Exercise 1 : Curry-Howard Isomorphism (8 points)

Give proofs of the following propositional formula using the Curry-Howard isomorphism between constructive logic and typed λ-calculus with products and sums (see Appendix A for details).

1. $(A \wedge B) \Rightarrow C \Rightarrow ((C \wedge A) \wedge B)$
2. $(A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow (A \vee B) \Rightarrow C$
3. $(A \vee B \Rightarrow C) \Rightarrow ((A \Rightarrow C) \wedge (B \Rightarrow C))$
4. $((A \Rightarrow B \vee C) \wedge (B \Rightarrow D) \wedge (C \Rightarrow D)) \Rightarrow (A \Rightarrow D)$

Exercise 2 : Subtyping for products (10 points)

The subtyping rule for products can be stated as:

$$\frac{S_1 <: T_1 \qquad S_2 <: T_2}{S_1 \times S_2 <: T_1 \times T_2} \tag{S-Prod}$$

In the course you were presented with the inversion lemma for subtyping with function types i.e., S-Arrow. Your task for this exercise is to write a proof for the following theorem for STLC with products and subtyping (see Appendices C and D).

Theorem 1. *If* $S_1 \times S_2 <: T$*, then either* $T = \texttt{Top}$ *or else* $T = T_1 \times T_2$*, with* $S_1 <: T_1$ *and* $S_2 <: T_2$*.*

Hint: proof the theorem by induction on the last used subtyping rule. State any lemmas that you use (without proof).

Exercise 3 : Tracking function purity (10 points)

In this exercise, we will work with a variant of STLC with mutable state. Our goal is to create a type system which tracks if functions are *pure* in the sense that they at most mutate state internal to themselves, or if they are *impure* and mutate some globally accessible state.

The calculus we consider does not support first-class references, like the ones you have seen in the course. Instead, our calculus will operate like Scala: we cannot mention a mutable variable without either extracting its value or assigning to it.

Our extensions to the syntax of STLC are as follows (note that we *replace* the function type):

$$
\begin{array}{rcll}
t, s & ::= & & \textbf{terms}: \\
& | & \textbf{letvar}\, x := s\, \textbf{in}\, t & \text{mutable variable binding} \\
& | & x := t & \text{mutable variable assignment} \\
& | & !x & \text{mutable variable dereference} \\
& | & t; t & \text{sequencing} \\
& | & \textbf{unit} & \text{unit} \\
\\
T, S & ::= & & \textbf{types}: \\
& | & T \xrightarrow{p} T & \text{function type} \\
& | & \textbf{Ref}\, T & \text{reference type} \\
& | & \textbf{Unit} & \text{unit type} \\
p & ::= & & \textbf{purity}: \\
& | & \text{pure} & \\
& | & \text{impure} &
\end{array}
$$

Additionally, we will use metavariables P and Q to specify a *purity set*, which is either a (potentially empty) set of variables, or Ω, conceptually representing a set of all variables. We extend set union and membership to work with Ω as follows:

$$\Omega \cup P = \Omega = P \cup \Omega$$

$$\forall\, x.\ x \in \Omega$$

Our goal is a typing judgement of the form $\Gamma \vdash t : T \,/\, P$, where P is the set of all variables that evaluating t could mutate. Here are some of the rules we need. In (T-Var), we disallow mentioning mutable variables as expressions – remember that our mutable variables are Scala-like. In (T-Assign), we *extend* the purity set P with x, the variable being mutated. In general, in order to make it coherent to mention variable names in the conclusion of a typing judgment, we require that all variables in the entire program are unique[1].

$$\frac{x : T \in \Gamma \qquad \forall\, S.\ T \neq \textbf{Ref}\, S}{\Gamma \vdash x : T \,/\, \varnothing} \tag{T-Var}$$

$$\frac{x : \textbf{Ref}\, T \in \Gamma \qquad \Gamma \vdash t : T \,/\, P}{\Gamma \vdash x := t : \textbf{Unit} \,/\, P \cup \{x\}} \tag{T-Assign}$$

Remember that in this exercise, we say a lambda is pure if it either doesn't assign to any variable, or it assigns to strictly local variables. For instance, the following lambda only mutates local state and therefore should be typed using $\varnothing$ as the purity set.

