gamma \vdash e_0 : \texttt{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{if } (e_0)\ e_1 \texttt{ else } e_2 : \tau}$$

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

TRFAE – **TFAE** with Recursion

$$\text{Expressions} \quad \mathbb{E} \ni e ::= \ \ldots \mid \texttt{def}\ x([x\texttt{:}\tau]^*)\texttt{:}\tau\ \texttt{=}\ e\texttt{;}\ e$$

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

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

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

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

ATFAE – **TRFAE** with Algebraic Data Types

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

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

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

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

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

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

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

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

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

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

Well-formedness of Types $\boxed{\Gamma \vdash \tau}$

$$\frac{}{\Gamma \vdash \texttt{num}} \qquad \frac{}{\Gamma \vdash \texttt{bool}}$$

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

PTFAE – TFAE 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 $\alpha \in \mathbb{X}_\alpha$

Note that this language restricts the number of function parameters to one for simplicity.

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

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

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

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

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

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

Well-formedness of Types $\boxed{\Gamma \vdash \tau}$

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

STFAE – TFAE with Records and Subtype Polymorphism

Expressions $\mathbb{E} \ni e ::= \ldots \mid \{[x = e]^*\} \mid e.x \mid \texttt{exit}$ Types $\mathbb{T} \ni \tau ::= \ldots \mid \{[x : \tau]^*\} \mid \bot \mid \top$

Note that this language restricts the number of function parameters to one for simplicity.

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

$$\ldots \qquad \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \ldots \qquad \sigma \vdash e_n \Rightarrow v_n}{\sigma \vdash \{x_1 = e_1, \ldots, x_n = e_n\} \Rightarrow \{x_1 = v_1, \ldots, x_n = v_n\}} \qquad \frac{\sigma \vdash e \Rightarrow \{x_1 = v_1, \ldots, x_n = v_n\} \qquad 1 \le i \le n}{\sigma \vdash e.x_i \Rightarrow v_i}$$

$$\text{Values} \quad \mathbb{V} \ni v ::= \ldots \mid \{[x = v]^*\}$$

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

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

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

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

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

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

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