1 Soundness of the Equivalence Algorithm (5 points)

Prove that the algorithm shown in class (and in Chapter 6 of ATTAPL) for deciding extensional equivalence of terms in the presence of `Unit` is sound with respect to definitional equivalence, i.e.,

\[ \text{If } \Gamma \vdash e_1 : \tau \text{ and } \Gamma \vdash e_2 : \tau \text{ and } \Gamma \vdash e_1 \leftrightarrow e_2 : \tau, \text{ then } \Gamma \vdash e_1 \equiv e_2 : \tau. \]

**Hint:** You may find it useful to rely on the fact that well-formed terms in this language have unique types (assuming we annotate \( \lambda \)-bound variables with their types).

2 Completeness in the Presence of a Top Type (5 points)

Suppose we add a `Top` type to the language considered in Chapter 6. The idea is that `Top` is a supertype (in the sense of subtyping) of every other type in the language, so every well-formed term can also be given type `Top` by subsumption. For instance, if our language supported product types (which would be a straightforward extension), the `Top` type might be useful for giving a “record-polymorphic” type to the `snd` function: `snd : \text{Top} \times \tau \rightarrow \tau \triangleq \lambda x. \pi_2(x)`. One could then apply `snd` to any term of product type whose second component had type \( \tau \), without concern for the type of the first component (since it’s guaranteed to be a subtype of `Top`).

Here are the extensions to the typing and equivalence judgments. They make use of a new subtyping judgment \( \vdash \tau_1 \leq \tau_2 \):

\[
\begin{align*}
\Gamma \vdash e : \tau & \quad \vdash \tau \leq \tau \\
\Gamma \vdash e : \tau & \quad \vdash \tau \leq \text{Top} \\
\Gamma \vdash e_1 \equiv e_2 : \text{Top} & \\
\Gamma \vdash e_1 \equiv e_2 : \tau & \quad \vdash \tau_1 \leq \tau_2 \\
\end{align*}
\]

Note that all terms are equivalent when considered at type `Top`. The reason for this is simple: if all you know about \( e_1 \) and \( e_2 \) is that they both have type `Top`, then there is nothing you can do with them, so there is no way to distinguish them, and thus they are extensionally equivalent.

**Problem:** Extend the equivalence algorithm to handle `Top` and extend the logical relations proof of completeness (i.e., that \( \Gamma \vdash e_1 \equiv e_2 : \tau \) implies \( \Gamma \vdash e_1 \leftrightarrow e_2 : \tau \)) accordingly. You don’t have to redo all the old parts of the proof; just show the new ones.

**Hint:** `Top` is kind of like `Unit`, so most of the new cases will be trivial. One will be non-trivial.
3  Adding Let to the Calculus with Definitions (5 points)

Suppose we add a \texttt{let} construct, \texttt{let }x = e_1 \texttt{ in } e_2, to the calculus with (acyclic) definitions presented in class, along with the following new typing and definitional equivalence rules:

\[
\frac{\Gamma \vdash e_1 : \tau_1}{\Gamma \vdash \texttt{let } x = e_1 \texttt{ in } e_2 : \tau} \quad \frac{\Gamma, x : \tau_1 \vdash e_2 : \tau}{\Gamma \vdash \texttt{let } x = e_1 \texttt{ in } e_2 : \tau} \quad \frac{\Gamma \vdash e_1 : \tau_1}{\Gamma, x = e_1 \vdash e_2 \equiv e : \tau} \quad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{let } x = e_1 \texttt{ in } e_2 \equiv e : \tau}
\]

Modify the algorithm so that it is sound and complete w.r.t. the new definitional equivalence, and prove completeness. (As usual, you don’t have to redo the whole proof; just show the new parts.)