[1] Technically this is known as Barendregt's convention.

$$\lambda\, x.\ \mathbf{letvar}\, y := 0\, \mathbf{in}\, y := 1; x$$

Your first three tasks are:

Task 1. (1 point) What typing rule should be used for sequencing $t; s$?

Task 2. (2 points) What typing rule *or rules* should be used for lambas $\lambda\, x.\ t$?

Task 3. (2 points) What typing rule should be used for variable binding $\mathbf{letvar}\, x := s\, \mathbf{in}\, t$?

In the remaining tasks, we consider evaluation. We will use store-based operational semantics similar to STLC with references. We will reduce variable binding as follows:

$$\frac{l \notin \mathrm{dom}(\mu)}{\mathbf{letvar}\, x := v\, \mathbf{in}\, t \mid \mu \longrightarrow [x \mapsto l]t \mid (\mu, l := v)} \quad \text{(E-LetVar)}$$

$$\frac{s \mid \mu \longrightarrow s' \mid \mu'}{\mathbf{letvar}\, x := s\, \mathbf{in}\, t \mid \mu \longrightarrow \mathbf{letvar}\, x := s'\, \mathbf{in}\, t \mid \mu'} \quad \text{(E-LetVar1)}$$

The other reduction rules are the same as in STLC with references. As a refresher, they are as follows:

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1\ t_2 \mid \mu \longrightarrow t_1'\ t_2 \mid \mu'} \quad \text{(E-App1)}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1\ t_2 \mid \mu \longrightarrow v_1\ t_2' \mid \mu'} \quad \text{(E-App2)}$$

$$(\lambda\ x.\ t_{12})\ v_2 \mid \mu \longrightarrow [x \mapsto v_2]t_{12} \mid \mu \quad \text{(E-AppAbs)}$$

$$\frac{\mu(l) = v}{!l \mid \mu \longrightarrow v \mid \mu} \quad \text{(E-DerefLoc)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{!t_1 \mid \mu \longrightarrow !t_1' \mid \mu'} \quad \text{(E-Deref)}$$

$$l := v_2 \mid \mu \longrightarrow \mathtt{unit} \mid [l \mapsto v]\mu \quad \text{(E-Assign)}$$

$$\frac{t \mid \mu \longrightarrow t' \mid \mu'}{x := t \mid \mu \longrightarrow x := t' \mid \mu'} \quad \text{(E-Assign1)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1; t_2 \mid \mu \longrightarrow t_1'; t_2 \mid \mu'} \quad \text{(E-Seq)}$$

$$v; t_2 \mid \mu \longrightarrow t_2 \mid \mu \quad \text{(E-SeqNext)}$$

Task 4. (3 points) Intuitively, we might expect calling pure functions to not modify the store. However, the following statement is false in our calculus:

If $\varnothing \vdash t : S \xrightarrow{\text{pure}} T \,/\, \varnothing$ and $\varnothing \vdash t\ s : T \,/\, \varnothing$ and $t\ s \mid \mu \longrightarrow^* v \mid \mu'$, then $\mu = \mu'$.

Demonstrate why with a counterexample.

Task 5. (2 points) Change the conclusion of the above statement so that correctly relates μ and μ' with an equality. Tautologies are worth no points.

Appendix A: **Curry-Howard Isomorphism**

The *Curry-Howard isomorphism* or *Curry-Howard correspondence* establishes a connection between type systems and logical calculi based on an observation that the ways we build types are structurally similar to the ways we build formulae.

