Sample solution for Pierce 15.5.2.

(1) If we drop the first premise of S-REF, we have

\[
\frac{T_1 <: S_1}{\text{Ref } S_1 <: \text{Ref } T_1} \quad (\text{S-REF1})
\]

Since \( \{x:\text{Bool},y:\text{Bool}\} <: \{x:\text{Bool}\} \), we have by S-REF1 that \( \text{Ref } \{x:\text{Bool}\} <: \text{Ref } \{x:\text{Bool},y:\text{Bool}\} \). So the program

\[
(\lambda r: \text{Ref } \{x:\text{Bool},y:\text{Bool}\}. !r.y) (\text{ref } \{x=true\})
\]

or, equivalently,

\[
\text{let } r = \text{ref } \{x=true\} \text{ as Ref } \{x:\text{Bool},y:\text{Bool}\} \text{ in } !r.y
\]

will typecheck, since we can apply T-SUB to the argument to make it match the declared type (or ascription) for \( r \). But the program will clearly get stuck attempting to access the \( y \) field from a record that has none.

(2) If we drop the second premise of S-REF, we have

\[
\frac{S_1 <: T_1}{\text{Ref } S_1 <: \text{Ref } T_1} \quad (\text{S-REF2})
\]

This gives \( \text{Ref } \{x:\text{Bool},y:\text{Bool}\} <: \text{Ref } \{x:\text{Bool}\} \). So the program

\[
\text{let } r = \text{ref } \{x=true,y=true\} \text{ in } (\text{Au:Ref } \{x:\text{Bool}\}. \ u := \{x=true\}) \ r; \ r.y
\]

or, equivalently,

\[
\text{let } r = \text{ref } \{x=true,y=true\} \text{ in } \\
\text{let } u = r \text{ as Ref } \{x:\text{Bool}\} \text{ in } \\
\quad \text{u := } \{x=true\}; \\
\quad r.y
\]

will typecheck, since we can apply T-SUB to \( r \) to make it match the declared type (or ascription) for \( u \). But the program will again get stuck attempting to access the \( y \) field from a record that has none.

Sample solution for Pierce 16.2.3.

Let \( s = (\lambda r: S.r)\{x=true,y=true\}, t = \{x=true,y=true\}, S = \{x:\text{Bool}\}, \text{ and } T = \{x:\text{Bool},y:\text{Bool}\} \). Then we have \( s \rightarrow^* t, \vdash s : S, \vdash t : T, \text{ and } T <: S \text{ but not vice-versa.} \)