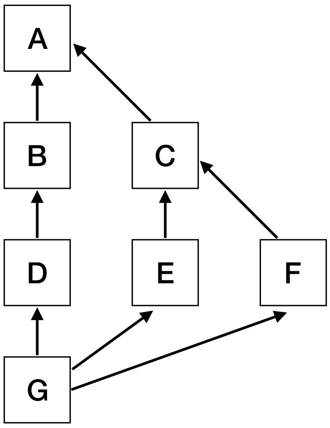
Exercise 1 : Specify Subtyping Relationship (10 points)

In this exercise we consider the system STLC extended with subtyping and a set of base types A, B, C, D, E, F, G. Subtyping between the *base types* is defined based on the following diagram:



In the diagram, the nodes represent types, and the arrows represent subtyping relationships between base types. For example, we have $B <: A$ and $E <: C$. The types in the system are defined as follows:

$$\begin{array}{lcll} \alpha & ::= & A \mid B \mid C \mid D \mid E \mid F \mid G & \text{base types} \\ S,T,U,L & ::= & \alpha \mid T \to T & \text{types} \end{array}$$

The subtyping rules are summarized below, which is standard:

$$\frac{\text{There is an arrow from } S \text{ to } T \text{ in the diagram}}{S \ <: \ T} \qquad \text{(S-Base)}$$

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

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

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

The *least upper bound* (LUB) and *greatest lower bound* (GLB) of types are specified as follows:

$$\begin{array}{lcl} LUB(T_1, T_2) = U & \iff & T_1 <: U \wedge T_2 <: U \wedge \forall U'.(T_1 <: U' \wedge T_2 <: U') \to U <: U' \\ GLB(T_1, T_2) = L & \iff & L <: T_1 \wedge L <: T_2 \wedge \forall L'.(L' <: T_1 \wedge L' <: T_2) \to L' <: L \end{array}$$

Part 1 (8 points). For each of the following pairs of types, compute LUB and GLB. If LUB or GLB does not exist, answer *None*.

1. B and C

2. A and $A \to A$

3. $D \to C$ and $A \to A$

4. $G \to A$ and $(G \to A) \to B$

5. $G \to D \to C$ and $G \to B \to A$

Part 2 (2 points). Can we extend subtyping relationship to make LUBs and GLBs always exist for given examples? What changes to types and subtyping rules are needed?

Exercise 2 : Curry-Howard Correspondence (10 points)

The well-known *Curry-Howard correspondence* describes a mapping between type theory and logic: propositions correspond to types and proofs correspond to programs. This correspondence is usually formulated only for *intuitionistic logics* (IL), in which the *law of excluded middle* (LEM) or equivalently the *law of double negation* (DNE) does not hold. Concretely, the following propositions are not provable in IL, thus by the correspondence there exists no programs that prove them:

- LEM: $\forall P.P \vee \neg P$
- DNE: $\forall P.\neg\neg P \to P$

This problem is about proving that intuitionistic logic with the law of excluded middle is equivalent to intuitionistic logic with the law of double negation, that is $IL + LEM = IL + DNE$.

Curry-Howard: Negation. In intuitionistic logic, $\neg P$ is the same as $P \to \bot$, where $\bot$ means absurdity. So the type $\neg\neg P$ is interpreted as $(P \to \bot) \to \bot$. We assume absurdity $\bot$ corresponds to the type $\bot$ in types, which is not inhabited. The following program *explode* is provided:

$$explode : \forall P.\bot \to P$$

The program *explode* has the type $\forall P.\bot \to P$. Logically, it says that from absurdity any proposition can be derived, which corresponds to a well-known principle in logic.

Curry-Howard: Universal quantification and System F. A second-order proposition of form $\forall P.T$ corresponds to a type in System F and can be proved by a System F term. For example, $\forall P.P \to P$ is proved by the program $\Lambda P.\lambda x : P.x$.

