Recall the definition of MinML from the textbook. Here is the notation that we shall use:

variables: \( x \) ::= \ldots

numbers: \( n \) ::= 0, 1, -1, 2, -2, \ldots

operations: \( \oplus \) ::= + | \ast | - | = | <

types \( \tau \)::= \text{int} | \text{bool} | \tau \rightarrow \tau

expressions: \( e \)::= x | n | \text{true} | \text{false} | e \oplus e | \text{if} \ e \text{ then } e \text{ else } e | \text{fun} f(x: \tau) : \tau \text{ is } e | e \ e

The typing rules are:

\[
\begin{align*}
\Gamma(x) = \tau & \quad \text{VAR} \quad & \Gamma \vdash n : \text{int} & \quad \text{NUM} \quad & \Gamma \vdash \text{true} : \text{bool} & \quad \text{TRUE} \quad & \Gamma \vdash \text{false} : \text{Bool} & \quad \text{FALSE} \\
\oplus \in \{+, \ast, -\} & \quad \text{ ARITH} \quad & \Gamma \vdash e_1 : \text{int} & \quad & \Gamma \vdash e_2 : \text{int} & \quad & \frac{}{\Gamma \vdash e_1 \oplus e_2 : \text{int}} \\
\oplus \in \{=, <\} & \quad \text{ CMP} \quad & \Gamma \vdash e_1 : \text{int} & \quad & \Gamma \vdash e_2 : \text{int} & \quad & \frac{}{\Gamma \vdash e_1 \oplus e_2 : \text{bool}} \\
\Gamma \vdash e_1 : \text{bool} & \quad \text{ FUN} \quad & \Gamma \vdash e_2 : \tau & \quad \Gamma \vdash e_3 : \tau & \quad \Gamma \vdash e_1 \text{ if } e_2 \text{ then } e_3 \text{ else } e_4 & \quad \frac{}{\Gamma \vdash \text{if} e_1 \text{ then } e_2 \text{ else } e_3 : \tau} \\
\Gamma \vdash \text{fun} f(x: \tau_1) : \tau_2 \text{ is } e : \tau_1 \rightarrow \tau_2 & \quad \text{APP} \quad & \Gamma \vdash e_1 : \tau_2 \rightarrow \tau & \quad \Gamma \vdash e_2 : \tau_2 & \quad \frac{}{\Gamma \vdash e_1 \ e_2 : \tau}
\end{align*}
\]

1. Suppose we added a \textbf{let}-form, in the style of the “arithmetic plus let” language discussed earlier in the course:

\[
e ::= \ldots | \text{let } x = e \text{ in } e
\]

(a) Give the typing rule for the \textbf{let}-form.
(b) Consider a small-step structural operational semantics for the language, i.e., a set of rules that lets us derive judgments of the form \( e \rightarrow^1 e' \) indicating that expression \( e \) “steps to” expression \( e' \). Show the rule(s) that must be added to handle the new \texttt{let}-form.

(c) Explain the part of a proof of \textit{progress} that deals with \texttt{let}.

(d) Explain the part of a proof of \textit{preservation} that deals with \texttt{let}.

(e) How does the C-machine (section 11.1 in the Harper text) have to be extended to handle \texttt{let}? Show any additional frames and any additional transitions.

2. Do the cases in the proofs of Theorem 11.1 (p. 73 in the Harper text), parts 1 and 2, involving \texttt{if}-expressions.