We'll consider the following language, the simply-typed lambda calculus extended with booleans.

$$
\begin{array}{lll}
\text{Contexts} & & \\
\Gamma ::= \emptyset & & \text{empty context} \\
\quad \mid \ \Gamma, x : \tau & & \text{context extension} \\
\text{Expressions} & & \\
e ::= x & & \text{variable} \\
\quad \mid \ e_1 \ e_2 & & \text{function application} \\
\quad \mid \ \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 & & \text{if} \\
\quad \mid \ v & & \text{values} \\
\text{Values} & & \\
v ::= \lambda x : \tau.\ e & & \text{function abstraction} \\
\quad \mid \ \texttt{true} & & \\
\quad \mid \ \texttt{false} & & \\
\text{Types} & & \\
\tau ::= \tau_1 \to \tau_2 & & \text{function type} \\
\quad \mid \ \texttt{Bool} & &
\end{array}
$$

Call-by-value operational semantics:

$$\frac{}{(\lambda x : \tau.\ e)\ v \longrightarrow e[x \mapsto v]}\ (\text{E-BETA})$$

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2}\ (\text{EC-LEFT}) \qquad \frac{e_2 \longrightarrow e_2'}{v_1\ e_2 \longrightarrow v_1\ e_2'}\ (\text{EC-RIGHT})$$

$$\frac{}{\texttt{if true then } e_1 \texttt{ else } e_2 \longrightarrow e_1}\ (\text{E-IFTRUE}) \qquad \frac{}{\texttt{if false then } e_1 \texttt{ else } e_2 \longrightarrow e_2}\ (\text{E-IFFALSE})$$

$$\frac{e_0 \longrightarrow e_0'}{\texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 \longrightarrow \texttt{if } e_0' \texttt{ then } e_1 \texttt{ else } e_2}\ (\text{EC-IF})$$

Static semantics:

$$\frac{}{\Gamma \vdash x : \Gamma(x)}\ (\text{T-VAR})$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.\ e : \tau_1 \to \tau_2}\ (\text{T-ABS}) \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ (\text{T-APP})$$

$$\frac{}{\Gamma \vdash \texttt{false} : \texttt{Bool}}\ (\text{T-FALSE}) \qquad \frac{}{\Gamma \vdash \texttt{true} : \texttt{Bool}}\ (\text{T-TRUE})$$

We want to prove the type system is sound. The following treatment is by Andrew Wright and Matthias Felleisen. We can view evaluation as a partial function

$$eval : Expr \rightharpoonup Value \cup \{\text{WRONG}\}$$

$eval$ maps an expression e to either value v or to WRONG, indicating a type error. The function is partial: the result is undefined if e does not terminate.

The simplest way to state a soundness property is that well-typed programs don't go WRONG:

Definition (Weak soundness). If $\vdash e : \tau$, then $eval(e) \neq \text{WRONG}$.

We use WRONG as shorthand for any expression e that has a type error. For example, `true true` or `if` $(\lambda x : \tau.\ e)$ `then` 0 `else` 1.

A stronger notion of soundness views a type τ as denoting a *set of values* V^τ. For instance, the type `Bool` denotes the set $V^{\texttt{Bool}} = \{\texttt{false}, \texttt{true}\}$.

Definition (Strong soundness). If $\vdash e : \tau$, and $eval(e) = a$, then $a \in V^\tau$.

The definition says that if a well-typed expression evaluates to an answer a, then a is a value of type τ. Strong soundness implies weak soundness because WRONG is not in any set V^τ.

Now, we can restate the definition of strong soundness purely syntactically by observing that

- $eval(e) = a$ iff $e \longrightarrow^* a$
- if $a \neq$ WRONG, then a is a value v
- $v \in V^\tau$ iff $\vdash v : \tau$.

We need to be careful to say that expressions might not evaluate to a value. We thus define *normal forms*:

Definition (Normal forms). e is a *normal form* if there is no e' such that $e \longrightarrow e'$.

Note that all values are in normal form, and all “stuck” expressions are also in normal form.

We thus have the following theorem:

Theorem (Soundness). If $\vdash e : \tau$ and $e \longrightarrow^* e'$ and e' is in normal form, then e' is a value v and $\vdash v : \tau$.