Task. Please prove the following propositions. The last two prove that $IL + LEM = IL + DNE$:

(1) $\forall P.P \to \neg\neg P$

Hint: find a term that inhabits $\forall P.P \to (P \to \bot) \to \bot$.

prog1 =

(2) $\forall P.\neg\neg(P \vee \neg P)$

prog2 = $\Lambda P.\lambda f : (P + P \to \bot) \to \bot.$

$let\ a : P \to \bot =$ ________________________________ in

$let\ b : (P \to \bot) \to \bot =$ ________________________________ in

______

(3) $(\forall P. P \vee \neg P) \rightarrow (\forall Q. \neg\neg Q \rightarrow Q)$

prog3 = $\lambda x : \forall P.(P + P \rightarrow \bot).\Lambda Q.\lambda f : (Q \rightarrow \bot) \rightarrow \bot.$
$case(x\ [Q])\ of$
$inl\ q \Rightarrow$ ____
$inr\ nq \Rightarrow$ ____

(4) $(\forall P. \neg\neg P \rightarrow P) \rightarrow (\forall Q. Q \vee \neg Q)$

prog4 =

Exercise 3 : Transitivity of Algorithmic Subtyping in $F_{<:}$ (10 points)

In this problem, we study algorithmic subtyping in System $F_{<:}$. System $F_{<:}$ is an extension of System F with subtyping of types and bounds on type variables. The types in System $F_{<:}$ are defined as follows:

$$T ::= \mathit{Top} \mid X \mid T \to T \mid \forall X <: T.T$$

One approach to formulate subtyping in $F_{<:}$ is algorithmic subtyping. The subtyping rules are given as follows:

$$\Gamma \vdash T <: \mathit{Top} \qquad \text{(S-TOP)}$$

$$\Gamma \vdash X <: X \qquad \text{(S-TVAR-REFL)}$$

$$\frac{X <: T \in \Gamma \qquad \Gamma \vdash T <: U}{\Gamma \vdash X <: U} \qquad \text{(S-TVAR-TRANS)}$$

$$\frac{\Gamma \vdash T_1 <: S_1 \qquad \Gamma \vdash S_2 <: T_2}{\Gamma \vdash S_1 \to S_2 <: T_1 \to T_2} \qquad \text{(S-FUN)}$$

$$\frac{\Gamma, X <: U \vdash S <: T}{\Gamma \vdash \forall X <: U.S \ <: \ \forall X <: U.T} \qquad \text{(S-ALL)}$$

For this problem, we may assume the typing environment Γ to be just a list of type bounds:

$$\Gamma ::= \emptyset \mid \Gamma, X <: T$$

The typing environment Γ is used in the rule S-TVAR-TRANS, and it is augmented in the rule S-ALL. For simplicity, in the rule S-ALL, we require the bound of two universal types to be the same type U.

Please prove the following theorem in the subtyping system.

Theorem 1 (Transitivity)**.** *If* $\Gamma \vdash S <: U$ *and* $\Gamma \vdash U <: T$, *then* $\Gamma \vdash S <: T$.

For reference: **Simply Typed Lambda Calculus**

The complete reference of the simply typed lambda calculus is:

$$
\begin{array}{lllr}
t & ::= & & \textbf{terms}: \\
 & | & x & \textit{variable} \\
 & | & \lambda\, x\!:\!\mathtt{T}.\ t & \textit{abstraction} \\
 & | & t\ t & \textit{application}
\end{array}
$$

$$
\begin{array}{lllr}
v & ::= & & \textbf{values}: \\
 & | & \lambda\, x\!:\!\mathtt{T}.\ t & \textit{abstraction} - \textit{value}
\end{array}
$$

$$
\begin{array}{lllr}
\mathtt{T} & ::= & & \textbf{types}: \\
 & | & \mathtt{T} \to \mathtt{T} & \textit{type of functions (right assoc.)}
\end{array}
$$

Evaluation rules:

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

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

$$(\lambda\, x\!:\ \mathtt{T}_1.\ t_1)\ v_2 \longrightarrow [x \to v_2]\ t_1 \qquad \text{(E-AppAbs)}$$

Typing rules:

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

$$\frac{\Gamma,\ x : \mathtt{T}_1 \vdash t_2 : \mathtt{T}_2}{\Gamma \vdash (\lambda\, x\!:\ \mathtt{T}_1.\ t_2) : \mathtt{T}_1 \to \mathtt{T}_2} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : \mathtt{T}_1 \to \mathtt{T}_2 \quad \Gamma \vdash t_2 : \mathtt{T}_1}{\Gamma \vdash t_1\ t_2 : \mathtt{T}_2} \qquad \text{(T-App)}$$