1. Consider the following program, written in lambda calculus with tuples, integers, and strings:

\[ \text{let app2} = \lambda f xy. (f x, f y) \text{ in} \]

\[ \text{app2} \ (\lambda x. x) \ 1 \ "\text{foo}" \]

Write down the type for \texttt{app2} in simply-typed lambda calculus with Hindley-Milner style polymorphism. Does this program exhibit any run-time errors (i.e., will its evaluation ever be stuck)? Does the program type check using Hindley-Milner style polymorphism? Explain what goes wrong. Can you give a type for \texttt{app2} that is polymorphic but not Hindley-Milner such that this program would type check? (Note: You will not be able to construct a most-general type for \texttt{app2} without using intersection types, which we have not discussed, but your type should work for this particular use of \texttt{app2}).

2. Consider the lambda calculus with Hindley-Milner style polymorphism. Here \( e ::= x \mid \lambda x.e \mid e\ e \mid \text{let } x = e \text{ in } e \) and types are given by \( \sigma ::= \forall \alpha.\tau \mid \tau \) and \( \tau ::= \alpha \mid \text{int} \mid \tau \to \tau \), with the following type rules:

\[
\begin{align*}
\text{VAR} & : \quad \Gamma(x) = \forall\alpha.\tau \quad \text{any } \tau' \\
\Gamma & \vdash x : \tau[\alpha \mapsto \tau'] \\
\text{LAM} & : \quad \Gamma, x : \tau \vdash e : \tau' \\
\Gamma & \vdash \lambda x.e : \tau \to \tau' \\
\text{APP} & : \quad \Gamma \vdash e_1 : \tau \to \tau' \quad \Gamma \vdash e_2 : \tau \\
\Gamma & \vdash e_1\ e_2 : \tau' \\
\text{LET} & : \quad \Gamma \vdash e_1 : \tau_1 \quad \alpha = FV(\tau_1) - FV(\Gamma) \\
\Gamma, x : \forall\alpha.\tau_1 & \vdash e_2 : \tau_2 \\
\Gamma & \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2
\end{align*}
\]

Here \( FV(\cdot) \) stands for the free variables of a type or type environment. Prove the polymorphic substitution lemma, which is used in proving subject reduction: If \( \Gamma, x : \forall\alpha.\tau_1 \vdash e_2 : \tau_2 \) where \( \alpha = FV(\tau_1) - FV(\Gamma) \) and \( \Gamma \vdash e_1 : \tau_1 \), then \( \Gamma \vdash e_2[x \mapsto e_1] : \tau_2 \).