According to the Curry-Howard isomorphism, proofs can be represented as programs and formulae they prove can be represented as types of those programs. Here is a (non-comprehensive) list of some examples of how concepts from constructive logic correspond to concepts from the simply typed lambda calculus.

Constructive logic	Simply typed lambda calculus
Formula	Type
$A \Rightarrow B$	$A \to B$
$A \wedge B$	$A \times B$
$A \vee B$	$A + B$
Proof of a formula	Term that inhabits a type

Appendix B: **The simply-typed lambda calculus**

$$
\begin{array}{lcll}
t & ::= & & \textbf{terms:} \\
 & | & x & \text{variable} \\
 & | & \lambda\, x\colon T.\, t & \text{abstraction} \\
 & | & t\ t & \text{application} \\
 & & & \\
v & ::= & & \textbf{values:} \\
 & | & \lambda\, x\colon T.\, t & \text{abstraction-value} \\
 & & & \\
T & ::= & & \textbf{types:} \\
 & | & T \to T & \text{type of functions}
\end{array}
$$

Evaluation rules:

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \qquad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \qquad \text{(E-App2)}$$

$$(\lambda\ x\colon T_1.\ t_1)\ v_2 \longrightarrow [x \to v_2]\ t_1 \qquad \text{(E-AppAbs)}$$

Typing rules:

$$\frac{x\colon T \in \Gamma}{\Gamma \vdash x : T} \qquad \text{(T-Var)}$$

$$\frac{\Gamma,\ x\colon T_1 \vdash t_2\ :\ T_2}{\Gamma \vdash (\lambda\, x\colon T_1.\ t_2) : T_1 \to T_2} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\ t_2 : T_2} \qquad \text{(T-App)}$$

Appendix C: **Subtyping extension to STLC**

$$\text{(S-Refl)}\ S <: S \qquad \text{(S-Trans)}\ \frac{S <: U \qquad U <: T}{S <: T}$$

$$\text{(S-Top)}\ S <: \texttt{Top} \qquad \text{(S-Arrow)}\ \frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2}$$

Appendix E: **Product extension to STLC**

$$
\begin{array}{rcll}
t & ::= & \ldots & \textbf{terms:} \\
 & | & \{t, t\} & \text{pair} \\
 & | & t.\mathtt{1} & \text{first projection} \\
 & | & t.\mathtt{2} & \text{second projection} \\
 & & & \\
v & ::= & \ldots & \textbf{values:} \\
 & | & \{v, v\} & \text{pair value} \\
 & & & \\
T & ::= & \ldots & \textbf{types:} \\
 & | & T_1 \times T_2 & \text{product type}
\end{array}
$$

Typing rules:

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2} \tag{T-Pair}$$

$$\frac{\Gamma \vdash t : T_1 \times T_2}{\Gamma \vdash t.\mathtt{1} : T_1} \tag{T-Proj1}$$

$$\frac{\Gamma \vdash t : T_1 \times T_2}{\Gamma \vdash t.\mathtt{2} : T_2} \tag{T-Proj2}$$

New evaluation rules:

$$\{v_1, v_2\}.\mathtt{1} \longrightarrow v_1 \tag{E-PairBeta1}$$

$$\{v_1, v_2\}.\mathtt{2} \longrightarrow v_2 \tag{E-PairBeta2}$$

$$\frac{t \longrightarrow t'}{t.\mathtt{1} \longrightarrow t'.\mathtt{1}} \tag{E-Proj1}$$

$$\frac{t \longrightarrow t'}{t.\mathtt{2} \longrightarrow t'.\mathtt{2}} \tag{E-Proj2}$$

$$\frac{t_1 \longrightarrow t_1'}{\{t_1, t_2\} \longrightarrow \{t_1', t_2\}} \tag{E-Pair1}$$

$$\frac{t_2 \longrightarrow t_2'}{\{v_1, t_2\} \longrightarrow \{v_1, t_2'\}} \tag{E-Pair2}$$