Note that the theorem assumes that e will evaluate to a normal form. It says nothing about expressions that *diverge*, that is, that go into infinite loops. This is okay. If an expression is going to get stuck, it will do so *before* going into an infinite loop. It cannot get stuck *after* an infinite loop because—being infinite—the loop will never terminate; talking about what happens after an infinite loop is meaningless.

To prove the theorem we show two properties: progress and type preservation. Progress states that a well-typed expression is either a value or it can take a step to another expression (i.e., is never “stuck”). Preservation states that if an expression e has a given type and it steps to another expression e', then e' has the same type. Put together, if e is not a value, e must be able to take a step (i.e., it does not get “stuck”) to another expression with the same type. Soundness follows immediately by induction on the number of steps taken.

The slogan is “well-typed programs do not get stuck.”

Lemma (Progress). If $\vdash e : \tau$, then either e is a value v or there is an e' such that $e \longrightarrow e'$.

Proof. By structural induction on e. Consider e by cases.

- $e = x$. Vacuous (x is ill-typed in the empty environment).
- $e = \texttt{true}$. Trivial (already a value).
- $e = \texttt{false}$. Trivial (already a value).
- $e = \lambda x.\ e_1$. Trivial (already a value).
- $e = \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2$. Since $\vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau$, by T-IF, we have $\vdash e_0 : \texttt{Bool}$, $\vdash e_1 : \tau$, and $\vdash e_2 : \tau$. Since $\vdash e_0 : \texttt{Bool}$, by the induction hypothesis either e_0 is a value or e_0 steps to e_0'. If e_0 is a value, there are three cases:
  - $e_0 = \texttt{true}$. Then by `E-IfTrue`, $e' = e_1$.
  - $e_0 = \texttt{false}$. Similar to the previous case.
  - $e_0 = \lambda x.\ e_0'$. In this case, $\vdash e_0 : \tau_0 \to \tau_0'$, but we already have $\vdash e_0 : \texttt{Bool}$. Therefore, this case cannot occur.

  If e_0 is not a value, then it must step to e_0'. By EC-IF, $e' = \texttt{if}\ e_0'\ \texttt{then}\ e_1\ \texttt{else}\ e_2$.
- $e = e_1\ e_2$. Since $\vdash e_1\ e_2 : \tau$, by T-APP, we have $\vdash e_1 : \tau_2 \to \tau$, and $\vdash e_2 : \tau_2$. Let’s consider e_1 and e_2 by cases.
  - If e_1 is not a value, then by the induction hypothesis, there is an e_1' such that $e_1 \longrightarrow e_1'$. By EC-LEFT, $e' = e_1'\ e_2$.
  - If e_1 is a value but e_2 is not, then by the induction hypothesis, there is an e_2' such that $e_2 \longrightarrow e_2'$. By EC-LEFT, $e' = e_1\ e_2'$.

- Finally, if both e_1 and e_2 are values, let's consider e_1 by cases.
    - $e_1 = \texttt{true}$. Then $\vdash e_1 : \texttt{Bool}$. But we already have $\vdash e_1 : \tau_2 \rightarrow \tau$. Therefore, this case cannot occur.
    - $e_1 = \texttt{false}$. Similar to the previous case.
    - $e_1 = \lambda x.\ e_1'$. By E-BETA, $e' = e_1'[x \mapsto e_2]$.

Lemma (Preservation). If $\Gamma \vdash e : \tau$ and $e \longrightarrow e'$, then $\Gamma \vdash e' : \tau$.

Proof. By structural induction on e. Consider e by cases.

- $e = \texttt{true}$ or $e = \texttt{false}$. Vacuous (a value; cannot take a step).
- $e = \lambda x.\ e_1$. Vacuous (a value; cannot take a step).
- $e = x$. Vacuous (cannot take a step).
- $e = \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2$. Since $\Gamma \vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau$, by T-IF, we have $\Gamma \vdash e_0 : \texttt{Bool}$, $\Gamma \vdash e_1 : \tau$, and $\Gamma \vdash e_2 : \tau$. Since $e \longrightarrow e'$ we have three cases:
    - E-IFTRUE. Then $e_0 = \texttt{true}$ and $e' = e_1$. By T-IF we have the premise $\Gamma \vdash e_1 : \tau$, so we're done.
    - E-IFFALSE. Similar to the previous case.
    - EC-IF. Then $e_0 \longrightarrow e_0'$ and $e' = \texttt{if}\ e_0'\ \texttt{then}\ e_1\ \texttt{else}\ e_2$. Since $\Gamma \vdash e_0 : \texttt{Bool}$, by the induction hypothesis we have $\Gamma \vdash e_0' : \texttt{Bool}$. Thus, by T-IF, we can derive $\Gamma \vdash \texttt{if}\ e_0'\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau$. Done.
- $e = e_1\ e_2$. Since $\vdash e_1\ e_2 : \tau$, by T-APP, we have $\Gamma \vdash e_1 : \tau_2 \rightarrow \tau$, and $\Gamma \vdash e_2 : \tau_2$. Since $e \longrightarrow e'$ we have three cases:
    - EC-LEFT. Then $e_1 \longrightarrow e_1'$ and $e' = e_1'\ e_2$. By the induction hypothesis $\Gamma \vdash e_1' : \tau_2 \rightarrow \tau$. Together with $\Gamma \vdash e_2 : \tau_2$, by T-APP, we can derive $\Gamma \vdash e_1'\ e_2 : \tau$.
    - EC-RIGHT. Similar to the previous case.
    - E-BETA. We have $e_1 = \lambda x : \tau_2.\ e_1'$, and we know e_2 is a value v, and $e' = e_1'[x \mapsto v]$. By T-ABS, we have $\Gamma, x : \tau_2 \vdash e_1' : \tau$ and by T-APP we have $\Gamma \vdash v : \tau_2$. We need to show that $\Gamma \vdash e_1'[x \mapsto v] : \tau$. To show this, we use the substitution lemma below, which immediately gives us the desired result.

Lemma (Substitution preserves types). If $\Gamma, x : \tau' \vdash e : \tau$ and $\Gamma \vdash v : \tau'$, then $\Gamma \vdash e[x \mapsto v] : \tau$.

Proof. The proof is by induction on the height of the derivation of $\Gamma, x : \tau' \vdash e : \tau$. Consider e by cases.

- $e = \texttt{true}$ or $e = \texttt{false}$. Trivial.
- $e = y \neq x$. Trivial.
- $e = x$. Then $e[x \mapsto v] = v$ and $\tau = \tau'$. By assumption $\Gamma \vdash v : \tau'$. Done.
- $e = \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2$. By T-IF, we have $\Gamma, x : \tau' \vdash e_0 : \texttt{Bool}$, $\Gamma, x : \tau' \vdash e_1 : \tau$, and $\Gamma, x : \tau' \vdash e_2 : \tau$. By the induction hypothesis, we have: $\Gamma \vdash e_0[x \mapsto v] : \texttt{Bool}$, $\Gamma \vdash e_1[x \mapsto v] : \tau$, and $\Gamma \vdash e_2[x \mapsto v] : \tau$. Thus, we can derive $\Gamma \vdash \texttt{if}\ e_0[x \mapsto v]\ \texttt{then}\ e_1[x \mapsto v]\ \texttt{else}\ e_2[x \mapsto v] : \tau$. Using the definition of substitution, we therefore have: $\Gamma \vdash (\texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2)[x \mapsto v] : \tau$. Done.
- $e = e_1\ e_2$. Similar to the previous case.
- $e = \lambda y : \tau_1.\ e_2$. By T-ABS, $\tau = \tau_1 \rightarrow \tau_2$. If $x = y$, then $e[x \mapsto v] = e$ and we're done. Otherwise, $e[x \mapsto v] = (\lambda y : \tau_1.\ e_2)[x \mapsto v] = \lambda y : \tau_1.\ e_2[x \mapsto v]$. By T-ABS, we have $\Gamma, x : \tau', y : \tau_1 \vdash e_2 : \tau_2$. Since $x \neq y$, we can rearrange the typing context and get $\Gamma, y : \tau_1, x : \tau' \vdash e_2 : \tau_2$. By the induction hypothesis, $\Gamma, y : \tau_1 \vdash e_2[x \mapsto v] : \tau_2$. Applying T-ABS, we have $\Gamma \vdash \lambda y : \tau_1.\ e_2[x \mapsto v] : \tau_1 \rightarrow \tau_2$. Done.